Fuzzing the Physical Space: Physics-Aware Testing of Black-Box Industrial Control Systems

Burak Sahin, David Oygenblik, Mingxuan Yao, Yizhi Huang, Brendan Saltaformaggio, Saman Zonouz
Georgia Institute of Technology

***Abstract*—Industrial Control Systems (ICS) operate essential physical processes in critical infrastructure sectors such as energy, water, and transportation. Existing ICS fuzzing techniques often assume white-box access or modify controller firmware, and black-box methods uncover only corrupted inputs, leaving a critical gap in detecting *physical security violations* that emerge gradually from valid commands on locked-down controllers. We present ICSFlux, the first physics-aware black-box fuzzing framework that systematically discovers physical security violations. ICSFlux leverages physical models and physical security constraints (defined during the ICS design phase) to infer the temporal evolution of physical states. By estimating potential trajectories that converge toward unsafe physical states, ICSFlux partitions the physical space by proximity to violation and directs test generation toward high-risk partitions. This physics-guided approach generalizes across heterogeneous ICS deployments and eliminates reliance on application-specific heuristics. We evaluate ICSFlux on 11 industrial testbeds (e.g., chemical manufacturing plant and water treatment utility) and discover 20 physical security violations.**

1. Introduction

Critical infrastructure sectors such as energy, water, and transportation rely on Industrial Control Systems (ICS) to operate essential physical processes [1]–[3]. The ICS lifecycle begins with the design phase and continues through procurement, installation, and commissioning [4]. Each stage builds on the design foundation, where the physical behavior of the system is first modeled. After the design phase, ICS deployment moves through procurement, installation, and commissioning [4], [5].

In the design phase, *physical models* and *physical security constraints* are defined for ICS physical processes [6]–[9]. The physical model [9]–[12] is a mathematical abstraction that formulates how a physical process – such as heating, pressure control, or fluid flow – responds to changes in its environment. Physical security constraints [13], [14] define safe operating conditions to prevent system failures and harm.

If a physical process is driven into physical states that violate physical security constraints, the result can be severe physical damage [15]–[26]. We refer to such outcomes as *physical security violations*. For example, a chemical production plant [27] must operate efficiently while maintaining physical security. The reactor pressure evolves according to the physical model (Eq. 17-21 in §A.1). The system must enforce its physical security constraint (Eq. 22) to prevent over-pressurization, which can ultimately lead to a reactor explosion (§3). This study focuses on revealing physical security violations.

ICS spans diverse physical processes across sectors, with proprietary controllers typically locked down [28], [29]. Some fuzzing approaches rely on white-box access and target memory corruption bugs [30]–[32]. Others focus on semantic bugs but still assume white-box access [33]–[35], modify controller internals such as control parameters [33], [34], or solely focus on finite-state logic [35]. Meanwhile, black-box approaches [36] uncover peripheral issues (e.g., misconfiguration or perception faults). They [33], [34], [36] do not uncover physical security violations that emerge only from valid commands on a locked-down controller. They also rely on application-tailored heuristics, limiting their generalizability across diverse ICS environments.

This gap motivates our focus on the following key insights. Although ICS encompasses heterogeneous physical processes, their physical models and physical security constraints are specified during the design phase. Physical models capture the temporal dynamics needed to reason about how states evolve over time. From this temporal evolution, we define a notion of physical space proximity: the distance between a current physical state and a physical security violation, measured by the time needed to reach that violation. By leveraging physical space proximity, we can structure fuzzing to achieve more comprehensive coverage of the physical space and direct scenario mutations toward likely physical security violations.

Building on these insights, we present ICSFlux, a physics-aware fuzzing framework for industrial control systems with black-box controllers. ICSFlux takes the physical model and physical security constraint as inputs and reasons about the temporal and causal evolution of the physical process. It first extracts causal differential constraints (§4.1) that characterize how actuator changes influence physical state variables, then constructs a causal evolution graph (§4.2) that captures directed, time-dependent transitions between physical states. Using this graph, ICSFlux partitions the physical state space into spatio-temporal causal compartments (§4.3), each bounded by empirically inferred constraints. These artifacts collectively form a constrained physical state space that

encodes the causal relations between controller actions and physical process behavior. Guided by this structure, ICSFlux systematically explores the physical state space to uncover violations that evolve gradually over time. During fuzzing, the physics-guided feedback engine (§4.4) monitors live process evolution to identify high-risk goals and directions, while the mutation engine (§4.5) perturbs operator commands and events in real time to drive the physical process toward unsafe outcomes. Together, these components enable physics-guided fuzzing that generalizes across eleven controllers in six ICS domains, revealing more than twenty physical security violations.

2. Preliminaries

ICS and Control Software. ICS regulates physical processes across various sectors such as energy, water, and manufacturing [1]–[3]. At their core lies the controller, which executes deterministic scan cycles to maintain process stability: it continuously (1) reads sensor inputs, (2) executes control logic, and (3) writes outputs to actuators [37]. This predictable timing forms the closed feedback loop shown in Fig. 1. A human–machine interface (HMI) enables operators to visualize system states and issue supervisory commands such as setpoint changes or manual overrides. Supervisory control and data acquisition (SCADA) systems aggregate telemetry from multiple sites and coordinate these commands across controllers.

Physical Models and Physical Security Constraints. In ICS design, engineers specify a physical model and physical security constraints as the foundation of the system [4], [7]. A physical model (e.g., $\dot{x} = Ax + Bu$ [9], [10], [38], [39]) captures the temporal dynamics of the physical process. It describes how states evolve in response to control actions and environmental changes.

Physical security constraints define the safe and unsafe operating regions of critical variables such as temperature, flow, pressure, and voltage. Violating these constraints can damage equipment or cause harm, so control software must prevent such violations. Both the physical model and physical security constraints guide subsequent stages, including control software design, commissioning, and runtime. They must be carefully developed and aligned with real-world dynamics. To securely operate ICSs, accurate models and well-defined constraints are essential. Our approach builds directly on these models and constraints to enable testing and fuzzing.

Black-box Controllers in ICS. In deployment, physical processes are delivered with vendor-supplied control software [42], [43]. This software is typically proprietary and inaccessible to ICS operators, whether it is coordinating robot arms [5], [42], stabilizing drones [44], [45], or regulating HVAC systems [43]. Vendors instead provide high-level configurability through coarse-grained state-machine interfaces (e.g., walk, turn, heat on/off) [5], [44], [46]–[48]. As a result, the underlying controller operates as a black box: the embedded control logic cannot be instrumented or modified, and interaction is limited to observable inputs, outputs, and the resulting physical process behavior. Unlike prior work requiring white-box controllers [30]–[35], [47], this paper focuses exclusively on black-box controllers.

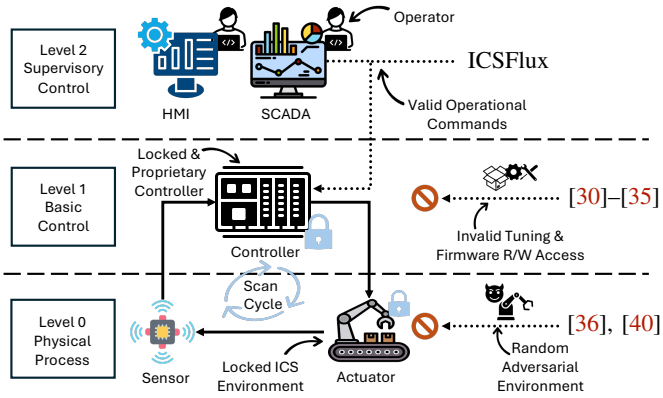

Figure 1: ICSFlux Threat Model. The figure situates ICSFlux within the Purdue model [41], operating at Level 2 on supervisory components. Dotted arrows mark typical attack surfaces. Prior works assume access to controller firmware [30]–[35] (Level 1) or modifiable adversarial environments [36], [40] (Level 0), which are infeasible assumptions for rigid, locked industrial systems.

Locked Physical and Control Layers. As shown in Fig. 1, *Level 0* physical processes in industrial plants operate within fixed, equipment-bounded environments. Hence, the attacker cannot introduce new physical components or alter the plant layout (e.g., inserting an adversarial valve or robotic arm) [8]. Likewise, *Level 1* control parameters—such as PID gains, filter coefficients, and calibration constants—are stored in protected memory regions and remain write-protected during normal operation [28], [29], with modifications permitted only in engineering or maintenance modes. Unlike prior work that assumes adversarial environments [36], [40] or modifies controller parameters [33], [34], ICSFlux operates within the rigid ICS operating environment, relying solely on valid operator commands issued from *Level 2*, the only layer permitted to influence ICS behavior during live operation.

3. Motivation

Physical Security Violations. Unlike typical software failures (e.g., memory corruption), these physical security violations often build up gradually as the controller, environmental conditions, and physical processes interact over time. For example, a chemical plant can exceed its physical security constraint through combinations of control commands over time, eventually leading to an explosion.

Threat Model. We assume an adversary can gain control of or otherwise misuse supervisory components such as human–machine interfaces (HMIs) or server-side applications (e.g., SCADA). This foothold may be achieved remotely, locally, or through insider access [49], [50]. The attacker's

key capability is issuing seemingly benign, valid operator commands, such as a pressure setpoint within safe limits, crafted to destabilize the physical process [51]. Regardless of the initial compromise vector, ICSFlux focuses on revealing physical security violations by exploring how valid operator commands (and environmental conditions) interact with the trusted control software and the underlying physical process.

As shown in Fig. 1, we assume the adversary cannot alter the underlying control stack or physical environment of the ICS. At *Level 0*, the physical process is bounded by real-world physics and the plant's installed actuators and sensors, preventing the attacker from adding new physical elements or imposing arbitrary adversarial dynamics [8]. At *Level 1*, controller configurations, such as PID gains, calibration constants, and other tuning parameters, are fixed after commissioning and remain protected during operation [28], [29]. As a result, techniques that rely on mutating the physical environment [36], [40] or modifying controller internals or firmware [33], [34] fall outside our scope. Instead, the adversary's actionable surface is *Level 2*, where they can misuse supervisory interfaces (e.g., HMI/SCADA) to issue valid but strategically crafted operator commands that influence the controller and drive the physical process toward physical security violations.

3.1. Motivating Example

To highlight the challenge of uncovering physical security violations in locked-down controllers, we present a scenario from a chemical-plant simulation [27], illustrated in Fig. 3. A controller maintains reactor pressure P below a maximum safe threshold $P_{\max}$. An attacker cannot modify controller internals such as control parameters or bypass input validation checks, but can issue valid operator commands. Their goal is to find a timed command sequence that drives the physical process into a physical security violation $P > P_{\max}$.

Initially, the plant operates safely. However, inspired by recent events, the adversary begins issuing small, valid setpoint adjustments, which appear indistinguishable from normal operator input. Over time, these seemingly benign actions shift the reactor into a physical state where pressure becomes highly sensitive to specific commands. When further commands are issued, such as increasing production setpoint and decreasing waste, the physical dynamics amplify their effect, as production overrules pressure control, causing pressure to surge past $P_{\max}$ and trigger a physical security violation.

The attacker need not compromise the controller or modify controller internals. Instead, the adversary operates at higher layers (Level 2 and above). This creates an asymmetry between the attacker and the defenders trying to discover physical security violations.

Therefore, finding such sequences is challenging because the controller is a black box. It continuously computes real-valued actuator commands using PID control, leaving no discrete state-machine transitions to flip or enumerate. Its internals and control parameters are locked down, and input validation blocks malformed or out-of-range commands. As a result, existing fuzzers that assume white-box visibility, modify controller behavior, or search for peripheral faults cannot expose the subtle control-logic failures that arise only from valid, timed commands interacting with the physical process.

How ICSFlux Investigates Physical Security Violations. ICSFlux takes as input (1) the plant's physical model and (2) physical security contraint. **Step 1:** ICSFlux sends valid operator commands and observes resulting actuator and state changes. This step creates causal samples that link operator commands to physical model variables. **Step 2:** From causal samples, ICSFlux identifies causal relations between commands, actuators, and physical states under physical constraints. For example, increasing the production setpoint lowers purge flow and raises reactor pressure (Table 3 Row 7). **Step 3:** Using these relations together with the physical model and the physical security constraint, ICSFlux performs reverse reasoning from the violation space. It constructs a causal evolution graph that captures physical states and transitions that could potentially lead to physical security violations. **Step 4:** Using the graph, ICSFlux clusters related physical states into compartments based on their temporal proximity to the violation space. Each compartment captures the relevant physical constraints and causal relations, forming a compact representation of the search space. **Step 5:** Finally, ICSFlux explores transitions between groups to broaden coverage while the groups themselves constrain initial-state mutations and reduce the search space. It then selects a concrete goal (e.g., $P > P_{\max}$) and uses causal relations to guide short, feasible mutations toward that goal. For example, to raise pressure, ICSFlux increases the product valve setpoint (which tends to raise P) and applies complementary tweaks (e.g., lowering a cost-related setpoint) that further increase pressure. These mutations can produce valid command sequences that override pressure control and drive P beyond $P_{\max}$, even when the pressure setpoint is below $P_{\max}$. This physics-guided strategy narrows the mutation search space. See Table 1 Row 1-7 for a detailed instance.

Note. The physical guidance, including constraints, causal relations, the graph, and the groups, is approximate. Physical models are never perfectly accurate or fully comprehensive, and these relations are empirically inferred because the controller behaves as a black box. To account for this uncertainty, ICSFlux applies small physics-guided mutations around the inferred results to introduce diversity in exploration and uncover feasible attack paths.

4. Design

The design of ICSFlux follows a sequential pre-fuzzing and runtime feedback workflow that transforms raw physical data into guided exploration of the physical state space. It begins with runtime probing, which collects synchronized measurements of control inputs, actuator signals, and physical states from the target process. From these samples, ICSFlux extracts causal differential constraints (§4.1) that capture how actuator or control

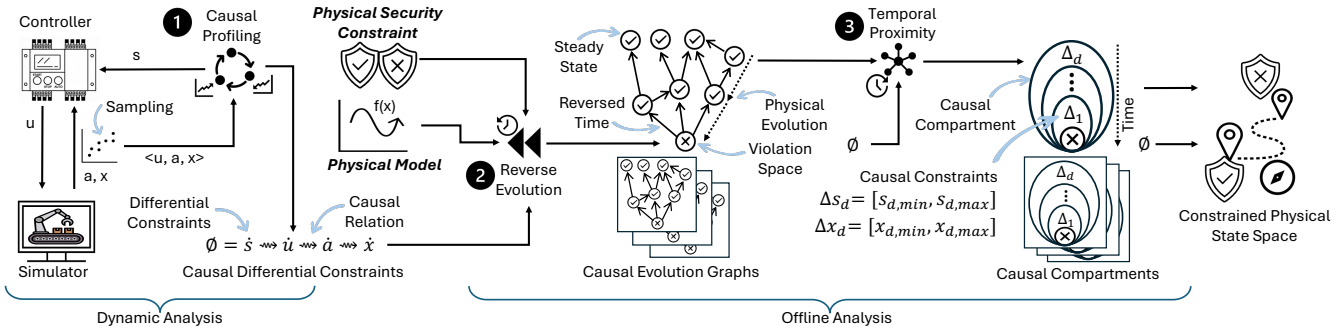

Figure 2: Pre-fuzzing analysis pipeline constructing the constrained physical state space. Starting from runtime probing, the system extracts causal differential constraints ❶, builds causal evolution graphs ❷, and partitions them into spatio-temporal causal compartments ❸. Together, these steps produce a constrained physical state space bounded by physical constraints and causal relations. The constrained physical state space serves as the foundation for guiding physics-aware mutation and exploration. **Inputs:** physical security constraint and physical model.

changes influence physical variables. These causal relations are then embedded into a causal evolution graph (§4.2), describing how valid control behaviors evolve over time. To manage the large and noisy space of states, ICSFlux partitions the graph into spatio-temporal causal compartments (§4.3)—regions that group physically and causally consistent states. Together, these components construct the constrained physical state space (CPSS), which bounds feasible state evolution under known physics and control dependencies. Finally, the physics-guided feedback engine (§4.4) and mutation engine (§4.5) operate on this constrained space during fuzzing, continuously generating and adapting actuator or command mutations to explore untested transitions and reveal potential physical security violations.

4.1. Extracting Causal Differential Constraints

ICS often operate with locked-down controllers whose internals are inaccessible. To understand how these controllers interact with the underlying physical process, ICSFlux performs perturbations on valid operator commands and observes the resulting actuator and process responses ❶. These observations form *causal differential constraints* (CDC) that express how command changes influence control inputs, actuators, and physical states.

ICSFlux empirically couples the controller with the physical process through symmetric perturbations of one command at a time. Let $\phi = (s, u, a, x)$ represent the combined vector of user commands, control inputs, actuator signals, and physical states. During symmetric probing, $\phi^+ = (s^+, u^+, a^+, x^+)$ and $\phi^- = (s^-, u^-, a^-, x^-)$ denote the system responses under positive and negative perturbations of a single command s_j (Eq. 1):

$$\begin{aligned} \phi^+ &\leftarrow B(s_t + \Delta s_j) \\ \phi^- &\leftarrow B(s_t - \Delta s_j) \end{aligned} \tag{1}$$

Here, s_t is the current vector of user commands, and only the jth is perturbed by a Δs_j. $B(\cdot)$ represents the ICS, returning the tuple $\phi = (s, u, a, x)$ for each input. These symmetric evaluations (positive and negative) yield paired samples that isolate the direct influence of a specific command on control inputs and physical responses (Line 4).

After obtaining paired samples, ICSFlux estimates derivatives using symmetric finite differences in the command space. This provides an interpretable approximation of how each command channel affects control outputs, actuators, and states (Eq. 2):

$$\begin{bmatrix} \dot{u} \\ \dot{a} \\ \dot{x} \end{bmatrix} \approx \frac{1}{2\,\Delta s_j} \begin{bmatrix} u^+ - u^- \\ a^+ - a^- \\ x^+ - x^- \end{bmatrix} \tag{2}$$

These expressions yield both the direction (sign) and magnitude (sensitivity) of the responses to command perturbations (Line 5). The probing loop continues until the observed differential estimates converge (Line 5), which monitors changes in the variance of recent samples to ensure convergence. This ensures that additional samples do not significantly alter the inferred relationships. This differential structure (D) converts raw empirical data into structured quantities suitable for causal modeling.

To determine the influence of each command or actuator on its successors, ICSFlux fits linear relations that summarize directional dependencies. The resulting coefficients encode monotonic relationships observed in the data and form the basis for causal reasoning.

Each pair (y, ξ) represents a possible causal edge, where ξ is an influencing variable (e.g., command or actuator) and y is an affected variable (e.g., actuator or state). Before constraining, Line 9 extracts the causal relations,which identifies and groups all variable pairs such as $s \rightarrow u$ and $u \rightarrow a$.

$$\begin{aligned} \beta(y \leftarrow \xi) &= \arg\min_{\beta} \sum_{t} (y_t - \beta \xi_t)^2, \\ \text{sign}(y \leftarrow \xi) &= \text{sign}(\beta), \\ \text{strength}(y \leftarrow \xi) &= |\beta| \cdot \text{std}(\xi) \\ \text{snr}(y \leftarrow \xi) &= \text{var}(\beta\xi)/\text{var}(y - \beta\xi). \end{aligned} \tag{3}$$

Algorithm 1: Pre-fuzzing pipeline for each violation term in a physical security constraint to construct the constrained physical state space.

Input: black-box controller B; physical model PM; target violation term $PSC(x_{tgt})$ within the physical security constraint
Output: constrained physical state space CPSS $= (\texttt{comps}, M_{\text{CDC}})$

// ❶ Collecting dynamic control variables
1 $D \leftarrow \emptyset$;
2 **foreach** $j \in J$ **do**
3 **while *not*** $\text{Converged}(D \mid j, \varepsilon)$ **do**
4 $\phi^+, \phi^- \leftarrow \text{ICSProbe}(B, s_t, \Delta s_j)$;
5 $D \leftarrow D \cup \{\langle t, j, \phi_t, \dot{\phi}_j \rangle\}$;

// Extracting causal differential constraints
6 $P \leftarrow \{\langle y, \xi \rangle : \xi \in \{s, u, a\},\ y \in \{u, a, x\}\}$;
7 $M_{\text{CDC}} \leftarrow \emptyset$;
8 **foreach** $\langle y, \xi \rangle \in P$ **do**
9 $S \leftarrow \text{Select}(D, y, \xi)$;
10 $(\beta, \text{sgn}, \text{imp}) \leftarrow \text{EstimateInfluence}(S, \xi)$;
11 $M_{\text{CDC}} \leftarrow M_{\text{CDC}} \cup \{\langle y, \xi, \text{sgn}, \text{imp}, \text{snr} \rangle\}$;

// ❷ Building causal evolution graph
12 $CEG \leftarrow \emptyset$; $X_{\text{tgt}}^{(0)} \leftarrow \text{UnsafeRegion}(\mathcal{PSC}(x_{tgt}))$;
13 $CEG.\text{add\_node}(X_{\text{tgt}}^{(0)})$;
14 $X_{\text{tgt}} \leftarrow X_{\text{tgt}}^{(0)}$;
15 **while *not*** $\text{IsEmpty}(X_{tgt})$ **do**
16 $\text{States} \leftarrow \text{ReverseEvolution}(X_{tgt}, M_{CDC}, PM)$;
17 $\text{States} \leftarrow \text{FilterPSV}(\text{States}, \text{PSC}(x_{tgt}))$;
18 $\text{AddPredecessor}(CEG, X_{tgt}, \text{States})$;
19 $X_{tgt} \leftarrow \text{UpdateTargets}(\text{States})$;

// ❸ Building causal compartments
// actuator->target state sensitivity
20 $r_T \leftarrow \text{EstimateJacobian}_{xa}(PM, M_{CDC})$;
// compartmentalizaton by target x_{tgt}
21 $da \leftarrow \text{ActuatorDeltaBounds}(M_{CDC})$;
22 $dx_{tgt} \leftarrow \text{TargetStepInterval}(r_T, da)$;
23 $C_i \leftarrow \text{BuildBands}(x_{\text{tgt}}^{(0)}, dx_{tgt})$;
24 $comps \leftarrow \emptyset$; $visited \leftarrow \emptyset$;
25 $Q \leftarrow \text{InitQueue}(CEG.root)$;
26 **while *not*** $\text{IsEmpty}(Q)$ **do**
27 $node \leftarrow \text{pop}(Q)$;
28 $i \leftarrow \text{LocateBandByTarget}(node.x_{\text{tgt}}, C_i)$;
29 $K \leftarrow \text{GetCompartment}(i, comps)$;
30 $K.add(node)$;
31 $push(Q, node.prev_nodes)$;

// constraining all physical variables
32 **foreach** $K \in comps$ **do**
  // for each $x_i \in x$
33 $K.constraints \leftarrow MinMaxVariables(K.nodes)$;
34 $K.relations \leftarrow InferLocalCausalRelations(K.nodes)$;
35 $CPSS \leftarrow (\texttt{comps}, M_{\text{CDC}})$;
36 **return** $CPSS$

Eq. 3 shows how ICSFlux creates the causal differential constraints (Line 10). β is the linear coefficient that quantifies how strongly an influencing variable affects the affected variable. β obtains the sign (direction of effect), strength (magnitude of effect), and signal-to-noise ratio. These results are stored in a structured form that summarizes the directional relationships between all variables (Line 11). The resulting matrix M_{CDC} aggregates edges that capture the empirical causal hierarchy ($\dot{s} \rightsquigarrow \dot{u} \rightsquigarrow \dot{a} \rightsquigarrow \dot{x}$), serving as the foundation for constructing the *causal evolution graph* (§4.2) and for guiding *physics-informed feedback engine* (§4.4).

4.2. Constructing Causal Evolution Graph

After extracting the causal differential constraints (§4.1), ICSFlux constructs a structured and temporal representation of how the physical state evolves under valid operator commands ❷. This structure, called the *causal evolution graph*, captures how continuous changes in commands influence actuator behavior and physical state transitions. Due to the black-box setting, the causal evolution graph provides an approximated but causally consistent structure showing how the physical state would evolve under valid commands toward physical security violations.

The causal evolution graph construction uses three key inputs: (1) causal differential constraints M_{CDC} learned from empirical probing; (2) the physical model, and (3) physical security constraints, given as inputs to ICSFlux. Together, these allow ICSFlux to generate physically plausible predecessor states that could have led to a physical security violation. The construction process begins in the violation space and recursively computes predecessors through feasible dynamics. The causal evolution graph is used to recursively generate potential predecessor physical states that trace how physical operations would drift toward physical security violations.

The reversed propagation begins by anchoring the graph to the violation region defined by the physical security constraints (Line 12-13). X_{tgt}^0 represents the final violation space. Closed-loop control systems are commonly approximated by a linearized model of the form $x_{t+1} \approx Ax_t + Bu_t$. x is the physical state vector, and u is the control input. Matrices A and B describe how current states and control actions influence the next state. However, the control law $u = \pi(x)$ or the matrix B is unknown when the controller is a black box. Therefore, ICSFlux replaces the unknown control inputs with the empirically learned causal differential constraints. ICSFlux constructs a hybrid Jacobian by combining the physical model and M_{CDC} (Line 16). ReversePropagation (Line 16) computes $A_{PM} = \partial f_{\text{PM}}(x, a)/\partial x$ and $B_{CDC} = \partial M_{CDC}(x, s)/\partial s$ (Line 16). These Jacobians capture the sensitivity of the physical model and the effect of the empirical controller, respectively. ICSFlux uses the following equation to compute potential predecessors:

$$x_{t-1} \approx f_{PM}^{-1}(x_{tgt} - B_{CDC}(s)) \tag{4}$$

After generating predecessor candidates, FilterPSV() (Line 17) filters out states that violate the physical security constraint. Surviving states are inserted into the graph using $AddPredecessor()$ (Line 18), linking each x_{pre} to its successor x_{tgt} through the corresponding differential reverse propagation using Eq. 4. This process is repeated until there are no more new valid predecessor states (Line 15)

The resulting causal evolution graph is an approximated backward structure that shows how valid operator commands can evolve toward physical security violations while remaining consistent with both the physical model and the empirical controller effect from causal differential

constraints. It provides the substrate for causal compartmentalization (§4.3), which groups these states by temporal proximity for guiding mutation.

4.3. Creating Causal Compartments

The purpose of this component is to cluster nodes in the causal evolution graph (§4.2) into spatio-temporal compartments that capture physically and causally consistent behavior towards the physical security violation boundary ❸. Because the controller is a black box, ICSFlux cannot directly access its control law or internal states. Instead, we approximate backward-reachable regions using the actuator–to–state sensitivities ($\partial x/\partial a$ obtained from the physical model and the actuator change bounds provided by the causal differential constraints. This approach allows ICSFlux to infer which causaliites in past states could lead to the target physical security violation, grouping those states into spatio-temporal compartments defined by feasible x-intervals.

ICSFlux begins by estimating how actuator bounds translate into bounded changes of the target variable x_{tgt}, and then propagates these relations through conservative accumulation to form reverse compartments.

Due to the black-box setting, the partial derivative ($\partial x/\partial a$) is computed specifically for the target variable x_{tgt} (Line 20), meaning the resulting constraints apply only to that target dimension and guide the formation of x_tgt-based compartments. Additional constraints for the remaining state variables are computed later (Line 32, further reducing the mutation search space by bounding all x dimensions. ICSFlux computes the actuator-to-target sensitivity vector ($r^T = \partial x_{tgt}/\partial a$ in Line 20) . Together with actuator delta boxes (Line 21), these captures the minimum and maximum possible target variable change under the learned actuator constraints (Line 22):

$$[\delta x_{tgt,\min},\, \delta x_{tgt,\max}] = \sum_j \Big[\min(r_j\,\Delta a_{\min,j},\, r_j\,\Delta a_{\max,j}), \max(r_j\,\Delta a_{\min,j},\, r_j\,\Delta a_{\max,j}) \Big]. \tag{5}$$

Eq. 5 defines the basic target-variable displacement interval. The algorithm extends these bounds conservatively by accumulating feasible displacements across reachable regions (Line 20-22). This operation ensures all possible x_{tgt} variations are enclosed and provides the cumulative range for each reversed compartment:

$$\delta x_{tgt} \in [i\delta x_{tgt,min}, i\delta x_{tgt,max}]. \tag{6}$$

Using this range (Eq. 6), ICSFlux constructs the reversed compartments (Line 23). Each compartment C_i corresponds to the feasible target-state range that could precede the violation by i^th layers:

$$C_i = \big[x^0_{tgt} - i\,\delta x_{tgt,\max},\; x^0_{tgt} - (i-1)\,\delta x_{tgt,\min}\big]. \tag{7}$$

ICSFlux traverses the causal evolution graph using a reversed queue initialized at x^0_{tgt} (Line 24-31). For every node, the algorithm reads its target state value ($node.x_{tgt}$), locates the matching interval (Eq. 7), and assigns it to K_i (Line 28). If a node's predecessors have not been visited, they are pushed onto the queue to continue exploration. This traversal organizes the nodes in the causal evolution graph according to their temporal proximity.

Finally, ICSFlux performs the full-dimensional constraining by bounding all physical state variables (Line 33-34). Once all compartments are finalized, ICSFlux computes per-variable bounds for every variable $z = [x; a]$ across all dimensions, producing range constraints $l^(_k i) \le z_k \le u^(_k i)$ for each partition K_i (Line 33). ICSFlux then fits data-driven local causal relations among the physical state and actuator variables inside each partition (Line 34). These steps generalize the single-target sensitivity derived in Line 20-22 to a complete local model capturing multivariate relations and envelopes for each compartment.

4.4. Physics-Guided Feedback Engine

The physics-guided feedback engine serves as the adaptive guidance layer, steering the mutation process through the constrained physical state space (Alg. 2). It monitors the current state of physical evolution, determines the direction of physical evolution, and continuously adjusts actuation strategies. The feedback engine aims to maximize violation discovery potential while ensuring the generated commands are valid.

Monitoring. At each iteration, the feedback engine reads the current state vector $\zeta = (z = [x; a], \tau)$ where x denotes the physical state, a the actuator vector, and τ the current timestamp. The physics-guided feedback engine identifies the active compartment K_{cur} and neighboring candidates C_{next} using constrained physical state space $CPSS$:

$$K_{cur} = \{\, K \in CPSS.\texttt{comps} \mid x_{tgt} \in [l^{(K)}tgt, u^{(K)}tgt] \,\} \tag{8}$$

This operation anchors the simulation to valid physical regions, ensuring that subsequent mutations and decisions are always constrained by feasible physics (Line 5–6).

Goal Selection. Each neighboring compartment candidate $c \in C_{next}$ is evaluated according to its risk score (Eq. 9):

$$\text{risk}(c) = 1 - \frac{\min_{x \in E_c} \text{dist}(x, \partial\text{Goal})}{\text{dist}_{\max}} \tag{9}$$

The risk score quantifies how close the current target variable x_{tgt} is to the goal boundary or transition target.ICSFlux guides search space towards the goal or targeted transition (Line 7).

Direction Selection. Once a goal g is chosen, the feedback engine computes a state-direction d_x that moves the system toward that goal (Line 8-9). This direction aligns with the gradient of a goal-distance function, defining the direction of evolution in physical space:

$$d_x = -\frac{\nabla_x \phi_{goal}(x)}{|\nabla_x \phi_{goal}(x)|2}. \tag{10}$$

To translate this direction into actionable signals for commands or events $(c \in s, a)$, ICSFlux uses the causal sensitivities in M_{CDC} and compartment constraints in $CPSS$:

$$\begin{aligned} \operatorname{sgn}(\Delta c) &= \chi^{(c)} \odot \operatorname{sign}\big(J_{xc}^{\top} d_x\big), \\ \overline{\Delta c} &= \operatorname{Proj}_{\mathcal{B}_{CDC}^{(c)} \cap \mathcal{B}_K^{(c)}}\Big(\varepsilon_c \cdot \operatorname{sgn}(\Delta c)\Big). \end{aligned} \tag{11}$$

Here, Jxc is the local sensitivity from channel c to the state x, $\chi^{(c)}$ masks infeasible directions according to $CPSS$ and M_{CDC}, and ε_c defines a step length scaled by the remaining distance to the goal boundary. When $c = s$, this yields the command direction and nominal delta used in command mutation; when $c = a$, it defines event adjustments used only under escalation conditions. These steps correspond to Line 8 and provide a unified rule for computing $(d_x, \overline{\delta s}, \overline{\delta a})$ used in the mutation stage.

Transition Recognition. After sending the control commands (Line 18), ICSFlux observes evolved state ζ from the simulation to determine progress and compartment transitions using Eq. 8. (Line 18–20). If $K_{new} = K_{cur}$, the system remains in the same compartment; otherwise, coverage and goal weights are updated.

Progress Evaluation. While running experiments, ICSFlux evaluates the current physical state x' within vector ζ' to determine how close the system approached the goal (Line 19). ICSFlux computes $prox$ (Eq. 12) and sets $hit = True$ when the current state x satisfies goal.

$$\text{prox} = 1 - \frac{\text{dist}(x, \partial\text{Goal})}{\text{dist}_{\max}} \tag{12}$$

Overall, ICSFlux monitors progress and explicitly selects a mode for the mutation engine each iteration: primarily command, but it elevates to event when progress stalls, and it deterministically triggers state re-initialization when progress still does not occur or when all neighboring compartments have been transitioned.

4.5. Mutation Engine

The mutation engine executes the guidance produced by the feedback engine, creating physically admissible perturbations of actuator commands, internal states, or event triggers within each compartment. Its purpose is to expand local coverage, recover from stagnation, and explore new physical transitions while respecting CPSS bounds.

The mutation engine dynamically switches between mutation modes: (i) command mutation when directed progress is possible, (ii) event mutation when progress stalls, or (iii) state mutation to reinitialize exploration in new compartments. These choices, made in Line 10–17, ensure continuous and diverse physical evolution.

Command Mutator. For each control step t, the admissible command-mutation region is the intersection of the causal constraint box and the active compartment envelope:

$$\Delta s_t \in B_{CDC}^{(s)} \cap B_{K_t}^{(s)} = [\Delta s_{min}, \Delta s_{max}] \cap [l^{K_t}, u^{K_t}] \tag{13}$$

Algorithm 2: Physics-Guided Feedback Engine and Mutation Engine

Input: Physical process P (providing current state ζ); constrained physical state space $CPSS$; causal differential constraints M_{CDC}; physical security constraints PSC; time budget T
Output: Discovered violation traces Π; coverage map Ω; execution log Λ

```
// Initialization
1  Π ← ∅;  Ω ← InitCoverage(CPSS);  Λ ← ∅;
2  comps ← CPSS.compartments;  ζ ← Observe(P);
3  while not TimeExceeded(T) do
       // Monitor and locate current context
4      ζ ← Observe(P);
5      (K_cur, C_next) ← LocateContext(ζ, comps) ;
6      Ω ← MarkCoverage(Ω, K_cur) ;
       // PGFE: strategy layer (goal, direction, and mode)
7      g ← SelectGoal(ζ, K_cur, C_next, CPSS, PSC)
8      (d_x, δs, δa) ← ComputeDirection(x, g, M_CDC, CPSS)
9      mode ← ChooseMode(Ω, K_cur, C_next, Λ)
       // Mutation Engine: tactics decided by PGFE
10     if mode = "Command" then
11         U ← CmdMutate(s, δs, K_cur, d_x, CPSS, M_CDC)
12     else if mode = "Event" then
13         U ← EventMutate(a, δa, K_cur, CPSS, M_CDC)
14     else
           // mode = "State" → reinitialize
15         K_tgt ← SelectReinitComp(K_cur, C_next, Ω, CPSS)
16         ζ ← StateMutateInit(K_tgt, CPSS, M_CDC)
17         continue
       // Execute and evaluate outcome
18     (ζ, res) ← ExecuteCommand(P, U)
19     (prox, hit) ← EvaluateProgress(ζ, PSC)
20     K_new ← LocateContext(ζ', comps)
       // Feedback and adaptation
21     Ω ← UpdateTransitionCoverage(Ω, K_cur, K_new)
22     Π ← Π ∪ {ζ};
23     ζ ← ζ'
24 return (Π, Ω, Λ)
```

Perturbations around the nominal command $\overline{\Delta s}$ from the physics-guided feedback engine are sampled as

$$\begin{aligned} \Delta s_t^{(k)} &= \overline{\Delta s}_t + \epsilon_t^{(k)}, \\ \epsilon_t^{(k)} &\sim \text{Proj}\mathcal{B}^{(s)}CDC \cap \mathcal{B}^{(s)}K_t\big(\mathcal{D}(0, \Sigma)\big), \end{aligned} \tag{14}$$

We mutate *command s only*; no direct state fuzzing occurs except when re-initializing via `StateMutateInit` (Line 16).

State Mutator. When the feedback engine requests a re-initialization (Line 16), the state mutator generates a new state x bounded by the selected compartment (using CPSS envelopes):

$$x_0^{(k)} \sim U(l^(K_j), u^{(K_j)}) \tag{15}$$

Event Mutator. If the feedback engine detects stagnation (Line 13), the event mutator injects actuator operator overrides/faults by mutating a:

$$E^{(k)} = (e_i, \tau_i + \delta i)| \quad |\delta i| < \Delta_{event} \tag{16}$$

These events force mode shifts that can expose new controller paths or fault-handling logic.

Overall, the mutation engine uses the mode and related constraints to drive the system towards the goal G.

5. Implementation

To evaluate ICSFlux, we built a Python prototype and tested it on eleven controllers across six industrial domains using datasets from prior works [27], [30], [31], [52], [53]. The dataset includes WAGO PFC200 [54], OpenPLC [55], and Codesys [56] controllers.

Simulation and communication setup. Controllers and simulators communicate over Modbus/TCP [57], allowing controllers to read sensor values from the simulator and send actuator commands back. ICSFlux leverages pymodbus [58] to remotely observe physical states and transmit control commands, eliminating the need for controller firmware modification or software installation. This setup ensures interoperability across heterogeneous controllers. To further enhance diversity beyond WAGO and OpenPLC, we deployed Codesys Runtime on Raspberry Pi [59] and BeagleBone Black [60], resulting in four distinct controller configurations.

Pre-fuzzing. This phase (≈2K LoC in Python) dynamically profiles physical process behavior. It collects controller commands, actuator outputs, and physical states through the Modbus interface. The resulting traces are used to constrain the physical state space according to observed relationships, the physical model, and defined physical security constraints.

Physics-guided fuzzing . The physics-guided feedback and mutation engines (≈4K LoC in Python) operate online with controllers and simulators via Modbus. The mutation engine issues operator-level setpoints and manual overrides through the HMI interface, while the feedback engine monitors real-time physical states to guide exploration. Based on the feedback, the mutation engine computes new command variations and sends them through Modbus to the controller.

6. Evaluation

We conducted a comprehensive evaluation to assess the efficacy of ICSFlux as a physics-guided fuzzing framework for industrial controllers. This evaluation focuses on several key aspects: revealing physical security violations (§6.1), analyzing the efficiency of physics-guided fuzzing (§6.2), assessing scalability and generalizability across controllers and domains (§6.3), and presenting representative case studies (§6.4).

Experimental setup. ICSFlux was evaluated on an Ubuntu 20.04.3 LTS environment running on WSL2 within Windows 11 Enterprise, equipped with a 12th Gen Intel(R) Core(TM) i7-12700H processor (14 cores at 2.40 GHz) and 16 GB of RAM. The testbed includes a WAGO PFC200, a virtual machine with OpenPLC, a Raspberry Pi 4, and a BeagleBone Black, each running the CodeSys Runtime (versions 4.4.0.0 and 4.11.0.0) as SoftPLC instances. All communication with controllers is performed over Modbus/TCP, providing a unified, vendor-agnostic interface. This standardization enables seamless integration with both hardware PLCs and software-based controllers, ensuring broad applicability.

Dataset. Experiments span 6 ICS domains using benchmark datasets from prior works, including GRFICS [27], MiniCPS [52], ICSFuzz [30], ICSPatch [53], and FieldFuzz [31]. The dataset collectively covers 11 controllers across various domains, including chemical, desalination, water treatment, and power grid. This diversity provides a robust platform to evaluate ICSFlux's scalability, effectiveness, and generalizability across heterogeneous ICS settings.

6.1. Effectiveness of ICSFlux

We first evaluate the overall physical security violation detection capability of ICSFlux across all tested controllers and domains (Tab. 1). ICSFlux evaluates each violation term derived from the physical security constraint. For instance, row 1 shows that ICSFlux reaches 100% physical-state coverage and triggers an explosion for the $P > P_{max}$ target in the chemical plant. The analysis thus focuses on unique violation terms rather than raw repetition counts. Each violation term was manually reviewed to confirm physical feasibility and controller vulnerability. 2 violation terms were deemed physically infeasible, and 5 showed no associated controller vulnerability, while the remaining 20 were validated as vulnerable. ICSFlux detected 20 violation terms triggered out of 20 vulnerable ones. These physical security violations occur across 6 / 6 (100%) industrial domains and 11 / 11 (100%) controllers, as summarized in Tab. 1. Because ICSFlux operates with closed-loop simulator feedback, no false positives were observed. The verified physical security violations encompass diverse unsafe outcomes - including reactor explosions, plant halts, tank overflows, and quality degradations. These findings demonstrate ICSFlux's capability to uncover gradual, multi-cycle physical security violations across heterogeneous systems. In summary, ICSFlux successfully triggered at least one physical security violation term in every tested domain and controller.

Comparison with Prior Fuzzers. This section compares ICSFlux with prior fuzzers such as ICSFuzz [30] and RVFuzzer [33]. Compared to them, ICSFlux consistently detects more physical security violations across a wider range of controllers and ICS domains, as summarized in Tab. 2. Each "✓" in the table indicates that at least one physical security violation term was triggered in that controller, while "−" indicates none were detected. ICSFuzz is specifically developed for Codesys-based controllers [56]. Therefore, to ensure fairness, we excluded the chemical plant [27] and water treatment [52] testbeds, which rely on OpenPLC [55] and a Python-based soft PLC, respectively. RVFuzzer was also evaluated only on the ICSFuzz dataset to maintain consistent comparisons across all three fuzzers. Other works [30], [31], [34], [35] require source-code access, which is unavailable for proprietary Codesys and WAGO controllers, making those tools

TABLE 1: Summary of the Effectiveness of ICSFlux across ICS Domains (**inputs:** Physical Model and Phys. Sec. Constraint)

ICS Domain	Physical Model	Phys. Sec. Constraint	Violation Term	Target (t_v) Predicate	Mutation Space Reduction			Phys. St. Coverage	Target (t_v) Reachable	Physical Impact
					Command	State	Event			
Chemical [27]	Eq. 17 - 21	Eq. 22	$P \in P_{range}$	$P > P_{max}$	75%	80%	65%	100%	Yes	Reactor Explosion
				$P < P_{min}$	75%	80%	65%	95.4%	No	Physically Infeasible
			$min\ F_{Purge}$	$F_{Purge} = F_{max}$	65%	70%	80%	94.4%	Yes	Increase Cost
			$max\ F_{Prod}$	$F_{Prod} = F_{min}$	60%	65%	80%	100%	Yes	Halt Production
			$L \in L_{range}$	$L \geq L_{max}$	65%	55%	80%	82.3%	No	Not Vulnerable
				$L = L_{min}$	65%	60%	80%	82.3%	No	Physically Infeasible
Water [52], [61]	Eq. 24-25	Eq. 27	$L_1 \in L_{range}$	$L_1 < L_{min}$	81%	76%	82%	77.4%	No	Not Vulnerable
				$L_1 > L_{max}$	81%	76%	82%	77.4%	No	Not Vulnerable
			$L_3 \in L_{range}$	$L_3 \leq L_{min}$	72%	75%	85%	100%	Yes	Halt Plant
				$L_3 \geq L_{max}$	72%	75%	85%	100%	Yes	Degrade Quality
Aircraft [30], [32]	Eq. 29	Eq. 30	0 < α < 30	$\alpha \leq 0$	83%	63%	56%	100%	Yes	Loss of altitude
				$\alpha \geq 30$	83%	63%	56%	100%	Yes	Instability
			-10 < θ < 20	$\theta \leq$ -10	77%	82%	80%	100%	Yes	Loss of Altitude
				θ >= 20	77%	82%	80%	100%	Yes	Loss of Control
			10 < ζ< 20	$\zeta \leq 10$	82%	79%	57%	100%	Yes	Loss of Altitude
				$\zeta \geq 20$	82%	79%	57%	100%	Yes	Instability
			-10 < β < 20	$\beta \leq$ -10	87%	79%	52%	100%	Yes	Loss of Altitude
				$\beta \geq 20$	87%	79%	52%	100%	Yes	Loss of Control
Desalination [30], [32]	Eq. 31	Eq. 32	0 < WD < 20000	WD $\leq$ 0	87%	67%	51%	80%	No	Physically Infeasible
				WD $\geq$ 20000	87%	67%	51%	100%	Yes	Degrade Quality
			25 < T < 110	T $\leq$ 25	88%	74%	60%	100%	Yes	Degrade Quality
				T $\geq$ 110	88%	74%	60%	100%	Yes	Damaging Product
Anaerobic [30], [32]	Eq. 33	Eq. 34	10 < T < 70	T $\leq$ 10	79%	80%	50%	100%	Yes	Degrade Quality
				T $\geq$ 70	79%	80%	50%	100%	Yes	Damage Product
Smart Grid [30], [32]	Eq. 35	Eq. 36	VG $\leq$ 750	VG > 750	79%	81%	50%	80%	No	Not Vulnerable
			VD $\leq$ 500	VD > 500	79%	81%	80%	80%	No	Not Vulnerable
			100 < BL	BL $\leq$ 100	89%	81%	80%	100%	Yes	Load Shedding

inapplicable. DriveFuzz [36], in contrast, focuses on autonomous driving systems using traffic-rule heuristics, limiting its generalizability. Within the comparable ICSFuzz dataset, ICSFlux uncovers violations in all 8/8 (100%) controllers, whereas ICSFuzz detects 1/8 (12.5%) and RVFuzzer detects 4/8 (50%). These results highlight ICSFlux's superior coverage and efficient guidance capability across equivalent controller environments, driven by its physics-guided strategy, demonstrating 2× higher coverage than RVFuzzer and 8× higher than ICSFuzz.

TABLE 2: Comparison of ICSFlux and prior fuzzers.

Contoller	Physical Security Violation		
	ICSFlux	ICSFuzz [30]	RVFuzzer [33]
Smart Grid	✓	–	–
NaOH Concentration	✓	–	✓
Water Distillate	✓	✓	✓
Brine Temperature	✓	–	✓
Angle of Attack	✓	–	–
Pitch Rate	✓	–	–
Pitch Angle	✓	–	–
Roll Angle	✓	–	✓
Total	8	1	4

6.2. Evalaution Efficiency

We now examine how ICSFlux achieves its results by evaluating the internal mechanisms of its feedback and mutation engines. This section isolates the contributions of the physics-guided feedback engine and the mutation-space reduction mechanisms.

Mutation-Space Reduction. We further analyze the quantitative impact of causal and physical constraints on the state, command, and event mutation spaces. ICSFlux first reduces the state space by partitioning the physical process into causal compartments, pruning infeasible initial conditions and narrowing each region by its bounded min–max ranges. This yields an average ≈64% state-space

reduction while maintaining ≈95% coverage of reachable partitions.

Within each compartment, causal dependencies constrain actuator-level commands and event triggers, further reducing redundant mutations. Across all controllers, ICSFlux achieved an average 73% command-space and 68% event-space reduction, with tightly coupled UAV controllers reaching up to 82%. The combined effect results in up to 80% total mutation-space reduction, concentrating tests on physically meaningful transitions without losing behavioral diversity.

This pruning allows ICSFlux to allocate computational effort toward meaningful explorations rather than random trials. On average, fuzzing throughput improved by 1.6×, confirming that constraint-guided mutation significantly enhances efficiency in real-time fuzzing scenarios.

Efficiency of Guidance. To quantify the benefit of physics-guided exploration, we compare ICSFlux with and without guidance. Both configurations leverage causal compartments, which constrain the physical state mutation space. The guided configuration, however, further tracks proximity to violation boundaries, dynamically adjusts command directions, and prioritizes compartment transitions, enhancing mutation efficiency using command and event directions. Across all controllers, physics-guided fuzzing reached the first physical security violation in an average of ≈60 minutes, whereas the compartment-only version required ≈180 minutes, achieving 3× faster convergence. It also produced nearly 20× more violations per 1 K tests, achieved a 92% compartment-transition success rate compared to 55%. These results demonstrate that while compartments improve baseline efficiency by diversifying initial physical states, guidance accelerates convergence by steering exploration toward target neighboring compartments and avoiding unnecessary command sequences.

Compared to prior fuzzers, ICSFlux achieves substantially faster convergence and higher violation density. On average, ICSFlux reached the first physical security violation within 60 minutes, whereas ICSFuzz required ≈21 hours (1268 minutes) and RVFuzzer ≈12 hours.

Overall, these findings demonstrate that physics-guided exploration not only accelerates convergence but also improves the effectiveness of fuzzing by ensuring that each test meaningfully advances the physical state toward safety-critical boundaries.

6.3. Scalability and Generalizability

Scalability. We next evaluate how ICSFlux scales across complexity in controllers and physical models.

Controller Complexity: **(a)** Across software layers, from simple Python-based logic (4K+ of LoC) to OpenPLC (10K+ LoC), to multi-threaded proprietary Codesys runtimes on Raspberry Pi and BeagleBone Black, and finally to full-fledged WAGO industrial controllers with multi-task scheduling and rich runtime services, ICSFlux maintains its effectiveness and efficiency. **(b)** These deployments differ in runtime semantics, ranging from finite-state logic to single and multi-loop PID control. **(c)** Treating each controller as a black box keeps ICSFlux agnostic to implementation language and internal threading architecture. **(d)** Quantitatively, across this four-tier spectrum, fuzzing throughput and detection efficiency vary by less than 10% due to the scan cycles, demonstrating stable scaling. **(e)** By extracting causal differential constraints from observed I/O traces, ICSFlux approximates nonlinear, nonmonotonic behaviors with local linear models, ensuring scalability across both software and physical model complexity. **(f)** Since ICSFlux does not need to be installed on the controller or modify its firmware, it sits in communication between the physical process and controller through the existing protocol interface. This design avoids runtime interference and introduces no overhead.

Domain Complexity: Scalability also extends to the number of terms in physical security constraints and the complexity of the physical model. **(a)** Each constraint term defines a safety boundary in the physical state space; more terms expand the search dimensionality. Even though ICSFlux breaks down physical security constraints to simpler terms, **(b)** Increasing the number of violation terms linearly raises analysis time during CPSS construction, but does not affect fuzzing throughput. **(c)** This overhead can be reduced by prioritizing critical terms or batching evaluations. **(d)** Across all testbeds, including multi-PID systems, ICSFlux exhibits near-linear scalability in pre-fuzzing and runtime phases, confirming that causal-linear approximations and targeted prioritization manage growth efficiently.

Generalizability. We next evaluate how ICSFlux generalizes across diversity in controllers and physical models.

Cross Controller: We evaluate ICSFlux across 11 controllers spanning four hardware configurations, WAGO PFC200, OpenPLC (virtualized), and Raspberry Pi and BeagleBone Black devices running Codesys runtimes to assess scalability, stability, and reproducibility. Each configuration differs in available memory, scan-cycle frequency, and network latency, resulting in diverse runtime conditions. During 24-hour fuzzing campaigns, ICSFlux maintained stable throughput within ±8% variation in executed tests per hour, demonstrating temporal consistency. The average throughput was approximately 50 tests per hour per controller. Controllers with faster scan cycles reached physical security violations sooner, confirming that ICSFlux adapts effectively to high-frequency control environments. Across all configurations, ICSFlux successfully triggered every manually identified violation term 11/11 (100%). The feedback engine and Modbus/TCP I/O remained the primary computational bottlenecks, but containerized deployment ensured reproducibility and consistent behavior across all controller architectures.

Cross Domain: To evaluate generalizability, we apply ICSFlux across six industrial domains, such as chemical, water, smart grid, desalination, UAV, and anaerobic systems. Despite variations in physical models, control logic, and process coupling, ICSFlux consistently identifies violations

TABLE 3: Summary of Extracted Causal Differential Constraints for the chemical plant [27], illustrated in §A.1.

Setpoint		Actuator		Physical State
$Material_A \uparrow$	$\rightarrow$	$Valve_A \uparrow$	$\rightarrow$	$Flow_A \uparrow \rightarrow \Delta Pressure \uparrow$
$Material_A \downarrow$	$\rightarrow$	$Valve_A \downarrow$	$\rightarrow$	$Flow_A \downarrow \rightarrow \Delta Pressure \downarrow$
$Material_B \uparrow$	$\rightarrow$	$Valve_B \uparrow$	$\rightarrow$	$Flow_B \uparrow \rightarrow \Delta Pressure \uparrow$
$Material_B \downarrow$	$\rightarrow$	$Valve_B \downarrow$	$\rightarrow$	$Flow_B \downarrow \rightarrow \Delta Pressure \downarrow$
$Pressure \uparrow$	$\rightarrow$	$Valve_{Purge} \downarrow$	$\rightarrow$	$Flow_{Purge} \downarrow \rightarrow \Delta Pressure \uparrow$
$Pressure \downarrow$	$\rightarrow$	$Valve_{Purge} \uparrow$	$\rightarrow$	$Flow_{Purge} \uparrow \rightarrow \Delta Pressure \downarrow$
$Product \uparrow$	$\rightarrow$	$Valve_{Product} \uparrow$	$\rightarrow$	$Flow_{Product} \uparrow \rightarrow \Delta Pressure \downarrow$
$Product \downarrow$	$\rightarrow$	$Valve_{Product} \downarrow$	$\rightarrow$	$Flow_{Product} \downarrow \rightarrow \Delta Pressure \uparrow$

in all domains. The number of discovered physical security violations ranges from 2 in small-scale water systems to 9 in highly coupled UAV controllers. Performance degradation across domains stays below 10% in mutation-space efficiency and under 15% in time-to-first violation, confirming consistent scalability. These findings demonstrate that the physics-guided fuzzing methodology generalizes effectively across heterogeneous process dynamics and controller architectures without requiring domain-specific tuning.

6.4. Case Study

Exploration Objective. A central goal of this case study is *not* to minimize the time required to trigger an explosion, but to demonstrate how a violation can emerge from a safe, nominal operating state through a causally coherent sequence of physically valid transitions. ICSFlux begins each campaign from the system's steady state and systematically explores the physical state space to ensure that all relevant compartments and transitions near the safety boundary are exercised. This methodology provides a comprehensive view of how pressure can drift from nominal to unsafe regions, revealing full end-to-end trajectories rather than isolated, time-optimized exploits. The emphasis is on *state-space coverage*, *causal interpretability*, and *realistic closed-loop evolution*, ensuring that every discovered trajectory represents a feasible physical path rather than one constructed from already dangerous or artificially advantageous starting conditions.

Reactor Explosing via Valid Operator Commands. To contextualize this trajectory, ICSFlux executed approximately 200 tests over about 3 hours (≈216000 scan cycles at 50 ms) before uncovering the first reactor explosion. During this period, ICSFlux produced roughly 350 operator command mutations and 200 physical space state mutations, of which around 22 resulted in successful compartment transitions. ICSFlux achieved 100% transition success rate. In a 24-hour campaign, ICSFlux generated 10 different explosion trajectories with 100% coverage, demonstrating that violation paths emerge only after sustained, coordinated perturbations.

Before fuzzing begins, ICSFlux performs pre-fuzzing analysis to extract causal differential constraints, construct the causal evolution graph, and derive the constrained physical state space, a process that completes in approximately 5 minutes on our evaluation setup. Using these outputs, ICSFlux begins from the safe steady state located in the compartment that contains the nominal operating point. From here, the physics-guided feedback engine identifies the neighboring compartment K_1 as the closest to the overpressure boundary and selects it as the next exploration goal.

To ground this reasoning, Tab. 3 summarizes the extracted causal differential constraints, which describe how setpoint adjustments propagate through actuators to affect physical-state variables. For example, increasing $Material_A$ inflow opens $Valve_A$, raises $Flow_A$, and consequently increases pressure. Conversely, decreasing the $Pressure$ setpoint drives ICSFlux to choose the opposite direction, opening the $Valve_{Purge}$ or reducing material inflows.

When the target violation term is $P > P_{max}$, ICSFlux selects the neighboring compartment that moves pressure upward and identifies the corresponding signed direction. Thus, ICSFlux chooses setpoint adjustments that increase pressure, steering the system toward the high-risk region while remaining within valid command limits. Using these constraints (Tab. 3), ICSFlux identifies which operator commands can effectively drive the system toward the next high-risk compartment and in which direction they should be perturbed. ICSFlux then generates signed command-direction hints—for instance, increasing Material-A and Material-B inflows, reducing purge, and decreasing product outflow—and passes these structured hints to the mutation engine for execution. On average, ICSFlux spent approximately 1 minute per compartment, with about 22 successful transitions out of approximately 200 attempts (≈9% success rate), reflecting the tight physical constraints defining each compartment.

The mutation engine applies these hints through bounded, monotonic command updates that remain within valid HMI ranges. Even though every command stays within its allowed interval, their combined effect steadily increases pressure across many scan cycles. ICSFlux uses the constrained physical state space to reduce the mutation search space: (1) physical-state mutation space is limited by compartment bounds (e.g., reducing reachable pressure range by ≈80%), (2) command-mutation space starts with roughly a 60% reduction and increases as ICSFlux approaches higher-risk neighboring compartments, reaching up to ≈90% reduction in the final compartment (≈75% reduction on average across the trajectory), and (3) event-mutation space is activated only when progress stalls (≈65% reduction in event triggers).

When ICSFlux detects that progress stalls under command mutation, it selectively triggers event mutations such as temporarily holding the purge valve closed. ICSFlux continuously monitors the current compartment, the goal compartment, and the signed direction vector, refreshing the budget provided to the mutation engine.

6.4.1. Extended Case Study: Pressure Safety Violations Under Conservative Setpoints. To further investigate robustness under more conservative safety settings, we

conducted additional experiments with progressively lower pressure thresholds. The original pressure guard was defined as $P \leq 3200$ kPa. Initial experiments revealed that although 3200 kPa is the nominal limit, the reactor becomes unstable once pressure exceeds 3100 kPa, frequently resulting in runaway escalation and explosions.

Experiments with Reduced Pressure Threshold. When repeating the full ICSFlux campaign under a tightened constraint of $P_{max} = 3100$, the system still exhibited multiple explosive trajectories. Although input mutations always respected the new guard (never issuing setpoints above 3100), the closed-loop dynamics exhibited oscillations and overshoot of $\pm$20–40 kPa around the setpoint. These oscillations occasionally drove pressure above the 3100 kPa safety boundary, triggering uncontrolled increases and eventual explosion. This demonstrates that even "safe" setpoints can indirectly cause hazardous outcomes due to internal controller dynamics.

Strict Pressure Threshold. Next, we imposed an even more conservative threshold, P_{max} = 3050 kPa, to prevent oscillatory overshoot from causing explosions. Under this stricter boundary, state-space mutation and command mutation alone were sufficient to pass 3050 kPa. However, ICSFlux did not discover explosions using only continuous operator-command perturbations because oscillatory pressure fluctuations no longer exceeded the 3050 limit.

However, once ICSFlux activated *event mutation*, new explosions emerged:

- *HMI Manual Override:* Manual override of the product valve reduced purge-valve opening, creating a sustained inflow/outflow imbalance. Although the controller correctly read pressure, valve, and flow values, it did not check whether the purge reduction had been manually overridden. As a result, it continued to push additional material to meet the product-level setpoint, eventually exceeding the safety envelope and causing an explosion.
- *Actuator Fault Injection:* When ICSFlux simulated a stuck purge valve actuator, the controller repeatedly attempted to open the valve to relieve pressure, but the actuator never responded. Despite receiving correct sensor readings for pressure, valve position, and flow (with no spoofing involved), the controller failed to check whether the valve was actually opening. Material inflows relied solely on the product-level setpoint, and the controller continued to push additional inflow material to maintain production. Because the controller lacked fallback logic for actuator failure and had no secondary pressure-based cross-checks, the pressure rose uncontrollably until the explosion.

Key Insight. Even under conservative safety limits, explosions can still occur due to a combination of unstable controller dynamics (e.g., oscillatory overshoot of the pressure setpoint), missing safety cross-checks for manual HMI overrides, and inadequate actuator-fault handling where the controller assumes that issued commands always translate to correct physical action. These findings show that ICSFlux exposes not only direct limit-violating inputs but also emergent unsafe behaviors arising from subtle controller–process interactions, overlooked override pathways, and actuator-level anomalies.

Evaluation Summary. Our evaluation demonstrates that ICSFlux provides reliable and systematic coverage of the physical state space, scales consistently across hardware platforms, and maintains efficiency even under increasing model and domain complexity. Quantitative results showed that physics-guided exploration improves convergence speed, reduces redundant mutations, and operates more efficiently compared to prior fuzzers. Across all tested controllers and domains, ICSFlux successfully exercised every reachable compartment and triggered all vulnerable violation terms, confirming the completeness of its search relative to the defined physical constraints. The case studies further illustrated how ICSFlux reconstructs realistic, multi-step violation trajectories from safe steady-state operation, revealing root causes that are otherwise difficult to observe.

7. Related Work

Syntactic Bug Detection. Coverage-guided fuzzers [62]–[64] and later dataflow-aware variants [65]–[74] have been highly effective at uncovering memory and logic bugs in traditional software systems. Within the ICS domain, Chen et al. [75] used supervised learning to identify PLC logic faults, ICSFuzz [30] applied fuzzing to PLC binaries, FieldFuzz [31] examined runtime-level vulnerabilities in Codesys, ICSPatch [53] used Data Dependence Graphs to localize faults, and ICSQuartz [32] leveraged compiler toolchains to track loop execution. However, these approaches focus on syntactic or structural bugs in PLC software and do not analyze semantic issues that manifest as gradual, multi-cycle physical security violations.

Semantic Bug Detection. Several works target cyber–physical semantic errors. Choi et al. [76] detect cyber–physical inconsistencies by mutating virtual field conditions in robotic vehicles. RVFuzzer [33] and ConTest [34] mutate environmental factors and configuration parameters to identify input-validation bugs. PGFuzz [35] uses propositional distance to guide mutations for high-level state-machine controllers, while DriveFuzz [36] mutates traffic scenarios to expose semantic bugs in autonomous driving systems. Many of these approaches require reading or modifying controller firmware [33]–[35], or assume an adversarial environment [36] where agents or environmental dynamics can be freely altered—assumptions that do not hold in ICS, where actuator sets are fixed and controller parameters are locked down. These semantic fuzzers primarily reveal physical-security violations caused by invalid inputs, adversarial conditions, or high-level state-machine errors, rather than exposing computational or logic flaws within the controller itself.

Physics-Based Attack Detection. Physics-based detection techniques [47], [77], [78] analyze real-time sensor behavior to detect anomalies caused by attacks during operation. They focus on runtime monitoring rather than pre-deployment analysis. While these approaches are

critical for intrusion detection, they do not proactively assess whether a controller, under valid operator commands, could naturally evolve into unsafe physical states. ICSFlux is not a replacement for such operational defenses; rather, it is complementary—providing pre-deployment analysis that identifies unsafe controller behaviors before they manifest, enabling stronger overall protection when used alongside physics-based detection systems.

Summary and Comparison. Across syntactic, semantic, and physics-based approaches, prior work either (1) targets traditional software bugs, (2) requires white-box controller access, or (3) assumes adversarial capabilities incompatible with locked-down ICS deployments. None provide a domain-general framework for discovering multi-cycle physical security violations that arise from interactions between a black-box controller and its physical process.

8. Discussions

Scalability of ICSFlux. ICSFlux requires only a brief one-time interface setup to map physical-model variables to sensor readings and actuator commands. After this step, scalability follows naturally from its black-box design: ICSFlux does not rely on firmware internals, runtime scheduling, or code size. In our evaluation, it maintained stable per-violation target effort across controllers ranging from lightweight Python soft-PLCs (5K+ LoC) to OpenPLC (10K+ LoC), and to more complex multithreaded Codesys and multiprocess WAGO runtimes. Domain scalability behaves similarly: larger or more nonlinear physical models simply introduce additional violation terms, but each term is fuzzed independently using its own local causal dynamics. As a result, even as system size grows, the per-violation target fuzzing cost remains predictable, demonstrating robust scalability across both controller and physical-model complexity.

Generalizability of ICSFlux. ICSFlux generalizes across controllers and industrial domains because it relies solely on observable I/O behavior rather than firmware access, source code, or runtime instrumentation. This allows it to operate uniformly on controllers implemented in different languages and architectures—from Python soft-PLCs and OpenPLC on x86 to ARM-based Codesys runtimes and full WAGO industrial devices with obfuscated, proprietary firmware. Domain generalizability follows the same principle: ICSFlux requires only the physical model and physical security constraints, enabling applications to chemical, water, and other process domains without domain-specific heuristics. Despite wide variation in dimensionality and nonlinear dynamics, ICSFlux adapts automatically through causal constraints and compartment guidance, making it broadly applicable across heterogeneous cyber–physical systems.

Prerequisites and Model Quality. ICSFlux relies on the physical model and physical security constraint to bound the physical state space. These models are defined early in engineering design and guide the entire ICS lifecycle, including commissioning, controller development, tuning, and safety verification. In safety-critical domains such as chemical or nuclear systems, a highly inaccurate model would already jeopardize plant operation. Therefore, deployed models tend to be realistic enough for tools like ICSFlux. If a model is incomplete, ICSFlux may miss some violations or guide less efficiently, but it does not produce false positives.

Protection of ICS. ICSFlux focuses on revealing control-logic faults and closed-loop behaviors that may result in physical security violations. It is not intended to replace operational defenses. Instead, ICSFlux is most effective when used alongside complementary security mechanisms such as anomaly-detection systems, intrusion detection, secure communication practices, and periodic security audits. These defenses address operational threats, while ICSFlux proactively identifies vulnerabilities in control logic before an adversary or natural fault triggers them. Used together, they provide a stronger overall security posture.

Imperfect Simulations. ICSFlux relies on software-based simulators, which approximate, but cannot perfectly match, real physical behavior. However, ICSFlux does not require exact physical fidelity. Its objective is to reveal whether controller decisions can drive the process toward unsafe regions. Because it reasons over compartment transitions and causal constraints, ICSFlux exposes structural vulnerabilities in controller logic rather than relying on precise physics. Our conservative-bound experiments (e.g., 3100 kPa and 3050 kPa) show that, even under stricter limits, ICSFlux consistently uncovers the same violations due to controller logic bugs, indicating that controller flaws, not simulation artifacts, drive the violations.

9. Conclusion

This paper presented ICSFlux, a physics-aware fuzzing framework for uncovering physical security violations arising from subtle controller misbehaviors. ICSFlux integrates dynamics profiling, extracted causal constraints, reversed propagation over a physical model, constrained physical state space, and physics-guided mutation to steer the physical state toward physical security violations using only valid Level-2 operator commands. Because ICSFlux treats the controller as a black box and relies solely on observable I/O behavior, it generalizes naturally across heterogeneous PLC platforms and diverse industrial domains.

In our evaluation, ICSFlux successfully identified all vulnerable physical-security violation terms across six domains and eleven controllers, uncovering unsafe behaviors ranging from reactor explosions to plant halts. These results show that ICSFlux can systematically expose gradual, multi-cycle vulnerabilities that emerge only through realistic control-process interactions. By enabling proactive identification of unsafe controller behaviors before they are triggered naturally or maliciously, ICSFlux provides a scalable and generalizable foundation for strengthening the safety and security of real-world industrial operations.

10. Acknowledgment

This material is based upon work supported by the U.S. Department of Energy, Office of Cybersecurity, Energy Security, and Emergency Response under Award Number DE-CR0000056, and the National Science Foundation (NSF).

11. Ethics considerations

All experiments in this study were conducted in controlled, simulated environments to avoid any disruption to live industrial control systems. We did not test on production ICS deployments or access proprietary vendor software beyond what was openly available. Any control programs and testbeds used in our evaluation were obtained from open-source repositories or research platforms. We did not attempt to reverse engineer closed-source binaries or firmware without explicit collaboration from manufacturers. This research was carried out strictly under ethical research guidelines, ensuring that no sensitive operational data, proprietary information, or real-world infrastructure was compromised. We are in the process of responsibly disclosing all controller-level vulnerabilities identified during evaluation to the corresponding research testbed maintainers, and coordination for validation is ongoing.

References

[1] Cybersecurity and Infrastructure Security Agency (CISA), *Industrial Control Systems*, https://www.cisa.gov/topics/industrial-control-systems, Accessed: August 6, 2025.

[2] Rockwell Automation, *A Comprehensive Guide to ICS Protection*, https://www.rockwellautomation.com/en-us/company/news/blogs/what-is-ics-security.html, Accessed: August 6, 2025.

[3] BitSight Technologies, *Industrial Control Systems (ICS)*, https://www.bitsight.com/glossary/industrial-control-systems-ics, Accessed: August 6, 2025, 2025.

[4] PA Consulting Group, "Manage Industrial Control Systems Lifecycle," Centre for the Protection of National Infrastructure (CPNI), Tech. Rep., 2015.

[5] ABB Robotics, *Abb irc5 controller product specification*, 2018. [Online]. Available: https://new.abb.com/products/robotics/controllers/irc5.

[6] T. Egel and S. Furry, "Physical Modeling Considerations for Control System Development," SAE Technical Paper, Tech. Rep., 2014.

[7] J. J. Ward, "Modeling and Simulating a Performance Hybrid Electric Vehicle," M.S. thesis, The Ohio State University, 2015.

[8] K. Stouffer, M. Pease, C. Tang, T. Zimmerman, V. Pillitteri, S. Lightman, A. Hahn, S. Saravia, A. Sherule, and M. Thompson, "Guide to industrial control systems (ICS) security," *NIST special publication*, 2023.

[9] K. J. Åström and R. Murray, *Feedback systems: an introduction for scientists and engineers*. Princeton university press, 2021.

[10] D. I. Urbina, J. A. Giraldo, A. A. Cardenas, N. O. Tippenhauer, J. Valente, M. Faisal, J. Ruths, R. Candell, and H. Sandberg, "Limiting the impact of stealthy attacks on industrial control systems," in *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*, 2016.

[11] D. E. Seborg, T. F. Edgar, D. A. Mellichamp, and F. J. Doyle III, *Process Dynamics and Control*, 4th ed. John Wiley & Sons, 2016.

[12] C. Kravaris and I. K. Kookos, *Understanding Process Dynamics and Control*. Cambridge University Press, 2021.

[13] *Functional safety – Safety instrumented systems for the process industry sector – Part 1: Framework, definitions, system, hardware and application programming requirements*, International Electrotechnical Commission (IEC), 2016.

[14] *Functional safety of electrical/electronic/programmable electronic safety-related systems – Parts 1 to 7*, International Electrotechnical Commission (IEC), 2010. [Online]. Available: https://webstore.iec.ch/en/publication/22273.

[15] N. Falliere, L. O. Murchu, E. Chien, *et al.*, *W32.Stuxnet Dossier*, https://pax0r.com/hh/stuxnet/Symantec-Stuxnet-Update-Feb-2011.pdf, [Accessed: 2025-04-01], 2011.

[16] U. D. of Homeland Security, Cybersecurity, and I. S. Agency, *Chemical Sector Landscape*, https://www.cisa.gov/sites/default/files/publications/Chemical%2520Sector%2520Landscape_compliant.pdf, [Accessed: 2025-04-01], 2019.

[17] N. Stone, *Industrial Control Systems are Exposed: Breaking Down the Risks*, https://www.bitsight.com/blog/industrial-control-systems-are-exposed-breaking-down-risks, [Accessed: 2025-04-01], 2023.

[18] Allianz, *Cyber attacks on critical insrastructure*, https://commercial.allianz.com/news-and-insights/expert-risk-articles/cyber-attacks-on-critical-infrastructure.html, [Accessed: 2025-04-01], 2016.

[19] AP News, *States and congress wrestle with cybersecurity after iran attacks small town water utilities*, https://apnews.com/article/water-utilities-hackers-cybersecurity-1c475f5d2ef3b5d52410c93bdeab3aad, [Accessed: 2025-04-01], 2023.

[20] Betsy Reed, *Robot kills worker at Volkswagen plant in Germany*, https://www.theguardian.com/world/2015/jul/02/robot-kills-worker-at-volkswagen-plant-in-germany, [Accessed: 2025-04-01], 2015.

[21] J. Vijayan, *6 malware tools designed to disrupt industrial control systems (ics)*, https://www.darkreading.com/threat-intelligence/6-malware-tools-designed-to-disrupt-ics-environments, [Accessed: 2025-04-01], 2016.

[22] CBS News, *Industrial robot crushes worker to death as he checks whether it was working properly*, https://www.cbsnews.com/news/

industrial-robot-crushes-worker-dead-south-korea/, [Accessed: 2025-04-01], 2023.
[23] CNN, *Federal officials investigating after pro-Iran group allegedly hacked water authority in Pennsylvania*, https://www.cnn.com/2023/11/28/us/pennsylvania-water-cyberattack/index.html, [Accessed: 2025-04-01], 2023.
[24] M. Giles, *Triton is the world's most murderous malware, and it's spreading*, https://www.technologyreview.com/2019/03/05/103328/cybersecurity-critical-infrastructure-triton-malware/, [Accessed: 2025-04-01], 2019.
[25] KnowBe4, *Cyber Attacks on Infrastructure: The New Geopolitical Weapon*, https://www.knowbe4.com/hubfs/Global-Infrastructure-Report-2024_EN_US.pdf, [Accessed: 2025-04-01], 2024.
[26] Poulsen, Kevin, *Tracking the Blackout bug*, https://www.theregister.com/2004/04/08/blackout_bug_report/, [Accessed: 2025-04-01], 2004.
[27] D. Formby, M. Rad, and R. Beyah, "Lowering the barriers to industrial control system security with {grfics}," in *2018 USENIX Workshop on Advances in Security Education (ASE 18)*, 2018.
[28] Control.com. "Practical pid controller features." [Accessed: 2025-11-10]. (2020), [Online]. Available: https://control.com/textbook/closed-loop-control/practical-pid-controller-features/.
[29] N. Inc., *Mini-ICS Inerting Control System Operations Manual*, [Accessed: 2025-11-10], 2018. [Online]. Available: https://www.mybacharach.com/wp-content/uploads/2021/02/Mini-ICS-Inerting-Control-System-Operations-Manual-MNA0031.pdf.
[30] D. Tychalas, H. Benkraouda, and M. Maniatakos, "ICSFuzz: Manipulating I/Os and repurposing binary code to enable instrumented fuzzing in ICS control applications," in *Proceedings of the 30th USENIX Security Symposium (Security)*, Virtual Conference, Aug. 2021.
[31] A. Bytes, P. H. N. Rajput, C. Doumanidis, M. Maniatakos, J. Zhou, and N. O. Tippenhauer, "Fieldfuzz: In situ blackbox fuzzing of proprietary industrial automation runtimes via the network," in *International Symposium on Research in Attacks, Intrusions and Defenses (RAID)*, Hong Kong, 2023.
[32] C. Villa, C. Doumanidis, H. Lamri, P. H. N. Rajput, and M. Maniatakos, "ICSQuartz: Scan Cycle-Aware and Vendor-Agnostic Fuzzing for Industrial Control Systems," in *Proceedings of the 2025 Annual Network and Distributed System Security Symposium (NDSS)*, San Diego, CA, Feb. 2025.
[33] T. Kim, C. H. Kim, J. Rhee, F. Fei, Z. Tu, G. Walkup, X. Zhang, X. Deng, and D. Xu, "RVFuzzer: Finding input validation bugs in robotic vehicles through Control-Guided testing," in *Proceedings of the 28th USENIX Security Symposium (Security)*, Santa Clara, CA, Aug. 2019.
[34] J. Wang, H. Zhang, C. Jiang, A. Clark, and N. Zhang, "ConTest: Taming the Cyber-physical Input Space in Fuzz Testing with Control Theory," in *Proceedings of the 32nd ACM Conference on Computer and Communications Security (CCS)*, Taipei, Taiwan, Oct. 2025.
[35] H. Kim, M. O. Ozmen, A. Bianchi, Z. B. Celik, and D. Xu, "PGFUZZ: Policy-Guided Fuzzing for Robotic Vehicles.," in *Proceedings of the 2021 Annual Network and Distributed System Security Symposium (NDSS)*, Virtual Conference, Feb. 2021.
[36] S. Kim, M. Liu, J. J. Rhee, Y. Jeon, Y. Kwon, and C. H. Kim, "Drivefuzz: Discovering autonomous driving bugs through driving quality-guided fuzzing," in *Proceedings of the 29th ACM Conference on Computer and Communications Security (CCS)*, Los Angeles, CA, Nov. 2022.
[37] L. Garcia, F. Brasser, M. H. Cintuglu, A.-R. Sadeghi, O. A. Mohammed, and S. A. Zonouz, "Hey, My Malware Knows Physics! Attacking PLCs with Physical Model Aware Rootkit," in *Proceedings of the 2017 Annual Network and Distributed System Security Symposium (NDSS)*, San Diego, CA, Feb. 2017.
[38] S. Paoletti, A. L. Juloski, G. Ferrari-Trecate, and R. Vidal, "Identification of hybrid systems a tutorial," *European journal of control*, vol. 13, no. 2-3, pp. 242–260, 2007.
[39] E. G. Gilbert, "Controllability and observability in multivariable control systems," *Journal of the Society for Industrial and Applied Mathematics, Series A: Control*, vol. 1, no. 2, pp. 128–151, 1963.
[40] C. Jung, A. Ahad, Y. Jeon, and Y. Kwon, "Swarmflawfinder: Discovering and exploiting logic flaws of swarm algorithms," in *2022 IEEE Symposium on Security and Privacy (SP)*, IEEE, 2022, pp. 1808–1825.
[41] T. J. Williams, "The purdue enterprise reference architecture," *Computers in industry*, vol. 24, no. 2-3, pp. 141–158, 1994.
[42] KUKA Roboter GmbH, *Kuka system software 8.3, operating and programming instructions for system integrators*, 2015. [Online]. Available: https://www.kuka.com.
[43] CAN in Automation (CiA), *Cia 402 drive profile for motion control*, 2017. [Online]. Available: https://www.can-cia.org.
[44] DJI, *Dji onboard and mobile sdk documentation*, 2022. [Online]. Available: https://developer.dji.com.
[45] PX4 Development Team, *Px4 autopilot user guide*, 2023. [Online]. Available: https://docs.px4.io/main/en/.
[46] User study on Unitree SDK, "Configuring unitree go2 edu with sdk to issue walk/stop commands," 2025, Describes how the Unitree Go2 SDK allows sending commands like walk, stop via Python SDK. [Online]. Available: https://hackmd.io/@c12hQ00ySVi6JYIERU7bCg/ByAOr12qJg.
[47] S. G. Abbas, M. O. Ozmen, A. Alsaheel, A. Khan, Z. B. Celik, and D. Xu, "SAIN: Improving ICS Attack Detection Sensitivity via State-Aware Invariants," in *Proceedings of the 33rd USENIX Security Symposium (Security)*, Philadelphia, PA, Aug. 2024.
[48] S. E. McLaughlin, S. A. Zonouz, D. J. Pohly, and P. D. McDaniel, "A Trusted Safety Verifier for Process Controller Code.," in *Proceedings of the 2014 Annual Network and Distributed System Security Symposium (NDSS)*, San Diego, CA, Feb. 2014.
[49] B. Perelman, *Top 3 Threats to Industrial Control Systems*, https://www.securityweek.com/top-3-threats-industrial-control-systems/, [Accessed: 2025-04-01], 2016.
[50] A. Ginter, *The top 20 cyber attacks on industrial control systems*, https://scadahacker.com/library/Documents/White_Papers/Waterfall%20-%20Top%2020%20Cyber%20Attacks%20on%20ICS.pdf#page=7.53, [Accessed: 2025-04-01], 2017.
[51] O. Koucham, S. Mocanu, G. Hiet, J.-M. Thiriet, and F. Majorczyk, "Detecting process-aware attacks in sequential control systems," in *Nordic Conference on Secure IT Systems*, Springer, 2016, pp. 20–36.
[52] D. Antonioli and N. O. Tippenhauer, "MiniCPS: A toolkit for security research on CPS networks," in *Proceedings of the First ACM workshop on cyber-physical systems-security and/or privacy*, 2015.
[53] P. H. N. Rajput, C. Doumanidis, and M. Maniatakos, "ICSPatch: Automated Vulnerability Localization and Non-Intrusive Hotpatching in Industrial Control Systems using Data Dependence Graphs," in *Proceedings of the 31st USENIX Security Symposium (Security)*, Boston, MA, Aug. 2022.
[54] CODESYS Group, *PFC200 Controller*, https://www.wago.com/us/pfc200, [Accessed: 2025-04-01], 2025.
[55] Autonomy, *OpenPLC: Open Source PLC Software*, https://autonomylogic.com/, [Accessed: 2025-04-01], 2025.
[56] C. Group, *CODESYS Runtime*, https://www.codesys.com/products/runtime/, [Accessed: 2025-04-01], 2025.
[57] M. Organization, *Modbus Protocol*, http://www.modbus.org/, [Accessed: 2025-04-01], 2025.
[58] Github, *PyModbus - A Python Modbus Stack*, https://github.com/pymodbus-dev/pymodbus, [Accessed: 2025-04-01], 2025.
[59] Raspberry Pi Foundation, *Raspberry Pi 5*, https://www.raspberrypi.com/products/raspberry-pi-5/, [Accessed: 2025-04-01], 2025.
[60] BeagleBoard.org Foundation, *BeagleBone Black*, https://www.beagleboard.org/boards/beaglebone-black, [Accessed: 2025-04-01], 2025.
[61] A. P. Mathur and N. O. Tippenhauer, "SWaT: A water treatment testbed for research and training on ICS security," in *2016 international workshop on cyber-physical systems for smart water networks (CySWater)*, IEEE, 2016.
[62] M. Zalewski, *American Fuzzy Lop (AFL)*, http://lcamtuf.coredump.cx/afl, [Accessed: 2025-04-01], 2020.
[63] LLVM Project, *LibFuzzer - a library for coverage-guided fuzz testing*, https://llvm.org/docs/LibFuzzer.html, [Accessed: 2025-04-01], 2020.
[64] Google, *OSS-Fuzz - continuous fuzzing for open source software*, https://google.github.io/oss-fuzz/, [Accessed: 2025-04-01], 2016.

[65] J. Chen, W. Diao, Q. Zhao, C. Zuo, Z. Lin, X. Wang, W. C. Lau, M. Sun, R. Yang, and K. Zhang, "IoTFuzzer: Discovering Memory Corruptions in IoT Through App-based Fuzzing.," in *Proceedings of the 2018 Annual Network and Distributed System Security Symposium (NDSS)*, San Diego, CA, Feb. 2018.

[66] Y. Chen, D. Mu, J. Xu, Z. Sun, W. Shen, X. Xing, L. Lu, and B. Mao, "Ptrix: Efficient hardware-assisted fuzzing for cots binary," in *Proceedings of the 14th ACM Symposium on Information, Computer and Communications Security (ASIACCS)*, Auckland, New Zealand, Jul. 2019.

[67] S. T. Dinh, H. Cho, K. Martin, A. Oest, K. Zeng, A. Kapravelos, G.-J. Ahn, T. Bao, R. Wang, A. Doupé, *et al.*, "Favocado: Fuzzing the Binding Code of JavaScript Engines Using Semantically Correct Test Cases.," in *Proceedings of the 2021 Annual Network and Distributed System Security Symposium (NDSS)*, Virtual Conference, Feb. 2021.

[68] P. Fiterau-Brostean, B. Jonsson, R. Merget, J. De Ruiter, K. Sagonas, and J. Somorovsky, "Analysis of DTLS implementations using protocol state fuzzing," in *Proceedings of the 29th USENIX Security Symposium (Security)*, Virtual Conference, Aug. 2020.

[69] I. Karim, F. Cicala, S. R. Hussain, O. Chowdhury, and E. Bertino, "ATFuzzer: dynamic analysis framework of AT interface for Android smartphones," *Digital Threats: Research and Practice*, vol. 1, no. 4, pp. 1–29, 2020.

[70] H. Kim, J. Lee, E. Lee, and Y. Kim, "Touching the untouchables: Dynamic security analysis of the LTE control plane," in *Proceedings of the 40th IEEE Symposium on Security and Privacy (S&P)*, San Francisco, CA, May 2019.

[71] S. Kim, M. Xu, S. Kashyap, J. Yoon, W. Xu, and T. Kim, "Finding semantic bugs in file systems with an extensible fuzzing framework," in *Proceedings of the 27th ACM Symposium on Operating Systems Principles (SOSP)*, Ontario, Canada, Oct. 2019.

[72] S. Österlund, K. Razavi, H. Bos, and C. Giuffrida, "ParmeSan: Sanitizer-guided greybox fuzzing," in *Proceedings of the 29th USENIX Security Symposium (Security)*, Virtual Conference, Aug. 2020.

[73] S. Rawat, V. Jain, A. Kumar, L. Cojocar, C. Giuffrida, and H. Bos, "VUzzer: Application-aware Evolutionary Fuzzing.," in *Proceedings of the 2017 Annual Network and Distributed System Security Symposium (NDSS)*, San Diego, CA, Feb. 2017.

[74] Z. Sialveras and N. Naziridis, "Introducing Choronzon: An approach at knowledge-based evolutionary fuzzing," in *Proceedings of the ZeroNights 2015*, Moscow, Russia, Nov. 2015.

[75] Y. Chen, C. M. Poskitt, and J. Sun, "Learning from mutants: Using code mutation to learn and monitor invariants of a cyber-physical system," in *Proceedings of the 39th IEEE Symposium on Security and Privacy (S&P)*, San Francisco, CA, May 2018.

[76] H. Choi, S. Kate, Y. Aafer, X. Zhang, and D. Xu, "Cyber-physical inconsistency vulnerability identification for safety checks in robotic vehicles," in *Proceedings of the 27th ACM Conference on Computer and Communications Security (CCS)*, Virtual Conference, Nov. 2020.

[77] M. Ike, K. Phan, A. Badapanda, M. Landen, K. Sadoski, W. Guo, A. Shah, S. Zonouz, and W. Lee, "Bridging both worlds in semantics and time: Domain knowledge based analysis and correlation of industrial process attacks," *arXiv preprint arXiv:2311.18539*, 2023. [Online]. Available: https://arxiv.org/pdf/2311.18539.

[78] M. Ike, K. Phan, K. Sadoski, R. Valme, and W. Lee, "Scaphy: Detecting modern ics attacks by correlating behaviors in scada and physical," in *Proceedings of the 44th IEEE Symposium on Security and Privacy (S&P)*, San Francisco, CA, May 2023.

Appendix A. Physical Models

A.1. Chemical Plant

We use the chemical plant simulation [27], which provides a simulation of the Tennessee Eastman (TE) process. The model captures the key material balances, valve–flow relationships, and pressure dynamics that govern the reactor vessel.

State Variables. The vessel state is $s = (N_A, N_B, N_C, N_D, L, P)$, where N_i are molar quantities, L is the liquid level (% of vessel volume), and P is the headspace pressure.

Valve–Flow Relations. Let $X_i \in [0, 100]$ be valve openings. The resulting flows are

$$\begin{aligned} F_A &= c_{v1}\,\frac{X_1}{100}, \\ F_B &= c_{v2}\,\frac{X_2}{100}, \\ F_{\text{Purge}} &= \begin{cases} c_{v3}\,\frac{X_3}{100}\,\sqrt{P - P_{\text{thr}}}, & P \geq P_{\text{thr}}, \\ 0, & P < P_{\text{thr}}, \end{cases} \\ F_{\text{Production}} &= \begin{cases} c_{v4}\,\frac{X_4}{100}\,\sqrt{P - P_{\text{thr}}}, & P \geq P_{\text{thr}},\ L > 0,\ X_4 > 0, \\ 0, & \text{otherwise.} \end{cases} \end{aligned} \tag{17}$$

Reaction Rate Species A and C react to form D with rate

$$\begin{aligned} RR &= k_{\text{par}}\left(P\,y_A\right)^{1.2}\left(P\,y_C\right)^{n_{\text{cpar}}}, \\ \text{where}\ \ y_A &= \frac{N_A}{N_V},\ y_C = \frac{N_C}{N_V},\ N_V = N_A + N_B + N_C. \end{aligned} \tag{18}$$

Mass Balances.

$$\begin{aligned} \dot{N}_A &= y_{A1}F_1 + F_2 - y_A F_{\text{purge}} - RR, \\ \dot{N}_B &= y_{B1}F_1 - y_B F_{\text{purge}}, \\ \dot{N}_C &= y_{C1}F_1 - y_C F_{\text{purge}} - RR, \\ \dot{N}_D &= RR - F_{\text{prod}}. \end{aligned} \tag{19}$$

Liquid Level Relation. We approximate level dynamics from net inflow/outflow:

$$\dot{L} \propto (F_1 + F_2) - F_{\text{prod}}. \tag{20}$$

Gas Law Headspace pressure follows the ideal gas law over vapor volume:

$$P = \frac{N_V\,R\,T}{VV}, \qquad VV = V_T - \frac{L}{100}\,V_T. \tag{21}$$

Guards and Physical Bounds.

$$\begin{aligned} X_i \in [0, 100], \quad L \in [0, 100]\%, \quad 0 \leq P \leq 3200, \\ F_{\text{Purge}} = F_{\text{Product}} = 0 \text{ if } P < P_{\text{thr}}, \\ F_{\text{Product}} = 0 \text{ if } L = 0 \text{ or } X_4 = 0. \end{aligned} \tag{22}$$

Lemma (Net vapor balance). Summing (19) over A, B, C and using $y_{A1} + y_{B1} + y_{C1} = 1$ and $y_A + y_B + y_C = 1$ yields

$$\dot{N}_V = \dot{N}_A + \dot{N}_B + \dot{N}_C = F_1 + F_2 - F_{\text{purge}} \ - \ 2\,RR. \tag{23}$$

This identity is *derived* from the PM; we do not assume it separately.


Figure 3: The simulation of chemical plant [27]. **Left:** The system is in steady state. **Right:** The system is driven to a reactor explosion.

A.2. Water Treatment

We summarize the discrete-time physical model for the SWaT raw-water tank (sensor LIT101, denoted L_1) and the ultrafiltration tank (sensor LIT301, denoted L_3), together with their physical security constraints.

Physical Model for Tank 1 (LIT101). Let $L_1(k)$ denote the water level in Tank 1 at discrete time step k (in meters), and let A be the tank cross-sectional area ($A = TANK\_SECTION$). The physical process is updated every PP_PERIOD_SEC seconds, corresponding to a time step $\Delta t = PP\_PERIOD\_HOURS$ (in hours). Let $Q_{\text{in}} = PUMP\_FLOWRATE\_IN$ and $Q_{\text{out}} = PUMP\_FLOWRATE\_OUT$ be the nominal inflow and outflow rates (m^3/h), and let $MV101(k), P101(k) \in \{0,1\}$ be the valve and pump states at time k. The raw-water tank dynamics implemented in the code can be written as:

$$\begin{aligned} V_1(k) &= A\,L_1(k) \\ Q_{\text{in},1}(k) &= \begin{cases} Q_{\text{in}}, & MV101(k)=1 \\ 0, & MV101(k)=0 \end{cases} \\ Q_{\text{out},1}(k) &= \begin{cases} Q_{\text{out}}, & P101(k)=1 \\ 0, & P101(k)=0 \end{cases} \\ V_1(k{+}1) &= V_1(k) + Q_{\text{in},1}(k)\,\Delta t - Q_{\text{out},1}(k)\,\Delta t \\ L_1(k{+}1) &= \max\left\{0,\ \frac{V_1(k{+}1)}{A}\right\}. \end{aligned} \tag{24}$$

Physical Model for Tank 3 (LIT301). Let $L_3(k)$ denote the water level in Tank 3 (ultrafiltration tank) at time step k. In the provided implementation, Tank 3 dynamics are driven by a simplified inflow/outflow balance tied to pump $P101$:

$$L_3(k{+}1) = \begin{cases} L_3(k) + (f_{\text{in},3} - f_{\text{out},3})\,\Delta t, & P101(k)=1 \\ L_3(k) - f_{\text{out},3}\,\Delta t, & P101(k)=0, \end{cases} \tag{25}$$

where $f_{\text{in},3} = TANK2\_INFLOW$ and $f_{\text{out},3} = TANK2\_OUTFLOW$ are the effective inflow and outflow rates from Tank 1 to Tank 3.

Physical Security Constraints (PSC). From the SWaT configuration, the safe operating ranges (in meters) are:

$$\begin{aligned} L_1^{LL} &= 0.25, \quad L_1^{HH} = 1.20, \\ L_3^{LL} &= 0.25, \quad L_3^{HH} = 1.20. \end{aligned} \tag{26}$$

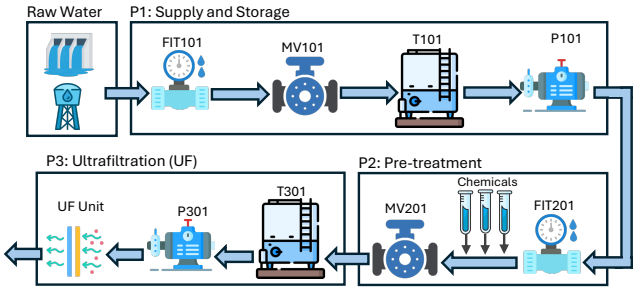

Figure 4: The simulation of water treatment [52]

The physical security constraint requires both tanks to stay within their respective bounds:

$$L_1^{LL} \le L_1(k) \le L_1^{HH} \ \wedge\ L_3^{LL} \le L_3(k) \le L_3^{HH} \tag{27}$$

Corresponding violation terms for ICSFlux-style analysis are:

$$\begin{aligned} &\text{Overfill}_1 : L_1(k) > L_1^{HH}, \quad \text{Underflow}_1 : L_1(k) < L_1^{LL}, \\ &\text{Overfill}_3 : L_3(k) > L_3^{HH}, \quad \text{Underflow}_3 : L_3(k) < L_3^{LL}. \end{aligned} \tag{28}$$

A.3. Aircraft

The following presents the abridged physical-model equations governing the drone's aerodynamic states, including angle of attack, pitch angle, pitch rate, and roll angle.

$$\begin{aligned} \alpha_{t+1} &= -0.1 \times \epsilon_1 + 0.03 \times V \times \sin(\theta_{t,rad}) \\ &-0.6 \times (\alpha_t - \alpha_0) + \alpha_t, \\ \theta_{t+1} &= -0.1 \times \epsilon_2 + 0.5 \times \theta_t, \\ \zeta_{t+1} &= 2.85 \times \epsilon_3 + \zeta_t, \\ \beta_{sp,degree} &= 0.5 \times \epsilon_4 + \beta_{curr,degree} \end{aligned} \tag{29}$$

Where:

- α: Angle of Attack
- ϵ_1: actuaotor command from the controller
- V: Aircraft speed.
- $\theta_{t,rad}$: Pitch angle in radians at time current time step.
- θ: Pitch angle at the current time step.
- ϵ_2: Elevator position control command from the controller
- ζ: change in pitch rate due to gust
- ϵ_3 : control input
- ϵ_4 actuator command from the controller
- β: roll angle, in degrees

Physical Security Constraints:

$$\begin{aligned} &0 < \alpha < 30 \wedge -10 < \theta < 20 \\ &\wedge -3 < \zeta < 3 \wedge -10 < \beta < 20 \end{aligned} \tag{30}$$

A.4. Desalination

The following presents the abridged physical-model equations governing the desalination plant.

$$T_{t+1} = 0.0215 \times \epsilon_1 + T_t,$$
$$WD_{t+1} = 1.2 \times \epsilon_2 + WD_t \tag{31}$$

- T: Temperature
- ϵ_1: Steam flow, actuator command from the controller
- ϵ_2: actuator command from the controller
- WD: quantity of water distillate

Physical Security Constraints:

$$25 < T < 110 \wedge 0 \leq WD \leq 20000 \tag{32}$$

A.5. Anaerobic

The following presents the abridged physical-model equations governing the desalination plant.

$$T_{t+1} = 0.174 \times CV + \frac{TV_t}{0.32},$$
$$C_{t+1} = \frac{C_t \times V_{influent}}{V_{influent} + (D_{base} + K * F)}, \tag{33}$$
$$pH_{t+1} = -3.1 \times log_{10}(\frac{T_{t+1}}{T_t}) + pH_t$$

- TV: current temperature converted to voltage
- CV: control voltage command from the controller
- F: flow rate of diluent
- C: Concentration of organic waste
- V: Volume of influent before dilution
- D: Baseline diluent flow rate

$$10^\circ C \leq T \leq 70^\circ C \tag{34}$$

A.6. Smart Grid

The following presents the abridged physical-model equations governing the smart grid.

$$VG_{t+1} = (K_v) \times (1 - \epsilon_1) + VG_t,$$
$$VD_{t+1} = \epsilon_2 + 0.00001 \times VD_t, \tag{35}$$
$$BL_{t+1} = BL_t + VG - (1.05 \times VD)$$

- VG: Generated voltage scaling factor of PWM signal amplitude
- VD: demanded voltage by the neighborhood.
- ϵ_1: actuator command from controller to stop charging
- ϵ_2: actuator command from controller to distribute voltage
- BL: battery level

Physical Security Constraint:

$$VD \leq 500 \wedge VG \leq 750 \wedge 100 \leq BL \tag{36}$$

Appendix B. Meta-Review

The following meta-review was prepared by the program committee for the 2026 IEEE Symposium on Security and Privacy (S&P) as part of the review process as detailed in the call for papers.

B.1. Summary

This paper presents ICSFlux, a physics-aware testing framework for industrial control systems with black-box controllers. Given a plant model and physical security constraints, it uses causal/temporal structure over physical states to guide the generation of valid supervisory command sequences toward unsafe physical behaviors.

B.2. Scientific Contributions

- Creates a New Tool to Enable Future Science.
- Provides a Valuable Step Forward in an Established Field.

B.3. Reasons for Acceptance

1) The work creates an end-to-end tool that identifies physical security violations in ICS, which can be used for future research on physics-aware ICS testing.
2) The paper provides a valuable step forward in ICS fuzzing/testing by guiding the input generation toward violations using causal/temporal structure over physical states.
3) The paper includes an extensive evaluation of ICSFlux on 11 industrial testbeds, demonstrating its generality.

B.4. Noteworthy Concerns

1) **"Black-box" framing.** While controller/firmware may be closed-sourced, the technique assumes access to a reasonably accurate plant model and explicit safety/security constraints.
2) **Positioning against CPS falsification/testing.** The paper does not clearly differentiate its contributions from robustness-guided CPS falsification and simulator-based CPS testing, which also guide search toward violations using quantitative robustness/distance signals and assume models/safety-constraints for their search.
3) **Dependence on Model Accuracy.** The effectiveness depends on the accuracy of the models, but the paper misses a sensitivity analysis with respect to errors/inaccuracies in the models.
4) **Assumptions and Deployment.** The paper does not clearly describe how ICSFlux would be used in practice and why its intended users lack access to the control logic (but they have access to the physical model and constraints).