

HyTwin: Hybrid Program Semantics for Digital Twin-based Security Interventions in Industrial Control Systems

Jainta Paul¹, Stefan Mitsch², and Luis Garcia¹

¹ University of Utah, Salt Lake City, Utah, USA
u1471999@utah.edu, la.garcia@utah.edu

² School of Computing, DePaul University, Chicago, Illinois, USA
smitsch@depaul.edu

Abstract. Industrial control systems (ICS) are increasingly targeted by sophisticated attacks on sensors and actuators, necessitating advanced frameworks that enable proactive mitigation. This paper introduces HyTwin, a formal framework that models both adversarial actions and corresponding mitigation strategies through digital twin-based interventions. HyTwin leverages differential dynamic logic (dL) to represent the temporal evolution of attacks and quantify the mitigation horizon, a critical parameter enabling precise reasoning about when and how to deploy fail-safe mechanisms during ongoing attacks. Our approach integrates temporal semantics with attack models to dynamically engage fail-safe controls. This work provides a rigorous framework for designing proactive countermeasures that preserve system safety, ensuring robustness in adversarial scenarios. The proposed framework establishes a foundation for advancing ICS security through verifiable temporal reasoning and contributes to bridging gaps between theoretical modeling and real-world industrial applications.

Keywords: Cyber-physical Systems · Differential Dynamic Logic · Digital Twin Security

1 Introduction

Digital twins have emerged as a transformative technology in industrial control systems (ICS), offering advanced capabilities for modeling, monitoring, and optimizing cyber-physical processes. Initially popularized for their potential in manufacturing optimization and predictive maintenance, digital twins have increasingly gained attention as a tool for enhancing security in ICS environments, particularly in addressing “forever-day vulnerabilities” where legacy systems cannot be updated [9, 3]. With the rise of sophisticated threats targeting ICS—ranging from supply chain attacks to sensor and actuator manipulation—the integration of real-time monitoring and proactive interventions enabled by digital twins has become a critical area of research.

While digital twins have demonstrated significant potential in industrial control systems, there remain critical gaps in their formal semantics and standardization, particularly for security applications. Current standards, such as ISO 23247, propose a reference architecture for digital twins in manufacturing, emphasizing composability and interoperability but stop short of addressing security-specific semantics or real-time interventions [6]. Similarly, frameworks like HyPLC [8] provide hybrid program representations of PLC code and physical plant models but are limited to verifying the safety of individual processes and do not incorporate adversarial scenarios. Efforts to extend dL for ICS security have primarily focused on modeling robustness against sensor spoofing attacks, introducing quantitative metrics for assessing resilience under adversarial conditions [5]. However, these approaches did not consider the integration of digital twins or temporal dynamics necessary for proactive intervention. This omission highlights the need for a unified framework that leverages temporal semantics to enable real-time fail-safe controls within digital twin architectures.

In this work, we present HYTWIN, a formal framework for enabling proactive digital twin interventions in industrial control systems using hybrid program representations. At its core, HYTWIN formalizes attack models relative to the operational semantics of ICS, capturing the dynamics of adversarial actions within a dL framework. We subsequently model how a digital twin would intervene to mitigate such attacks, ensuring the preservation of safety properties under adversarial conditions. Central to our approach is the formal integration of temporal reasoning into digital twin architectures. By explicitly modeling the progression of attacks and the timing of interventions, HYTWIN introduces a rigorous basis for dynamically triggering fail-safe controls. We introduce a *mitigation horizon* to provide actionable insights into the urgency of interventions, enabling proactive decision-making based on the system’s temporal dynamics.

To illustrate the efficacy of our framework, we present a formal methodology and demonstrate its applicability to a realistic ICS setup, the miniSWaT [15] industrial control system testbed. Specifically, we show how the formally verified and validated digital twin interventions of HYTWIN can parameterize attack detectors to identify known attack signatures and guide proactive mitigation strategies, ensuring safety properties are preserved under adversarial conditions.

Contributions. Our contributions are as follows:

- We present HYTWIN (Section 3.1), a formal framework for modeling adversarial attack behaviors on ICS sensors and actuators. HYTWIN defines attack dynamics relative to the operational semantics of ICS using differential dynamic logic (dL), enabling rigorous analysis of attacks (Section 3.2).
- We formalize digital twin-based intervention strategies within HYTWIN, providing a systematic approach for triggering and executing mitigations in response to evolving attack scenarios. This formalization ensures the preservation of safety properties under adversarial conditions (Section 3.3).
- We introduce the concept of a *mitigation horizon*—a principled basis for guiding proactive interventions and parameterizing attack detectors to recognize known attack signatures (Section 3.4).

2 Preliminaries

In this section, we summarize background theory, related work, and introduce notation.

2.1 Digital Twin-Based Interventions in ICS

Digital twins are virtual counterparts of physical systems, providing a synchronized, real-time view of their state and behavior. In industrial control systems, digital twins have been widely recognized for their potential to enhance system safety and security by enabling real-time monitoring, anomaly detection, and proactive intervention [2, 10]. Digital twins can leverage formal methods to reason about system dynamics and drive time-sensitive responses to adversarial actions [4].

A common component of digital twin-based intervention is integrating a *physics-aware model*, which encapsulates the expected state and behavior of the ICS based on physical laws, operational constraints, and control logic. Prior works [1, 19] leverage digital twins to continuously monitor the physical system (the *real twin*) for discrepancies between observed and expected behavior. Such discrepancy detection is critical for identifying adversarial actions or system faults. However, existing approaches often stop at ad-hoc anomaly detection, lacking both formal semantics to rigorously model system behavior under adversarial conditions and the structured intervention strategies needed to dynamically mitigate threats and ensure system safety.

2.2 Differential Dynamic Logic (dL)

Differential dynamic logic dL [17] expresses hybrid systems written as hybrid programs. The syntax of *hybrid programs* is described by the following grammar where α, β are hybrid programs, x is a variable and $e, f(x)$ are arithmetic expressions (terms) in $+, -, \cdot, /$ over the reals, Q is a dL formula:

$$\alpha, \beta ::= x := e \mid x := * \mid ?Q \mid x' = f(x) \& Q \mid \alpha \cup \beta \mid \alpha; \beta \mid \alpha^*$$

Assignment $x := e$ assigns the value of term e to x (e.g., compute valve position to meet flow); nondeterministic assignment $x := *$ assigns any real value to x . Tests $?Q$ abort execution and discard the run if Q is not true, possibly backtracking to other nondeterministic alternatives. Differential equations $x' = f(x) \& Q$ are followed along a solution of $x' = f(x)$ for any duration as long as the evolution domain constraint Q is true at every moment along the solution (e.g., tank level l changes according to flow f , but does not become negative $l' = f \& l \geq 0$). Nondeterministic choice $\alpha \cup \beta$ runs either α or β (e.g., open or close a valve), sequential composition $\alpha; \beta$ first runs α and then β on the resulting states of α (e.g., first control, then physics), and nondeterministic repetition α^* runs α any natural number of times (e.g., repeated control and environment loop). Nondeterministic assignments with tests restrict the chosen

values to some set (e.g., choose flow within pipe limits: $f := *; ?0 \leq f \leq F$). Conditionals and loops can be expressed: if P then α else $\beta \equiv (?P; \alpha) \cup (? \neg P; \beta)$ and while P do $\alpha \equiv (?P; \alpha)^*; ? \neg P$.

The formulas of dL describe properties of hybrid programs and are described by the following grammar where P, Q are formulas, f, g are terms, $\sim \in \{<, \leq, =, \neq, \geq, >\}$, x is a variable and α is a hybrid program:

$$P, Q ::= f \sim g \mid \neg P \mid P \wedge Q \mid P \vee Q \mid P \rightarrow Q \mid P \leftrightarrow Q \mid \forall x P \mid \exists x P \mid [\alpha]P \mid \langle \alpha \rangle P$$

The operators of first-order real arithmetic are as usual with quantifiers ranging over the reals. For any hybrid program α and dL formula P , the formula $[\alpha]P$ is true in a state iff P is true after all runs of α . Its dual, $\langle \alpha \rangle P$ is true in a state iff P is true after at least one run of α .

The semantics of dL is a Kripke semantics in which the states of the Kripke model are the states of the hybrid system. A state in the set of all states \mathcal{S} is a map $\omega : \mathcal{V} \rightarrow \mathbb{R}$, assigning a real value $\omega(x)$ to each variable $x \in \mathcal{V}$ in the set of all variables \mathcal{V} . We write $\llbracket Q \rrbracket$ to denote the set of states in which formula Q is true, $\omega \in \llbracket Q \rrbracket$ if formula Q is true at state ω , $\omega \llbracket e \rrbracket$ to denote the real value of term e in state ω , and ω_x^e to denote the state ν that agrees with ω except that $\nu(x) = \omega \llbracket e \rrbracket$. We use $\mathcal{V}(\alpha)$ to refer to all variables mentioned in program α , and $\text{BV}(\alpha)$ to denote the bound variables of program α , see [17]. The semantics of hybrid programs is expressed as a state-transition relation $\llbracket \alpha \rrbracket \subseteq \mathcal{S} \times \mathcal{S}$ [17]. The differential equations and nondeterministic alternatives in hybrid programs make them an expressive specification language, but require computationally expensive methods similar to online reachability analysis for execution, which is detrimental to their runtime use in monitoring. Next, we review ModelPlex [12] to shift much of this computational complexity offline.

Notation. We use $S_{\downarrow x}$ to denote state S restricted to variables x . Variables in definitions are implicitly vectorial; we use $u : \in \text{hp}(x)$ for the potentially nondeterministically chosen vectorial values u by a program $\text{hp}(x)$ with free vectorial variables x . We use $\begin{pmatrix} x_1 \\ \dots \\ x_n \end{pmatrix} := \begin{pmatrix} v_1 \\ \dots \\ v_n \end{pmatrix}$ to abbreviate the sequence of assignments $x_1 := v_1; \dots; x_n := v_n$.

2.3 ModelPlex

ModelPlex [12] combines a universal offline safety proof $[\alpha]P$ with an existential reachability check whether two concrete states ω, ν are connected by the program α , i.e., whether $(\omega, \nu) \in \llbracket \alpha \rrbracket$. The safety proof witnesses that *all* states reachable by program α satisfy P , while passing the reachability check witnesses that the two concrete states ω, ν are connected by the program α , and so state ν inherits the safety property, i.e., $\nu \in \llbracket P \rrbracket$.

The set \mathcal{M} of ModelPlex formulas $\phi \subseteq \mathcal{S} \times \mathcal{S}$ is generated by the following grammar ($\sim \in \{\leq, <, =, \neq, >, \geq\}$ and θ, η form the arithmetic expressions of the set \mathcal{T} of ModelPlex terms in $+, -, \cdot, /$ over the reals, i.e., $\theta : \mathcal{S} \times \mathcal{S} \rightarrow \mathbb{R}$):

$$\phi, \psi ::= \theta \sim \eta \mid \neg \phi \mid \phi \wedge \psi \mid \phi \vee \psi$$

When a ModelPlex formula $\phi \in \mathcal{M}$ is satisfied over states ω, ν , we write $(\omega, \nu) \models \phi$. Since any concrete program α and, thus, ModelPlex formula, will be phrased over finitely many variables, a ModelPlex formula ϕ can be encoded as a standard dL formula ϕ_{dL} using a fresh variable x^+ for each variable $x \in \text{BV}(\alpha)$: $(\omega, \nu) \models \phi$ then is shorthand notation for $\omega_{x^+}^{\nu(x)} \in \llbracket \phi_{\text{dL}} \rrbracket$. The reachability check $(\omega, \nu) \in \llbracket \alpha \rrbracket$ is thus equivalently phrased in dL as a monitor specification $\langle \alpha \rangle \bigwedge_{x \in \text{BV}(\alpha)} (x = x^+)$ [12]. To exclude a set $\bar{X} \subseteq \text{BV}(\alpha)$ of variables from monitoring, we use $\langle \alpha \rangle (\exists_{x^+ \in \bar{X}} x^+ \bigwedge_{x \in \text{BV}(\alpha)} (x = x^+))$ [13].

A ModelPlex formula $\phi \in \mathcal{M}$ is guaranteed to not miss alarms for program α when $\phi_{\text{dL}} \rightarrow \langle \alpha \rangle \bigwedge_{x \in \text{BV}(\alpha)} (x = x^+)$ is valid [12]; in contrast to online reachability analysis, this approach shifts computation offline by using theorem proving to translate a hybrid systems model into a ModelPlex formula.

ModelPlex formulas are quantifier-free real arithmetic formulas, and are therefore computationally inexpensive to evaluate from concrete measurements at runtime [7], which makes them attractive for use in security monitoring. This paper exploits ModelPlex for security monitoring, where a monitor derived from a formal model of the system behavior is used to detect deviations from expected behavior (i.e., a monitor violation witnesses deviation from expected behavior), and monitors derived from attack models are used to detect presence of certain attacks (i.e., a satisfied monitor witnesses presence of an attack). ModelPlex monitors can be interpreted quantitatively [18] to obtain a robustness measure of the monitor decision in addition to a Boolean verdict.

2.4 Hybrid Program Modeling for PLC Scan Cycle Verification

In a prior work, HyPLC [8], dL was used to formalize a specific hybrid program structure, termed *scan cycle normal form*, to model a PLC (Programmable Logic Controller) scan cycle. This model reflects the cyclic PLC process of scanning inputs, executing control logic, and setting outputs to actuators within a fixed duration ε . The scan cycle normal form has the shape

$$(i : \in \text{read}(x); u : \in \text{ctrl}(i); t := 0; \{x' = f(x, u), t' = 1 \ \& \ t \leq \varepsilon\} \ .$$

In this model, $i : \in \text{read}(x)$ represents inputs (e.g., from sensors), while $u : \in \text{ctrl}(i)$ selects control actions u based on the inputs i . The plant $t := 0; \{x' = f(x, u), t' = 1 \ \& \ t \leq \varepsilon\}$ evolves the system state x according to the control actions u , governed by differential equations $x' = f(x, u)$, while keeping track of the scan cycle time constraint ε with a clock t ; if necessary, the plant can be extended with discrete modifications of the variables x . A program in scan cycle normal form is guaranteed to maintain property S under assumptions A if the following formula is valid:

$$A \rightarrow [(i : \in \text{read}(x); u : \in \text{ctrl}(i); t := 0; \{x' = f(x, u), t' = 1 \ \& \ t \leq \varepsilon\})^*] S$$

Our framework builds on the scan cycle normal form to integrate digital twin-based interventions, further extending the safety guarantees of PLC-driven systems under adversarial conditions.

3 Digital Twin-based Security Interventions

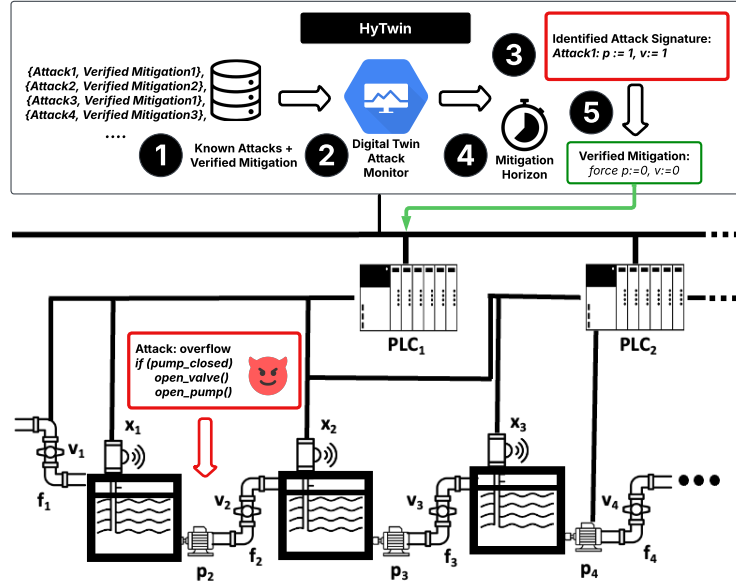


Fig. 1. HyTWIN Framework for digital twin-based security interventions.

3.1 HyTWIN Overview

The *HyTwin* framework provides a structured approach to detecting and mitigating security threats in industrial control systems (ICS) using digital twin-based interventions. The *HyTwin* framework (Figure 1) consists of five key components: **1 Known Attacks & Verified Mitigations**: The system maintains a database of previously encountered attacks and their verified countermeasures. This repository informs the selection of mitigation strategies for newly detected threats. **2 Digital Twin Attack Monitor**: This component continuously observes sensor and control signals to detect anomalies indicative of adversarial activity. It utilizes formalized attack models to systematically identify deviations from expected behavior. **3 Identified Attack Signature**: Once an anomaly is detected, the system classifies it based on known attack patterns. The attack monitor maps the observed deviation to a predefined attack signature. **4 Mitigation Horizon**: The framework calculates a mitigation horizon, which determines the time window available for counteracting an attack before system safety is compromised. **5 Mitigation Execution**: If an attack is confirmed and falls within the mitigation horizon, the framework enforces a verified countermeasure. These countermeasures are selected from the repository of known mitigations or adapted dynamically based on system constraints.

To illustrate this framework in practice, we introduce a use case water treatment system. In this scenario, HYTWIN detects an attack that attempts to force a valve into an unsafe position. By leveraging its digital twin attack monitor, the system identifies the attack signature, calculates the available mitigation horizon, and executes an intervention to restore the correct system state.

Simplified water tank use case. To illustrate our approach, as a running example, we use a simple water tank component taken from the first of six control processes of a water treatment testbed [11], depicted in Figure 1. This process is responsible for taking in water from a raw water source and feeding it into a tank. This water will then be pumped out into a second tank to be treated with chemicals. For this first process, PLC1 is responsible for controlling the inflow of water for both tanks by opening or closing valves, v_1 and v_2 , as well as the outflow of water to the second tank by running the pump, p_2 . The PLC monitors the water level of both water tanks, x_1 and x_2 , to ensure that v_1 and v_2 , respectively, are closed before each respective tank overflows beyond an upper bound, H_1 and H_2 . The PLC is additionally responsible for protecting the outflow pump, p_2 , by ensuring that the pump is off if the water level of x_1 is below a lower threshold, L_1 .

The first component of the process is formalized in the dL model of a water treatment testbed based on [8].

Example 1 (Water Tank). The water tank model uses flow sensors f_1 and f_2 , a controller to control the inflow and outflow of the first tank, and a differential equation model of the tank levels.

$$\begin{aligned}
\text{tank} &\equiv (i : \in \text{read}(x); u : \in \text{ctrl}(i); t := 0; \text{plant}_{\text{tank}}) \\
i : \in \text{read}(x) &\equiv \hat{x}_1 := x_1; \hat{x}_2 := x_2; f_1 := *; f_2 := *; ?0 \leq f_1 \leq F_1 \wedge 0 \leq f_2 \leq F_2 \\
u : \in \text{ctrl}(i) &\equiv \text{inflow}; \text{outflow}; \text{protect} \\
\text{inflow} &\equiv \begin{cases} v_1 := 0 & \text{if } f_1 > \frac{H_H - \hat{x}_1}{\varepsilon} \\ v_1 := 1 & \text{else if } \hat{x}_1 \leq L_1 \\ ?\text{true} & \text{else} \end{cases} \\
\text{outflow} &\equiv \begin{cases} p_2 := 1; v_2 := 1 & \text{if } \hat{x}_2 \leq L_2 \\ ?\text{true} & \text{else} \end{cases} \\
\text{protect} &\equiv \begin{cases} p_2 := 0; v_2 := 0 & \text{if } v_1 \cdot f_1 - v_2 \cdot p_2 \cdot f_2 < \frac{L_L - \hat{x}_1}{\varepsilon} \\ & \vee v_2 \cdot p_2 \cdot f_2 > \frac{H_H - \hat{x}_2}{\varepsilon} \\ ?\text{true} & \text{else} \end{cases} \\
\text{plant}_{\text{tank}} &\equiv x'_1 = v_1 \cdot f_1 - v_2 \cdot p_2 \cdot f_2, x'_2 = v_2 \cdot p_2 \cdot f_2, t'_1 = 1 \ \& \ Q_{\text{tank}} \wedge t \leq \varepsilon \\
Q_{\text{tank}} &\equiv x_1 \geq 0 \wedge x_2 \geq 0
\end{aligned}$$

Infinite-time safety of the tank model is expressed in **dL** as follows:

$$\begin{aligned}
A_{\text{tank}} &\rightarrow [\text{tank}^*]S_{\text{tank}} \\
A_{\text{tank}} &\equiv L_L < L_1 \leq x_1 \leq H_1 < H_H \\
&\quad \wedge L_L < L_2 \leq x_2 \leq H_2 < H_H \\
&\quad \wedge v_1 = 0 \wedge v_2 = 0 \wedge p_2 = 0 \\
S_{\text{tank}} &\equiv L_L \leq x_1 \leq H_H \wedge L_L \leq x_2 \leq H_H
\end{aligned}$$

A proof in **dL** witnesses infinite-time safety of the tank model; the proof bases on finding an inductive invariant I_{tank} , i.e., the formulas $A_{\text{tank}} \rightarrow I_{\text{tank}}$, $I_{\text{tank}} \rightarrow S_{\text{tank}}$, and $I_{\text{tank}} \rightarrow [\text{tank}]I_{\text{tank}}$ are valid. ■

Using [14], the component results obtained from analyzing PLC1 integrate with other component results into system-wide guarantees. In order to make this component resilient to attacks, in the following sections we formalize attack capabilities and develop formal techniques to analyze how long mitigation strategies are able to keep the component operational under attack.

3.2 Attack Model Formalization

In our approach, we assume an attacker that tempers with sensors, control code, or actuation, but does not have physical access to alter the actual system state (e.g., the attacker can change water level measurements, but not modify the true water level physically). Note, that the examples in the formalization are deliberately simplistic to convey intuition rather than illustrate true attacks and mitigation strategies. To define classes of attacks, our approach summarizes attack capabilities of the attacker in a hybrid program, see Def. 1.

Definition 1 (Attack Program and Signature). *An attack program is a hybrid program $A(x) : \mathcal{S}_{\downarrow x} \rightarrow \mathcal{S}$ with free variables x that produces a modified (compromised) state. The set of reachable states of an attack program $\llbracket A(x) \rrbracket$ is called the attack signature.*

An attack program transforms system state according to attacker objectives, often while maintaining stealthiness properties. This transformation can take place instead of, or before and/or after control to affect sensor readings, actuator commands, or both.

Definition 2 (System under Attack). *The structure of a system in scan-cycle normal form with inputs i , control program $u : \in \text{ctrl}(i)$, and dynamics $x' = f(x, u) \ \& \ Q \wedge t \leq \varepsilon$ under attack $A(i)$ is:*

$$\begin{aligned}
\text{attacked}(A(i)) &\equiv (i : \in \text{read}(x); && \text{(sense)} \\
&\quad (u : \in \text{ctrl}(i) \cup u : \in A(i)); && \text{((compromised) control)} \\
&\quad t := 0; && \text{(reset timer)} \\
&\quad \{x' = f(x, u), t' = 1 \ \& \ Q \wedge t \leq \varepsilon\} && \text{(dynamics)} \\
&\quad)^* && \text{(repeat)}
\end{aligned}$$

The attack structure describes attacks that target the control output u by modifying, for instance, the control code itself, the sensor inputs, the actuator values, or the communication between the controller and the actuator. The nondeterministic choice $u:\in\text{ctrl}(i) \cup u:\in\text{A}(i)$ models that the attack may happen continuously or intermittently at an arbitrary time during the execution for a nondeterministic duration. Sensor and input attacks can be modeled in $\text{A}(i)$ by altering the sensor and input values before forwarding to $u:\in\text{ctrl}(i)$. The physical plant dynamics operate on the true system state x and the potentially compromised control choices u .

Definition 3 (Attack Trace). *An attack trace is a trace s_0, s_1, \dots, s_n of states traversed by n steps of an attacked system that ends in violating a safety property S , so $(s_{i-1}, s_i) \in \llbracket \text{attacked}(\text{A}(i)) \rrbracket$ for $1 \leq i \leq n$ and $s_n \models \neg S$.*

Note that an attack program can be nondeterministic and symbolic, which means its attack signature can express many attack traces; by analyzing attack mitigation strategies for attack programs, rather than for attack traces, we reduce the offline analysis burden and certify general mitigation strategies for entire classes of attacks.

We start our analysis of attack mitigation strategies from known and recorded attack traces. We implemented attacks and recorded attack traces on the miniSWAT testbed [15] inspired by known attacks on the SWAT testbed [11]. These are typically collected after the fact, or by existing predicate-based attack detection systems. Such attack detection predicates describe the effect of an attack on observable quantities, which can be formalized with attack programs as follows:

Definition 4 (Attack Detector). *An attack detector $\text{D}(x_0, x)$ characterizes attacks by their effect on a previous state x_0 and a current state x and the relationship between them. The formula $\text{D}(x_0, x)$ is satisfied when an attack is detected; the attack signature characterized by $\text{D}(x_0, x)$ is formalized in dL with the attack program $x_0 := x; x := *; ?\text{D}(x_0, x)$.*

In a cyber-physical system, any deviation from expected behavior can be safety-critical, regardless of whether it is caused by an attack or hardware malfunction. ModelPlex [12] and SCADMAN [1] are techniques to monitor such deviation. Their monitoring formulas are expressible as attack detectors: SCADMAN includes manually crafted attack detectors; in contrast, ModelPlex monitors are satisfied when observed behavior agrees with expected behavior, so the negation of a ModelPlex monitor formula is an attack detector.

We classify ICS attacks into three fundamental categories with increasing sophistication, each representing a distinct programmatic approach to compromising system state. The following grammar defines the shapes of common attacks:

$$\begin{aligned}
 \text{attack} ::= & u := c && \text{(Direct attack)} \\
 & | \tilde{u}:\in\text{ctrl}(i); u:\in\tilde{u} \pm \delta && \text{(Bounded attack)} \\
 & | \bigcup_j (?p_j(i); \text{attack}) \cup ?\text{true} && \text{(State-dependent attack)}
 \end{aligned}$$

Direct attack $u := c$ assigns a malicious value c to control choice u , representing the most basic form of attack.

Bounded attack $\tilde{u} := \text{ctrl}(i); u := \tilde{u} \pm \delta$ nondeterministically perturbs the control output \tilde{u} within bounds δ .

State-dependent attack $\bigcup_j (?p_j(i); \text{attack}) \cup ?\text{true}$ inspects the system state to decide when to engage.

Example 2 (Direct Actuator Attack). The attacker directly forces valve v_1 open even when the raw water tank level x_1 exceeds the control upper limit H_1 : $v_1 := 1$. ■

Note, that the loop $*$ in the overall system under attack model allows activating a series of different attack steps, one on each loop iteration.

Definition 5 (Stealthy Attack). An attack is stealthy if it evades detection, i.e., the attack signature of a stealthy attack for a set of attack detectors \mathcal{D} is formalized in dL by the attack program $x_0 := x; x := *; ?(\neg \bigvee_{D \in \mathcal{D}} D(x_0, x))$.

In order to analyze the effectiveness of attack mitigation strategies, we formalize attack mitigation as hybrid programs that are able to re-establish safety.

3.3 Attack Detection and Mitigation

Attack detection and mitigation is a central aspect of system resilience; in this section, we formalize attack monitors, mitigation strategies, as well as bounds on the duration for which a system remains operational under attack.

Definition 6 (Attack Mitigation). An attack mitigation $M(\tilde{x}) : \mathcal{S}_{\downarrow \tilde{x}} \rightarrow \mathcal{S}$ is a program to establish a property P relative to assumptions A even in the presence of an attack, i.e., for an effective mitigation counteracting attack $A(x)$ the following formula is valid:

$$A \rightarrow [A(x)] \langle M(\tilde{x}) \rangle P$$

The attack mitigation characterizes how a known attack unfolds in terms of steps modeled in the program $A(x)$, and how an effective mitigation must react. Note that the mitigation and the attack may, but not necessarily, access different variables. A mitigation is effective if *there exists an execution* ($\langle M(\tilde{x}) \rangle$) for all possible attack behaviors ($[A(x)]$) such that property P is true in the end.

Example 3 (Direct Actuator Attack Mitigation). A mitigation for the direct actuator attack of Example 2 through a redundant uncompromised PLC overwrites compromised IO memory values with correct values, which sets v_1 to closed:

$$A_{\text{tank}} \rightarrow \underbrace{[v_1 := 1]}_{A(x)} \underbrace{\langle v_1 := 0 \rangle}_{M(\tilde{x})} \underbrace{[\text{plant}_{\text{tank}}]}_P (L_L \leq x_1 \leq H_H)$$

■

From the attack mitigation characterization, we use ModelPlex to derive an attack monitor that is *satisfied when the attack is detected*:

Definition 7 (Attack Monitor). *An attack monitor $\text{mon}(A(x))$ is characterized by $\langle A(x) \rangle \bigwedge_{v \in BV(A(x))} (v^+ = v)$; to exclude unobservable internal variables $I \subseteq BV(A(x))$ of the attack, we use $\langle A(x) \rangle \exists I^+ \bigwedge_{v \in BV(A(x))} (v^+ = v)$. A dL proof certifies that $\text{mon}(A(x)) \in \mathcal{M}$ does not miss alarms, i.e., $\text{mon}(A(x)) \rightarrow \langle A(x) \rangle \bigwedge_{v \in BV(A(x))} (v^+ = v)$ is valid. The attack monitor is activating an effective attack mitigation $M(\tilde{x})$ when two states ω and ν satisfy the monitor (i.e., presence of attack) and the monitor is started in states satisfying the attack mitigation assumption A :*

- Monitor startup: $\nu \in \llbracket A \rrbracket$
- Monitoring and mitigation: $\begin{cases} M(\tilde{x}) & \text{if } (\omega, \nu) \models \text{mon}(A(x)) \\ ?\text{true} & \text{otherwise} \end{cases}$

By [12], a satisfied monitor means that—from an attack mitigation proof of $A \rightarrow \llbracket (A(x)) \langle M(\tilde{x}) \rangle P \rrbracket$ —we inherit the property $\langle M(\tilde{x}) \rangle P$ at runtime, which means that we have a proof that mitigation $M(\tilde{x})$ will be effective in addressing the attack $A(x)$.

Example 4 (Direct Actuator Attack Monitor). The attack monitor specification $\langle v_1 := 1 \rangle (v_1^+ = v_1)$ characterizes the behavior of the attack program of Example 2. A proof in dL certifies that the program- and quantifier-free monitor formula $v_1^+ = 1$ does not miss alarms. This formula is evaluated at runtime by providing the current state values for v_1^+ . ■

3.4 Mitigation Analysis

For analyzing an attack mitigation strategy, it is crucial to understand how much time is left until mitigation is impossible. To characterize the execution time of a hybrid program, we extend it with a fresh mitigation clock c .

Definition 8 (Timed Program). *A timed program $\text{timed}_c(\alpha)$ is a program α extended with a fresh mitigation clock variable $c \notin \mathcal{V}(\alpha)$, recursively defined as follows:*

$$\begin{aligned} \text{timed}_c(x := e) &\triangleright x := e \\ \text{timed}_c(?P) &\triangleright ?P \\ \text{timed}_c(\alpha \cup \beta) &\triangleright \text{timed}_c(\alpha) \cup \text{timed}_c(\beta) \\ \text{timed}_c(\alpha^*) &\triangleright \text{timed}_c(\alpha)^* \\ \text{timed}_c(x' = f(x) \ \& \ Q) &\triangleright x' = f(x), c' = 1 \ \& \ Q \end{aligned}$$

We use the mitigation clock in a timed program $\text{timed}_c(\alpha)$ to characterize the *safety horizon* of a program α , i.e., the cutoff time until when a property P necessarily is preserved by program α and when it may start violating the property.

Definition 9 (Safety Horizon). *The safety horizon τ of a program α relative to assumptions A and conditions P is characterized by the following formula:*

$$A \rightarrow \exists \tau ([c := 0; \text{timed}_c(\alpha); ?c \leq \tau] P \\ \wedge \forall \tilde{\tau} > \tau \langle c := 0; \text{timed}_c(\alpha); ?c \leq \tilde{\tau} \rangle \neg P)$$

It is beneficial for further analysis to determine a witness for the existentially quantified τ . Such a witness can serve as the cutoff time until when mitigation of the attacks characterized by an attack program $A(x)$ is possible with a mitigation strategy $M(\tilde{x})$, which is expressed in terms of the safety horizon as follows:

Definition 10 (Mitigation Horizon). *The mitigation horizon for a mitigation $M(\tilde{x})$ of an attack $A(x)$ under assumptions A and conditions S is the safety horizon τ for $P \equiv \langle M(\tilde{x}) \rangle S$ and a program α following the system under attack model of Def. 2.*

The mitigation horizon of a component characterizes for how long a component remains operational under attack.

Example 5 (Direct Actuator Attack Mitigation Horizon). The mitigation horizon of mitigation $v_1 := 0$ for a water tank under attack (Example 1 extended to the shape of Def. 2)

$$\text{tank}_{A_{v_1}} \equiv (i \in \text{read}(x); (u \in \text{ctrl}(i) \cup u \in A(i)); t := 0; \text{plant}_{\text{tank}}) \\ u \in A(i) \equiv v_1 := 1$$

is computed from the characterization

$$A_{\text{tank}} \rightarrow \exists \tau ([c := 0; \text{timed}_c(\text{tank}_{A_{v_1}})^*; ?c \leq \tau] P \\ \wedge \forall \tilde{\tau} > \tau \langle c := 0; \text{timed}_c(\text{tank}_{A_{v_1}})^*; ?c \leq \tilde{\tau} \rangle \neg P) \\ \text{with } P \equiv \langle v_1 := 0 \rangle [\text{plant}_{\text{tank}}] (L_L \leq x_1 \leq H_H)$$

A symbolic program- and quantifier-free witness of τ (e.g. $\tau = \frac{(H_H - \hat{x}_1)}{f_1}$ for the worst-case behavior of an attack at tank level \hat{x}_1 and outflow $v_2 \cdot p_2 \cdot f_2 = 0$) can provide additional insight for implementing mitigation strategies. ■

For attacks that avoid direct detection and mitigation (e.g., an attack that can only be detected by observing the system state over some period, or an attack that keeps overwriting the inflow valve even when the mitigation is active), the mitigation horizon becomes part of the attack mitigation analysis:

Example 6 (Delayed Actuator Attack Mitigation). If the outflow is sufficiently strong (at least the inflow), opening the outflow valve v_2 and turning on the outflow pump p_2 before the mitigation horizon τ is exceeded mitigates the actuator attack on the first water tank component:

$$A_{\text{tank}} \rightarrow \exists \tau ([c := 0; (\text{tank}_{A_{v_1}})^*; ?c \leq \tau] \langle (\frac{v_2}{p_2}) := (\frac{1}{1}) \rangle [\text{plant}_{\text{tank}}] S)$$

Otherwise, if the outflow is not strong enough, the outflow mitigation becomes part of the attack model and mitigation horizon analysis informs us how much time is left until the attack must be removed from the system:

$$\begin{aligned}
A_{\text{tank}} \rightarrow \exists \tau (& [\alpha(\tau)] \langle (\begin{smallmatrix} v_1 \\ p_1 \end{smallmatrix}) := (\begin{smallmatrix} 0 \\ 0 \end{smallmatrix}) \rangle [\text{tank}^*] S \\
& \wedge \forall \tilde{\tau} > \tau \langle \alpha(\tilde{\tau}) \rangle \neg \langle (\begin{smallmatrix} v_1 \\ p_1 \end{smallmatrix}) := (\begin{smallmatrix} 0 \\ 0 \end{smallmatrix}) \rangle [\text{tank}^*] S \\
& \text{where } \alpha(T) \equiv c := 0; \text{timed}_c(\text{tank}_{Av_1})^*; ?c \leq T; \\
& \quad (\begin{smallmatrix} v_2 \\ p_2 \end{smallmatrix}) := (\begin{smallmatrix} 1 \\ 1 \end{smallmatrix}) ; (\text{tank}_{Av_1})^*
\end{aligned}$$

■

We now combine attack monitoring $\text{mon}(A(x))$ and attack mitigation $M(\tilde{x})$ to formalize a shielded system that is provably robust to attacks $A(x)$.

Definition 11 (Shielded System). *A shielded system under attack (Def. 2) is equipped with an attack monitor (Def. 7) that switches to a verified effective attack mitigation (Def. 6) on detection of an attack. The notation α^d refers to mitigation being under angelic choice [16] (mitigation decides how to react).*

$$\begin{aligned}
\text{shielded}_{M(i)}(A(i)) \equiv & (i : \in \text{read}(x); \\
& (u : \in \text{ctrl}(i) \cup u : \in A(i)); \\
& (\text{if } \text{mon}(A(i)) \text{ then } (u : \in M(i))^d); \\
& t := 0; \\
& \{x' = f(x, u), t' = 1 \ \& \ Q \wedge t \leq \varepsilon\} \\
&)^*
\end{aligned}$$

Theorem 1 (Shielded System is Safe). *A safe unattacked system*

$$(i : \in \text{read}(x); u : \in \text{ctrl}(i); t := 0; \{x' = f(x, u), t' = 1 \ \& \ t \leq \varepsilon\})^*$$

in scan-cycle normal form is robust against attacks $A(i)$ when shielded with a monitor that does not miss alarms to invoke an effective attack mitigation $M(i)$ that preserves $[t := 0; \{x' = f(x, u), t' = 1 \ \& \ t \leq \varepsilon\}]I$, i.e., if the formula

$$A \rightarrow [(i : \in \text{read}(x); u : \in \text{ctrl}(i); t := 0; \{x' = f(x, u), t' = 1 \ \& \ t \leq \varepsilon\})^*]S$$

is valid and proved using an inductive invariant I such that $A \rightarrow I$, $I \rightarrow S$, and $I \rightarrow [i : \in \text{read}(x); u : \in \text{ctrl}(i); t := 0; \{x' = f(x, u), t' = 1 \ \& \ t \leq \varepsilon\}]I$ are valid, if $\text{mon}(A(i)) \rightarrow \langle A(i) \rangle \bigwedge_{v \in BV(A(i))} (v^+ = v)$ is valid (the monitor does not miss alarms), and if $I \rightarrow [A(i)] \langle M(i) \rangle [t := 0; \{x' = f(x, u), t' = 1 \ \& \ t \leq \varepsilon\}]I$ is valid (mitigation is effective), then the following formula is valid as well:

$$A \rightarrow [\text{shielded}_{M(i)}(A(i))]S .$$

Proof. After expanding conditionals “if P then α ” to a nondeterministic choice $(?P; \alpha) \cup ?\text{true}$, safety of a shielded system follows from a safe unattacked system according to the following two cases:

Unsatisfied monitor (no attack) Safety proof of unattacked system applies;
Satisfied monitor (attack) From a satisfied monitor $(\omega, \nu) \models \text{mon}(A(i))$ we get $(\omega, \nu) \in \llbracket A(i) \rrbracket$ by monitor correctness

$$\models \text{mon}(A(i)) \rightarrow \langle A(i) \rangle \bigwedge_{v \in \text{BV}(A(i))} (v^+ = v) ,$$

which in turn gives $\langle M(i) \rangle [t := 0; \{x' = f(x, u), t' = 1 \ \& \ t \leq \varepsilon\}] I$ by [12] using effective attack mitigation³

$$\models I \rightarrow [A(i)] \langle M(i) \rangle [t := 0; \{x' = f(x, u), t' = 1 \ \& \ t \leq \varepsilon\}] I .$$

As a result, $(u : \in M(i))^d$ preserves $[t := 0; \{x' = f(x, u), t' = 1 \ \& \ t \leq \varepsilon\}] I$. \square

4 Water Treatment Attack Mitigation Case Study

Valve Attack When the tank level x_2 is below H_2 so that in normal operation the pump p_2 is running and valve v_2 open to refill the tank, the attacker directly forces valve v_2 closed in an attempt to increase pipe pressure and stress: $v_2 := 0$ when $p_2 = 1$ and $x_2 < H_2$.

A mitigation for this valve attack forces the valve open with a redundant PLC when the pump is running:

$$A_{\text{tank}} \rightarrow [\text{if } p_2=1 \wedge x_2 < H_2 \text{ then } v_2 := 0][\text{if } p_2=1 \text{ then } v_2 := 1](p_2=1 \rightarrow v_2=1)$$

This ensures the safety condition that the valve must be open when the pump is running to prevent pipe burst. The attack monitor specification

$$\langle \text{if } p_2 = 1 \wedge x_2 < H_2 \text{ then } v_2 := 0 \rangle (v_2^+ = v_2)$$

characterizes the behavior of the above attack program. A proof in dL certifies that the monitor formula $(v_2^+ = 0 \wedge p_2 = 1 \wedge x_2 < H_2)$, does not miss alarms when detecting the undesired combination of closed valve with running pump, i.e., the following formula is valid:

$$(v_2^+ = 0 \wedge p_2 = 1 \wedge x_2 < H_2) \rightarrow \langle \text{if } p_2 = 1 \wedge x_2 < H_2 \text{ then } v_2 := 0 \rangle (v_2^+ = v_2) .$$

Stealthy Sensor Attack A stealthy sensor attack attempts to avoid detection. In this example, we therefore start from a level perturbation detector $|\hat{x}_2 - \tilde{x}_2| > \delta$ that is designed to detect changes in the tank level x_2 that are unexpected with some small perturbation δ relative to the expected twin-predicted level \tilde{x}_2 . The comparison is based on the measured level \hat{x}_2 .

Following Def. 5, the attack signature of attacks that have access to the twin-predicted value \tilde{x}_2 to modify \hat{x}_2 while remaining stealthy to the level perturbation detector is $\hat{x}_2 := *; ?\neg|\hat{x}_2 - \tilde{x}_2| > \delta$. The attack program $\hat{x}_2 := \tilde{x}_2 - \delta$

³ Note, that the attack mitigation must preserve $[t := 0; \{x' = f(x, u), t' = 1 \ \& \ t \leq \varepsilon\}] I$ rather than the plain safety conditions as in the examples above.

a refinement of this attack signature that attempts the maximum undetectable perturbation to make the tank level seem lower than it is (i.e., the attack attempts to make the tank overflow).

The parameter δ is typically chosen according to the accuracy of the level sensors and the fidelity of the twin-predicted level; the safety horizon for $\text{tank}_{A_{x_2}}$

$$\begin{aligned} \text{tank}_{A_{x_2}} &\equiv i : \in \text{read}(x); (u : \in \text{ctrl}(i) \cup u : \in A_{x_2}(i)); \text{plant}_{\text{tank}} \\ u : \in A_{x_2}(i) &\equiv \hat{x}_2 := \tilde{x}_2 - \delta \\ i : \in \text{read}(x) &\equiv \hat{x}_1 := x_1; \hat{x}_2 := x_2; f_1 := *; f_2 := *; ?0 \leq f_1 \leq F_1 \wedge 0 \leq f_2 \leq F_2; \\ &\quad \underbrace{\tilde{x}_2 := *; ?|\hat{x}_2 - \tilde{x}_2| \leq \delta}_{\text{Twin-predicted level}} \end{aligned}$$

gives an estimate for the robustness of the system state in terms of the mitigation horizon (here, time to overflow) τ :

$$\begin{aligned} A_{\text{tank}} \rightarrow \exists \tau (& [c := 0; \text{timed}_c(\text{tank}_{A_{x_2}})^*; ?c \leq \tau](x_2 \leq H_H) \\ & \wedge \forall \tilde{\tau} > \tau \langle c := 0; \text{timed}_c(\text{tank}_{A_{x_2}})^*; ?c \leq \tilde{\tau} \rangle \neg (x_2 \leq H_H) \end{aligned}$$

A generic mitigation strategy against stealthy attacks can combine a witness for the time to overflow τ with some threshold $T > 0$ to engage emergency draining of the tank: if $\tau < T$ then $v_3 := 1; p_3 := 1$.

Sensor Spike Attack Emergency draining a tank as a generic mitigation strategy makes the system vulnerable to attacks that change the sensor readings in a way that triggers the mitigation, e.g., an attack $\hat{x}_2 := *; ?\hat{x}_2 > H_H$ causes time to overflow become negative, which triggers emergency draining.

$$\begin{aligned} \text{tank}_{A_{x_2\uparrow}} &\equiv i : \in \text{read}(x); (u : \in \text{ctrl}(i) \cup u : \in A_{x_2\uparrow}(i)); \text{plant} \\ u : \in A_{x_2\uparrow}(i) &\equiv x_0 := \hat{x}_2; \hat{x}_2 := *; ?\hat{x}_2 > H_H \wedge \frac{|\hat{x}_2 - x_0|}{\varepsilon} \leq F_L; \underbrace{\left(\frac{v_3}{p_3}\right) := \left(\frac{1}{1}\right)}_{\text{emergency draining}} \end{aligned}$$

Since sudden spikes are easily detectable from limits on the flow F_L , the attack must be performed when the tank level is close to the tank capacity. In this case, mitigation analysis with a witness for τ informs us for how long emergency draining should be active at most to not violate the lower tank level:

$$\begin{aligned} A_{\text{tank}} \rightarrow \exists \tau (& [c := 0; \text{timed}_c(\text{tank}_{A_{x_2\uparrow}})^*; ?c \leq \tau]P \\ & \wedge \forall \tilde{\tau} > \tau \langle c := 0; \text{timed}_c(\text{tank}_{A_{x_2\uparrow}})^*; ?c \leq \tilde{\tau} \rangle \neg P \\ & \text{with } P \equiv \langle \left(\frac{v_3}{p_3}\right) := \left(\frac{0}{0}\right) \rangle [\text{tank}^*](L_2 \leq x_2) \end{aligned}$$

This formula characterizes the time τ to underflow, until when the emergency draining $\left(\frac{v_3}{p_3}\right) := \left(\frac{0}{0}\right)$ must be stopped and the attack must be removed from the system so that under normal (unattacked) operation conditions the tank does not drain below minimum level L_2 .

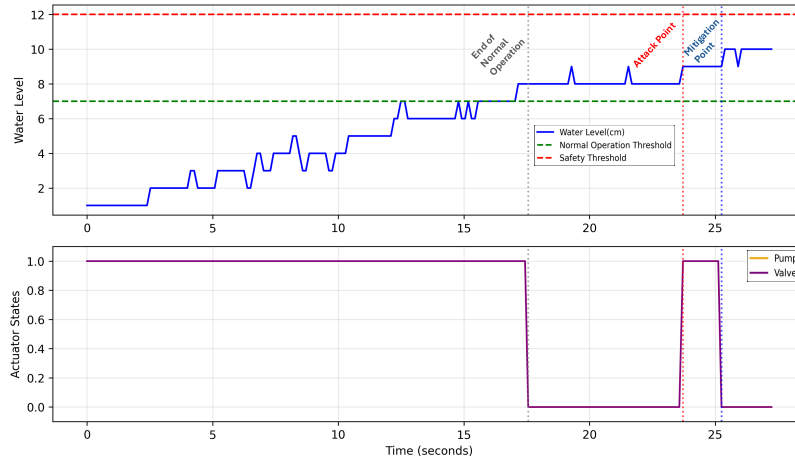


Fig. 2. Attack scenario with rapid mitigation response showing successful prevention of safety violations

5 Evaluation

The experimental evaluation⁴ was conducted in the miniSWaT testbed [15], where we implemented and analyzed a specific attack scenario targeting the system’s actuators. The attack vector focused on compromising two components, pump p_2 and valve v_2 , forcing them to remain in an open state beyond the normal operational threshold of water level. This sequence was executed twice under different mitigation response conditions. In Scenario 1, as shown in Figure 2, our detection system successfully identified the attack signature, enabling rapid mitigation deployment. The swift response prevented the water level from exceeding the safety threshold, demonstrating the effectiveness of timely intervention. In contrast, Scenario 2, as shown in Figure 3, revealed potential vulnerabilities when mitigation measures are delayed. The extended response time allowed the water level to breach the safety threshold, violating the system’s safety properties. These results validate the importance of the mitigation horizon in guiding timely responses and maintaining system safety under adversarial conditions.

6 Conclusion and Future Work

This paper presented HYTWIn, a formal framework leveraging digital twins to enable proactive security interventions in industrial control systems. By integrating temporal semantics and formalized attack models, HYTWIn provides a rigorous methodology for dynamically mitigating adversarial actions while preserving system safety properties. Our evaluation on real water treatment testbeds

⁴ The code and traces are available here: <https://github.com/iotrustlab/HyTwin>.

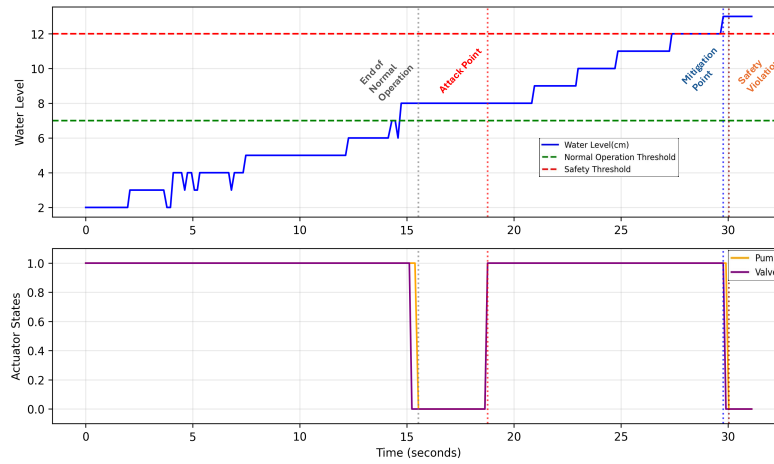


Fig. 3. Attack scenario with delayed mitigation response demonstrating safety threshold violation

demonstrated the practical efficacy of the mitigation horizon in guiding timely interventions and maintaining resilience under representative attack scenarios.

Future work will explore how the fidelity of digital twins impacts their effectiveness in detecting and mitigating adversarial behavior. Specifically, we aim to investigate the trade-offs between model complexity and computational efficiency, addressing how runtime fidelity affects the alignment between the digital and real twins. Additionally, extending the formal semantics of digital twins to include runtime properties—such as the emulation fidelity of PLC scan cycles—will further enhance their capability to adaptively respond to evolving threats. Finally, while this work formalized mitigation strategies, future research will focus on synthesizing resilient controllers to ensure continuous operation under attack, particularly for critical infrastructure that cannot afford downtime.

Acknowledgment

We would like to thank the anonymous reviewers for their constructive comments. This material is based upon work supported by the National Science Foundation under Grants No. CCF2427581 and CCF2425711. Any opinions, findings, and conclusions made in this material are those of the authors and do not necessarily reflect the views of the funding agency.

References

1. Adepuz, S., Brassler, F., Garcia, L., Rodler, M., Davi, L., Sadeghi, A.R., Zonouz, S.: Control behavior integrity for distributed cyber-physical systems. In: 2020

- ACM/IEEE 11th International Conference on Cyber-Physical Systems (ICCPS). pp. 30–40 (2020). <https://doi.org/10.1109/ICCPS48487.2020.00011>
2. Alam, K.M., El Saddik, A.: C2PS: A Digital Twin Architecture Reference Model for the Cloud-Based Cyber-Physical Systems. *IEEE Access* **5**, 2050–2062 (2017). <https://doi.org/10.1109/ACCESS.2017.2657006>
 3. de Azambuja, A.J.G., Giese, T., Schützer, K., Anderl, R., Schleich, B., Almeida, V.R.: Digital twins in industry 4.0—opportunities and challenges related to cyber security. *Procedia CIRP* **121**, 25–30 (2024)
 4. Cao, Y., Currie, C., Onggo, B.S., Higgins, M.: Simulation Optimization for a Digital Twin Using a Multi-Fidelity Framework. In: 2021 Winter Simulation Conference (WSC). pp. 1–12. IEEE, Phoenix, AZ, USA (Dec 2021). <https://doi.org/10.1109/WSC52266.2021.9715498>
 5. Chong, S., Lanotte, R., Merro, M., Tini, S., Xiang, J.: Quantitative Robustness Analysis of Sensor Attacks on Cyber-Physical Systems. In: Proceedings of the 26th ACM International Conference on Hybrid Systems: Computation and Control. pp. 1–12. ACM, San Antonio TX USA (May 2023). <https://doi.org/10.1145/3575870.3587118>
 6. Ferko, E., Bucaioni, A., Pelliccione, P., Behnam, M.: Standardisation in digital twin architectures in manufacturing. In: 2023 IEEE 20th International Conference on Software Architecture (ICSA). pp. 70–81. IEEE (2023)
 7. Franchetti, F., Low, T.M., Mitsch, S., Mendoza, J.P., Gui, L., Phaosawasdi, A., Padua, D., Kar, S., Moura, J.M., Franusich, M., Johnson, J., Platzer, A., Veloso, M.M.: High-assurance spiral: End-to-end guarantees for robot and car control. *IEEE Control Systems Magazine* **37**(2), 82–103 (2017). <https://doi.org/10.1109/MCS.2016.2643244>
 8. Garcia, L., Mitsch, S., Platzer, A.: Hyplc: Hybrid programmable logic controller program translation for verification. In: Proceedings of the 10th acm/ieee international conference on cyber-physical systems. pp. 47–56 (2019)
 9. Gehrman, C., Gunnarsson, M.: A digital twin based industrial automation and control system security architecture. *IEEE Transactions on Industrial Informatics* **16**(1), 669–680 (2019)
 10. Mathur, A.P.: Reconfigurable Digital Twin to Support Research, Education, and Training in the Defense of Critical Infrastructure. *IEEE Security & Privacy* **21**(4), 51–60 (Jul 2023). <https://doi.org/10.1109/MSEC.2023.3281272>
 11. Mathur, A.P., Tippenhauer, N.O.: 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). pp. 31–36. IEEE (2016)
 12. Mitsch, S., Platzer, A.: Modelplex: verified runtime validation of verified cyber-physical system models. *Formal Methods Syst. Des.* **49**(1-2), 33–74 (2016). <https://doi.org/10.1007/S10703-016-0241-Z>
 13. Mitsch, S., Platzer, A.: Verified runtime validation for partially observable hybrid systems. *CoRR abs/1811.06502* (2018)
 14. Müller, A., Mitsch, S., Retschitzegger, W., Schwinger, W., Platzer, A.: Tactical contract composition for hybrid system component verification. *International Journal on Software Tools for Technology Transfer* **20**(6), 615–643 (Nov 2018). <https://doi.org/10.1007/s10009-018-0502-9>
 15. Paul, J., Ponce, L., Zhang, M., Garcia, L.: Towards cross-physical-domain threat inference for industrial control system defense adaptation. In: Proceedings of the 2024 Workshop on Re-design Industrial Control Systems with Security. pp. 57–64 (2023)

16. Platzer, A.: Differential game logic. *ACM Trans. Comput. Log.* **17**(1), 1 (2015). <https://doi.org/10.1145/2817824>
17. Platzer, A.: A complete uniform substitution calculus for differential dynamic logic. *J. Autom. Reason.* **59**(2), 219–265 (2017). <https://doi.org/10.1007/S10817-016-9385-1>
18. Qian, M., Mitsch, S.: Reward shaping from hybrid systems models in reinforcement learning. In: *NASA Formal Methods Symposium*. pp. 122–139. Springer (2023)
19. Thalpage, N.S., Nisansala, T.A.D.: Exploring the opportunities of applying digital twins for intrusion detection in industrial control systems of production and manufacturing—a systematic review. *Data Protection in a Post-Pandemic Society* pp. 113–143 (2023)