CPSim: Simulation Toolbox for Security Problems in Cyber-Physical Systems

MENGYU LIU, University of Notre Dame, Notre Dame, United States
LIN ZHANG, University of Pennsylvania, Philadelphia, United States
WEIZHE XU, University of Notre Dame, Notre Dame, United States
SHIXIONG JIANG, University of Notre Dame, Notre Dame, United States
FANXIN KONG, University of Notre Dame, Notre Dame, United States

There are various applications of Cyber-Physical systems (CPSs) that are life-critical where failure or malfunction can result in significant harm to human life, the environment, or substantial economic loss. Therefore, it is important to ensure their reliability, security, and robustness to the attacks. However, there is no widely used toolbox to simulate CPS and target security problems, especially the simulation of sensor attacks and defense strategies against them. In this work, we introduce our toolbox CPSim, a user-friendly simulation toolbox for security problems in CPS. CPSim aims to simulate common sensor attacks and countermeasures to these sensor attacks. We have implemented bias attacks, delay attacks, and replay attacks. Additionally, we have implemented various recovery-based methods against sensor attacks. The sensor attacks and recovery methods configurations can be customized with the given APIs. CPSim has built-in numerical simulators and various implemented benchmarks. Moreover, CPSim is compatible with other external simulators and can be deployed on a real testbed for control purposes.[1]

CCS Concepts: • **Computer systems organization → Embedded and cyber-physical systems**; • **Security and privacy → Systems security;**

Additional Key Words and Phrases: cyber-physical system, toolbox, simulation

ACM Reference Format:
Mengyu Liu, Lin Zhang, Weizhe Xu, Shixiong Jiang, and Fanxin Kong. 2024. CPSim: Simulation Toolbox for Security Problems in Cyber-Physical Systems. *ACM Trans. Des. Autom. Electron. Syst.* 29, 5, Article 90 (September 2024), 16 pages. https://doi.org/10.1145/3674904

1 Introduction

Cyber-physical systems (CPSs) seamlessly integrate computational resources, physical elements, sensing, and actuation, enhancing the functionality of critical components and services

[1]This toolbox has been presented and published at the 29th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS'23) Demo session.

This work was supported in part by NSF CNS-2333980.
Authors' Contact Information: Mengyu Liu, University of Notre Dame, Notre Dame, Indiana, United States; e-mail: mliu9@nd.edu; Lin Zhang, University of Pennsylvania, Philadelphia, Pennsylvania, United States; e-mail: cpsec@seas.upenn.edu; Weizhe Xu, University of Notre Dame, Notre Dame, Indiana, United States; e-mail: wxu3@nd.edu; Shixiong Jiang, University of Notre Dame, Notre Dame, Indiana, United States; e-mail: sjiang5@nd.edu; Fanxin Kong, University of Notre Dame, Notre Dame, Indiana, United States; e-mail: fkong@nd.edu.

ACM 1084-4309/2024/09-ART90
https://doi.org/10.1145/3674904

across domains such as healthcare, aviation, transportation, and manufacturing [19, 23]. CPSs bridge the gap between physical processes and computational systems, allowing for more efficient, automated, and intelligent interaction with the physical environment. CPSs play a critical role in emergency management and disaster response, providing essential data for informed decision-making and automated response mechanisms, thereby enhancing resilience and mitigating the impact of crisis [1, 12, 17]. However, malicious attacks on CPSs can have severe consequences, including economic harm, compromised personal safety, and societal disruptions. For instance, an attack on industrial control systems could lead to machinery malfunction, causing property damage or even catastrophic industrial accidents. Upon its discovery in 2010, the Stuxnet cyber worm, which specifically targeted programmable logical controllers used in industrial settings, had already spread to an estimated 60,000 computers globally. This sophisticated malware represented a significant escalation in cyber threats, demonstrating an unprecedented level of complexity and an ability to disrupt critical industrial processes [8]. Recently, there have been multiple cases reported by media about the vulnerability of CPSs [20, 21, 28]. These cases also motivate the development of new safety and security solutions.

To tackle the above security problem, researchers have developed several algorithms to enhance the safety of the CPSs [15, 16, 33, 34], but there is no widely used simulation toolbox targeting security problems in CPSs [26, 27, 30]. CPSs often operate in complex, real-world environments where physical testing can be costly, dangerous, or impractical. Simulation allows for extensive testing and validation of the system's performance under a wide range of conditions without the risks associated with physical trials. Through simulation, designers can experiment with different configurations and scenarios to optimize system performance. This is particularly important in CPS, where the interplay between physical and computational components can lead to unexpected behaviors. Furthermore, developing and deploying physical prototypes for complex systems can be extremely expensive. Simulation provides a more cost-effective way to explore design alternatives, identify problems, and refine system behaviors before investing in physical prototypes.

Compared to existing simulators, most CPS-related simulators [11, 24] focus on enhancing fidelity and information accuracy, often neglecting issues related to safety. Among the few that do address safety [26], they typically concentrate only on the impacts of attacks without considering aspects of recovery.

However, it is difficult to develop a CPS simulation toolbox for security problems due to two main challenges:

— The implementation complexity of CPS system security components. Attack detectors aim to identify attacks early on, while attack recovery methods mitigate their impact and guide the system's physical states toward predetermined targets. Yet, assessing the effectiveness and efficiency of these security measures presents challenges. Gathering benchmark plants, designing controllers, customizing attacks, constructing defense approaches, and evaluating performance requires substantial efforts.
— The coupling of each component in CPS. Integrating new features or incorporating solutions into existing simulators is also difficult using existing methods. Additionally, the simulation of physical system behaviors in tandem with cyber states is crucial [13]. This dual-layered approach is imperative for understanding the intricate interactions between the physical components and their digital counterparts within CPSs. By accurately modeling both the physical and cyber aspects, users can gain a comprehensive insight into the system's overall behavior, enabling them to anticipate potential vulnerabilities, optimize performance, and devise effective countermeasures against cyber threats. Such simulations are particularly vital in complex environments where the physical and digital realms are deeply intertwined and where the consequences of system failures or cyber attacks can be far-reaching[14].

To address the above challenges, we propose CPSim, a simulation and security toolbox with high extensibility and adaptability. CPSim aims to simulate common sensor attacks and countermeasures to these sensor attacks. The design of CPSim has two main features:

— Users can easily toggle between different experiment parameters and deploy defense prototypes in response to various attacks.
— CPSim is compatible with existing simulators for CPSs under different scenarios.

The source code of CPSim can be found at https://github.com/lion-zhang/CPSim. The CPSim documentation can be found at https://sim.cpsec.org/en/latest/.

2 Related Works

Compared to other existing simulators, most CPS-related simulators [11, 18, 24] focus on enhancing fidelity and information accuracy, often neglecting issues related to safety. Researchers in Reference [24] proposed a novel high-fidelity simulation framework that provides both **Software-in-the-Loop (SITL)** and **Hardware-in-the-Loop (HITL)** testing. Researchers in Reference [11] developed the 3D simulator Gazebo, which is the foundation for many simulators. However, Gazebo does not specifically target security problems in CPS. Ma et al. [18] have developed a wireless CPS simulator for edge computing that enhances the high-fidelity simulation.

Some of the CPS simulators focus on simulating specific systems such as power systems [3–5, 29]. In Reference [5], the authors introduced an event-driven co-simulation scheme employing the network simulator NS2 and OpenDSS for simulating Cyber-Physical Power Systems. In Reference [3], the researchers developed a co-simulation framework by integrating OpenDSS and OMNeT++ to perform power system simulations and communication network analyses, aiming to evaluate wide-area monitoring and control applications. In Reference [4], a co-simulation framework has been developed to simulate power routing algorithms for microgrid applications by merging OMNeT++ with the **Real-Time Digital Simulator (RTDS)**. While these works offer significant insights and contributions, they overlook the simulation of attacks and security issues within these systems. This omission limits their applicability in real-world scenarios, where security vulnerabilities and malicious attacks are common. Consequently, further research is needed to incorporate comprehensive simulation models that evaluate and mitigate potential security threats.

Among the few that do address security [2, 6, 26], they typically concentrate on the impacts of attacks without considering defending strategies. In Reference [26], the researchers propose a simulation MATLAB toolbox, epanetCPA. The toolbox provides both demand-driven and pressure-driven simulations, allowing users to realistically analyze cyber-physical attacks and their impacts under various conditions. In Reference [2], researchers propose MiniCPS, which enhances Mininet to offer lightweight real-time network emulation and extends it by incorporating tools to simulate typical CPS components, such as programmable logic controllers. However, they only provide a single method to mitigate the attacks by analyzing the traffic of the network systems. In Reference [6], researchers simulate and observe the effects of a common network attack on the availability of the system. Unfortunately, no defense strategy is included and discussed.

Some researchers have targeted sensor attacks and defense strategies against them [7, 12, 31, 33, 34]. In Reference [12], researchers systematically introduce how to apply recovery-based methods to against sensor attacks. In Reference [7], researchers propose a novel framework to create a software sensor that estimates the system states under sensor attacks. In References [31, 33, 34], the researchers propose a series of recovery algorithms against sensor attacks under different scenarios. We have implemented all of them in CPSim.

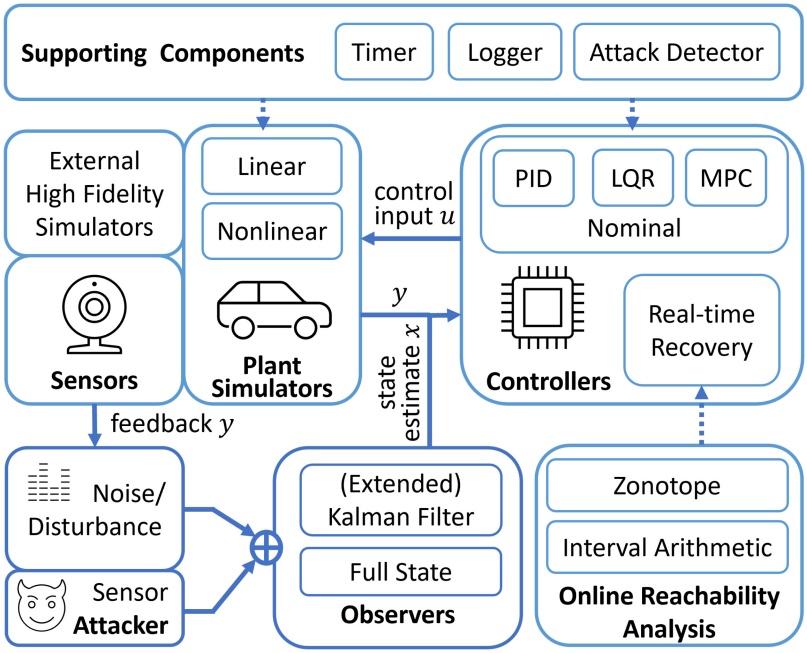


Fig. 1. Design overview of simulation and security toolbox.

3 Architecture

The proposed toolkit consists of three components: a simulator engine, security analysis elements, and supporting components. Illustrated in Figure 1, the simulator replicates the operations of a CPS where sensors capture the system's conditions and transmit measurements to observers. Alternatively, the simulator can be replaced by external simulators for high-fidelity simulation. In practice, sensor measurements can be affected by external uncertainties and intentional attacks, depending on the settings and requirements. The observers utilize these measurements to estimate the current state of the system for the controllers. The controllers, in turn, generate control inputs that are implemented in physical plants. The plant simulators then update the system states based on the system dynamics and applied control inputs.

To enhance the security of the CPS, various attack detectors and real-time attack recovery controllers are integrated to identify and respond to potential attacks. Additionally, there are some reachability analysis tools implemented to help estimate the safety status and deadlines [32]. Furthermore, these reachability analysis tools serve a dual purpose by aiding in the enforcement of operational deadlines. They provide a temporal analysis to ensure that the system's responses occur within acceptable time frames, which is critical for time-sensitive operations where delays could lead to unsafe conditions or system failures. The integration of these advanced security and analytical capabilities is imperative for maintaining the integrity, reliability, and safety of the CPS. By offering both detection and response mechanisms, as well as predictive capabilities, the system aims to shield itself from the dire consequences of cyberattacks while ensuring that it operates within safe parameters and adheres to strict temporal constraints.

The supporting components are responsible for the data streaming and timing-related scheduling. The timer device within the simulation framework plays a pivotal role, as it emulates the system clock, a central piece in the coordination and synchronization of the CPS's operations. It meticulously triggers control steps at predefined intervals, ensuring that the sequence

```python
import numpy as np
# import built-in CSTR system
from cpsim.models.nonlinear.continuous_stirred_tank_reactor import CSTR

# simulation settings
max_index = 1000  # simulation steps
dt = 0.02  # sampling time
ref = [np.array([0, 300])] * (max_index + 1)  # reference/target state
noise = None   # noise settings
cstr_model = CSTR('test', dt, max_index, noise)   # simulation object

# simulation control loop
for i in range(0, max_index + 1):
    assert cstr_model.cur_index == i
    cstr_model.update_current_ref(ref[i])   # update reference/target state
    cstr_model.evolve()      # evolve the system

# print results
import matplotlib.pyplot as plt
# how to get the simulation results
t_arr = np.linspace(0, 10, max_index + 1)
ref = [x[1] for x in cstr_model.refs[:max_index + 1]]
y_arr = [x[1] for x in cstr_model.outputs[:max_index + 1]]
u_arr = [x[0] for x in cstr_model.inputs[:max_index + 1]]
# plot the state x[1]
plt.title('CSTR x[1]')
plt.plot(t_arr, y_arr, t_arr, ref)
plt.show()
```

Fig. 2. An example code of a simulation using CPSim.

of actions within the CPS adheres to the designated schedule. This precise timing is critical for maintaining the fidelity of the simulation, mirroring real-world operations where timing is often as crucial as the control actions themselves. The logger records the system's historical data from other components. It captures a comprehensive log of key operational data points, including state estimates, sensor measurements, and control inputs. By chronicling this data, the logger provides a valuable archive that can be used for retrospective analysis, troubleshooting, and system optimization. It plays a forensic role in the event of system anomalies or failures, allowing engineers and system analysts to trace back through the event sequence and identify the root causes.

Figure 2 displays a Python code script example of the simulation of a **Continuous Stirred Tank Reactor (CSTR)** system. In each simulation of CPSim, there are three main steps to produce the simulation results: initialize the simulation, build the main control loop, and output the simulation results.

Lines 1–3 are responsible for importing necessary Python modules and classes. A CSTR class is imported from a cyber-physical system modeling package for simulating a continuously stirred tank reactor. Lines 5–9 set up the parameters for the simulation. The number of simulation steps, the sampling time, the reference or target state, and the noise settings are defined here. Notably, noise is set to None, indicating that noise is not currently factored into the simulation. Lines 12–16 represent the main loop where the simulation is executed. For each iteration, it asserts the current simulation index, updates the reference state, and evolves the system to the next state. Lines 18–27 are responsible for handling the results. Matplotlib is imported for plotting, and arrays for time, the reference state, system output, and control input are created. The block concludes with the setup for a plot that would display the system's response overtime against the reference state.

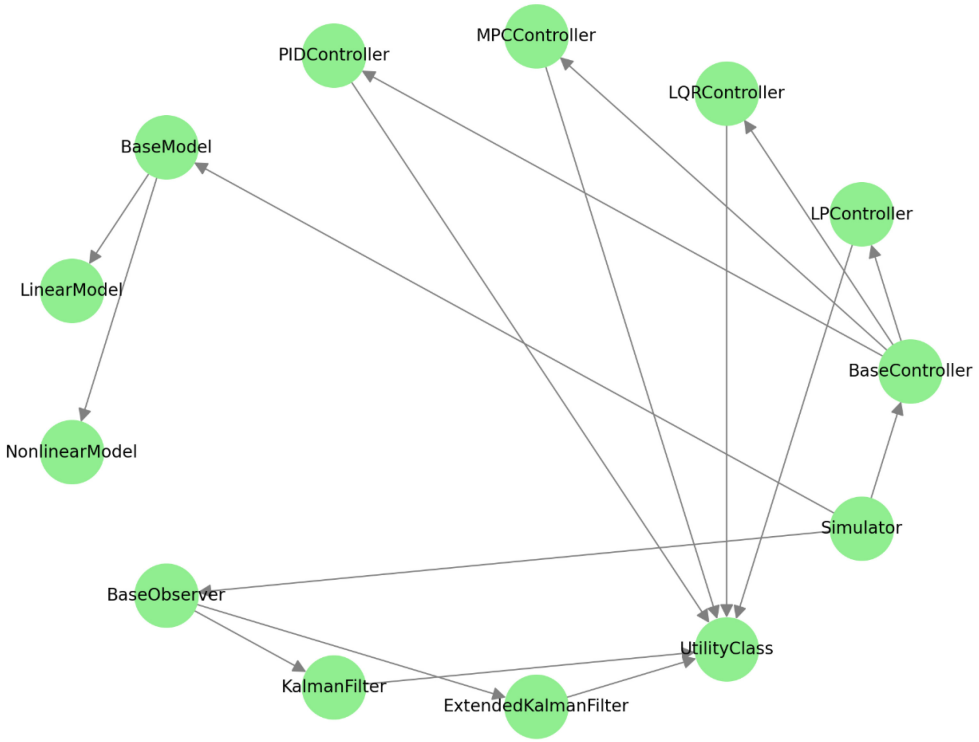


Fig. 3. The OOP layout diagram of CPSim.

4 Features

(i) User-centric Interface: CPSim distinguishes itself with its user-centric design, setting it apart from many toolboxes that rely on C or C++—languages that can pose a challenge for less experienced researchers. The choice of Python 3 for its development underscores its commitment to simplicity and readability, key factors that enhance its accessibility for a broader range of researchers. Python's widespread popularity further augments this aspect.

The design of CPSim using **Object-Oriented Programming (OOP)** principles represents a significant stride in developing robust, scalable, and maintainable systems. OOP, characterized by its use of classes and objects, offers a modular approach that aligns perfectly with the multifaceted nature of CPS. In these systems, various physical processes and computational elements interact closely, and OOP allows for each of these components to be encapsulated within individual objects. This encapsulation not only promotes a clear separation of concerns but also enhances the reusability of code. Figure 3 is a layout of CPSim design from a lower level. This circular layout provides a clearer and more consolidated view of the classes and their relationships, including inheritance and associations. The arrangement should help in visualizing the structure and connections within CPSim.

By adopting OOP, developers can create models for each physical process and computational function as distinct objects, each with its own properties and methods. This method greatly simplifies complex system designs, making them more manageable and comprehensible. It also facilitates the simulation and testing of individual components in isolation, as well as in integrated settings, enhancing the reliability and effectiveness of the entire CPS.

Furthermore, OOP's inheritance feature enables developers to create hierarchical relationships between objects, which is invaluable in representing real-world systems where certain components naturally share attributes and behaviors. This hierarchical modeling can significantly streamline the development process, reduce redundancy, and allow for more intuitive system expansions or updates.

(ii) Customizability and Modularity: One of CPSim's standout features is its high extensibility. The simulator engine is designed to function independently from the toolbox's other components, a design choice that greatly simplifies the process of adding new elements. This modular approach allows users to easily integrate additional modules or components into the toolbox, tailoring it to meet their specific simulation and analysis needs. Such customizability is invaluable in a research setting, where specific requirements can vary greatly from one project to another. This feature empowers users to adapt the toolbox to their particular research objectives, enhancing its utility and effectiveness.

One of the key areas where CPSim allows for user-driven extension is in the creation of customized benchmarks. Benchmarks are essential for evaluating the performance and resilience of CPS under different conditions. Users can design these benchmarks to reflect specific real-world scenarios or theoretical models, enabling them to conduct more relevant and targeted research. This capability is particularly valuable for testing systems under a variety of controlled conditions to understand their behavior and limitations. The user can define the system dynamics in state-space format for linear models and in ODE format for non-linear models. The new benchmark's dynamics and controller must be specified and integrated into the simulation class.

Additionally, CPSim's architecture supports the integration of custom controllers. Controllers are at the heart of CPS, dictating the system's response to input data and environmental conditions. By allowing users to implement their own controllers, CPSim provides a versatile platform for experimenting with various control strategies. This flexibility is vital for exploring innovative control algorithms, including those based on emerging technologies such as artificial intelligence and machine learning. Users can define their own controller class by inheriting the controller class.

Furthermore, the ability to simulate different types of attacks is an indispensable feature of CPSim, especially considering the growing importance of cybersecurity in CPS. Users can create and integrate custom attack scenarios to evaluate the robustness and security of systems. This feature is critical for identifying vulnerabilities, developing mitigation strategies, and ensuring the resilience of CPS against malicious cyber threats. The details can be found at the documentation site under custom components.

(iii) Versatility and Integration Capabilities: CPSim's flexibility extends beyond mere numerical simulations. The toolbox is designed for easy deployment in well-regarded high-fidelity simulators such as AirSim and CARLA, making it a versatile choice for a wide array of simulation environments. Additionally, its compatibility with the **Robot Operating System (ROS)**—a collection of open-source software libraries and tools for robot development—broadens its applicability. This integration demonstrates CPSim's adaptability in diverse simulation contexts, ranging from autonomous vehicle research to robotic system development.

Additionally, users can replace the simulator engine with a real testbed. CPSim's capability to be deployed on testbeds adds another layer of versatility and practicality to its already robust set of features. Testbeds are essential in the field of CPS, as they provide a controlled yet realistic environment for validating the performance, reliability, and security of these systems under various conditions.

Deploying CPSim on a testbed allows for real-time, hands-on testing and experimentation with actual hardware and software configurations. This is crucial for bridging the gap between

theoretical simulations and real-world applications. In a testbed setting, researchers and developers can observe how CPS behaves in scenarios that closely mimic their intended operational environments. This includes testing under various physical conditions, network configurations, and different types of hardware.

The use of a testbed also enables comprehensive stress testing of CPS under extreme or unexpected conditions, which might be difficult to replicate in a purely simulated environment. This includes testing the resilience of systems against physical disruptions, cyber attacks, and network failures. Such real-world testing is vital for identifying and rectifying potential issues that might not be evident in a simulation-only environment.

5 Attacks and Noises

It is important to simulate the noises in the system and potential attacks injected by malicious attackers for security problems in CPS. Simulating attacks helps identify potential vulnerabilities in CPS. These systems often integrate physical devices with digital networks, making them susceptible to various attack vectors. By simulating attacks, weaknesses can be discovered and addressed before they can be exploited by malicious actors. Moreover, simulating attacks enables organizations to develop effective response strategies. This includes not only technical responses but also procedural and human responses, ensuring that all aspects of the system can react appropriately to mitigate damage from a real attack. Many industries have regulatory requirements for cybersecurity. Simulating attacks can help ensure that CPS meets these standards, reducing legal and financial risks associated with non-compliance.

Simulating noises helps in testing the system's reliability and its ability to operate correctly in the presence of such interferences, which are common in real-world environments. Furthermore, by introducing noise into the system during testing, engineers can evaluate the error and fault tolerance capabilities of CPS. This helps in designing systems that can continue to operate correctly or fail safely in the presence of unexpected disturbances. In the CPSim toolbox, we simulate attacks that compromise the integrity or availability of sensor measurements.

5.1 Attack Types

We have implemented three common attacks in this toolbox: bias attack, delay attack, and replay attack. A bias attack usually refers to the intentional introduction of systematic errors or biases into the system. This type of attack is particularly relevant in systems that rely on sensor data or algorithmic decision-making. For example, in a navigation system using GPS data, an attacker might introduce a small, consistent error in the GPS readings, causing the system to slowly drift off course. A delay attack targets the timeliness of information in CPS. The attacker deliberately delays the transmission of critical data or commands, disrupting the normal operation of the system. In an industrial control system, delaying the communication between sensors and controllers could result in machinery operating on outdated information, potentially causing safety hazards or production issues. A replay attack involves capturing valid data transmissions and retransmitting them later. The key aspect of this attack is that the data itself is legitimate but is reused or replayed at an inappropriate time or in an incorrect context. The details of these three attacks can be found in Reference [31].

In all these attacks, the integrity and reliability of the CPS are compromised. The subtlety of bias and delay attacks makes them particularly insidious, as they may not be immediately detectable. Replay attacks exploit the lack of temporal validation in communication protocols. These attacks highlight the importance of robust security measures, including encryption, authentication, and real-time monitoring, to protect against various forms of cyber threats.

We have implemented Gaussian noises in the toolbox and will implement more noises in the future. Since the noises can vary depending on the scenario, we make the noise distribution as an argument when initializing the simulation class.

It should be noted that our system is not specifically designed to counter cyber attacks; instead, it focuses on sensor attacks. More specifically, these attacks involve adversaries manipulating the values entered into the system by sensors, either through physical means or via the network. We do not have a network layer in our toolbox; instead, we try to simulate attacks from the software level.

6 Recovery Controllers

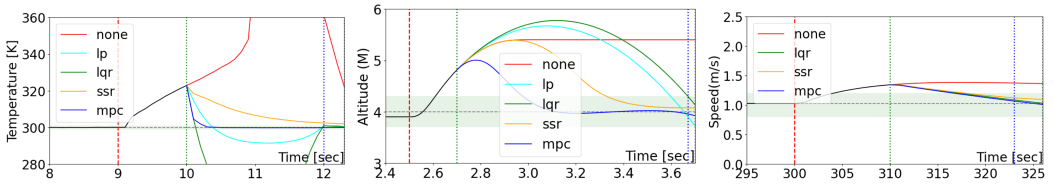
Recovery in CPS is vital due to the deeply interconnected nature of these systems, which blend computational elements with physical processes. In environments where CPSs are integral—ranging from automotive systems and medical devices to industrial controls—recovery mechanisms are essential to ensure safety, preventing malfunctions or anomalies from leading to unsafe conditions. Moreover, the reliability of these systems is paramount; they must operate without fail even amidst hardware failures, software glitches, or unexpected environmental changes. This reliability is further challenged by the growing threat of cyber attacks, making robust recovery processes a shield against potential security breaches that could compromise the integrity and confidentiality of operations. It is important to react to the attacks to maintain the safety of the CPSs. Therefore, we have implemented several recovery controllers to recover the system from a risky status to a safe state.

Software-sensor-based recovery refers to the process of restoring the normal operation of a system using software sensors after a disruption, such as a cyber attack or a fault [7, 12]. After the detection of sensor attacks, the baseline replaces the corrupted physical sensor data with the software sensor data predicted by the linear system model.

Linear programming (LP)-based recovery is a mathematical approach used to restore the optimal operation of a system following a disruption, such as equipment failure, resource depletion, or a cyber attack[31]. This baseline approach constructs the issue of system recovery as an optimization problem that can be solved using linear programming. The objective of solving this optimization problem is to determine a sequence of control actions that will guide the system's state back to a predefined acceptable range, ensuring that this is achieved within a specified time frame that is considered safe.

Linear-Quadratic Regulator (LQR)-based recovery is a control strategy used in systems engineering to bring a system that has experienced disturbances or deviations back to its desired state or trajectory [34]. This approach is grounded in control theory and uses state feedback to ensure the system's performance optimally meets a set of criteria.

MPC-based recovery controller using MPC methods to bring the system back to a safe state after the attacks have been detected [33]. This approach uses data-driven methods to identify a local linear model for the MPC problem and generate a control sequence for the recovery. The MPC's ability to adhere to various constraints, such as operational limits and safety requirements, is also a critical feature, especially during recovery phases. The entire system operates within a closed-loop feedback mechanism, enabling continuous monitoring and real-time adjustments, ensuring that control actions are consistently effective and relevant. This data-driven approach to developing an MPC-based recovery controller provides a sophisticated solution for managing systems with complex dynamics. It offers enhanced predictive accuracy, flexibility in handling nonlinear behaviors, and adaptability to changing operational conditions, making it an invaluable tool in maintaining system stability and efficiency, especially in environments that demand rapid responses to disturbances.



(a) CSTR (b) Quad (c) vessel

Fig. 4. Recovery examples of three non-linear benchmarks in CPSim. The systems are under sensor attacks. The red vertical line indicates the start time of the sensor attack sequence, the green vertical line represents when the detector triggers an alarm and the system initiates recovery, and the blue vertical line signifies the deadline for recovery. The green regions are safe sets. "none" refers to no recovery, "lp" refers to linear programming recovery, "lqr" refers to linear quadratic regulator recovery, "ssr" refers to software sensor recovery, and "MPC" refers to model predictive control recovery.

7 Experiments

In this section, we will show the results of the algorithms described in Section 6 on various benchmarks in the toolbox. Additionally, we will show some results that integrate this toolbox with high-fidelity simulators and a real four-wheel testbed. We have implemented several common linear benchmarks in CPSim. In the real world, most systems are non-linear, therefore, to demonstrate the capabilities of **CPSim (Cyber-Physical System Simulator)** for non-linear systems, it is essential to provide example codes that users can refer to as a starting point. These example codes cover a range of non-linear system characteristics to showcase the versatility and robustness of CPSim in handling complex dynamics.

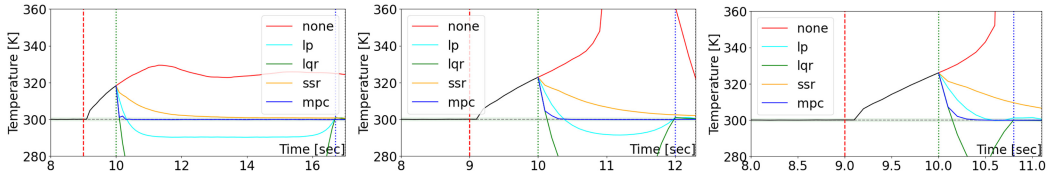
The **Continuous Stirred Tank Reactor (CSTR)** [10] is a common benchmark problem in the field of process control and chemical engineering. It serves as an important example for testing and demonstrating the capabilities of control systems, including those used in CPS. The CSTR benchmark typically involves controlling the concentration and temperature within the reactor. Maintaining the desired temperature is crucial for the efficiency and safety of the reaction. The heat generated or absorbed by the reaction, along with the heat exchange with the environment, contributes to the system's non-linearity.

The quadrotor benchmark [22] is a significant and challenging problem in the field of robotics and control systems, particularly relevant to computer science and real-time systems. Quadrotors, also known as quadcopters, are a type of **unmanned aerial vehicle (UAV)** with four rotors. They are widely used for various applications, including aerial photography, surveillance, and research. The dynamics of quadrotors are complex and highly non-linear, making them an ideal benchmark for testing control algorithms and real-time system performance. In this benchmark, the task is to control the altitude of the quadrotor. Altitude control is directly related to the thrust generated by the quadrotor's rotors. Adjusting the rotor speeds precisely to achieve the desired altitude is challenging and important to real-world applications.

The vessel benchmark [9, 25] is designed to simulate the dynamics of naval vessels and control them for autonomous surveys. Designing CPS that can operate autonomously for extended periods without human intervention is critical. This involves ensuring not just operational endurance but also the ability to make decisions and adapt to new situations over time. Implementing control systems to manage the vessel's propulsion and maintain stability in varying sea conditions is challenging. This involves controlling rudders, thrusters, and other propulsion systems. Figure 4 provides an example of recovering non-linear systems from sensor attacks. The settings can be found in Table 1. It is convenient to set the attack state, reference, time frequency, and other

Table 1. Settings Used in Each Benchmark

	CSTR	Quadrotor	Naval Vessel
attack state	Temperature	Altitude	Speed
reference	300 K	4 m	1.03 m/s
dt	0.1 s	0.01 s	1 s
noise	uniform distribution [0, 0.1]	uniform distribution [0, 5e-6]	uniform distribution [0, .015]
detection delay	1 s	0.2 s	10 s
bias	−30 K	−1.5 m	−0.3 m/s
safe set	(250, 360) K	(0, 200) m	(0, 150) m/s
target set	(299, 301) K	(4.8, 5.2) m	(0.8, 1.2) m/s
control limits	(200, 300) K	(−50, 200) N	[(0, 4), (−2, 2)]
MPC frequency	10 Hz	10 Hz	1 Hz
Orig Controller	P = 0.50 I = 1.33 D = −0.05	P = 100 I = 0 D = −19	P = 0.45, 0.1 I = 0.05, 0 D = 0, −3.5



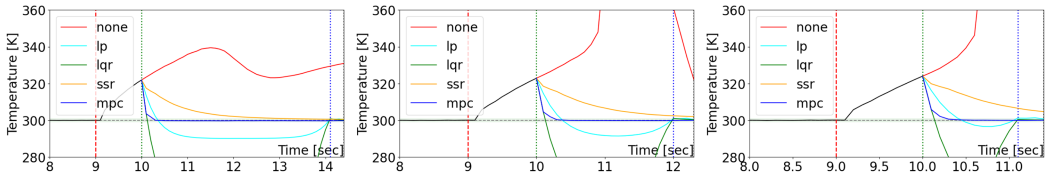
(a) Bias1 (b) Bias2 (c) Bias3

Fig. 5. Sensitivity analysis of recovery for CSTR in CPSim. The systems are under bias sensor attacks. The bias values are $\{-25, -30, -35\}$.

parameters using the built-in API of CPSim. This provides a user-friendly interface to simulate and test customized security algorithms on CPS benchmarks. The recovery control algorithms introduced in Section 6 have been tested on these systems to show the capabilities of CPSim. CPSim can simulate the dynamics of non-linear CPSs under attack and during the recovery, it is efficient to design security experiments and implement them using CPSim.

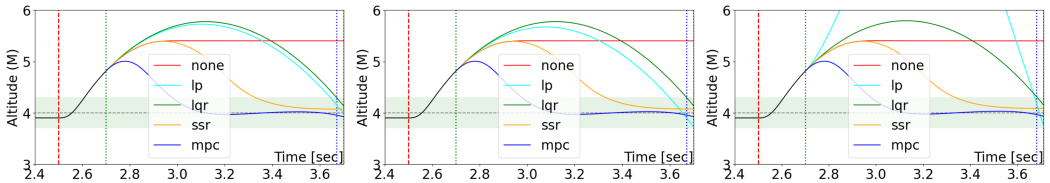
To be noticed, sometimes, the recovery algorithms may not find any feasible solutions from the optimization. From Figure 4(c), we can see linear programming does not provide any solution for the vessel benchmark. Real-time systems often have stringent constraints related to time, resources, and performance. If these constraints are too rigid or conflicting, then it may render the optimization problem infeasible. For example, if the time constraints for a task are tighter than what the system can physically achieve, then the optimization algorithm may fail to find a solution that satisfies all constraints. Moreover, CPSs can have a large and complex problem space, especially in systems with a high degree of concurrency or those requiring intricate synchronization. In such cases, the sheer complexity of the problem can make it difficult for recovery algorithms to find a feasible solution, particularly within the time limits imposed by real-time requirements.

Figure 5 shows the recovery performance on the CSTR benchmark under different levels of attacks. The values of the bias attacks are set to $\{-25, -30, -35\}$. The image shows that as the bias values increase, the deviation of the system state at the time of the detector's alarm (indicated by the green vertical line) becomes larger, making recovery more challenging and the deadline for recovery (indicated by the blue vertical line) closer. It is important to test the robustness of



(a) Noise1 (b) Noise2 (c) Noise3

Fig. 6. Sensitivity analysis of recovery for CSTR in CPSim. The systems are under bias sensor attacks. The noise upper bounds are $\{0.05, 0.1, 0.15\}$.



(a) Noise1 (b) Noise2 (c) Noise3

Fig. 7. Sensitivity analysis of recovery for quadrotor in CPSim. The systems are under bias sensor attacks. The noise upper bounds are $\{0.01, 0.05, 0.2\} \times 10^{-4}$.

the algorithms under attacks to identify and mitigate potential vulnerabilities. Additionally, under real-time scenarios, testing the algorithms under various attacks can evaluate the temporal performance of the algorithms such as deadlines and recovery delay. Under attack conditions, the ability of these systems to maintain deadline adherence is a crucial measure of their robustness and reliability. Testing how algorithms perform under attack can reveal their ability to prioritize critical tasks and maintain functionality even when under attack. Figure 6 shows the recovery performance on the CSTR benchmark under different levels of noise. The upper bounds of the uniform noises are set to $\{0.05, 0.1, 0.15\}$. We visualize how the recovery algorithms from CPSim work on the CSTR benchmark when the system is in a more and more noisy environment. The image shows that as the noise upper bounds increase, the deadline for recovery (indicated by the blue vertical line) becomes closer. Different recovery algorithms have different levels of resistance to noise. For example, we can see the MPC recovery algorithm has a more robust performance under noises compared to SSR recovery.

CPSs are often deployed in critical applications where failure or incorrect operation can have severe consequences. Testing these systems under different noise conditions ensures they can maintain their performance and reliability even in the presence of disturbances. This is especially important for systems that operate in unpredictable or harsh environments. Exposure to different noise conditions during testing allows developers to evaluate and enhance the system's error detection, handling, and recovery mechanisms. This is critical for ensuring system stability and continuity of operation under adverse conditions. Testing under various noise scenarios provides a more comprehensive understanding of the system's performance limits. It helps in identifying performance bottlenecks and areas where the system might fail or degrade under stress, leading to more informed design and optimization decisions. Similarly, we have implemented sensitivity analysis on the quadrotor as well. Figure 7 shows the recovery performance for the quadrotor benchmark; the noise upper bounds are set to $\{0.01, 0.05, 0.2\} \times 10^{-4}$. Similar to Figure 6, the MPC algorithm demonstrates robust performance across different noise upper bounds.

(a) Attacked (b) Recovering (c) Recovered

Fig. 8. Simulation example of the connection between CPSim and SVL simulator. The car is under sensor attack and being recovered.

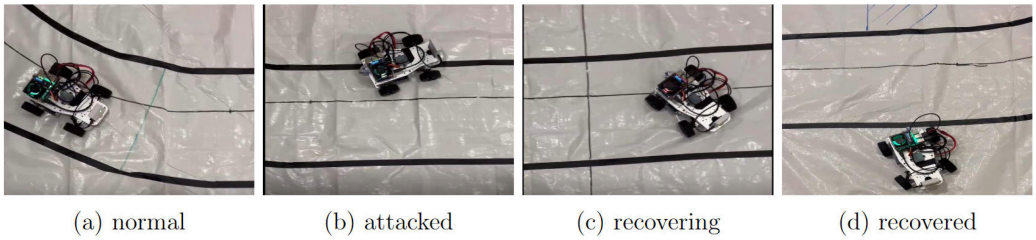
The LG SVL Simulator is a powerful tool for the development and testing of autonomous vehicle systems, particularly within the domain of CPS [24]. This simulator offers a comprehensive environment for researchers, engineers, and developers to rigorously test and validate various aspects of autonomous vehicle technology, from individual components to full system integration. SVL Simulator provides a highly realistic and detailed simulation environment. This includes accurate representations of urban and rural landscapes, diverse weather conditions, and varied traffic scenarios. Such a rich simulation environment is crucial for testing autonomous vehicles in a wide range of realistic conditions, which they are likely to encounter in the real world. One of the key strengths of the LG SVL Simulator is its ability to simulate a wide array of sensors used in autonomous vehicles. This includes LiDAR, cameras, radar, GPS, and IMUs. The fidelity of these sensor simulations is essential for developing effective sensor fusion algorithms, which are critical for the perception systems in autonomous vehicles.

CPSim is compatible with external high-fidelity simulators such as SVL. SVL Simulator provides a highly realistic environment, including detailed urban and rural scenarios, diverse traffic conditions, and variable weather. Integrating this with CPSim can significantly improve the realism of simulations for autonomous vehicle systems. By combining CPSim's focus on CPS security with SVL's dynamic and richly detailed environments, developers can test autonomous systems in more complex and realistic scenarios than what might be achievable with either tool alone.

Figure 8 shows the interface of STL while connecting to CPSim through ROS. In the left subfigure, the car started to experience a bias sensor attack. In the middle figure, the car is recovering using the LQR recovery algorithms. In the right figure, the car is recovered and stopped at the shoulder of the road. For more details about this high-fidelity simulator experiment, please check the demo video on the documentation website.

It is also possible to deploy CPSim on a real testbed. Autonomous vehicles sense states and environments, make decisions, and control mobility. Our robotic vehicle testbeds, whose hardware architecture is shown in Figure 10, simulate these functions through the following stages:

Our testbed is equipped with an **Inertial Measurement Unit (IMU)**, **Ultra-wideband (UWB)**, and encoder sensors that measure attitude, position, and velocity, respectively. We can also use cameras and LiDAR to collect additional environmental data. However, these sensors are vulnerable to sensor attacks. A Raspberry Pi 4B with **Robot Operating System (ROS2)** serves as the main controller. It collects sensor data, estimates vehicle states, and generates control signals. The system uses different controllers for longitudinal and lateral control. For cruise control, a PID controller stabilizes the testbed's velocity based on encoder feedback. For lane-keeping, a Stanley Controller uses the front axle as its reference point, considering both heading error and cross-track error. We can also deploy the proposed attack detection and recovery algorithms on this system.



(a) normal (b) attacked (c) recovering (d) recovered

Fig. 9. Deploy CPSim on a real testbed.

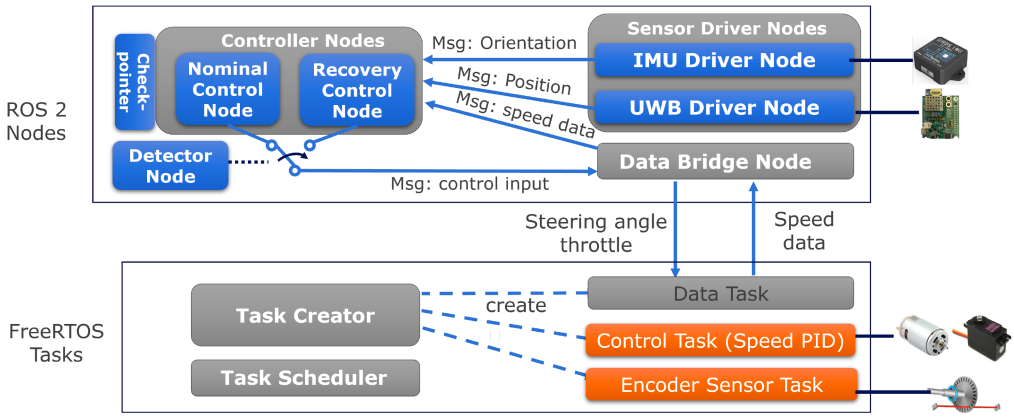


Fig. 10. Deploy CPSim on a real testbed.

An STM32 microcontroller with a FreeRTOS system receives control signals from the Raspberry Pi through a **Universal Asynchronous Receiver/Transmitter (UART)** protocol. Running a real-time operating system, it performs time-sensitive tasks such as generating **Pulse Width Modulation (PWM)** signals to drive actuators. The actuator stage includes components such as motors and servos that execute vehicle movements according to control signals.

Figure 9 illustrates the process of our deployed algorithm recovering the testbed from a sensor attack. Figure 9(a) shows the testbed operating normally on the track. Figure 9(b) depicts the testbed beginning to deviate upwards due to a bias sensor attack. Figure 9(c) represents the testbed during the recovery process. Figure 9(d) shows the testbed after it has been recovered to the target set, which in this case is set to be the shoulder on the lower side of the road.

It is important to note that in Figures 8, 9, and 10, we merely demonstrate the functionality of integrating CPSim with high-fidelity simulators such as SVL and real testbeds. For detailed experimental procedures and parameters, please refer to Reference [32]. It should be noted that there are some limitations in this version of CPSim, such as the recovery methods deployed being based solely on reachability analysis [33]. Additionally, the number of benchmarks is relatively small, with only three types available. However, the system does support users adding their own benchmarks conveniently.

8 Conclusion

In conclusion, we proposed a user-friendly simulation toolbox, CPSim, for security problems in CPS. Users can customize their benchmarks and controllers inheriting the built-in classes.

Moreover, we provide comprehensive experiments and examples of how to use the built-in recovery algorithms against sensor attacks. We believe this open-source toolbox can benefit many researchers in CPS fields, especially junior students without expertise in implementing experiments from scratch.

Acknowledgement

The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the National Science Foundation (NSF).

References

[1] Francis Akowuah and Fanxin Kong. 2021. Real-time adaptive sensor attack detection in autonomous cyber-physical systems. In *Proceedings of the IEEE 27th Real-Time and Embedded Technology and Applications Symposium (RTAS'21)*. IEEE, 237–250.

[2] Daniele Antonioli and Nils Ole Tippenhauer. 2015. MiniCPS: A toolkit for security research on CPS networks. In *Proceedings of the 1st ACM Workshop on Cyber-physical Systems-security and/or Privacy*. 91–100.

[3] Dhananjay Bhor, Kavinkadhirselvan Angappan, and Krishna M. Sivalingam. 2016. Network and power-grid co-simulation framework for smart grid wide-area monitoring networks. *J. Netw. Comput. Applic.* 59 (2016), 274–284.

[4] Kianoosh Boroojeni, M. Hadi Amini, Arash Nejadpak, Tomislav Dragičević, Sundaraja Sitharama Iyengar, and Frede Blaabjerg. 2016. A novel cloud-based platform for implementation of oblivious power routing for clusters of micro-grids. *IEEE Access* 5 (2016), 607–619.

[5] Gianni Celli, Paolo Atillio Pegoraro, Fabrizio Pilo, Giuditta Pisano, and Sara Sulis. 2014. DMS cyber-physical simulation for assessing the impact of state estimation and communication media in smart grid operation. *IEEE Trans. Power Syst.* 29, 5 (2014), 2436–2446.

[6] Rohan Chabukswar, Bruno Sinopoli, Gabor Karsai, Annarita Giani, Himanshu Neema, and Andrew Davis. 2010. Simulation of network attacks on SCADA systems. In *Proceedings of the 1st Workshop on Secure Control Systems*.

[7] Hongjun Choi, Sayali Kate, Yousra Aafer, Xiangyu Zhang, and Dongyan Xu. 2020. Software-based realtime recovery from sensor attacks on robotic vehicles. In *Proceedings of the 23rd International Symposium on Research in Attacks, Intrusions and Defenses (RAID'20)*. 349–364.

[8] James Farwell and Rafal Rohozinski. 2011. Stuxnet and the future of cyber war. *Survival* 53, 1 (2011), 23–40.

[9] Thor I. Fossen. 2011. *Handbook of Marine Craft Hydrodynamics and Motion Control*. John Wiley & Sons.

[10] John D. Hedengren. 2008. A nonlinear model library for dynamics and control. *Yeast* 7 (2008), 24.

[11] Nathan Koenig and Andrew Howard. 2004. Design and use paradigms for Gazebo, an open-source multi-robot simulator. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS'04)*. IEEE, 2149–2154.

[12] Fanxin Kong, Meng Xu, James Weimer, Oleg Sokolsky, and Insup Lee. 2018. Cyber-physical system checkpointing and recovery. In *Proceedings of the ACM/IEEE 9th International Conference on Cyber-Physical Systems (ICCPS'18)*. IEEE, 22–31.

[13] Edward A. Lee. 2008. Cyber physical systems: Design challenges. In *Proceedings of the 11th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing (ISORC'08)*. IEEE, 363–369.

[14] Edward A. Lee. 2021. Determinism. *ACM Trans. Embed. Comput. Syst.* 20, 5 (2021), 1–34.

[15] Mengyu Liu, Lin Zhang, Pengyuan Lu, Kaustubh Sridhar, Fanxin Kong, Oleg Sokolsky, and Insup Lee. 2022. Fail-safe: Securing cyber-physical systems against hidden sensor attacks. In *Proceedings of the IEEE Real-Time Systems Symposium (RTSS'22)*. IEEE, 240–252.

[16] Mengyu Liu, Lin Zhang, Vir Phoha, and Fanxin Kong. 2023. Learn-to-respond: Sequence-predictive recovery from sensor attacks in cyber-physical systems. In *Proceedings of the IEEE Real-Time Systems Symposium (RTSS'23)*. IEEE.

[17] Pengyuan Lu, Lin Zhang, Mengyu Liu, Kaustubh Sridhar, Oleg Sokolsky, Fanxin Kong, and Insup Lee. 2024. Recovery from adversarial attacks in cyber-physical systems: Shallow, deep, and exploratory works. *Comput. Surv.* 56, 8 (2024), 1–31.

[18] Yehan Ma, Chenyang Lu, Bruno Sinopoli, and Shen Zeng. 2020. Exploring edge computing for multitier industrial control. *IEEE Trans. Comput.-Aid. Des. Integ. Circ. Syst.* 39, 11 (2020), 3506–3518.

[19] Naser Hossein Motlagh, Tarik Taleb, and Osama Arouk. 2016. Low-altitude unmanned aerial vehicles-based internet of things services: Comprehensive survey and future perspectives. *IEEE Internet Things J.* 3, 6 (2016), 899–922.

[20] Automotive News. 2023. Tesla Model 3 hacked by cybersecurity team in minutes. Retrieved from https://www.autonews.com/mobility-report/tesla-model-3-hacked-cybersecurity-team-minutes

[21] The Hacker News. 2016. Hacker Hijacks a Police Drone from 2 Km Away with $40 Kit. Retrieved from https://thehackernews.com/2016/04/hacking-drone.html

[22] Raul Quinonez, Jairo Giraldo, Luis Salazar, Erick Bauman, Alvaro Cardenas, and Zhiqiang Lin. 2020. SAVIOR: Securing autonomous vehicles with robust physical invariants. In *Proceedings of the 29th USENIX Security Symposium (USENIX Security'20)*. 895–912.

[23] Ragunathan Rajkumar, Insup Lee, Lui Sha, and John Stankovic. 2010. Cyber-physical systems: The next computing revolution. In *Proceedings of the Design Automation Conference (DAC'10)*. IEEE, 731–736.

[24] Guodong Rong, Byung Hyun Shin, Hadi Tabatabaee, Qiang Lu, Steve Lemke, Mārtiņš Možeiko, Eric Boise, Geehoon Uhm, Mark Gerow, Shalin Mehta, Eugene Agafonov, Tae Hyung Kim, Eric Sterner, Keunhae Ushiroda, Michael Reyes, Dmitry Zelenkovsky, and Seonman Kim. 2020. rongLgsvl simulator: A high fidelity simulator for autonomous driving. In *Proceedings of the IEEE 23rd International Conference on Intelligent Transportation Systems (ITSC'20)*. IEEE, 1–6.

[25] Asgeir J Sørensen. 2005. Marine cybernetics. *Lecture Notes for TMR4240 Marine Control Systems, Dept. Of Marine Tehcnology, NTNU*, Citeseer. Trondheim, Norway. http://www.ivt.ntnu.no/imt/courses/tmr4240

[26] Riccardo Taormina, Stefano Galelli, H. C. Douglas, Nils Ole Tippenhauer, Elad Salomons, and Avi Ostfeld. 2019. A toolbox for assessing the impacts of cyber-physical attacks on water distribution systems. *Environ. Model. Softw.* 112 (2019), 46–51.

[27] Quinn Thibeault, Jacob Anderson, Aniruddh Chandratre, Giulia Pedrielli, and Georgios Fainekos. 2021. PSY-TaLiRo: A Python toolbox for search-based test generation for cyber-physical systems. In *Proceedings of the 26th International Conference on Formal Methods for Industrial Critical Systems (FMICS'21)*. Springer, 223–231.

[28] WIRED. 2022. Security News This Week: Attackers Keep Targeting the US Electric Grid. Retrieved from https://www.wired.com/story/attacks-us-electrical-grid-security-roundup/

[29] Rajaa Vikhram Yohanandhan, Rajvikram Madurai Elavarasan, Premkumar Manoharan, and Lucian Mihet-Popa. 2020. Cyber-physical power system (CPPS): A review on modeling, simulation, and analysis with cyber security applications. *IEEE Access* 8 (2020), 151019–151064.

[30] Massimiliano Zanin, Ernestina Menasalvas Ruiz, Alejandro Rodríguez-González, Christian Wolff, Juana Wendt, Elisa A. Herrmann, and Pavel Smrz. 2020. Developing a data analytics toolbox to support CPS-based services. In *Proceedings of the 9th Mediterranean Conference on Embedded Computing (MECO'20)*. IEEE, 1–7.

[31] Lin Zhang, Xin Chen, Fanxin Kong, and Alvaro A. Cardenas. 2020. Real-time recovery for cyber-physical systems using linear approximations. In *Proceedings of the 41st IEEE Real-Time Systems Symposium (RTSS'20)*. IEEE.

[32] Lin Zhang, Mengyu Liu, and Fanxin Kong. 2023. Demo: Simulation and security toolbox for cyber-physical systems. In *Proceedings of the IEEE 29th Real-Time and Embedded Technology and Applications Symposium (RTAS'23) Brief Presentations*. IEEE.

[33] Lin Zhang, Kaustubh Sridhar, Mengyu Liu, Pengyuan Lu, Xin Chen, Fanxin Kong, Oleg Sokolsky, and Insup Lee. 2023. Real-time data-predictive attack-recovery for complex cyber-physical systems. In *Proceedings of the IEEE 29th Real-Time and Embedded Technology and Applications Symposium (RTAS'23)*. IEEE, 209–222.

[34] Lin Zhang, Zifan Wang, Mengyu Liu, and Fanxin Kong. 2022. Adaptive window-based sensor attack detection for cyber-physical systems. In *Proceedings of the 59th ACM/IEEE Design Automation Conference*. 919–924.

Received 30 November 2023; revised 5 June 2024; accepted 12 June 2024