Enhancing CRYSTAL: Preventive Recovery in Brief

Fereidoun Moradi[1], Zahra Moezkarimi[1], and Marjan Sirjani[1]

School of Innovation, Design and Engineering, Mälardalen University, Västerås, 722 20, Sweden
fereidoun.moradi,zahra.moezkarimi,marjan.sirjani@mdu.se

Abstract

We propose a model-based mechanism for recovering the system into its normal state when it is partially damaged by CPS attacks. The CRYSTAL framework includes mechanisms for monitoring and detecting cyber and physical attacks that exploit CPS-related vulnerabilities in communication channels and components. We enhance the framework with a module that generates repair actions instead of just terminating the system in case of successful attack detection. When an attack is detected at runtime, the module drops the malicious action and transfers the system back to its normal execution. The repair actions are defined at the design phase using counterexamples generated by model checking, and our approach is able to automatically identify and use such repair actions.

1 Introduction

CRYSTAL framework is introduced in [4] for building safe and secure Cyber-Physical Systems (CPS). CRYSTAL provides a set of methods and tools for modeling the system, defining and augmenting attack models in the system model (a Timed Rebeca model [6]), then abstracting the model and creating a Tiny Digital Twin, and finally, developing a monitor to detect cyberattacks. A Tiny Digital Twin is an abstract behavioral model of the system [5]. It is automatically derived from the Timed Rebeca model by excluding actions that are not observable from the monitor perspective. The Tiny Digital Twin is constructed at the design phase and is used by the monitor at the operational phase to ensure that the system is functioning as intended. The architecture of CRYSTAL is inspired by the MAPE-K[1] feedback loop [3]. The monitor is strategically positioned between the control part and the sensor and actuator components in CPS applications. It observes the visible inputs and outputs of the controllers, traverses state transitions in the Tiny Digital Twin, and detects any misbehavior occurring during system operation.

In CRYSTAL, whenever the target system is about to execute an action that is not specified in the Tiny Digital Twin, the monitor drops the action and stops the system execution. In this short paper, we present the idea of a recovery mechanism that reacts to malicious actions and transfers the system back to its normal execution after the malicious action is dropped. The recovery mechanism can be considered as Runtime Enforcement [1] since the system is enforced to run according to its specification.

2 Recovery Mechanism

We propose a preventive recovery mechanism. The preventive recovery mechanism starts when an execution deviates from the intended behavior specified by Tiny Digital Twin. In CRYSTAL, the STRIDE threat modeling [7] is used as a guideline to define attack scenarios and build attack models. The attack models are used to verify the correct behavior specified in the model and

[1]The acronym MAPE-K stands for **M**onitor, **A**nalyze, **P**lan, **E**xecute, plus **K**nowledge.

discover all *counterexamples* that violate the safety properties of interest. We derive all *known failures* from counterexamples generated by the Timed Rebeca model checker while exploring the state space. The reactions to known failures might be different.

We propose a preventive recovery mechanism that addresses deviations from the reference model (Tiny Digital Twin) by two main techniques consisting of a sequence of *repair actions*. The first technique is applied when there is an attack changing the actuation command of the controllers, like a successful *tampering* attack on a controller. Here, we modify the incorrect actuation command of the controllers and reconfigure the mode of the controllers to a safe state. The reconfiguration is applied based on the state information available on counterexamples and the state transitions on the Tiny Digital Twin. The second technique is needed when there is false sensor data, like when there is an *injection* of false sensor data. Here, we need to replace the incorrect values with the correct ones, and for that we need redundant sensors. The reaction to a deviation is determined based on *counterexamples* and may vary across different CPS applications. It may involve reverting the system to its previous state and executing an alternative action.

3 A Small Example

Figure 1 shows a Pneumatic Control System (PCS) [2] case study. The control system regulates the movement of two cylinders in multiple directions. Each cylinder is controlled by a dedicated controller to regulate the movement in either left-right or up-down directions. The desired sequence of movements of the cylinders is as follows: (1) CylinderB moves down (picks up a particle), (2) CylinderB moves up, (3) CylinderA moves right (pushes CylinderB to the right), (4) CylinderB moves down (leaves the particle) and (5) then up, (6) CylinderA moves left. We assume that ControllerA starts its linear motion from the left at the top of location X.

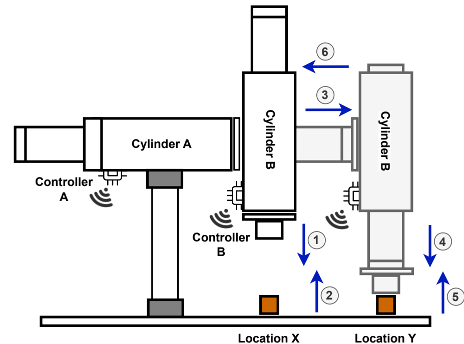

Figure 1: PCS with two cylinders (adapted from [2]). The cylinders work together to pick up a particle from location X and move it to location Y.

For this case study, we checked 3037 attack scenarios on the Timed Rebeca model of PCS. Only in 383 cases, these attacks caused the violation of safety properties (and hence a failure), and 383 counterexamples are generated by the model checker when verifying the PCS Timed Rebeca model augmented with attack scenarios. In this small example, we consider two counterexamples out of the 383 total counterexamples, as shown in Table 1.

By comparing each counterexample with its corresponding correct execution path in Tiny Digital Twin, we identify an action or a sequence of actions that returns the control system to a state where it can perform a different execution path to avoid failure. These actions are called *repair actions*. We keep both counterexamples and their corresponding repair actions in a repository. At runtime, if the monitor detects a malicious action present in a counterexample, it executes its corresponding repair actions from the repository.

[2] https://github.com/fereidoun-moradi/Reconfigurable-Pneumatic-System

Table 1: The counterexamples (known failures) for safety properties

#	safety property	counterexample	repair action
1	!((motionR&&motionU)||(motionL&&motionD))	$S67 \xrightarrow{controllerb.getsense[0]} S65,...,S70 \xrightarrow{cyla.actuate[1]} S83$	dropping and reconfig.
2	!(locXa_outRange||locXb_outRange)	$S67 \xrightarrow{controllerb.getsense[0]} S65,...,S74 \xrightarrow{controllera.getsense[2]} S84$	dropping
cyla: CylinderA, cylb: CylinderB, actuate[1]: moves right/down, actuate[-1]: moves left/up			

Table 1 shows the safety properties along with counterexamples and their corresponding repair actions. The properties are defined using the values of the variables loc and motion for two cylinders in the PCS Timed Rebeca model. Property #1 ensures that both cylinders do not move diagonally. In this system, CylinderB cannot move up (motionU) or down (motionD) while CylinderA is moving to the right (motionR) or left (motionL). Property #2 ensures that both CylinderA and CylinderB only have motion between the initial position and the end position in locations X or Y.

In the counterexample for property #1, as shown in the state transitions in Figure 2(a), in state S67, controllerA and controllerB receive the location information of the cylinders through sensor data, i.e., controllerb.getsense[0] and controllera.getsense[0] (state S67 to state S70). At state S70, according to the outgoing transition, the intended action is cylb.actuate[−1] that actuates CylinderB to move downward and pick up the particle. However, at the current state, the action cyla.actuate[1] is transmitted by controllerA to move CylinderA to the right (see the dotted red outgoing transition from S70 to S83). The monitor detects a deviation at state S70 and consequently drops the action cyla.actuate[1]. To recover the system from the deviation, the controllerA is reconfigured. According to the transition controllera.getsense[0] from S65 to S70, the recovery module keeps state S70 as the current state and executes the monitor to proceed with the system execution.

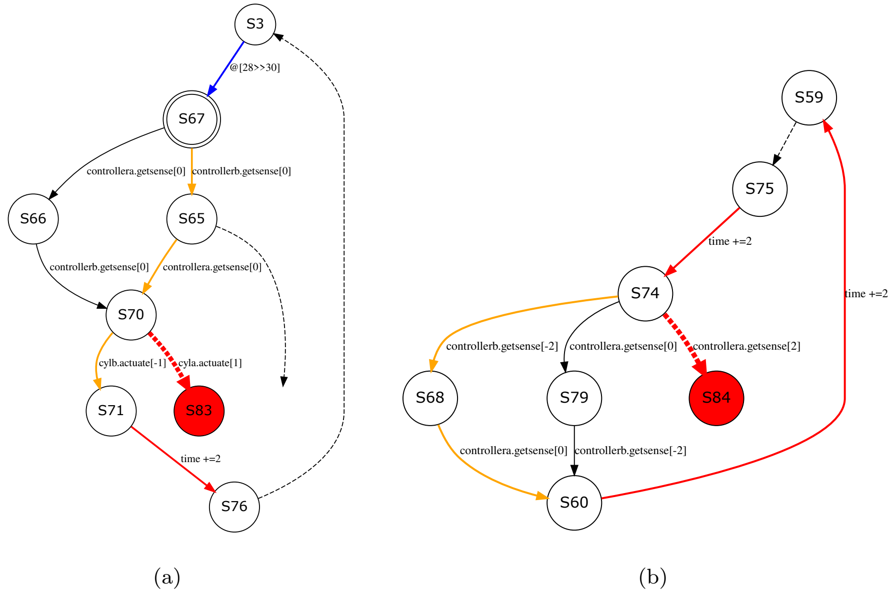

Figure 2: The violation of property #1 (a) and property #2 (b) are shown on a subset of the state transitions in the Tiny Digital Twin for the PCS case study.

In the counterexample for property #2, controllerA receives sensor data controllera.getsense[2] at state S74 as shown in Figure 2(b). At state S74, CylinderB has been moved down and the location information is transmitted to the controllerB through sensor data controllerb.getsense[−2]. However, at the current state, incorrect sensor data is injected into the system. The monitor detects the deviation and drops the sensor data controllera.getsense[2]. In this case, the incorrect sensor data can be injected either by an attacker or by a compromised sensor. The recovery module does not modify any values on the controllers, but it triggers the monitor (when incorrect sensor data is dropped) to proceed and check the status of the system execution. This failure can be recovered using a redundant sensor if the original sensor has been compromised.

4 Future Direction

We address known failures based on the insights gained from property violations during the design phase. This research study also involves the development of a pathfinding method, aimed at identifying the recovery paths within the Tiny Digital Twin, with a focus on achieving the shortest possible recovery times. In addition, we work on addressing *unknown* failures, such as attacks that are not discovered during the design phase (e.g., stealthy attacks). For these challenges, we propose self-healing mechanisms that rely on both the Tiny Digital Twin and a set of meta-rules prepared by experts.

References

[1] Yliès Falcone, Leonardo Mariani, Antoine Rollet, and Saikat Saha. Runtime failure prevention and reaction. *Lectures on Runtime Verification: Introductory and Advanced Topics*, pages 103–134, 2018.

[2] Zivana Jakovljevic, Vuk Lesi, and Miroslav Pajic. Attacks on distributed sequential control in manufacturing automation. *IEEE Transactions on Industrial Informatics*, 17(2):775–786, 2020.

[3] Jeffrey Kephart and David Chess. The vision of autonomic computing. *Computer*, 36(1):41–50, 2003.

[4] Fereidoun Moradi, Sara Abbaspour, Bahman Pourvatan, Zahra Moezkarimi, and Marjan Sirjani. Crystal framework: Cybersecurity assurance for cyber-physical systems. submitted to Journal of Logical and Algebraic Methods in Programming (JLAMP), 2023.

[5] Fereidoun Moradi, Bahman Pourvatan, Sara Abbaspour Asadollah, and Marjan Sirjani. Tiny Twins for detecting cyber-attacks at runtime using Concise Rebeca Time Transition System. *Journal of Parallel and Distributed Computing*, page 104780, 2023.

[6] Arni Hermann Reynisson, Marjan Sirjani, Luca Aceto, Matteo Cimini, Ali Jafari, Anna Ingólfsdóttir, and Steinar Hugi Sigurdarson. Modelling and simulation of asynchronous real-time systems using Timed Rebeca. *Sci. Comput. Program.*, 89:41–68, 2014.

[7] Adam Shostack. *Threat modeling: Designing for security.* Wiley, 2014.