An Actor-based Approach for Security Analysis of Cyber-Physical Systems

(Technical Report)

Fereidoun Moradi*[1], Sara Abbaspour Asadollah[1], Ali Sedaghatbaf[1], Aida Čaušević[1], Marjan Sirjani[1], Carolyn Talcott[2]
[1] *School of Innovation, Design and Engineering, Mälardalen University, Västerås, Sweden*
[2] *SRI International, Melno Park, CA, USA*
{fereidoun.moradi, sara.abbaspour, ali.sedaghatbaf, aida.causevic, marjan.sirjani}@mdh.se
clt@csl.sri.com

***Abstract*—Cyber-Physical Systems are networks of interconnected computing, networking and physical devices. Communicating through a network makes these systems vulnerable to possible malicious attacks. These systems may play a crucial role in controlling critical infrastructures and their security is of paramount importance. Formal modeling and verification can be effectively used for evaluating secure designs, particularly at the architecture level of complex systems. In this work, we present an actor-based approach for security analysis of Cyber-Physical Systems at the design phase. We use Timed Rebeca, an actor-based modeling language, to model the behavior of components and potential attacks, and verify the security properties using Rebeca model checking tool. We employ STRIDE model as a reference for classifying the attacks. To demonstrate the applicability of our approach, we use a Secure Water Treatment (SWaT) system as a case study. We analyze the architecture of the SWaT system using three different attack schemes in which various parts of the system network and physical devices are compromised. At the end, we identify single and combined attack scenarios that violate security properties.**

***Index Terms*—Cyber-Physical System (CPS), Rebeca Model Checker (RMC), Secure Water Treatment (SWaT), Attack scenarios.**

I. Introduction

Cyber-Physical System (CPS) refer to a system in which physical, computational and communication components are integrated to achieve a larger goal [1]. Generally, a CPS includes three kinds of components i.e. sensors, controllers and actuators. Sensors are responsible to gather data about the state of a physical process and submit them to the controllers. By analyzing the data, if the controllers detect a need for some changes in the process, they apply those changes by sending appropriate commands to the actuators [2]. Despite the advantages of combining cyber and physical spaces, connection to the Internet makes CPS exposed to several attacks, which may lead to undesirable changes in the physical process [3].

There are a variety of use cases for CPS such as water treatment systems, manufacturing products, health care services, and electrical grid operations [4], [5], [6]. Despite the advantages of combining cyber and physical spaces, connection to the Internet makes CPS exposed to several attacks, which may lead to undesirable changes in the physical process [7], [8]. For example, as reported by Kaspersky ICS CERT [9] in February 2019, some parts of the production line of Japanese Optics manufacturer HOYA in Thailand was crashed by a Malware [10]. Once the malware could spread over a hundred of the company computers, it stole user credentials, and distributed a cryptocurrency miner. The incident also affected computers at HOYA headquarters in Japan that were connected to the network, disrupting the invoice issue process.

As another example, an attack took place on a US power facility in March 2019 that caused interruption of electrical grid operations [11]. The firewall on a network device was compromised by an anonymous remote entity in 10 hours. The attackers exploited a known vulnerability in the device which caused the firewall to reboot repeatedly. During each reboot, the firewall was out of service for about 5 minutes. The direct impact of the attack was communication outages between the control center and devices.

To tackle CPS attacks, it is required to consider security of CPS beyond the IT systems standard information security [12], [13], and several researchers have proposed formal or simulation methods to analyse the security of CPS [14], [15], [16]. The work presented in this paper is a step towards an actor-based approach for assessing the security aspects of CPSs. We use Timed Rebeca as an actor-based modeling language [17], [18], [19] to model the behavior of CPS components and attack scenarios, and we utilize STRIDE [20] model as a reference for classifying potential attacks on a CPS.

As an actor-based language, Rebeca [21], [22] is well-suited for modeling complex behaviors in event-based asynchronous distributed systems [23]. Timed Rebeca is supported by a model checking tool suite Afra [24] and can be used for verifying CPS [25]. In this work, beside modeling a cyber-physical system, we propose a model of attack for both kinds of attacks on communication and components. Using Timed Rebeca, an attacker is modeled as an actor to jeopardise the communication, and a compromised component is modeled as an actor with possible malfunction. In addition, we use the security threats category, STRIDE, to systematically map the reported CPS attacks in [26], [27], [28] to the STRIDE threat types and identify the attacks in our models. By model checking we analyze security of the CPS design to recognize where the potential attack scenarios can successfully cause a failure in the system. The output counter-example gives us the trace of events leading to a security failure which can then be used for developing mitigation plans.

We demonstrate the applicability of this method in practice using a case study on Secure Water Treatment (SWaT) system [29]. The natural mapping between the communicating entities in the problem domain and actors in Rebeca models makes the approach easy to understand and reuse [30].

The paper is organized as follows. In Section II, we introduce Rebeca, and our approach for security analysis is introduced in Section III. Section IV shows how our attack models can be classified within the STRIDE model. In Section V, we describe the case study and evaluate our experimental results. Section VI discusses the related work and Section VII concludes the paper and gives a summary of our future works.

II. An Actor-based modeling language: Rebeca

Rebeca is an actor-based modeling language with formal foundation used for modeling concurrent and reactive systems with asynchronous message passing [21], [31]. A Rebeca model consists of the definition of *reactive classes*, each describing the type of a certain number of *actors* (called *rebecs*, we use both terms rebec and actor interchangeably in the Rebeca context). Each reactive class declares the size of its message queue, a set of *state variables*, and the messages to which it can respond. Each rebec has a set of *known rebecs* to which it can send messages. The behavior of a rebec is determined by its *message servers*. Each rebec takes a message from its message queue and executes the corresponding message server. Taking a message from the queue to execute it can be seen as an event. Communication takes place by asynchronous message passing, which is non-blocking for both sender and receiver.

Rebeca comes with a formal semantics that makes it suitable for model checking purposes. Additionally, the language supports temporal logic to specify desired properties. Timed Rebeca [17], [19] is an extension of Rebeca where computation time and network delay can be modeled. In Timed Rebeca, each rebec has its own local clock, but there is also a notion of global time based on synchronized distributed clocks of all rebecs. Messages that are sent to a rebec are put in its message bag together with their arrival time, and their deadline. Methods are executed atomically, but the passing of time during the execution of methods can be modeled.

Afra tool [24] is an IDE with a dedicated model checker, Rebeca Model Checker (RMC), for verifying Rebeca family models. The tool provides development environment for models, property specification, model checking, and counter-example visualization.

III. Methodology

As depicted in Figure 1, the proposed method for CPS security analysis includes the following steps: (1) the Rebeca model of the CPS is developed from the system design specifications, (2) the potential attack scenarios against the system are modeled, (3) the security properties are defined in terms of assertions or temporal logic, and (4) Afra is used to identify the events trace leads to a security failure. The above steps are elaborated in the following subsections.

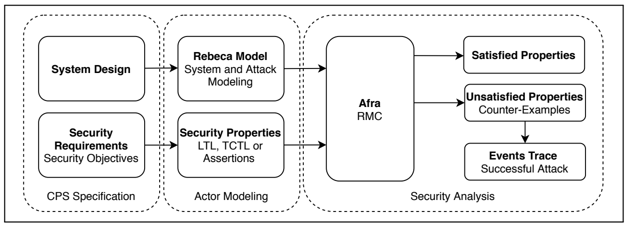


Figure 1: The overview of the actor-based security analysis process.

A. Building the Rebeca model of the Cyber-Physical System

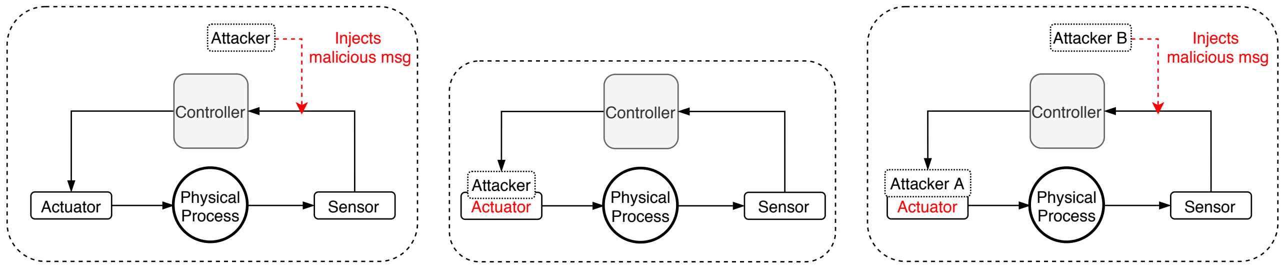
We consider each CPS component or physical process as an actor. We realise four types of actors in our Rebeca model, controllers, sensors, actuators and physical processes. Generally, the interaction scenarios between these actors follow a closed-loop feedback. Sensor observes the physical component's status, and sends the sensed data to the controller denoting the state of the physical component. Based on the received sensed data, the controller sends the control command to the actuator, and the actuator performs the actual physical change. The Rebeca model of a CPS includes reactive classes corresponding to the four categories of actors. In real cases, we may have different kinds of actors belonging to each category (e.g., temperature sensors, speed sensors, etc.), and each kind may be defined by a distinct reactive class.

B. Attack Modeling

According to the malicious behaviour on communication channels and components three cases are considered as follows: (1) attacker targets the communication channel between two components through injecting malicious messages, (2) attacker manipulates the internal behavior of one or more components e.g. through malicious code injection, and (3) one or more attackers perform a coordinated attack to launch malicious behaviour on both the communication channels and the components. To illustrate these cases, we define three attack schemes.

Scheme-A: Attack on Communication indicates a situation in which an attacker injects malicious messages into the communication channels between the controller and its associated sensor or actuator. These messages may mislead the receiver and cause a system security failure. For example, as depicted in Figure 2(a), attacker compromises the channel between the sensor and the controller, and injects a malicious data message that shows a state different from the real state of physical process. Note that the controller is not aware of the communication interruption, thus accepts the injected data and gives the faulty command to the actuator. Actuator performs the unintended action and may damage the physical process.

In the Rebeca model, a separate reactive class is defined to model the attacker's behavior in this scheme. This reactive class includes at least one message server to send malicious message(s), e.g. the sensed data message, to the target channel(s) at an appropriate time. To perform exhaustive security check, a set of Rebeca models is built that contains one or more attacker actors that target different channels at different



(a) ***Scheme-A: Attack on Communication.*** Example: Attacker injects malicious data into the communication channel between a sensor and the controller.

(b) ***Scheme-B: Attack on Components.*** Example: Attacker compromises an actuator.

(c) ***Scheme-C: Combined Attack.*** Example: Two attackers attack in a coordinated way.

Figure 2: Three attack schemes in Rebeca model for security analysis of CPS.

injection times during CPS operation. These Rebeca models are inputs of executing CPS security analysis.

Scheme-B: Attack on Components indicates a situation in which a number of components are compromised and do not function correctly. Attackers may have direct access to the components and perform physical attacks on them. They may damage some sensors/actuators or inject malicious code into the controllers. For example, as Figure 2(b) shows, an attacker may compromise an actuator and perform an action over physical process different from the command issued by the controller. This action of the compromised actuator will effect the physical process state and sensor feedback report.

This scheme is modeled in the Rebeca model as an additional message server inside the reactive class corresponding to the target component. This message server models the incorrect functionality. In the above example, the Rebeca model includes the compromised actuator actor which has a message server sending the malicious message to the physical process actor once receiving a control command from the controller. Similar to Attack Scheme-A, all the possible Rebeca models including one or more compromised components are built and the models are analysed in the model checking step.

Scheme-C: Combined Attack is a combination of the previous two attack schemes in which both the system components and communication channels are compromised by attackers. Usually, this happens when more than one attacker try to attack the system in a coordinated way. Figure 2(c) illustrates a CPS with presence of two attackers in which attacker A compromises actuator to launch an alteration on the physical process, and attacker B injects a false data message into the channel from the sensor to the controller. This coordinated operation of attackers makes an unexpected change on the physical process without the controller awareness. Indeed, the injected data message is sent to the controller falsely showing that the expected action is performed rather than the malicious alteration.

The modeling of this scheme would include various combinations of the defined attackers and compromised components as actors in a Rebeca model. We can choose many kinds of attack scenarios with assumption of compromised network or components in Rebeca model and check the attacks damage on the CPS system.

C. Model Checking and Security Analysis

The security objectives will be the basis for defining the security properties to be verified. Afra supports LTL, TCTL and assertions for property specification. The most important security objectives are *confidentiality*, *integrity* and *availability* presented in Table I and explained in Section IV.

We use RMC to automatically verify each of the specified security properties. If RMC detects that a property is not satisfied by the Rebeca model, it provides the modeler with a counter-example detailing the sequence of events that would lead to a security violation. The sequence of events determines a successful attack. Realising the possible successful attacks can be the basis for applying appropriate countermeasures. In some cases, it may be enough to change the security policies to protect the system against the attacks, and in some cases we may need a security component such as an intrusion detection system (IDS) to keep the system safe against intruders. As our future work, we would incorporate and check these solutions in the model

The common problem in model checking is state-space explosion. A Rebeca model of a CPS in principle has a recurrent bounded behavior. Although we model time, the model checking tool is able to distinguish when a newly generated state is already visited and the only difference is in the logical time stamps. If needed, while running the model checker we can use assertions to stop the process and look into the state space. In any case we can have a bound on the growing time stamps to stop the model checking at a certain time.

IV. Attack Classification

STRIDE[1] is designed as a model for identifying different types of threats that a system may experience and the corresponding security objective which might be violated [20]. In Table I, we classify the significant attacks on CPS (reported in

[1]The acronym STRIDE stands for **S**poofing, **T**ampering, **R**epudiation, **I**nformation Disclosure, **D**enial of Service, and **E**levation of Privilege.

[26], [27], [28]) based on the STRIDE categories. The cyber and physical attacks exploit emerging CPS-related vulnerabilities in the two aspects of *communication* and *component*, and are shown in Table I as *Scheme-A* and *Scheme-B*. *Scheme-A* consists of the attack scenarios which are secretly recording or modifying the data transmitted over the channels (e.g., eavesdropping and MITM). *Scheme-B* includes the attacks that inject malicious code into the software components or perform a malicious alteration on a physical component (e.g., malware and physical attack).

Table I: Attack Classification using STRIDE model.

Threat Type (Security Objective)	*Cyber and Physical Attack*	*Scheme-A*	*Scheme-B*
Spoofing (Authentication)	Masquerade attack		[27]
	Packet spoofing attack	[28]	
Tampering (Integrity)	Man-in-the-middle (MITM)	[27]	
	Injection attack	[28]	[28]
	Replay attack	[27]	
	Malware (Virus or Worms)		[28]
	Physical attack	[28]	[26]
Reputation (Non-Repudiation)	On-Off attack		[26]
Information Disclosure (Confidentiality)	Eavesdropping	[27]	
	Malware (Spyware)		[28]
	Side-channel attack		[28]
	Physical attack	[28]	[26]
Denial of Service (Availability)	Resource exhaustion attack	[27]	[28]
	Interruption attack	[27]	
	Malware (Ransomware)		[28]
	Physical attack	[28]	[26]
Elevation of Privilege (Authorization)	Malware (Rootkit)		[28]

Scheme-A: Attack on Communication typically consists of the followings. In the *Spoofing attack*, the attacker transmits a message with a spoofed identity into the network. *Man-in-the-middle (MITM)* requires the attacker puts herself in between two communicating parties and changes the messages. To launch MITM attack, the attacker impersonates herself as one of targeted parties. *Injection attack* indicates that the attacker injects invalid messages into the network (i.e., packet injection). *Replay attack* is an intentional repetition of sending a message to mislead the receiver. *Eavesdropping attack* takes advantage of unsecured channels to steal the information transmitted over the network. *Side-channel attack* is an attack in which the attacker uses own technical knowledge of the system to compromise the system security. *Resource exhaustion attack* represents a situation that the network resources are overwhelmed by a flood of messages transmitted from the attacker. *Interruption attack* makes a service unavailable for legitimate use, and finally *Physical attack* aims to damage a communication link.

Scheme-B: Attack on Components includes the attacks as follows. *Masquerade attack* refers to a situation in which the attacker impersonates herself as one of the communicating parties. *Injection attack* is used by an attacker to inject a malicious code into a component (i.e., code injection). *Malware* is a malicious software designed to manipulate the behavior of components. *Side-channel attack* is an attack in which the attacker gains knowledge about the system by observing the behavior of some component(s). Finally, *physical attack* manipulates some component(s) physically.

We can model each of the explained attacks using our methodology based on Rebeca. In Section V-B, we show a Rebeca model and explain how some of these attacks can be modeled using a separate attacker actor or an actors that mimic the compromised component.

V. Case Study and Evaluation

In this section, we discuss an experimental study on the SWaT testbed [29]. We first present the SWaT architecture and its security objectives. Then, we provide details on the Rebeca model, and finally, we discuss the security analysis results.

A. SWaT Design

The SWaT testbed is a scaled-down version of an industrial water treatment system. This testbed is used for several research and training purposes in the iTrust research center [29]. The aim of constructing the SWaT testbed is to assess the impact of various attacks on the system and analyze the effectiveness of different mitigation techniques.

The water treatment process in SWaT system consists of three stages as depicted in Figure 3. These stages include supplying raw water into the system, Ultra-Filtration (UF) and Reverse Osmosis (RO). In each stage, there is a PLC responsible for controlling a water tank. The PLC is directly connected to some actuators (i.e., valves or pumps) through a local network. A simple password-based authentication is the only mechanism employed to control access to the network, which makes SWaT system vulnerable to eavesdropping or packet injection attacks [14].

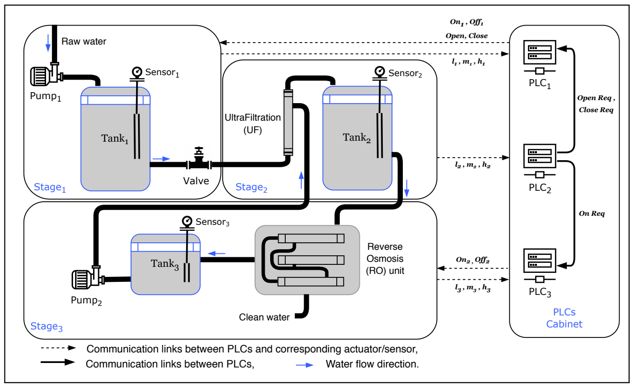


Figure 3: An abstract architecture of the SWaT system

At any stage during the execution of the water treatment process, each pump can be in *On* or *Off* state, and respectively each valve can be in one of the two states *Open* or *Close*. Also, three states are considered for the big tanks (i.e., Tank$_1$ and Tank$_2$): Low(l), Medium(m), and High(h), and two states for the small tank (Tank$_3$): Low(l) and High(h). During the system operation, whenever the water level of a tank changes to h, the associated sensor reports the change to the responsible PLC. That PLC will close the valve or turn off the pump that is pouring water into the tank. Also, the PLC may open a valve, turn a pump on, or send *open*/*on* requests to other PLCs when the water level in the tank is either *l* or *m*.

In the following, we elaborate the control logic of each PLC (see Figure 3): whenever the level of water in Tank_1 is *l*, ***PLC***$_1$ turns on Pump_1 and closes Valve, so that raw water flows into the tank and no water flows out. Valve is opened only if the water level is *h* or *m* and also PLC_1 has received an *open* request from PLC_2. In addition, whenever PLC_1 receives a close request from PLC_2, it will close Valve. ***PLC***$_2$ is in charge of controlling Tank_2, and sends a *close* request to PLC_1 whenever water level in Tank_2 is *h*. Hence, the flow of water from Tank_1 to Tank_2 is stopped. The PLC sends an *open* request to PLC_3 when the level of water is reduced to *m*. If no response is received from PLC_3, the PLC asks PLC_1 to open Valve. It will continue sending requests until one of the other PLCs replies. ***PLC***$_3$ is in charge of controlling Tank_3 by turning off Pump_2 whenever the level of water in Tank_3 is *l*. The purpose is to stop the flow of water to Tank_2 and prevent Tank_3 from underflow. If the water level is *h*, Tank_3 turns Pump_2 on, so that water is depleted to Tank_2 after passing through RO unit.

A dataset collected from SWaT system operation is available in iTrust homepage for research purposes [32]. The dataset includes data about network traffic and sensor and actuator status during normal operation of the system. The dataset indicates that one millimeter increase or decrease in water level of Tank_1 and Tank_2 takes approximately two seconds. The sensors of Tank_1 or Tank_2 report the water level in millimeters. The capacity of Tank_3 is half capacity of Tank_1 and Tank_2, and its sensor reports only low and high levels of water to the corresponding PLC.

B. Security objectives and Threats

We assume that malicious attackers have the ability of injecting arbitrary packets into the communication channels between PLCs and sensors/actuators, and also they are able to alter the functionality of sensors/actuators. Here we use the STRIDE terminology to explain the possible attack scenarios. An attacker may break through the network authentication, disguise himself/herself as an actual system component (*spoofing threat*) and inject a packet into the channel between sensor and PLC (*tampering threat*). The *integrity* objective of the system is jeopardized when an attacker wants to mislead the PLC (*reputation threat*) by sending a packet that contains a value different from the real value of the water tank status. Another attack scenario is possible when an attacker wants to jeopardize the *availability* of the system by sending the same message to a communication channel several times. This repetition causes the channel to be overwhelmed with several packets (*denial of service threat*). It is even possible that the attacker changes the state of an actuator through bypassing the actual commands coming from the PLC.

In this experiment we focus on the integrity of SWaT system following the STRIDE model. In fact, we use model checking to detect the undesirable events that might happen while attackers tamper the channels (e.g., by injecting packets) and compromise sensors/actuators by altering their functionality (e.g., physical attack).

C. SWaT Actor Model

The actor model of the SWaT system is depicted in Figure 4. In this model, each shape represents an actor which corresponds to a component in the SWaT abstract architecture presented in Figure 3. Each arrow models a message passed between two components. In the model, the messages that may be the targets of attackers are distinguished from the secure ones. The red points with numbers from one to six indicate the possible compromised channels where the attackers may inject messages. The channels between PLCs and sensors/actuators can be compromised due to the lack of strong authentication methods and tamper-resistant mechanisms. However, the PLCs communicate with each other through a separate protected network. For example, the *open_Req* or the *on_Req* message passed in the secured channel between the PLCs may not be the target of any attacker. However, the messages (*l, m, and h*) which are transmitted from the sensors to the PLCs indicate the water level and may be tampered by an attacker to affect the decisions made by the PLCs. The blue points represent the components that may behave maliciously. Typically, the malicious behaviour of the component leads to a faulty data transmission. For instance, whenever a pump is compromised, it may transmit message *Turn On* to the connected tank once it receives the command *Turn Off* from the corresponding PLC.

In the SWaT actor model, we assume that the water level in each tank is low in the initial state. Also, the water treatment process begins by pumping raw water to Tank_1 and it ends when the cleaned water flows out of Tank_3. During the process execution, each sensor sends water level information to the corresponding PLC periodically.

Based on the iTrust dataset (see Section V-A) in the SWaT system the sensing period is 1 second, and the water level is changed every 1000 seconds. We use these values for setting the value of parameters (i.e., *sensing_interval* and *operationTimeTank*) in the Rebeca model, and also in the logic of the code.

D. The Rebeca Model of the SWaT System

Here, we provide a detailed explanation of the Rebeca model developed for SWaT system. The complete model is available in [33].

Listing 1 shows an abstract view of the SWaT Rebeca model. The main block includes the declarations of all rebecs defined in the SWaT actor model (see Figure 4) together with an attacker rebec. In each declaration, the first parameter list includes the known rebecs, those which the declared rebec communicates with. For example, the known rebecs of PLC_1 are Valve, Pump_1 and Sensor_1. The second parameter list

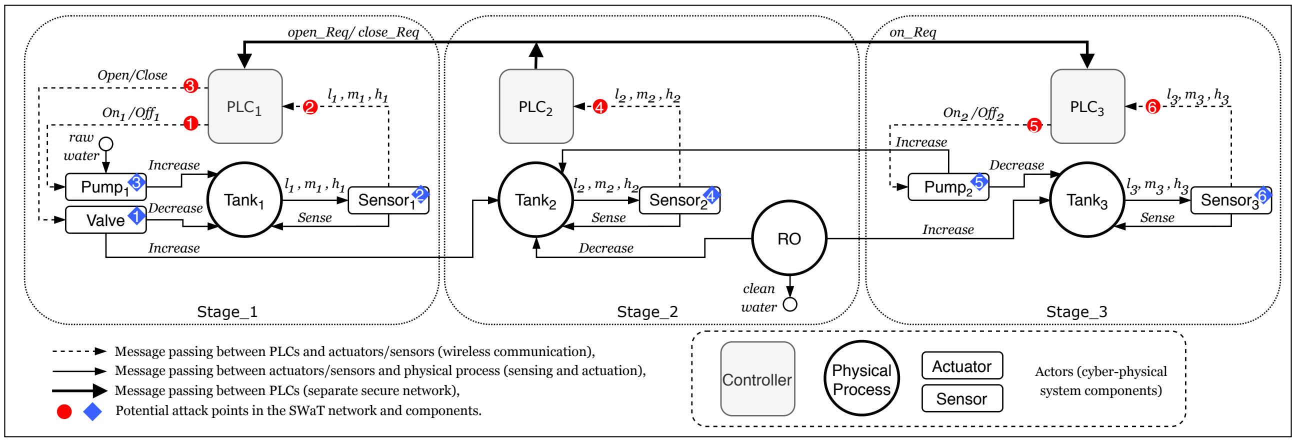

open_Req/ close_Req
on_Req
Open/Close
On1 /Off1
PLC1
PLC2
PLC3
l1 , m1 , h1
l2 , m2 , h2
l3, m3 , h3
On2 /Off2
raw water
Increase
Decrease
Pump1
Valve
Tank1
Sensor1
Sense
Tank2
Sensor2
Tank3
Sensor3
Pump2
RO
clean water
Stage_1
Stage_2
Stage_3
Message passing between PLCs and actuators/sensors (wireless communication),
Message passing between actuators/sensors and physical process (sensing and actuation),
Message passing between PLCs (separate secure network),
Potential attack points in the SWaT network and components.
Controller
Physical Process
Actuator
Sensor
Actors (cyber-physical system components)

Figure 4: SWaT actor model.

includes the parameters to be passed to the constructor of the rebec.

In addition to the main block, the Rebeca model includes the reactive classes defining the behavior of the SWaT actors. For example, the PLC$_1$ reactive class has three known rebecs which are instances of reactive classes Pump$_1$, Valve and Sensor$_1$. The PLC$_1$ reactive class includes a Boolean state variable *openReqPlc2* whose value indicates whether a water request is received from PLC$_2$ or not. This variable is initialized to *false* in the constructor of PLC$_1$. Two Boolean state variables *pump1On* and *valveOpen* indicate the current status of Pump$_1$ and Valve respectively. The definition of PLC$_1$ includes three message servers i.e., *processSensorData*, *openReq* and *closeReq*. The message server *processSensorData* processes the sensor data and issues commands *on* or *off* to Pump$_1$ and *open* or *close* to Valve accordingly. The message servers *openReq* and *closeReq* are activated once a message is received from PLC$_2$.

The reactive class Pump$_1$ includes four massage servers *on*, *off*, *KeepOnpumping* and *maliciousAct*. The message servers *on* and *off* update the value of the state variable *On* based on the commands received from PLC$_1$. The message server *KeepOnpumping* calls *increaseWater* which takes *operationTimeTank* units of time and increases the level of water for one level in the tank. This continues until the message server *off* receives the turn off message. Due to space limitations, we exclude the explanation of other reactive classes from this paper. Interested readers may refer to [33] for more details.

E. Attack Models in Rebeca

In the Rebeca code, we model compromised actors (Scheme-B Attacks) using two parameters that are passed to all the actors that can be compromised (see Listing 1). The first parameter sets the status of the actor, and the second parameter sets the time of the attack. For example, the reactive class of Pump$_1$ includes a variable *maliciousAction* that can be set to change the status of the component to be compromised or not compromised. If this variable is set to be compromised then although the pump receives a message to turn its status to *on*, it turns it to *off*. For changing the variable *maliciousAction* at different times in each run of the model, a message is sent to Pump$_1$ at a certain model time. This model time can be configured and is passed to the pump as a parameter. Similar to the compromised mode of Pump$_1$, whenever the value of the input parameter *compromised* is true for Valve, then both message servers *open* and *close* behave maliciously (for example the message server *open* changes the value of state variable *Open* to *false*). The message server *maliciousAct* corresponding to each sensor activates compromised mode for the sensor, which causes the sensor to report invalid water level to PLC$_1$.

In addition to the reactive classes that define the normal and compromised behavior of SWaT components, the Rebeca model includes a reactive class named *Attacker* that models the behaviour of potential attackers targeting channels to inject messages (Scheme-A Attacks).

As we assume that attackers may target the communication channels between any two components in SWaT system, the *knownrebecs* section of reactive class *Attacker* includes all the other rebecs defined in the Rebeca model. The constructor of this class has three arguments representing the target channel, malicious message content, and attack time. Since there are six channels in the system, the value of the first argument would be a number between 1 and 6. Based on the value passed to this argument, the message server responsible for sending malicious messages to the corresponding channel is invoked by the constructor. Message content is another numeric argument whose value indicates either the water level in a tank, an *on/off* command for Pump, or an *open/close* command for Valve. Finally, the third argument represents the time during the system operation that the malicious message is sent to a channel.

F. Properties of Interest

The goal of attacks on the SWaT system is to cause an overflow or underflow in one of the tanks. An overflow may harm some of the critical units such as the UF or RO and lead to flow out unclean water, and an underflow may damage a valve or a pump. Accordingly, the properties presented in Figure 5 are considered to be verified on the Rebeca model of SWaT system.

```
property {
    define {
        t1_overFlow = tank1.overFlow;
        t1_underFlow = tank1.underFlow;
        t2_overFlow = tank2.overFlow;
        t2_underFlow = tank2.underFlow;
        t3_overFlow = tank3.overFlow;
        t3_underFlow = tank3.underFlow;}
    Assertion{
        safety_tank1_over: !(t1_overFlow);
        safety_tank1_under: !(t1_underFlow);
        safety_tank2_over: !(t2_overFlow);
        safety_tank2_under: !(t2_underFlow);
        safety_tank3_over: !(t3_overFlow);
        safety_tank3_under: !(t3_underFlow);}
}
```

Figure 5: Security properties considered in the SWaT system.

G. Model Checking and Abstract States

Figure 6 represents an abstract view of the state transition diagram of the SWaT system during a normal operation. The diagram is derived manually from the state space generated automatically by Afra. Each state shows the water level in the three tanks and the status of the pumps and the valve. Each transition between two states indicates an increase or/and decrease of the water level of some tank(s). Whenever a *waterIncrease* or *waterDecrease* occurs in a tank, then the attached sensor informs the corresponding PLC to update the status of the pumps and the valve based on the sensed data. Each state in Figure 6 represents a set of states and transitions in the state space generated by model checking. In each of these abstract states the total amount of progress of time in the including transitions is shown. The state space generated through model checking by Afra includes 42k states and 53k transitions.

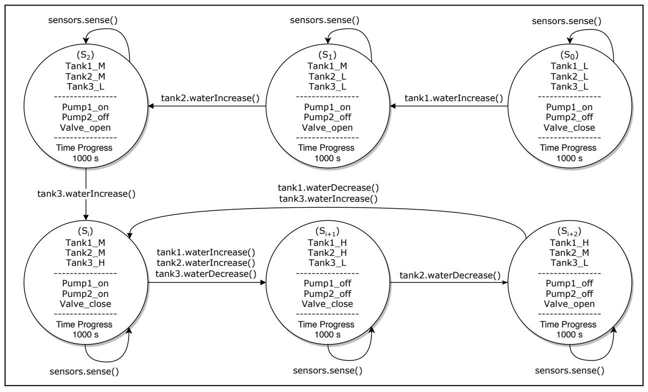


Figure 6: The abstracted state transition of SWaT system.

H. Model Checking and Security Analysis

In order to analyze the security properties of the SWaT system using the developed Rebeca model, we follow three attack schemes presented in Section III-B. The outcome of the analysis includes the attack scenarios which lead the system to security violation. To cover all possible attack scenarios by model checking, we need to generate all combinations of different values for the input parameters of the attacker and the compromised components, and verify the model for each combination. A Python script is developed to automate input value generation and accumulation of the verification results. This approach is similar in its nature to the automated verification technique using symbolic modeling and constraint solving in [34]. Here we use an algorithmic approach to enumerate all the possible attack scenarios. In total we modeled 105 communication attacks and 84 attacks on components, and also the combination of these attacks (resulting in 8820 attack scenarios). Totally, out of all above possible attack scenarios 29 cases successfully violate the system security which we report in Tables II, III and IV.

Table II presents the outcomes of the analysis process for *Attack on Communication (Scheme-A)*. The results indicate at which system state the injected message has caused security violation. For example, assume that the system is in state S_0 (see the state transition diagram in Figure 6), and the attacker injects a malicious message into the channel between $Sensor_1$ and PLC_1 (see channels in Figure 4). This message wrongly reports the level of water in $Tank_1$ as being High. $Tank_1$ will underflow afterwards, because *Turn off Pump1* and *Open Valve* are issued by PLC_1 after receiving the message (line 5 in Table II).

Table II: Model checking results in Attack on Communication (Scheme-A).

#	*Tank*	*Property*	*Injected Message*	*Communication Channel*	*System State*
1	$Tank_1$	Overflow	Water level in $Tank_1$ is low	$Sensor_1$ to PLC_1	S_{i+1}
2	$Tank_1$	Overflow	Turn on $Pump_1$	PLC_1 to $Pump_1$	S_{i+1}
3	$Tank_1$	Overflow	Water level in $Tank_1$ is low	$Sensor_1$ to PLC_1	S_{i+2}
4	$Tank_1$	Overflow	Turn on $Pump_1$	PLC_1 to $Pump_1$	S_{i+2}
5	$Tank_1$	Underflow	Water level in $Tank_1$ is high	$Sensor_1$ to PLC_1	S_0
6	$Tank_2$	Overflow	Water level in $Tank_2$ is medium	$Sensor_2$ to PLC_2	S_{i+1}
7	$Tank_2$	Overflow	Open Valve	PLC_1 to Valve	S_{i+1}
8	$Tank_3$	Overflow	Water level in $Tank_3$ is high	$Sensor_3$ to PLC_3	S_i
9	$Tank_3$	Overflow	Open Valve	PLC_1 to Valve	S_i
10	$Tank_3$	Underflow	Turn on $Pump_2$	PLC_3 to $Pump_2$	S_0
11	$Tank_3$	Underflow	Turn on $Pump_2$	PLC_3 to $Pump_2$	S_1
12	$Tank_3$	Underflow	Water level in $Tank_3$ is high	$Sensor_3$ to PLC_3	S_2
13	$Tank_3$	Underflow	Turn on $Pump_2$	PLC_3 to $Pump_2$	S_2
14	$Tank_3$	Underflow	Water level in $Tank_3$ is high	$Sensor_3$ to PLC_3	S_{i+2}
15	$Tank_3$	Underflow	Turn on $Pump_2$	PLC_3 to $Pump_2$	S_{i+2}

Table III shows the results of model checking on the Rebeca model for *Attack on Components (Scheme-B)*. These results indicate at which system state the compromised component causes security violation. For example, assume that the system is in state S_{i+1} and $Sensor_2$ is compromised. This sensor sends a wrong report about the water level of $Tank_2$ to PLC_2. This report indicates the level of water as being Medium, whereas the real level is High. Upon receiving this report, PLC_2 opens Valve and causes $Tank_2$ to overflow (line 5 in Table III).

The analysis results in Table IV indicate that by using the modeling method presented in *Combined Attack (Scheme-C)*, such collaborative attack can be easily detected. For example assume that the system is in state S_0 and an attacker injects message *Open Valve* into the communication link between PLC_1 and Valve, and at the same time another attacker compromises $Pump_1$ to be turned off, then $Tank_1$ will underflow

Table III: Model checking results in Attack on Components (Scheme-B).

#	*Tank*	*Property*	*Compromised Component*	*Malicious Behaviour*	*System State*
1	Tank_1	Overflow	Sensor_1	Water level in Tank_1 is low	S_{i+1}
2	Tank_1	Overflow	Pump_1	Turn on	S_{i+1}
3	Tank_1	Overflow	Sensor_1	Water level in Tank_1 is low	S_{i+2}
4	Tank_1	Underflow	Sensor_1	Water level in Tank_1 is high	S_0
5	Tank_2	Overflow	Sensor_2	Water level in Tank_2 is medium	S_{i+1}
6	Tank_3	Overflow	Sensor_2	Water level in Tank_2 is low	S_i
7	Tank_3	Overflow	Valve	Open	S_i
8	Tank_3	Underflow	Pump_2	Turn on	S_1
9	Tank_3	Underflow	Sensor_3	Water level in Tank_3 is high	S_2
10	Tank_3	Underflow	Pump_2	Turn on	S_{i+1}
11	Tank_3	Underflow	Sensor_3	Water level in Tank_3 is high	S_{i+2}

(line 1 in Table IV). As another example, if the system is in state S_1, Sensor_2 is compromised and a malicious message of high water level for Tank_3 is injected into the channel between Sensor_3 and PLC_3, then Tank_3 will underflow (line 3 in Table IV). Note that the scenarios presented in Table IV are those in which the single attacks (message injection or the compromised component) do not cause a security failure separately, but the combination leads to the security violation. If we assume that the system is robust against the scenarios in Table II and Table III, the system may still be vulnerable against the collaborative attacks in Table IV.

VI. Related Work

Several modeling and simulation methods have been proposed for analyzing the security of CPSs. In this section, we review the ones most related to the method presented in this paper.

Wasicek et al. [35] propose an aspect-oriented technique to model attacks against CPSs. They use Ptolemy [36] as the modeling and simulation framework, and demonstrate the practicality of their technique through modeling four types of attacks on an automotive control system. They also illustrate how Ptolemy [36] can be used to simulate the behavior of system components and detect anomalies.

Taormina et al. [15] propose another simulation-based approach that is implemented in a MATLAB toolbox to analyze the risk of cyber-physical attacks on water distribution systems. In this approach, they analyze the hydraulic response of water networks to several attacks implemented in a MATLAB toolbox. The implemented attacks are extracted from a graphical attack model which includes the cyber and physical elements that may be the targets of attack together with the types of attacks they might be subject to. Using this approach we can analyze the security of water networks toward several kinds of attacks. In [1], [16], the authors rely on simulation to perform their analyses. They propose a new metric to quantify the impact of attacks on components of the target CPS. This metric can be used to perform cost-benefit analysis on security investments.

Furthermore, there are several formal methods examine CPS security. In [14], Kang et al. use Alloy to model SWaT behavior and potential attackers. They can discover the undetected attacks which cause safety failure (e.g., water tank overflow). The study is considered as run-time monitoring, where compares actual invariant of the SWaT system and output state in the Alloy model checker during system operation. Although some important attack scenarios are identified using this approach, each run of the analysis considers only one point of the system to attack. Using the actor-based method presented in this paper, we are able to detect scenarios in which several attackers attack to different components and communication channels at various times of system operation. Furthermore, defining attacker actors in the Rebeca model helps us to detect sabotages in the system by analyzing the behavior of various kinds of attack scenarios that exploit communication and components vulnerabilities.

Rocchetto and Tippenhauer [37] present another formal method for discovering feasible attack scenarios on SWaT. ASLan++ is the formal language used for modeling the physical layer interactions and CL-AtSe is a tool used to analyze the state space of the model and discover the potential attack scenarios. As the result, they succeed to find eight attack scenarios. A distinctive feature of this method is providing support for modeling different attacker profiles. However, only one profile can be active at each moment, whereas Rebeca allows us to have multiple profiles active simultaneously.

Fritz and Zhang [38] consider CPSs as discrete-event systems and model them using a variant of Petri nets. They propose a method based on permutation matrices to detect deception attacks. In particular, they can detect attacks by changing the input and output behavior of the system and analyzing its effect on the system behavior. Covert attacks and replay attacks are two kinds of attacks modeled and analyzed in this study. However, the combinations of attacks are not considered.

VII. Conclusion and Future Work

In this paper, we present an approach to model and analyze the security properties of CPS using formal methods. We define three attack schemes targeting communication channels, components, and the combination of each, and then verify if the attacks could compromise the system security. In this approach, we use an actor-based modeling language Rebeca. The language facilitates modeling and analysis of the normal system behavior as well as the malicious behavior of potential attackers.

We present a case study on a Secure Water Treatment (SWaT) System. This case study shows how each component in a cyber-physical system can be directly mapped to an actor in a Rebeca model. We demonstrate how the Afra model checking tool makes it possible to discover various potential attack scenarios. The presented approach enables the evaluation of the attack scenarios in a practical case study where some of the scenarios were not easily manually analyzable.

As future work, we intend to extend the application of our method to security analysis during run-time system operation and also analyze mitigation strategies together with attack scenarios.

Table IV: Model checking results in Combined Attack (Scheme-C).

#	*Tank*	*Property*	*Injected Message*	*Communication Channel*	*Compromised Component*	*Malicious Behaviour*	*System State*
1	$Tank_1$	Underflow	Open Valve	PLC_1 to Valve	$Pump_1$	Turn Off	S_0
2	$Tank_3$	Underflow	Water level in $Tank_2$ is medium	$Sensor_2$ to PLC_2	$Sensor_3$	Water level in $Tank_3$ is high	S_0
3	$Tank_3$	Underflow	Water level in $Tank_3$ is high	$Sensor_3$ to PLC_3	$Sensor_2$	Water level in $Tank_2$ is medium	S_1

Acknowledgment

This research is supported by Swedish Foundation for Strategic Research (SSF) via the Serendipity project.

References

[1] R. Lanotte, M. Merro, R. Muradore, and L. Viganò, "A formal approach to cyber-physical attacks," in *IEEE 30th Computer Security Foundations Symposium (CSF)*, pp. 436–450, IEEE, 2017.

[2] S. Adepu, A. Mathur, J. Gunda, and S. Djokic, "An agent-based framework for simulating and analysing attacks on cyber physical systems," in *Algorithms and Architectures for Parallel Processing*, pp. 785–798, Springer, 2015.

[3] "The industrial control systems cyber emergency response team." https://www.us-cert.gov/ics. [Online; accessed April 23, 2020].

[4] D. B. Rawat, C. Bajracharya, and G. Yan, "Towards intelligent transportation cyber-physical systems: Real-time computing and communications perspectives," in *SoutheastCon 2015*, pp. 1–6, IEEE, 2015.

[5] Y. Zhang, M. Qiu, C.-W. Tsai, M. M. Hassan, and A. Alamri, "Health-cps: Healthcare cyber-physical system assisted by cloud and big data," *IEEE Systems Journal*, vol. 11, no. 1, pp. 88–95, 2015.

[6] C. Lv, X. Hu, A. Sangiovanni-Vincentelli, Y. Li, C. M. Martinez, and D. Cao, "Driving-style-based codesign optimization of an automated electric vehicle: a cyber-physical system approach," *IEEE Transactions on Industrial Electronics*, vol. 66, no. 4, pp. 2965–2975, 2018.

[7] J. Malchow and J. Klick, "Sicherheit in vernetzten systemen: 21. dfn-ws," 2014.

[8] E. Byres, "Project shine: 1,000,000 internet-connected scada and ics systems and counting," *Tofino Security*, 2013.

[9] "Threat landscape for industrial automation systems." https://ics-cert.kaspersky.com/reports. [Online; accessed December 12, 2019].

[10] P. Bell, "Cyber threat report may 06, 2019," 2019.

[11] E. Kovacs, "Dos attack blamed for u.s. grid disruptions," 2019.

[12] W. Stallings, L. Brown, M. D. Bauer, and A. K. Bhattacharjee, *Computer security: principles and practice*. Pearson Education, 2012.

[13] D. Gollmann, P. Gurikov, A. Isakov, M. Krotofil, J. Larsen, and A. Winnicki, "Cyber-physical systems security: Experimental analysis of a vinyl acetate monomer plant," in *Proceedings of Cyber-Physical System Security*, pp. 1–12, ACM, 2015.

[14] E. Kang, S. Adepu, D. Jackson, and A. P. Mathur, "Model-based security analysis of a water treatment system," in *Proceedings of Software Engineering for Smart Cyber-Physical Systems*, pp. 22–28, ACM, 2016.

[15] R. Taormina, S. Galelli, N. O. Tippenhauer, E. Salomons, and A. Ostfeld, "Characterizing cyber-physical attacks on water distribution systems," *Journal of Water Resources Planning and Management*, 2017.

[16] R. Lanotte, M. Merro, A. Munteanu, and L. Viganò, "A formal approach to physics-based attacks in cyber-physical systems," *ACM Transactions on Privacy and Security (TOPS)*, vol. 23, no. 1, pp. 1–41, 2020.

[17] A. H. Reynisson, M. Sirjani, L. Aceto, M. Cimini, A. Jafari, A. Ingólfsdóttir, and S. H. Sigurdarson, "Modelling and simulation of asynchronous real-time systems using timed rebeca," *Sci. Comput. Program.*, vol. 89, pp. 41–68, 2014.

[18] M. Sirjani and E. Khamespanah, "On time actors," in *Theory and Practice of Formal Methods*, pp. 373–392, Springer, 2016.

[19] E. Khamespanah, M. Sirjani, Z. Sabahi-Kaviani, R. Khosravi, and M. Izadi, "Timed rebeca schedulability and deadlock freedom analysis using bounded floating time transition system," *Sci. Comput. Program.*, vol. 98, pp. 184–204, 2015.

[20] A. Shostack, *Threat modeling: Designing for security*. Wiley, 2014.

[21] M. Sirjani, A. Movaghar, A. Shali, and F. S. De Boer, "Modeling and verification of reactive systems using rebeca," *Fundamenta Informaticae*, vol. 63, no. 4, pp. 385–410, 2004.

[22] M. Sirjani, "Rebeca: theory, applications, and tools," in *Formal Methods for Components and Objects FMCO 2006*, pp. 102–126, 2006.

[23] M. Sirjani and M. M. Jaghoori, "Ten years of analyzing actors: Rebeca experience," in *Formal Modeling: Actors, Open Systems, Biological Systems - Essays*, pp. 20–56, 2011.

[24] "Afra: an integrated environment for modeling and verifying rebeca family designs." https://rebeca-lang.org/alltools/Afra, 2019. [Online; accessed November 09, 2019].

[25] M. Sirjani, E. Khamespanah, and E. Lee, "Model checking software in cyberphysical systems," in *COMPSAC 2020*, 2020.

[26] J. Giraldo, D. Urbina, A. Cardenas, J. Valente, M. Faisal, J. Ruths, N. O. Tippenhauer, H. Sandberg, and R. Candell, "A survey of physics-based attack detection in cyber-physical systems," *ACM Computing Surveys (CSUR)*, vol. 51, no. 4, pp. 1–36, 2018.

[27] S. Choi, J.-H. Yun, and S.-K. Kim, "A comparison of ics datasets for security research based on attack paths," in *International Conference on Critical Information Infrastructures Security*, Springer, 2018.

[28] J.-M. Flaus, *Cybersecurity of industrial systems*. J. Wiley & Sons, 2019.

[29] A. P. Mathur and N. O. Tippenhauer, "Swat: a water treatment testbed for research and training on ics security," in *Cyber-physical Systems for Smart Water Networks (CySWater)*, pp. 31–36, IEEE, 2016.

[30] M. Sirjani, "Power is overrated, go for friendliness! expressiveness, faithfulness, and usability in modeling: the actor experience," in *Principles of modeling - essays dedicated to Edward A. Lee*, pp. 423–448, 2018.

[31] "Rebeca homepage." http://rebeca-lang.org/Rebeca, 2019. [Online; accessed June 03, 2019].

[32] iTrust, "Secure water treatment (swat) dataset." https://itrust.sutd.edu.sg/itrust-labs_datasets/dataset_info/, 2019. [Accessed September 17, 2019].

[33] "Rebeca homepage." http://rebeca-lang.org/allprojects/CRYSTAL, 2020.

[34] J. R. Burch, E. M. Clarke, D. E. Long, K. L. McMillan, and D. L. Dill, "Symbolic model checking for sequential circuit verification," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 13, no. 4, pp. 401–424, 1994.

[35] A. Wasicek, P. Derler, and E. A. Lee, "Aspect-oriented modeling of attacks in automotive cyber-physical systems," in *ACM/EDAC/IEEE Design Automation Conference (DAC)*, 2014.

[36] J. Buck, S. Ha, E. A. Lee, and D. G. Messerschmitt, "Ptolemy: A framework for simulating and prototyping heterogeneous systems," in *Readings in hardware/software co-design*, pp. 527–543, 2001.

[37] M. Rocchetto and N. O. Tippenhauer, "Towards formal security analysis of industrial control systems," in *ACM Asia Conference on Computer and Communications Security*, pp. 114–126, ACM, 2017.

[38] R. Fritz and P. Zhang, "Modeling and detection of cyber attacks on discrete event systems," *IFAC-PapersOnLine*, vol. 51, no. 7, pp. 285–290, 2018.

```
env boolean p1Compromised = false; env int p1Compromised_time = 0;
env boolean s1Compromised = false; env int s1Compromised_time = 0;
env boolean vCompromised = false; env int vCompromised_time = 0;
env boolean s2Compromised = false; env int s2Compromised_time = 0;
env boolean p2Compromised = false; env int p2Compromised_time = 0;
env boolean s3Compromised = false; env int s3Compromised_time = 0;
env int chl = 1;
env int malMsg = 0;
env int attackTime = 0;
env int sensing_interval = 1;
env int operationTimeTank = 1000;
//............................................
reactiveclass PLC1(5){
    knownrebecs{
        Pump1 pump1;
        Valve valve;
        sensorTank1 sensor1;}
    statevars{
        boolean openReqPlc2, pump1On, valveOpen;
        int waterLevelTank1;}
    PLC1(){
        openReqPlc2 = false;
        waterLevelTank1 = 0;
        pump1On = false;
        valveOpen = false;}
    msgsrv processSensorData(int waterLevel){
        if (waterLevel == 1){
            if (waterLevelTank1 != waterLevel){
                pump1.on(); pump1On = true;}
        } else if (waterLevel == 2 && openReqPlc2 == true
            && pump1On == true && valveOpen == false){
                 if (waterLevelTank1 != waterLevel){
                     openReqPlc2 = false; valve.open(); valveOpen = true;}
        } else {...}
        waterLevelTank1 = waterLevel;}
    msgsrv openReq(){ openReqPlc2 = true;}
    msgsrv closeReq(){ valve.close();}
}
reactiveclass PLC2(5){...}
reactiveclass PLC3(5){...}
//............................................
reactiveclass Tank1(10){
    knownrebecs{
        sensorTank1 sensor;}
    statevars{
        boolean underFlow,low,medium,high,overFlow;
        int status;}
    Tank1(){
        underFlow = false; overFlow = false;
        low = true; medium = false; high = false;}
    msgsrv status(){
        if (underFlow){sensor.reportStatus(0);
        } else if (low){sensor.reportStatus(1);
        } else {...}}
    msgsrv waterIncrease(){
        delay(operationTimeTank);
        ... //changes water level status
        if (low == true) {low = false; medium = true; high = false;
        } else if (medium == true) {low = false; medium = false; high = true;
        } else if (high == true) {
        overFlow = true; low = false; medium = false; high = false;}}
    msgsrv waterDecrease(){...}
}
reactiveclass Tank2(10){...}
reactiveclass Tank3(10){...}
//............................................
reactiveclass Pump1(10){
    knownrebecs{
        Tank1 tank1;}
    statevars{
        boolean On, maliciousAction;}
    Pump1(boolean compromised, int compTime){
        on = false;
        maliciousAction = false;
        if (compromised == true) { self.maliciousAct() after(compTime);}}
    msgsrv on(){
        if(maliciousAction == true) { on = false; maliciousAction = false;
        } else if (on == true) { //do nothing
        } else { on = true; tank1.waterIncrease();
            self.KeepOnpumping() after(operationTimeTank);}}
    msgsrv KeepOnpumping(){
        if (on == true) {
            tank1.waterIncrease();
            self.KeepOnpumping() after(operationTimeTank);}}
    msgsrv off(){
        if(maliciousAction == true) { on = true; tank1.waterIncrease();
            self.KeepOnpumping() after(operationTimeTank);
            maliciousAction = false;
        } else {on = false;}}
    msgsrv maliciousAct(){ maliciousAction = true;}
}
reactiveclass Pump2(10){...}
reactiveclass Valve(10){...}
//............................................
reactiveclass SensorTank1(10){
    knownrebecs{
        Tank1 tank1;
        PLC1 plc1;}
    statevars{
        boolean underFlow,overFlow;
        boolean low,medium,high;
        boolean maliciousAction;
        int sensing_interval;}
    SensorTank1(boolean compromised, int compTime){
        ... // initialize boolean variables with false
        sensing_interval = 1; maliciousAction = false;
        if (compromised == true) {
            self.maliciousAct() after(compTime);}
        self.sense();}
    msgsrv sense(){tank1.status();}
    msgsrv reportStatus(int waterLevel) {
        if (waterLevel == 1) {
            if (maliciousAction == true) {
                plc1.processSensorData(2);
                ... //other boolean variables are false.
                medium = true;
            } else {
            low = true;
            ... //other boolean variables are false.
            plc1.processSensorData(1);}
        }else if (waterLevel == 1) {
            low = true;
            plc1.processSensorData(1);}
        else {
        ... //sends sensed data to PLC1.
        tank1.status() after(sensing_interval);
    }
    msgsrv maliciousAct(){maliciousAction = true;}
}
reactiveclass SensorTank2(10){...}
reactiveclass SensorTank3(10){...}
//............................................
reactiveclass reverseOsmosisUnit(5){
    knownrebecs{
        Tank2 tank2;
        Tank3 tank3;}
    statevars { boolean drinkableWater;}
    reverseOsmosisUnit(){ drinkableWater = false;}
    msgsrv waterIncreaseTank3(){
        tank2.waterDecrease(); tank3.waterIncrease();}
    msgsrv cleanWater(){
        tank2.waterDecrease(); drinkableWater = true;}
}
//............................................
reactiveclass Attacker(3){
    knownrebecs{
        PLC1 plc1;
        PLC2 plc2;
        PLC3 plc3;
        Pump1 pump1;
        Pump2 pump2;
        Valve valve;}
    Attacker(int chl, int maliciousMsg, int attackTime){
        if (chl == 1) { self.channelPlc1P1(maliciousMsg, attackTime);
        } else if (chl == 2) {self.channelPlc1S(maliciousMsg, attackTime);
        } else {...}}
    msgsrv channelPlc1P1(int msg, int attackTime){
        if(msg == 1) { pump1.on() after(attackTime);
        } else if(msg == 0) { pump1.off() after(attackTime);}}
    msgsrv channelPlc1S(int msg, int attackTime){
        plc1.processSensorData(msg) after(attackTime);}
    ... //message servers
}
main{
    PLC1 plc1(pump1,valve,sensor1):();
    PLC2 plc2(plc1,plc3,sensor2):();
    PLC3 plc3(pump2,tank3,sensor3):();
    Tank1 tank1(sensor1):();
    Tank2 tank2(sensor2,unit):();
    Tank3 tank3(sensor3,tank2):();
    sensorTank1 sensor1(tank1,plc1):(s1Compromised,s1Compromised_time);
    sensorTank2 sensor2(tank2,plc2):(s2Compromised,s2Compromised_time);
    sensorTank3 sensor3(tank3,plc3):(s3Compromised,s3Compromised_time);
    Pump1 pump1(tank1):(p1Compromised,p1Compromised_time);
    Pump2 pump2(tank2,tank3):(p2Compromised,p2Compromised_time);
    Valve valve(tank1,tank2):(vCompromised,vCompromised_time);
    reverseOsmosisUnit unit(tank2,tank3):();
    Attacker attacker(plc1,plc2,plc3,pump1,pump2,
        valve):(chl,malMsg,attackTime);
}
```

Listing 1: An abstract version of the SWaT system Rebeca model.