2024 WODES Workshop on Discrete Event Systems Rio de Janeiro, Brazil

17th IFAC Workshop on Discrete Event Systems
April 29-May 1, 2024. Rio de Janeiro, Brazil

Persistent Attacks on the Supervisory Control Layer of DES ⋆

Michel R. C. Alves * **Karen Rudie** ** **Patrícia N. Pena** ***

* *Embraer, Av. Brg. Faria Lima, 2170 - Putim, São José dos Campos - SP, 12227-901, Brazil. Email: michel.alves@embraer.com.br*
** *Department of Electrical and Computer Engineering, and Ingenuity Labs Research Institute, Queen's University, Kingston, Ontario K7L 3N6, Canada. Email: karen.rudie@queensu.ca*
*** *Department of Electronics Engineering, Universidade Federal de Minas Gerais, Belo Horizonte, Brazil. Email: ppena@ufmg.br.*

Abstract: This work is set in the context of systems modeled as discrete-event systems. The system is composed of several subsystems whose behavior is coordinated by the action of a controller, which, in turn, is a structure composed, among other elements, by supervisors. It is considered that the controller and each of the subsystems are connected to a communication network and they are subject to the action of a malicious agent that is able to modify the messages sent and received. A new attack model is presented, which considers the different types of actions an attacker is able to perform in a real system. Additionally, a new class of attackers is introduced, called persistent attackers. Moreover, a new design technique for this class of attackers is presented, where the goal of the attacker is to perform the attack while remaining stealthy.

Keywords: Cyber-attacks, discrete-event systems, persistent attackers, supervisory control theory

1. INTRODUCTION

Attacks on industrial control systems predominantly fall under two categories: deception attacks and DoS (Denial of Service) attacks, jeopardizing data integrity and availability, respectively (Yuan et al., 2019). Commonly, known vulnerabilities in device-to-network interfaces serve as prime targets for malicious entities executing deception attacks (Pan et al., 2020). Rashidinejad et al. (2019) highlight that security approaches in DES are typically categorized based on three key aspects: i) attack location; ii) impact on transmitted data; and iii) security mechanisms.

Concerning the attack location, DES-related approaches often focus on attacks occurring in the communication channels from the plant to the controller, from the controller to the plant, or in both directions (Meira-Góes et al., 2020; Zhang et al., 2022). This differs from the portrayal of attack locations in cybersecurity literature (Shareef, 2022; Prinsloo et al., 2019), where practical communication channels don't draw such clear boundaries.

This work proposes an attack model in which the attacker infects the network interface of devices, allowing it to have full control of the communication. Additionally, we explore this attack model and present an attacker called a *persistent attacker*, which has the goal to cause the production of off-specification products and/or cause small delays in the production process, while remaining undetected. Furthermore, we propose a method for the design of persistent attackers.

This paper is organized as follows. Section 2 presents some basic preliminaries concepts about languages and automata. In Section 3 we present the attack model and explain how the attacker's actions are represented. Section 4 introduces the design method for persistent attackers and Section 5 presents some discussion and concludes the work.

2. PRELIMINARIES

In this section we introduce some basic notation that will be employed in this work. Let Σ be a finite set of events. Behaviors of DES are modeled using strings of events of Σ. The set Σ^* is composed of all finite strings of events in Σ, including the empty string ε. The concatenation of strings $s, u \in \Sigma^*$ is written as su. Any subset $L \subseteq \Sigma^*$ is called a language. In this paper, finite state automata are used to recognize languages.

Definition 1. (Finite-State automaton). A finite-state automaton is a quintuple $G = (Q, \Sigma, \delta, q_0, Q_m)$ where Q is a finite set of states, Σ is a finite set of events, $\delta : Q \times \Sigma \to Q$ is a partially defined transition function, $q_0 \in Q$ is the initial state, and $Q_m \subseteq Q$ is a finite set of marked states.

The notation $\delta(q, \sigma)!$ represents that $\delta(q, \sigma)$ is defined for some $q \in Q$ and $\sigma \in \Sigma$. The transition function δ can be extended to a function $Q \times \Sigma^* \to Q$ according to $\delta(q, \varepsilon) := q$ and $\delta(q, s\sigma) := \delta(\delta(q, s), \sigma)$, with $q \in Q$,

⋆ This work has been supported by the National Council for Scientific and Technological Development - CNPq under grant 443656/2018-5, Brazil, and the Natural Sciences and Engineering Research Council of Canada - NSERC.

$s \in \Sigma^*$ and $\sigma \in \Sigma$. With abuse of notation, we sometimes treat δ as a set Δ and $(q, \sigma, q') \in \Delta$ if and only if $\delta(q, \sigma) = q'$. The automaton is said to be deterministic if $(q, \sigma, q'), (q, \sigma, q'') \in \Delta$ always implies that $q' = q''$. The language $\mathcal{L}(G)$, defined as $\mathcal{L}(G) := \{s \in \Sigma^* \mid \delta(q_0, s)!\}$ is the generated language. The marked language $\mathcal{L}_m(G)$ is defined as $\mathcal{L}_m(G) := \{s \in \Sigma^* \mid \delta(q_0, s) \subseteq Q_m\}$. The natural projection $P : \Sigma^* \to \Sigma_i^*$ is an operation over languages that maps strings in Σ^* to strings in Σ_i^*, $\Sigma_i \subseteq \Sigma$. This operation is defined for strings as $P(\varepsilon) = \varepsilon$ and $P(s\sigma) = P(s)$ if $s \in \Sigma^*$, $\sigma \notin \Sigma_i$ and $P(s\sigma) = P(s)\sigma$ if $s \in \Sigma^*$, $\sigma \in \Sigma_i$. The concept is extended to languages by defining $P(L) := \{t \in \Sigma_i^* | t = P(s) \text{ for some } s \in L\}$. To simplify the notation, consider that $\mathcal{L}_G$ represents the generated language of an automaton G, i.e., $\mathcal{L}_G = \mathcal{L}(G)$.

A state $q \in Q$ of an automaton G is accessible if $\delta(q_0, s) = q$, for some $s \in \mathcal{L}(G)$, and coaccessible if $\delta(q, s) \in Q_m$ for some $s \in \Sigma^*$. The accessible, $Ac(G)$, and coaccessible, $CoAc(G)$, components of an automaton G are obtained by eliminating, respectively, the non-accessible and non-coaccessible states and associated transitions. The $Trim$ operation is defined as $Trim(G) := CoAc[Ac(G)] = Ac[CoAc(G)]$. We use the symbol || to denote the parallel composition, as defined in (Cassandras and Lafortune, 2021).

In this work, we adopt the framework of supervisory control proposed by Ramadge and Wonham (1989). We use the notation $Sup\mathcal{C}(K, G)$ to denote the supremal controllable sublanguage of K with respect to $\mathcal{L}(G)$.

3. ATTACK MODEL

3.1 System Setup

We consider a system comprised of multiple subsystems, which are coordinated by a centralized controller, as shown in Fig. 1a. The subsystems and controller are connected to a bus network, through which they exchange information. Because of the network's topology, whenever a message is sent by one of the devices connected to it, all other devices will receive the message. Each device has the ability to check if the message was addressed to itself and process the information in the positive case. Otherwise, the message is ignored. In this work the ideas of (Dietrich et al., 2002;

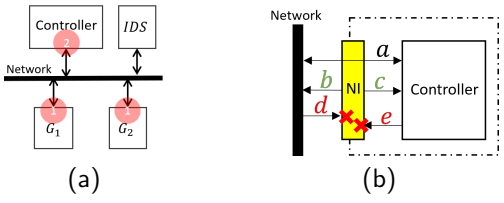


Fig. 1. System setup. (a) System with IDS. The red circles represent the possible attack locations. (b) Attacker's actions. The acronym NI stands for Network Interface.

Vieira et al., 2017), regarding the implementation of control systems modeled as DES, are adopted. The controller has the role of generating commands to be sent to the subsystems while the subsystems generate responses that are sent to the controller. A command can be associated with a controllable event, while a response is associated with an uncontrollable event. As a consequence, any subsystem will ignore any uncontrollable event received while the controller ignores any controllable event that it might receive. The controller is considered to be an entity that has a single or a set of supervisors (the design method chosen to obtain the supervisor is out of scope) encoded in it and is able to coordinate the subsystems by deciding which controllable event to be triggered next. In order to make this decision, the controller takes into account the control action of the supervisor(s) and an event priority rule. More details about the concept of such controller can be found at (Alves et al., 2022).

Additionally, the IDS (Intrusion Detection System) is a special device that monitors the network, verifying if the events that are being exchanged respect the legal behavior of the system. If it detects an abnormal behavior, it triggers an alarm for a human operator. It is important to highlight that, although the IDS has a defensive purpose, it is not a defense strategy proposed by this work. In reality, the IDS is an additional obstacle that the attacker has to overcome. We assume that the IDS is immune to attacks.

An attack can happen at location 1 or 2, represented by the red circles in Fig. 1a. Any device in the network has an internal controller that is responsible to generate events and to handle received events and a Network Interface (NI), which is the bridge between the device's internal controller and the network. In this work, we consider only attackers that have infected the device's network interface at location 2 in Fig. 1a. How the attacker managed to infect the device's network interface is out of scope of this work.

Once the attacker has infected the network interface of the controller, it can control all the communication between the device and the network, by inserting or erasing events. An attacker that is able to act on all events is an attacker that is able to perform multiple types of attacks including the ones that cause the system to block and the ones that can even produce physical damage in the machines, depending on the physical system. Such attacks can achieve the attacker's goal with the cost of being exposed to a human operator, who may perform a scan of the system in order to find the attacker. In contrast, in this work, the interest is in the types of attacks that allow the attacker to remain hidden even after achieving a successful attack, so that it can keep attacking without being noticed. Such attackers are called *persistent attackers* (Veerasamy, 2020). A persistent attacker is considered to be successful if it is able to cause the production of off-specification products or to introduce small delays in the process, while the discrete behavior of the controlled system under attack is indistinguishable from the behavior of the controlled system with no attack, as seen by the IDS. In this work, it is assumed that the attacker knows the models of plants and supervisors, as well as the physical process.

3.2 Attacker's Actions

An attacker A acting over a subsystem G_i is an agent that implements an attack function, which is a set of rules that gives the available options of actions that the attacker can take according to the evolution of G_i. In this paper we

represent the attack function as an automaton and more details on how to obtain it will be given in the next section. An attacker that has infected the network interface of the system's controller is able to act over all the events related to the subsystems under control of such controller. However, to limit the scope of this paper, we will assume that the attacker will only act over the events of a given subsystem G_i. The possible actions are shown in Fig. 1b.

The actions represented in Fig. 1b are associated to the following sets of events:

- Σ_i: (action a) events that are not tampered with by the attacker;
- $\Sigma_i^{+N} := \{\sigma^{+N} | \sigma \in \Sigma_i\}$: (action b) events that are inserted in the network. In this case, all devices in the network, except the controller, can see the event;
- $\Sigma_i^{+C} := \{\sigma^{+C} | \sigma \in \Sigma_i\}$: (action c) events that are inserted in the controller's observation. Inserting an occurrence of a controllable event into the controller's observation does not have any effect;
- $\Sigma_i^{-C} := \{\sigma^{-C} | \sigma \in \Sigma_{i,uc}\}$: (action d) events that are erased, preventing them from getting to the controller. All devices in the network, except the controller, can see the event.
- $\Sigma_i^{-N} := \{\sigma^{-N} | \sigma \in \Sigma_{i,c}\}$: (action e) events that are erased, preventing them from getting to the network. Notice that the attacker can only erase controllable events in Σ_i since these are the events generated by the controller. Only the controller can see the event.

Events in Σ_i^{-C} and Σ_i^{-N} represents the fact that the legitimate event occurred and then it was erased by the attacker before reaching the controller or the network, respectively. The set of all events, the ones that are legitimate and the ones that represent the actions of the attacker, is represented by $\Sigma_{ai} := \Sigma_i \cup \Sigma_i^{+N} \cup \Sigma_i^{-N} \cup \Sigma_i^{+C} \cup \Sigma_i^{-C}$ and is called the *attacked alphabet*. In the remainder of the text, strings defined in Σ_{ai} are denoted by w while strings defined in Σ_i are denoted by s.

Because the attack is happening at the controller's network interface, there are two different points of view: the one from the controller and the one from all other devices in the network, that share the same observation. To obtain only the events from the attacked alphabet that are seen from the network's point of view, a reporter map, called the *network projection* $P^N : \Sigma_{ai}^* \to \Sigma_i^*$ can be applied:

$$P^N(\varepsilon) := \varepsilon \tag{1}$$

$$P^N(\sigma^a) := \begin{cases} \sigma \text{ if } \sigma^a \in (\Sigma_i \cup \Sigma_i^{+N} \cup \Sigma_i^{-C}) \\ \varepsilon \text{ if } \sigma^a \in \Sigma_{ai} \setminus (\Sigma_i \cup \Sigma_i^{+N} \cup \Sigma_i^{-C}). \end{cases} \tag{2}$$

$$P^N(w\sigma^a) := P^N(w)P^N(\sigma^a), \text{ for } w \in \Sigma_{ai}^*, \sigma^a \in \Sigma_{ai} \tag{3}$$

The network projection erases the superscript a from σ^a if $a \in \{_{+N,-C}\}$ or erases the event σ^a entirely if $a \notin \{_{+N,-C}\}$ (i.e., $a \in \{_{-N,+C}\}$). If a string $w \in \Sigma_{ai}^*$ represents the sequence of actions of the attacker, then a string $s = P^N(w)$ is the sequence of events observed by the network. Additionally, according to (Cunha and Cury, 2007), because $\Sigma_i \subseteq \Sigma_{ai}$, then the network projection is equivalent to a natural projection.

Additionally, although an attacker at location 2 is able to act over events of all subsystems connected to the network, to limit the scope of this paper, we consider that the system G has only 2 subsystems, G_1 and G_2, and that the attacker will only act over the events of G_1. Note that the system G can have more subsystems if they are embedded in G_2.

4. DESIGN METHOD FOR A PERSISTENT ATTACK

In this section, we define a persistent attacker and we propose a design method for persistent attackers.

Let G_1 and G_2, with alphabets Σ_1 and Σ_2 (where $\Sigma_1 \cap \Sigma_2 = \emptyset$), respectively, be subsystems of a system G and E_1 and E_2 be specifications such that $\Sigma_1 \cap \Sigma_2 \cap \Sigma_{E_1} \neq \emptyset$, that is, E_1 affects both G_1 and G_2, and $\Sigma_1 \cap \Sigma_{E_2} = \emptyset$, which means that E_2 does not affect G_1. Define $G := G_1||G_2$. Let $\mathcal{S}$ be the supervisor that enforces the restrictions modeled by E_1 and E_2 over the plant G in a minimally restrictive way, i.e., $\mathcal{L}(\mathcal{S}/(G||E_1||E_2)) = Sup\mathcal{C}(E1 \cap E2, G)$. The language $K = \mathcal{L}(G||E_1||E_2)$ is called the desired behavior.

Now, consider that A is an attacker acting over a system G, which is represented by A/G. A supervisor $\mathcal{S}$ enforces the system's specifications in a closed-loop manner. The attacker modifies the original behavior $\mathcal{L}(G)$ to a behavior of G under attack of A, denoted by $\mathcal{L}(A/G) \subseteq \Sigma_{ai}^*$. The concept of a persistent attacker is formalized in the next definition.

Definition 2. (Persistent Attacker). An attacker A acting over a system G is called a persistent attacker if $P^N(\mathcal{L}(A/G)) \subseteq \mathcal{S}$.

According to Def. 2, an attacker is persistent if for every string $w \in \mathcal{L}(A/G)$, then $P^N(w) \in \mathcal{S}$, which means that the closed-loop behavior of the subsystem under attack is indistinguishable from the closed-loop behavior with no attack, when observed from the network's point of view.

In order to obtain an attacker that will not reveal itself, it is important to know the impact of its actions. The design method consists of building automata that show the impact of the attacker's actions from the point of view of the controller and network. These automata are called *controller estimator* and *network estimator*, respectively, and they represent the behavior the attacker is able to do, considering only the open-loop behavior. However, in order to be persistent, the attacker has to consider the impact of its actions over the system under control. Using the estimators, and the specifications for the system control, it is possible to prune part of the attacker's behavior to ensure that it will be a persistent attacker, resulting in the non-exposing attack structure under control Ψ. The procedure is summarized in Alg. 1.

Algorithm 1: Attacker design procedure

Input: G_1, G_2, E_1, E_2
Result: $G_1^A = (Q_A, \Sigma_{ai}, \delta_A, q_0, Q_m)$
$\Theta_{G_1}^C \leftarrow$ ControllerEstimator(G_1)
$\Theta_{G_1}^N \leftarrow$ NetworkEstimator(G_1)
$\Theta_{G_1} \leftarrow \Theta_{G_i}^C || \Theta_{G_i}^N$
$G' \leftarrow \Theta_{G_1} || G_2$
$\Theta_{E_1} \leftarrow$ SpecEstimator(E_1)
$K' \leftarrow G' || \Theta_{E_1} || E_2$
$\Psi \leftarrow Sup\mathcal{C}(K', G')$

(a) (b)

Fig. 2. Automata of Example 3. (a) Automaton G_1. (b) $\Theta^C_{G_i}$.

4.1 Controller Estimator

The controller estimator is an automaton that shows the impact of the attacker's actions over the plant, from the controller's point of view. The attacker is able to distinguish between legitimate events and the ones that are inserted by it, which is not true for the controller. For example, the controller cannot distinguish a legitimate occurrence of an uncontrollable event σ and its occurrence created by the attacker, σ^{+C}. Thus, upon seeing either of these two events, the controller will infer that event σ occurred in the plant, if it was feasible at that moment.

Some of the attacker's actions have no impact on the controller estimator, which are events that represent the action of inserting events only into the network or erasing an event, thus preventing it from reaching the controller. For this reason, these events do not appear in the controller estimator. Algorithm 2 shows how to obtain such an estimator, starting from the automaton G_1. We use Example 3 to illustrate the algorithm.

Algorithm 2: Controller estimator $\Theta^C_{G_1}$

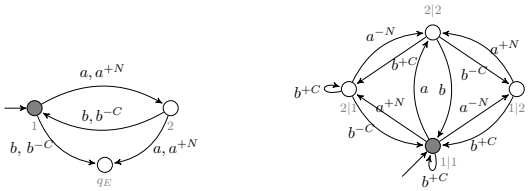
Input: $G_1 = (Q, \Sigma_1, \delta_1, q_0, Q_m)$
Result: $\Theta^C_{G_1} = (Q^C, \Sigma^C, \delta^C, q_0, Q^C_m)$
$\Sigma^C \leftarrow \Sigma_1 \cup \Sigma_1^{+C} \cup \Sigma_1^{-N}$
$\Theta^C_{G_1} \leftarrow G_1$
for $(q, \sigma, q') \in \Delta^C$ **do**
  if $(\sigma \in \Sigma_{1,c})$ **then**
    $\Delta^C \leftarrow \Delta^C \cup \{(q, \sigma^{-N}, q')\}$
  else
    $\Delta^C \leftarrow \Delta^C \cup \{(q, \sigma^{+C}, q')\}$
for $q \in Q^C$ **do**
  for $\sigma^a \in \Sigma_1^{+C} \wedge \sigma \in \Sigma_{1,uc}$ **do**
    if $(q, \sigma, q') \notin \Delta^C$, *with* $q' \in Q^C$ **then**
      $\Delta^C \leftarrow \Delta^C \cup \{(q, \sigma^{+C}, q)\}$

Example 3. Consider the automaton G_1 of Fig. 2a, with $\Sigma_{1,c} = \{a\}$ and $\Sigma_{1,uc} = \{b\}$. We want to obtain the controller estimator, $\Theta^C_{G_1}$, for this automaton, with Alg. 2. The result is the automaton $\Theta^C_{G_1}$, shown in Fig. 2b. □

4.2 Network estimator

The network estimator $\Theta^N_{G_1}$ is an automaton that shows the impact of the attacker's actions over the plant, from the network's point of view. Unlike the attacker, any other device in the network cannot distinguish between legitimate events or the ones that are inserted by the attacker.

Some of the attacker's actions have no impact on the network estimator, which are events that represent the action of inserting events only into the controller's observation or erasing an event, preventing it from reaching the network. For this reason, these events do not appear in the network estimator. Additionally, although it is possible for the attacker to insert a false occurrence of an uncontrollable in the network at any time, by doing so, it can potentially reveal itself because the attacker is not able to erase the legitimate occurrence of an uncontrollable event. Thus, we rule out the insertion of uncontrollable events in the network. Algorithm 3 shows how to obtain such an estimator, starting from the automaton G_1, which is illustrated in Example 4.



(a) (b)

Fig. 3. (a) Automaton $\Theta^N_{G_1}$ of Example 4. (b) Non-exposing attack structure Θ_{G_1} of Example 5.

Algorithm 3: Network estimator $\Theta^N_{G_1}$

Input: $G_1 = (Q, \Sigma_1, \delta_1, q_0, Q_m)$
Result: $\Theta^N_{G_1} = (Q^N, \Sigma^N, \delta^N, q_0, Q^N_m)$
$\Sigma^N \leftarrow \Sigma_1 \cup \Sigma_1^{+N} \cup \Sigma_1^{-C}$
$\Theta^N_{G_1} \leftarrow G_1$
for $q \in Q$ **do**
  for $\sigma \in \Sigma_1$ **do**
    if $(q, \sigma, q') \in \Delta^N$, *with* $q' \in Q$ **then**
      if $\sigma \in \Sigma_{1,c}$ **then**
        $\Delta^N \leftarrow \Delta^N \cup \{(q, \sigma^{+N}, q')\}$
      else
        $\Delta^N \leftarrow \Delta^N \cup \{(q, \sigma^{-C}, q')\}$

Example 4. Consider the automaton G_1 of Example 3. We want to obtain the network estimator, $\Theta^N_{G_1}$, for this automaton, using Alg. 3. The result is the automaton $\Theta^N_{G_1}$ obtained is shown in Fig. 3a. □

4.3 Local Non-Exposing Attack Structure

Using the controller and network estimator, it is possible for the attacker to obtain an automaton that shows the impact of its actions from the points of view of the controller and network. This automaton, called *Local Non-Exposing Attack Structure*, can be used to guide the attacker on which action to choose to do next, achieving its goal and remaining stealthy. The *local* term comes from the fact that this automaton considers only the subsystem G_1, ignoring how it interacts with other subsystems.

The local non-exposing attack structure Θ_{G_1} is given by:

$$\Theta_{G_1} = \Theta^C_{G_1} || \Theta^N_{G_1} \tag{4}$$

An example showing how to obtain the local non-exposing attack structure will be given next.

Example 5. Continuing from Examples 3 and 4, to obtain the local non-exposing attack structure, the parallel composition $\Theta_{G_1} = \Theta^C_{G_1} || \Theta^N_{G_1}$, must be obtained. The result is the local non-exposing attack structure Θ_{G_1}, as shown in Fig. 3b. □

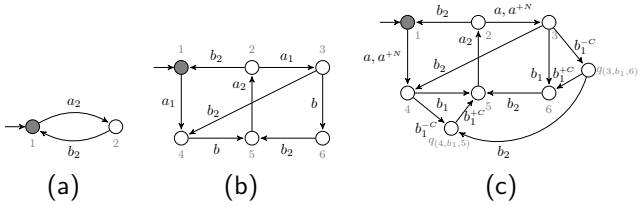


Fig. 4. Automata of Example 6. (a) Automaton G_2. (b) Specification E_1. (c) Automaton Θ_{E_1}.

4.4 Specification Estimators

Knowing the actions that it can do regarding a subsystem G_1 is not enough for the attacker to remain stealthy, since some of its actions can induce the controller to trigger events that might violate the system specifications. Thus, the attacker has also to consider the impact of its actions over the specification. For this, the attacker also builds an estimator Θ_{E_1} based on the controllable specification. In order to build this estimator, Alg. 4 is applied. Example 6 illustrates the algorithm.

Algorithm 4: Specification Estimator Θ_{E_1}

Input: $E_1 = (Q, \Sigma_1, \delta_{E_1}, q_0, Q_m)$, Σ_c^A, Σ_{uc}^A
Result: $\Theta_{E_1} = (Q_E, \Sigma_{a1}, \delta_E, q_0, Q_E^m)$
$\Theta_{E_1} \leftarrow E_1$
for $(q, \sigma, q') \in \Delta_{E_1}$, *where* $q \neq q'$ **do**
  if $(\sigma \in \Sigma_{1,c} \cap \Sigma_c^A)$ **then**
    $\Delta_E \leftarrow \Delta_E \cup \{(q, \sigma^{+N}, q')\}$
  else if $(\sigma \in \Sigma_{1,uc} \cap \Sigma_{uc}^A)$ **then**
    $Q_E \leftarrow Q_E \cup \{q_{(q,\sigma,q')}\}$
    $\Delta_E \leftarrow \Delta_E \cup \{(q, \sigma^{-C}, q_{(q,\sigma,q')}), (q_{(q,\sigma,q')}, \sigma^{+C}, q')\}$
    for $(q', \sigma', q'') \in \Delta_E$ *where* $\sigma' \in \Sigma_{uc1} \setminus \Sigma_{uc}^A \wedge (q''', \sigma, q'') \in \Delta_E$ **do**
      $\Delta_E \leftarrow \Delta_E \cup \{(q_{(q,\sigma,q')}, \sigma', q_{(q''',\sigma,q'')})\}$
for $(q, \sigma, q) \in \Delta_{E_1}$ **do**
  $\Delta_E \leftarrow \Delta_E \cup \{(q, \sigma^{-C}, q), (q, \sigma^{+C}, q)\}$
  for $(q, \sigma', q'') \in \Delta_E$ *where* $q'' = q_{(q,\sigma',q')}$ **do**
    $\Delta_E \leftarrow \Delta_E \cup \{(q'', \sigma^{-C}, q''), (q'', \sigma^{+C}, q'')\}$

Example 6. Suppose we have a second subsystem, whose behavior is modeled by the automaton of Fig. 4a, where $\Sigma_{c_2} = \{a_2\}$. Additionally, suppose the controller is enforcing a specification E_1, shown in Fig. 4b.

The attacker will now apply Algorithm 4 in order to obtain the estimator for the specification. The alphabets Σ_c^A and Σ_{uc}^A received as input in Alg. 4 represent the set of controllable and uncontrollable events, respectively, that are under attack. In this example, $\Sigma_c^A = \{a\}$ and $\Sigma_{uc}^A = \{b\}$. Figure 4c shows the specification estimator, which shows the impact of the attacker's actions on the specification. □

4.5 Non-Exposing Attack Structure Under Control

Now that the attacker has the automata that show the impact of its actions, it can obtain the non-exposing behavior under control. For this, we can take advantage of the supervisor synthesis, which will give the behavior that prevents the attacker from executing an action that will reveal its presence, considering that the system is under control. For this, the controllability of events needs to be defined. All events representing the attacker's actions can be considered as controllable. Events that are not tampered with, i.e., events in Σ_i, keep their controllability status. Thus, the non-exposing behavior under control Ψ can be obtained as follows.

In (5) the global uncontrolled behavior is obtained, which already considers the attacker's actions over G_1.

$$G' = \Theta_{G_1} || G_2 \tag{5}$$

Then, the desired behavior under control is obtained by (6) and finally the non-exposing attack structure under control is obtained by (7).

$$K' = \mathcal{L}_{G'} || \mathcal{L}_{\Theta_{E_1}} || \mathcal{L}_{E_2} \tag{6}$$

$$\mathcal{L}_\Psi = SupC(K', \mathcal{L}_{G'}) \subseteq K' \tag{7}$$

Example 7. Continuing from Example 6, G' and K' are obtained from (5) and (6), respectively. Their automata are not presented here due to space limitation. Finally, the non-exposing attack structure under control Ψ is obtained by (7) and the automaton is shown in Fig. 5.

Notice that upon initialization, the attacker can choose between three actions : *i*) wait for the controller to trigger event a; *ii*) erase event a once it is triggered, preventing it from reaching the network and; *iii*) insert a false occurrence of event a into the network (a^{+N}). The automaton Ψ shows then what are possible next actions for the attacker, according to the evolution of the entire system.

The attacker can insert a delay in the production process by, among other possibilities, executing actions a^{-N} and a^{+N}, since the subsystem G_1 will only start working after event a^{+N}. The attacker can also anticipate events by inserting a^{+N} before the controller triggers the legitimate event a. In this case, the attacker needs to have the possibility of erasing the legitimate occurrence of a or otherwise it will be revealed by the IDS' observation of two occurrences of event a. □

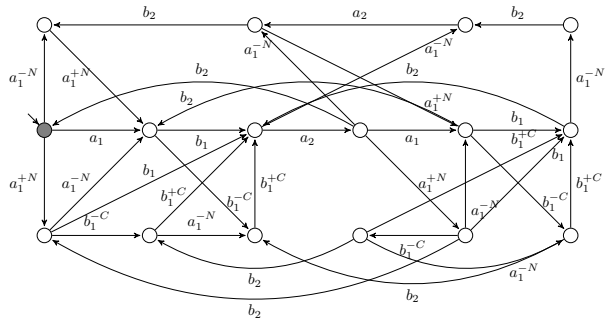


Fig. 5. Non-exposing attack structure Ψ of Example 7.

Next we present two lemmas that we will use to prove the main result. Due to space limitation, we omit the lemmas' proofs.

Lemma 8. Let G_1 be a subsystem and Θ_{G_1} be the local non-exposing attack structure given by (4). Then, $P^N(\Theta_{G_1}) = G_1$.

Lemma 9. Let E_1 be a specification and Θ_{E_1} be the specification estimator generated by Alg. 4. Then, $P^N(\Theta_{E_1}) = E_1$.

Next we present a result that allows us to use the automaton Ψ for the design of a persistent attacker.

Theorem 10. Let G_1 and G_2 be the automata that model the uncontrolled behavior of subsystems and E_1 and E_2 the automata that model the specifications such that $\Sigma_{E_1} \cap \Sigma_1 \cap \Sigma_2 \neq \emptyset$ and $\Sigma_{E_2} \cap \Sigma_1 = \emptyset$. Let $\mathcal{S} = Sup\mathcal{C}(K, G)$ be a supervisor, with $G = \mathcal{L}(G_1||G_2)$ and $K = \mathcal{L}(G||E_1||E_2)$, which is controllable. The language generated by the non-exposing attack structure under control Ψ, obtained by the application of Alg. 1, models a persistent attacker, i.e.,

$$P^N(\mathcal{L}_\Psi) \subseteq \mathcal{S}. \tag{8}$$

Proof. To prove (8), it is necessary to prove that for any string $s \in P^N(\mathcal{L}_\Psi)$, it is also true that $s \in \mathcal{S}$. Consider a string $w \in \mathcal{L}_\Psi$ such that $P^N(w) = s$. From (7), we have that

$$w \in K' = \mathcal{L}_{\Theta_{G_1}}||\mathcal{L}_{G_2}||\mathcal{L}_{\Theta_{E_1}}||\mathcal{L}_{E_2}. \tag{9}$$

Applying the network projection over both sides of (9):

$$P^N(w) \in P^N(\mathcal{L}_{\Theta_{G_1}}||\mathcal{L}_{G_2}||\mathcal{L}_{\Theta_{E_1}}||\mathcal{L}_{E_2}). \tag{10}$$

Using the property of natural projections that allows us to say that $P^N(\mathcal{L}_{\Theta_{G_1}}||\mathcal{L}_{G_2}||\mathcal{L}_{\Theta_{E_1}}||\mathcal{L}_{E_2}) \subseteq P^N(\mathcal{L}_{\Theta_{G_1}})||$ $P^N(\mathcal{L}_{G_2})||$ $P^N(\mathcal{L}_{\Theta_{E_1}})||P^N(\mathcal{L}_{E_2})$, then (10) becomes:

$$P^N(w) \in P^N(\mathcal{L}_{\Theta_{G_1}})||P^N(\mathcal{L}_{G_2})||P^N(\mathcal{L}_{\Theta_{E_1}})||P^N(\mathcal{L}_{E_2}). \tag{11}$$

Using Lemmas 8 and 9, it is possible to rewrite (11) as

$$P^N(w) \in \mathcal{L}_{G_1}||P^N(\mathcal{L}_{G_2})||\mathcal{L}_{E_1}||P^N(\mathcal{L}_{E_2}). \tag{12}$$

Additionally, because G_2 and E_2 were not modified, then the network projection will not have any impact on them. Thus, (12) can be written as

$$s \in \mathcal{L}_{G_1}||\mathcal{L}_{G_2}||\mathcal{L}_{E_1}||\mathcal{L}_{E_2} = K. \tag{13}$$

With the assumption that K is controllable, then it is possible to say that $s \in \mathcal{S}$. □

Thus, because $P^N(\mathcal{L}_\Psi) \subseteq \mathcal{S}$, the non-exposing behavior under control Ψ can be used as guide to a persistent attacker on how to choose its actions.

5. CONCLUSION

Understanding how an attacker can act over a system and remain undetected is crucial to strengthening the security of DES since it provides valuable insights into potential vulnerabilities, thus aiding in the development of robust defense mechanisms. This work introduces a method for designing a persistent attacker, which allows it to remain hidden while manipulating the behavior of a controlled system. Our method considers an attack model where the attacker has infected the network interface of the ICS devices, which is consistent with the literature on cybersecurity. One of the main contributions of this work is to bring attention to the fact that cyber-attacks in practice often occur in the devices of a network, instead of the network itself. Additionally, the algorithms presented are based on the models of plants and specifications, which is generally less costly when compared to methods that start from the supervisor automata, which are often bigger automata.

Moreover, while the constructed non-exposing attack structure under control, represented by Ψ, offers insights into potential actions for the attacker, it does not dictate the timing of these actions. Future research could delve into refining this framework to incorporate temporal elements, thereby enhancing the attacker's strategy.

An important avenue for further exploration lies in the development of defense techniques against persistent attackers. Creating robust defense mechanisms capable of detecting and mitigating such stealthy attacks is paramount in safeguarding controlled systems.

ACKNOWLEDGEMENTS

We thank one of the anonymous reviewers whose comment that our original version of Alg. 4 did not handle self-loops correctly led to our current version of Alg. 4.

REFERENCES

Alves, M.R.C., Rudie, K., and Pena, P.N. (2022). A Security Testbed for Networked Control Systems. *IFAC-PapersOnLine*, 55(28), 128–134.

Cassandras, C.G. and Lafortune, S. (2021). *Introduction to Discrete Event Systems*. Springer International Publishing, 3rd edition.

Cunha, A.E.C. and Cury, J.E.R. (2007). Hierarchical supervisory control based on discrete event systems with flexible marking. *IEEE Transactions on Automatic Control*, 52, 2242–2253.

Dietrich, P., Malik, R., Wonham, W.M., and Brandin, B.A. (2002). Implementation Considerations in Supervisory Control. In *B. Caillaud, P. Darondeau, L. Lavagno, X. Xie, Synthesis and Control of Discrete Event Systems*, 185–201. Kluwer Academic Publishers.

Meira-Góes, R., Kang, E., Kwong, R.H., and Lafortune, S. (2020). Synthesis of Sensor Deception Attacks at the Supervisory Layer of Cyber–Physical Systems. *Automatica*, 121, 109172.

Pan, X., Wang, Z., and Sun, Y. (2020). Review of PLC Security Issues in Industrial Control System. *Journal of Cyber Security*, 2, 69–83.

Prinsloo, J., Sinha, S., and von Solms, B. (2019). A Review of Industry 4.0 Manufacturing Process Security Risks. *Applied Sciences (Switzerland)*, 9(23).

Ramadge, P.J.G. and Wonham, W.M. (1989). The control of discrete event systems. *Proceedings of the IEEE*, 77, 81–98.

Rashidinejad, A., Lin, L., Wetzels, B., Zhu, Y., Reniers, M., and Su, R. (2019). Supervisory Control of Discrete-Event Systems Under Attacks: An Overview and Outlook. In *Proceedings of the 18th European Control Conference*, 1732–1739. EUCA.

Shareef, T. (2022). 9 Times Hackers Targeted Cyber-attacks on Industrial Facilities. Make use of. Available at https://www.makeuseof.com/cyberattacks-on-industry-hackers. Accessed on 23/09/2022.

Veerasamy, N. (2020). *Chapter 2 - Cyberterrorism – the spectre that is the convergence of the physical and virtual worlds*. Academic Press.

Vieira, A.D., Santos, E.A.P., De Queiroz, M.H., Leal, A.B., De Paula Neto, A.D., and Cury, J.E.R. (2017). A Method for PLC Implementation of Supervisory Control of Discrete Event Systems. *IEEE Transactions on Control Systems Technology*, 25(1), 175–191.

Yuan, Y., Yang, H., Guo, L., and Sun, F. (2019). *Analysis and Design of Networked Control Systems under Attacks*, volume 1. CRC Press, 1st edition.

Zhang, Q., Seatzu, C., Li, Z., and Giua, A. (2022). Sensor and actuator attacks in discrete event systems. *IFAC-PapersOnLine*, 55, 38–45.