2020 59th IEEE Conference on Decision and Control (CDC)
Jeju Island, Republic of Korea, December 14-18, 2020

On Attack Mitigation in Supervisory Control Systems: A Tolerant Control Approach

Jingshi Yao, Xiang Yin and Shaoyuan Li

Abstract— **This paper investigates attack mitigation problem in supervisory control of discrete event systems. We consider the scenario where the system is subject to actuator enablement attack. We explicitly distinguish between controllable events and defendable events; the former are events that can be disabled by the normal supervisor but may be subject to attack, while the latter are events that can be defensed (disabled definitely) by the mitigation module but possibly with higher costs. The objective is to design an attack mitigation strategy to prevent serious damage from attack. We formulate the attack mitigation problem as a tolerant control problem under partial observation. Particularly, in addition to guarantee safety, we aim to maximize the desirable behavior (normal specification) while minimize the tolerable behavior (safe but not desirable). We provide an effective online algorithm for solving this problem, which yields a novel attack mitigation strategy that generalizes the existing one in the literature. Specifically, we show that the proposed strategy may still prevent damage even when the safe-controllability condition, which is required by the existing strategy, does not hold.**

I. Introduction

Cyber-attacks are increasingly becoming pervasive in safety-critical cyber-physical systems (CPSs) such as industrial control systems, intelligent transportation systems and smart grids. To describe high-level behaviors of CPSs, cyber or cyber-physical control systems are usually abstracted as transition systems with discrete state spaces and event-driven dynamics, or discrete event systems (DESs). Due to the importance of security and safety concerns in CPSs, detection and mitigation of attacks has been becoming a very active research area in the past few years under the framework of supervisory control of DESs; see, e.g., [1]–[16] and the recent survey [17].

In the context of DES, the study of attack detection and mitigation dates back to the work of Thorsley and Teneketzis [18], where a language-measure-based dynamic programming approach is proposed to synthesize an optimal supervisor in the presence of an attacker that may allow the occurrences of disabled events, i.e., actuator enablement attack. The problem of synthesizing supervisor under sensor insertion attack and removal attack is considered in [14]. In [15], the authors consider both sensor attacks and actuator attacks in a unified model. The problem of synthesizing attack-robust supervisor is studied in [4], [8], [19]. In [2], [3], [11], the authors investigate, from the attacker point of view, how to synthesize successful attacks such that the system can be damaged while preventing itself from been detected.

This work was supported by the National Natural Science Foundation of China (61803259, 61833012) and by Shanghai Jiao Tong University Scientific and Technological Innovation Funds.

J. Yao, X. Yin and S. Li are with Department of Automation and Key Laboratory of System Control and Information Processing, Shanghai Jiao Tong University, Shanghai 200240, China. E-mail: {yaojingshi,yinxiang,syli}@sjtu.edu.cn.

Recently in [1], Carvalho et al. proposed a general framework for attack detection and mitigation for supervisory control systems, where four types of attacks are considered. Automaton model is provided to describe the closed-loop system under each type of attacks. A generic attack detection and mitigation strategy, which is independent from the specific attack type, is also proposed. The strategy consists of the following two parts: (i) first, a diagnoser is used to detect whether or not the system has been attacked; and (ii) once the the diagnoser detects attack unambiguously, the mitigation module will disable/defend all controllable events to prevent the system from reaching unsafe states. Whether or not such a mitigation strategy works is characterized by the condition of safe-controllability that was modified based on the original concept proposed in [20] for the purpose of fault-tolerant control.

Essentially, the attack mitigation strategy in [1] follows the philosophy that "*take action to defend after detecting attack for sure*". However, such a strategy may be unnecessarily conservative in the sense that it may be too late to take actions when the attack is detected. In general, the attack mitigation module may start to take early actions in advance when it is *possible* that the system has been attacked. In fact, this point has been mentioned in [1] that "*we adopt simple and conservative approach to defend attacks, and have left the refinement of our methodology to account for more sophisticated defense mechanisms*". Our work aims to fill in this gap and to improve the existing mitigation strategy.

In this paper, we follow the basic framework of [1] for attacks in supervisory control systems. However, we consider a more general setting by explicitly distinguishing between controllable events and defendable events, where controllable events stand for events that can be disabled by the normal supervisor but may be subject to attack, but defendable events stand for events that can be defensed (disabled definitely) by the mitigation module but possibly with higher costs. Also, different from the mitigation strategy in [1], we propose a more general attack mitigation strategy that allows the system to defend even before the attack is precisely detected.

More specifically, we formulate the attack mitigation problem as a *tolerant control problem*. We treat the system under possible attacks as a new plant and our goal is to strictly prevent the new system from reaching unsafe states while maximizing the *desirable behavior*, which is the closed-loop language without attack. Behavior between the desirable behavior and the unsafe behavior is considered as the *tolerable behavior* for which we further need to minimize. We show that our approach is more general than that of [1] in the sense that the proposed strategy may successfully prevent damage from attack even when the safe-controllability condition does not hold.

Our approach is motivated by the tolerant control problem studied in [21]. However, the solution in [21] is only applicable for systems with full observation. In our setting for attack mitigation, we need to solve a tolerant control problem under *partial observation* which has never been solved in the literature to our knowledge. Technically, our approach methodology also generalizes existing techniques for solving the standard supervisory control problem under partial observation [22]–[25] to the tolerant control setting by considering both desirable language and tolerable language into account. Finally, we emphasize that we only focus on actuator enablement attack in this paper to state our main results. However, the proposed tolerant control approach is generic and can be applied to any type of attack given the attacked behavior can be modeled.

II. Preliminaries

We use standard notations in discrete-event systems [26]. A discrete-event system is modeled as a deterministic finite-state automaton (DFA) $G = (X, \Sigma, \delta, x_0)$, where X is the finite set of states, Σ is the finite set of events, $\delta : X \times \Sigma \to X$ is the (partial) transition function and $x_0 \in X$ is the initial state. The transition function is also extended to $\delta : X \times \Sigma^* \to X$ in the usual manner [26]. For simplicity, $\delta(x_0, s)$ is abbreviated as $\delta(s)$. The language generated by G is $\mathcal{L}(G) = \{s \in \Sigma^* : \delta(s)!\}$, where "!" means "is defined" and Σ^* denotes the set of all finite strings over Σ including the empty string ϵ. The prefix closure of language L is $\overline{L} = \{t \in \Sigma^* : \exists u \in \Sigma^* \text{ s.t. } tu \in L\}$. We denote by $s \leq t$ if $s \in \overline{\{t\}}$ and by $s < t$ if $s \leq t$ and $s \neq t$.

In the supervisory control framework, the event set is partitioned as: $\Sigma = \Sigma_c \dot\cup \Sigma_{uc}$, where Σ_c and Σ_{uc} are the sets of controllable and uncontrollable events, respectively. We denote by $\Gamma = \{\gamma \in 2^\Sigma : \Sigma_{uc} \subseteq \gamma\}$ the set of admissible control decisions or control patterns. The event set is further partitioned as $\Sigma = \Sigma_o \dot\cup \Sigma_{uo}$, where Σ_o and Σ_{uo} are the sets of observable and unobservable events, respectively. The natural projection $P : \Sigma^* \to \Sigma_o^*$ is defined as the mapping that replaces each unobservable event in a string by ϵ.

A partial-observation supervisor is a mapping $S_P : P(\mathcal{L}(G)) \to \Gamma$ that enables/disables events dynamically based on its observation. We denote by $\mathcal{L}(S_P/G)$ the closed-loop language under control, which is defined recursively by:

- $\epsilon \in \mathcal{L}(S_P/G)$; and
- for any $s \in \Sigma^*, \sigma \in \Sigma$, we have $s\sigma \in \mathcal{L}(S_P/G)$ iff $s \in \mathcal{L}(S_P/G), s\sigma \in \mathcal{L}(G)$ and $\sigma \in S_P(P(s))$.

In the problem of attack mitigation, it is assumed that there already exists a designed supervisor and the supervisor is assumed to be *recognized* by a DFA $H = (X_H, \Sigma, \delta_H, x_{0,H})$ such that:

(i) $\forall \sigma \in \Sigma_{uc}, x \in X_H : \delta_H(x, \sigma)!$; and

(ii) $\forall x, x' \in X_H, \sigma \in \Sigma_{uo} : \delta_H(x, \sigma) = x' \Rightarrow x = x'$.

Intuitively, the first condition says that the supervisor cannot disable uncontrollable events and the second condition says that only observable events can trigger decision changes. Therefore, $H \| G$ is essentially the model of the closed-loop system, i.e., $\mathcal{L}(H \| G) = \mathcal{L}(S_P/G)$, where "$\|$" is the standard parallel composition operator.

III. Attack Model and Mitigation Strategies

A. Actuator Enablement Attack Model

We first review the model of actuator enablement attack (AE-attack) proposed in [1]. In this setting, we assume that $\Sigma_v \subseteq \Sigma_c$ is a set of *vulnerable actuator events*, which can be either observable or unobservable. The attacker is able to enable (attack) a vulnerable event even when it is disabled by the supervisor, e.g., by replacing the control command in the communication channel. Then the set of attacked actuator events is defined by $\Sigma_{c,v}^a = \{\sigma^a : \sigma \in \Sigma_v\}$ and we define $\Sigma_a = \Sigma \cup \Sigma_{c,v}^a$. Intuitively, σ^a represents the occurrence of σ when it is originally disabled by the supervisor but is enabled by the attacker. We also define the compression operator $C : \Sigma_a \to \Sigma$ by: $C(\sigma) = \sigma$ if $\sigma \in \Sigma$ and $C(\sigma^a) = \sigma$ if $\sigma^a \in \Sigma_{c,v}^a$. The compression operator is also extended to $C : \Sigma_a^* \to \Sigma^*$ and to $C : 2^{\Sigma_a^*} \to 2^{\Sigma^*}$. Also, we define the dilation operator $D : \Sigma \to 2^{\Sigma_a}$ as the inverse of C, which is also extended to $D : \Sigma^* \to 2^{\Sigma_a^*}$ and to $D : 2^{\Sigma^*} \to 2^{\Sigma_a^*}$.

In order to model the closed-loop system under possible AE-attacks, one can construct two new DFAs G_a and H_a based on G and H, respectively. Specifically, $G_a = (X, \Sigma_a, \delta_a, x_0)$ is a DFA such that

$$\forall x \in X, \sigma \in \Sigma_a : \delta(x, C(\sigma))! \Rightarrow \delta_a(x, \sigma) = \delta(x, C(\sigma)).$$

Intuitively, G_a is obtained by adding a parallel transition labeled by $\sigma^a \in \Sigma_{c,v}^a$ for each feasible transition labeled by $\sigma \in \Sigma_v$, which captures the possibility of AE-attack on σ. Also, DFA $H_a = (X_H, \Sigma_a, \delta_{H,a}, x_{0,H})$ is defined by: for any $x \in X_H, \sigma \in \Sigma_a$, we have

$$\delta_{H,a}(x, \sigma) = \begin{cases} \delta_H(x, \sigma) & \text{if } \delta_H(x, \sigma)! \\ x & \text{if } \sigma \in \Sigma_{c,v}^a \wedge \delta_H(x, \sigma)\neg! \end{cases}$$

Intuitively, H_a is obtained by adding self-loop labeled by $\sigma^a \in \Sigma_{c,v}^a$ for each $\sigma \in \Sigma_c$ that is originally disabled by the supervisor. This captures the fact that σ is AE-attacked.

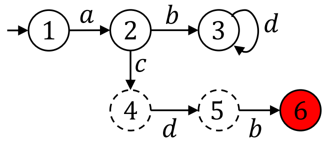
As shown in [1], the closed-loop system under all possible AE-attacks with the given supervisor can be computed by taking the parallel composition of the modified plant G_a and the modified supervisor recognizer H_a. We denote by

$$G_M = H_a \| G_a = (X_M, \Sigma_a, \delta_M, x_{0,M})$$

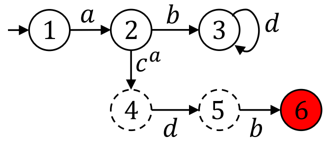
the closed-loop system under attack.

B. Mitigation Strategy in [1] and its Limitation

In the framework of [1], all strings in $\mathcal{L}(S_P/G)$ are considered as desirable. That is, the normal supervisor S_P is designed to achieve the desirable specification. However, not all strings outside of $\mathcal{L}(S_P/G)$ are considered unsafe



(a) G



(b) G_M

Fig. 1: Systems for Examples 1, 2 and 3, where dashed states denote states that are not desirable but tolerable and red states denote unsafe states.

immediately. Specifically, $X_{bad} \subseteq X$ is assumed to be the set of unsafe states capturing states in which the plant is physically damaged. Behavior between unsafe and desirable is called tolerable. When the system is attacked, unsafe states may be reachable in $\mathcal{L}(G_M)$ (in terms of the second component for the plant).

In order to prevent the system from reaching unsafe states, [1] proposes an attack mitigation strategy. Specifically, the mitigation module is assumed to be able to *defend* all controllable events Σ_c; its observation mapping is specified by $P_C : \Sigma_a^* \to \Sigma_o^*$, where $P_C := P \circ C$, since it cannot distinguish between σ and σ^a.

Then a simple diagnoser-based mitigation strategy was proposed by [1] as follows. First, a diagnoser is used to infer, based on mapping P_C, whether an event σ^a has occurred. The mitigation module will remain silent before the attack is detected. Once the attack is detected, the mitigation module will defend by (possibly physically) disabling all events in Σ_c for the purpose of safety. It is shown that such a strategy can successfully prevent unsafe states when the system is GF-safe controllable.

However, as we mentioned earlier, the mitigation strategy in [1] as reviewed above is conservative as it follows the idea of "start to defend only when knows attack for sure". This setting unnecessarily restricts the power of the mitigation module. In some cases, it may be worthwhile taking early actions even when the attack is just suspected. This point is illustrated by the following example.

Example 1: Let us consider system G shown in Figure 1(a). We assume that $\Sigma_c = \{c, d\}$ and $\Sigma_o = \{a, b, d\}$. We assume that the plant is controlled by supervisor S_P that always disables event c in order to achieve desirable specification $\overline{\{ab\}\{d\}^*}$. Now we assume that $\Sigma_v = \{c, d\}$ are vulnerable events and unsafe state is $X_{bad} = \{6\}$. Then the closed-loop system under AE-attack is shown in Figure 1(b). The strategy in [1] cannot prevent the system from reaching X_{bad}. This is because the diagnoser cannot infer the occurrence of c^a for sure as it is unobservable. When the attack is detected by observing ad, it is already too late as unsafe state will be reached uncontrollably via b. Therefore, this system is not safe controllable. However, one possible solution is to defend event c in advance when a is observed; in the case, the mitigation module can prevent the system from reaching unsafe state even without detecting the attack precisely. This example is straightforward as we are defending at the boundary for c. In the following subsection, we will consider a more general setting where boundary events may not be defendable.

Remark 1: (Mitigation Module v.s. Supervisor) Here we would like to emphasize that, although both the original supervisor and the mitigation module can disable/defend events in Σ_c, their physical interpretations are different. A normal disablement in the supervisory control framework is usually done by sending a disable decision to actuators via control channels which may be attacked. On the other hand, the mitigation module may take physical actions forcing to disable (defend) events in Σ_c; this is more reliable, in the sense of non-attackable, but is also more costly in general. This is why one need to separate the roles of the supervisor and the mitigation module, although the latter can also be treated as a supervisor mathematically.

C. Defendable Events

In the framework of [1], it is assumed that the mitigation module can "disable" exactly the same set of events as the supervisor can do, i.e., Σ_c are both normal controllable events and dependable events. As we discussed earlier, the physical meaning of disable decision issued by the mitigation module and disable decision issued by the supervisor can be different. Here, we consider a more general setting by precisely distinguish between these two sets of events. Specifically, we assume that Σ_d is a set of *defendable events*, where we have

$$\Sigma_d \subseteq \Sigma_v \subseteq \Sigma_c.$$

That is, Σ_d are vulnerable in the original closed-loop system but can be forced to be disabled (defended) possibly with higher costs by the mitigation module. Then controllable events are essentially partitioned into three categories in terms of their controllability:

- $\Sigma_c \setminus \Sigma_v$: events that can be disabled directly by the supervisor in the normal manner as they are not subject to attack.
- $\Sigma_v \setminus \Sigma_d$: events that cannot be reliably disabled, i.e., for any $\sigma \in \Sigma_v \setminus \Sigma_d$, σ^a can always be enabled by the attacker even when σ is disabled by the supervisor.
- Σ_d: events that can be reliably disabled (defended) by the mitigation module (but possibly in a more costly manner).

Then the mitigation module is essentially a new supervisor M that treats G_a as a new plant. [1]

We illustrate our setting with the role of defendable events by the following examples.

[1]One can also consider G_M as a new plant; this has no essentially difference from the synthesis point of view. Here we consider G_a as a plant to directly synthesize the combined decision of the supervisor and the mitigation module. Then, at each instant, one can compare the synthesized decision and the original decision to distinguish whether a disable decision is made by the normal supervisor or by the mitigation module.

Example 2: Consider again the system in Figure 1(a) with $\Sigma_c = \{c, d\}$ and $\Sigma_o = \{a, b, d\}$. We assume that $\Sigma_v = \{c, d\}$ and $\Sigma_d = \{d\}$; hence, the previous strategy in Example 1 for depending the boundary c cannot be applied. Yet, we can design a mitigation strategy M by defending d when a is observed. In this case, although M does not know the occurrence of c^a for sure, it still defends d for the purpose of safety. This strategy yields language $\overline{\{ab\}\{d\}^* \cup \{ac\}}$. Although ac is outside of the desirable behavior, it is still tolerable as it does not reach X_{bad}. More interestingly, no desirable behavior is sacrificed as $\mathcal{L}(S_P/G) = \overline{\{ab\}\{d\}^*} \subset \overline{\{ab\}\{d\}^* \cup \{ac^a\}}$. This example shows that the failure of the mitigation strategy in [1] does not necessarily imply the non-existence of a successfully strategy that can guarantee safety without compromising the desirable behavior.

The following example shows a more general scenario, in which such a "non-sacrifice" solution does not exist but we may allow the mitigation module to sacrifice some desirable behavior in order to guarantee safety.

Example 3: Let us still consider system G shown in Figure 1(a). However, we assume $\Sigma_c = \{c, d\}$ and $\Sigma_o = \{a, d\}$; note that event b is unobservable now. The plant is also controlled by a normal supervisor S_P that always disables event c in order to achieve desirable specification $\overline{\{ab\}\{d\}^*}$. We assume that $\Sigma_v = \{c, d\}$ and $\Sigma_d = \{d\}$. In this case, it is not possible to avoid reaching 6 without compromising the desirable behavior. For the purpose of safety, one can choose to defend d when a is observed. However, since b is unobservable, event d at state 3 is also disabled. This strategy yields language $\overline{\{ab, ac\}}$. In this case, desirable strings $abdd\dots$ are sacrificed to avoid reaching X_{bad}.

IV. A General Attack Mitigation Problem

With the previous discussions, we now formally formulate the general attack mitigation problem as a tolerant control problem under partial observation in this section.

Specifically, we consider the system under possible attacks $G_a = (X, \Sigma_a, \delta_a, x_0)$. In order to distinguish the desirable behavior and the safe behavior, we define

$$K_{des} = D(\mathcal{L}(S_P/G)) = D(\mathcal{L}(H\|G))$$

as the *desirable language*, which is the normal language generated by S_P/G without attack. Note that we put the dilation operator $D(\cdot)$ because σ and σ^a are the same event in G_a when the supervisor is to be specified. Also, not all strings outside of K_{des} is considerable as unsafe; those strings between the desired language and the unsafe language are considered tolerable. Therefore, we define

$$K_{tol} = \{s \in \mathcal{L}(G_a) : \forall t \le s, \delta_a(x_0, t) \notin X_{bad}\}$$

as the *tolerable language*. Clearly, we have $K_{des} \subseteq K_{tol}$ as the normal language should not reach unsafe states.

For the sake of simplicity, we assume that the state space of G_a is partitioned as

$$X = X_D \dot\cup X_T \dot\cup X_F$$

such that $K_{des} = \{s \in \mathcal{L}(G_a) : \delta_a(x_0, s) \in X_D\}$ and $K_{tol} = \{s \in \mathcal{L}(G_a) : \delta_a(x_0, s) \notin X_F\}$. Therefore, X_D, X_T and X_F represent, respectively, desirable states, tolerable states and unsafe states. Note that this assumption is without loss of generality as we can always refine the state-space of G_a in linear-time such that the partition holds.

In our approach, since the mitigation module is to be determined and we do not need to detect the attack precisely, there is no need to distinguish between σ and σ^a. Therefore, we simplify G_a as

$$\tilde{G}_a = (X, \Sigma_a, \tilde{\delta}_a, x_0)$$

which is obtained by only keeping the transition labeled with σ^a for each pair of transitions in $\tilde{G}_a$ labeled with σ and σ^a. Or equivalently, $\tilde{G}_a$ is obtained by replacing each event $\sigma \in \Sigma_v$ by σ^a in G. Then the set of controllable events for $\tilde{G}_a$ is defined by

$$\Sigma_{c,a} = \{\sigma^a : \sigma \in \Sigma_d\} \cup (\Sigma_c \setminus \Sigma_v)$$

and we denote by Γ_a the set of control patterns w.r.t. $\Sigma_{c,a}$. The set of uncontrollable events in $\tilde{G}_a$ is denoted by $\Sigma_{uc,a}$. The observation is specified by mapping $P_C = P \circ C$.

Our goal is to design $M : P_C(\mathcal{L}(\tilde{G}_a)) \to \Gamma_a$ which can be considered as the combined strategy of the mitigation module and the original supervisor such that (i) the new closed-loop system is safe; and (ii) it affects the original closed-loop language K_{des} in a least-restrictive manner. This is formulated as the following problem.

Problem 1: (Attack Mitigation Problem) Given plant $\tilde{G}_a$ with controllable $\Sigma_{c,a}$ and observation mapping $P_C : \Sigma_a^* \to \Sigma_o^*$, desirable language K_{des} and tolerable language K_{tol}, find a supervisor $M : P_C(\mathcal{L}(\tilde{G}_a)) \to \Gamma_a$ such that

(1) $\mathcal{L}(M/\tilde{G}_a) \subseteq K_{tol}$;

(2) For any M' such that $\mathcal{L}(M'/\tilde{G}_a) \subseteq K_{tol}$, we have
$$\mathcal{L}(M/\tilde{G}_a) \cap K_{des} \not\subset \mathcal{L}(M'/\tilde{G}_a) \cap K_{des}.$$

(3) For any M' such that $\mathcal{L}(M'/\tilde{G}_a) \subseteq K_{tol}$ and $\mathcal{L}(M/\tilde{G}_a) \cap K_{des} = \mathcal{L}(M'/\tilde{G}_a) \cap K_{des}$, we have
$$\mathcal{L}(M'/\tilde{G}_a) \not\subset \mathcal{L}(M/\tilde{G}_a).$$

The first condition in Problem 1 says that no unsafe state should be reachable, i.e., the system should always be within the tolerable language. The second condition says that the system should achieve as much desirable behavior as possible. Finally, the third requirement says that the behavior of the system should be as restrictive as possible outside of K_{des}. Later in our solution, we will show that condition (2) can only be achieved in a maximal manner rather than a supremal manner. However, condition (3) can be achieved in an infimal manner in the sense that $\mathcal{L}(M'/\tilde{G}_a) \not\subset \mathcal{L}(M/\tilde{G}_a)$ can be replaced by $\mathcal{L}(M/\tilde{G}_a) \subseteq \mathcal{L}(M'/\tilde{G}_a)$.

Note that, to guarantee safety, the system should not only avoid reaching X_F but also avoid reaching states that can reaching X_F uncontrollably. Therefore, we define

$$X_F^+ = \{x \in X : \exists s \in (\Sigma \setminus \Sigma_{c,a})^* \text{ s.t. } \tilde{\delta}_a(x, s) \in X_F\}$$

as the set of extended unsafe states. To avoid the case of no solution, we assume that the initial state x_0 cannot reach X_F via uncontrollable events, i.e., $x_0 \notin X_F^+$.

Finally, we define some operators that will be used to solve the problem. The *unobservable reach* of a set of states $q \subseteq X$ under a set of events $\gamma \in \Gamma_a$ is defined by

$$\mathrm{UR}_\gamma(q) = \left\{ x \in X : \begin{array}{c} \exists x' \in q, \exists w \in \gamma^* \text{ s.t.} \\ x = \tilde{\delta}_a(x', w) \wedge P_C(w) = \epsilon \end{array} \right\}.$$

The *observable reach* of a set of states $q \subseteq X$ upon the occurrence of event $\sigma \in \Sigma_o$ is defined by

$$\mathrm{NX}_\sigma(q) = \{x \in X : \exists x' \in q \text{ s.t. } x = \tilde{\delta}_a(x', \sigma) \vee x = \tilde{\delta}_a(x', \sigma^a)\}.$$

Then the *extended unobservable reach* of a set of states $q \subseteq X$ under a set of events $\gamma \in \Gamma_a$ is defined by

$$\mathrm{UR}_\gamma^+(q) = \mathrm{UR}_\gamma(q) \cup \left(\cup_{\sigma \in \Sigma_o \cap C(\gamma)} \mathrm{NX}_\sigma(\mathrm{UR}_\gamma(q)) \right).$$

V. General Solution

In this section, we provide an online solution to Problem 1. Our approach is motivated by the standard online supervisory control algorithm for safety only [22], [23]. However, in our case, we need to handle both desirable behavior and tolerant behavior.

A. Online Mitigation Strategy

The proposed online mitigation strategy is described in Algorithm 1. The general idea is to iteratively estimate the current state of the closed-loop system based on the decision and observation history by using operators $\mathrm{UR}(\cdot)$ and $\mathrm{NX}(\cdot)$ in an alternative manner. Specifically, we use parameter α to track the current observation, and $\mathcal{E}(\alpha)$ and $\hat{\mathcal{E}}(\alpha)$ represent, respectively, the current-state estimate with and without the unobservable tail. The initial setting is $\alpha = \epsilon$ since no event is observed and $\hat{\mathcal{E}}(\alpha) = \{x_0\}$ which is the initial state. Once a new control decision γ is issued, $\hat{\mathcal{E}}(\alpha)$ is updated to $\mathcal{E}(\alpha) = \mathrm{UR}_\gamma(\hat{\mathcal{E}}(\alpha))$, and once a new event σ is observed, $\hat{\mathcal{E}}(\alpha)$ is updated to $\hat{\mathcal{E}}(\alpha) = \mathrm{NX}_\sigma(\mathcal{E}(\alpha))$.

Algorithm 1: Online Mitigation M

```
Input: AE-attacked model $G_a = (X, \Sigma_a, \delta_a, x_0)$
       with $\Sigma_{uc,a}$, observation mapping $P_C$
Output: control decision $\gamma$ at each instant
1  $\alpha \leftarrow \epsilon$ ;
2  $\hat{\mathcal{E}}(\alpha) \leftarrow \{x_0\}$;
3  $\gamma \leftarrow$ ChooseAction($\hat{\mathcal{E}}(\alpha)$);
4  defense by disabling events not in $\gamma$;
5  $\mathcal{E}(\alpha) \leftarrow \mathrm{UR}_\gamma(\hat{\mathcal{E}}(\alpha))$;
6  while new event $\sigma \in \Sigma_o$ is observed do
7      $\alpha \leftarrow \alpha\sigma$;
8      $\hat{\mathcal{E}}(\alpha) \leftarrow \mathrm{NX}_\sigma(\hat{\mathcal{E}}(\alpha))$;
9      $\gamma \leftarrow$ChooseAction($\hat{\mathcal{E}}(\alpha)$);
10     defense by disabling events not in $\gamma$;
11     $\mathcal{E}(\alpha) \leftarrow \mathrm{UR}_\gamma(\hat{\mathcal{E}}(\alpha))$;
```

The key of Algorithm 1, which is also the main part different from the previous supervisory control algorithm, is how to choose a control decision γ at each instant based on the state estimate $\hat{\mathcal{E}}(\alpha)$; this part is done by procedure ChooseAction$(\cdot)$. To describe the functionality of ChooseAction$(\cdot)$, for each state estimate $\hat{\mathcal{E}}(\alpha) \in 2^X$, we define $\Gamma_\gamma^{des}(\hat{\mathcal{E}}(\alpha))$ as the set of events that are feasible in X_D via $\hat{\mathcal{E}}(\alpha)$ under γ, i.e.,

$$\Gamma_\gamma^{des}(q) = \left\{ \sigma \in \Sigma_a : \begin{array}{c} \exists x \in \hat{\mathcal{E}}(\alpha), s\sigma \in \gamma^* \text{ s.t.} \\ \tilde{\delta}_a(x, s\sigma) \in X_D \wedge P_C(s) = \epsilon \end{array} \right\}$$

Similarly, we define

$$\Gamma_\gamma(\hat{\mathcal{E}}(\alpha)) = \left\{ \sigma \in \Sigma_a : \begin{array}{c} \exists x \in \hat{\mathcal{E}}(\alpha), s\sigma \in \gamma^* \text{ s.t.} \\ \tilde{\delta}_a(x, s\sigma)! \wedge P_C(s) = \epsilon \end{array} \right\}$$

as the set of events that are feasible within its unobservable reach. Then ChooseAction($\hat{\mathcal{E}}(\alpha)$) essentially returns a control decision $\gamma \in \Gamma_a$ satisfying the following properties:

(i) $\mathrm{UR}_\gamma^+(\hat{\mathcal{E}}(\alpha)) \cap X_F^+ = \emptyset$;

(ii) For any γ' such that $\mathrm{UR}_{\gamma'}^+(\hat{\mathcal{E}}(\alpha)) \cap X_F^+ = \emptyset$, we have
$$\Gamma_\gamma^{des}(\hat{\mathcal{E}}(\alpha)) \not\subset \Gamma_{\gamma'}^{des}(\hat{\mathcal{E}}(\alpha)).$$

(iii) For any γ' such that $\mathrm{UR}_{\gamma'}^+(\hat{\mathcal{E}}(\alpha)) \cap X_F^+ = \emptyset$ and $\Gamma_\gamma^{des}(\hat{\mathcal{E}}(\alpha)) = \Gamma_{\gamma'}^{des}(\hat{\mathcal{E}}(\alpha))$, we have
$$\Gamma_\gamma(\hat{\mathcal{E}}(\alpha)) \subseteq \Gamma_{\gamma'}(\hat{\mathcal{E}}(\alpha)).$$

Therefore, ChooseAction($\hat{\mathcal{E}}(\alpha)$) essentially search for a control decision γ in a greedy manner such that (i) no unsafe state can be reached within its extended unobservable reach; and (ii) no safe decision can activate more feasible events than γ in the desirable language; and (iii) γ is the smallest set within those safe decisions activating same feasible events in the desirable language as γ. Note that such a decision γ may not be unique as decisions are compared in terms of set inclusion. To find such a γ, one direct approach is to perform a brute force search to test whether each $\gamma \in \Gamma_a$ satisfies the above three properties. Such a search take exponential time in the number of controllable events. Here we provide a more efficient approach to find such a decision γ, which is specified in Algorithm 2.

The basic idea for finding γ is to "add-and-test". We start to grow γ from the set of uncontrollable events by incrementally adding new controllable events in it. Specifically, we use set `EventList` to denote the set of potential events to be added. At each instant, we pick its ith event `EventList[`i`]` in the list to test whether it can be added and then take one of the following three actions:

- Delete the event from the list: this situation is captured by lines 6 and 7, which happens when adding the event may lead to unsafe states;
- Add the event to the control decision: this situation is captured by lines 9-11, which happens when adding the event is safe and it indeed introduces a new desirable behavior;
- Skip the event currently: this situation happens when adding an event is safe but do not introduce anything

desirable: either it is not feasible or it only introduces tolerable but not desirable strings. Note that we just skip the event rather than deleting it because it may become useful when some other events are added. This case is captured by line 14 in which we only add the counter.

Note that, once a new event is added, the counter i is reset to 0 in line 12. However, the counter increases when the event is skipped. The while-loop terminates when all events are added or the counter achieves the cardinality of the event list. The latter means that we have tested all events in the current list and nothing can be added anymore.

Algorithm 2: ChooseAction

```
Input: estimate $\hat{\mathcal{E}}(\alpha)$, controllable events $\Sigma_{a,c}$ and
       attacked system $\tilde{G}_a = (X, \Sigma_a, \tilde{\delta}_a, x_0)$
Output: control action $\gamma$
1  $\gamma \leftarrow \Sigma_{uc,a}$;
2  $i \leftarrow 0$;
3  EventList $\leftarrow \Sigma_{a,c}$;
4  while EventList $\neq \emptyset$ and $i < |$EventList$|$ do
5      $\sigma \leftarrow$ EventList$[i]$;
6      if $\mathrm{UR}^{+}_{\gamma\cup\{\sigma\}}(\hat{\mathcal{E}}(\alpha)) \cap X^{+}_{F} \neq \emptyset$ then
7          EventList $\leftarrow$ EventList $\setminus \{\sigma\}$;
8      else
9          if $\exists x \in \mathrm{UR}_{\gamma}(\hat{\mathcal{E}}(\alpha)) : \delta(x,\sigma) \in X_D$ then
10             $\gamma \leftarrow \gamma \cup \{\sigma\}$;
11             EventList $\leftarrow$ EventList $\setminus \{\sigma\}$;
12             $i \leftarrow 0$;
13         else
14             $i \leftarrow i + 1$
```

B. Correctness Proof and Complexity Analysis

Now we formally prove the correctness of Algorithms 1. Hereafter, M is always denoted as the supervisor (mitigation strategy) synthesized by Algorithm 1.

First, we show that the synthesized supervisor M is safe.

Lemma 1: $\mathcal{L}(M/\tilde{G}_a) \subseteq K_{tol}$.

Second, we show that M achieves tolerable but not desirable behavior in the most restrictive manner.

Lemma 2: For any M' such that $\mathcal{L}(M'/\tilde{G}_a) \subseteq K_{tol}$ and $\mathcal{L}(M'/\tilde{G}_a) \cap K_{des} = \mathcal{L}(M'/\tilde{G}_a) \cap K_{des}$, we have $\mathcal{L}(M/\tilde{G}_a) \subseteq \mathcal{L}(M'/\tilde{G}_a)$.

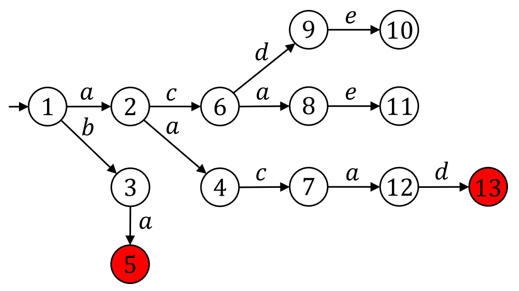
Finally, we show that M achieves the desirable behavior in the least restrictive manner.

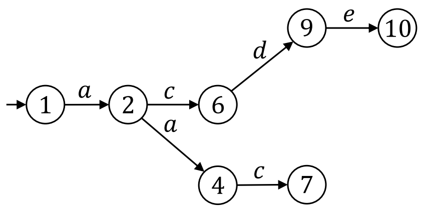
Lemma 3: For any M' such that $\mathcal{L}(M/\tilde{G}_a) \subseteq K_{tol}$, we have $\mathcal{L}(M/\tilde{G}_a) \cap K_{des} \not\subset \mathcal{L}(M'/\tilde{G}_a) \cap K_{des}$.

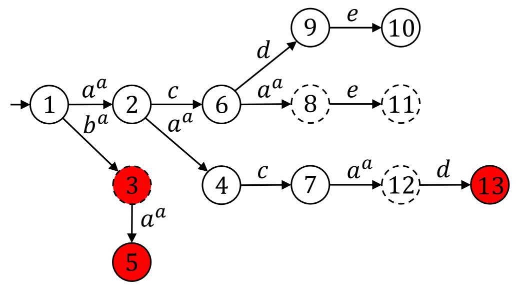
Based on the above three lemmas, we obtain the following theorem.

Theorem 1: Algorithm 1 correctly solve Problem 1.

Remark 2: Based on our solution, one can see that requirement (3) in Problem 1 can actually be replaced by $\mathcal{L}(M/\tilde{G}_a) \subseteq \mathcal{L}(M'/\tilde{G}_a)$. That is, tolerable but non-desirable behavior can be achieved in an *infimal* manner. However, requirement (2) in Problem 1 cannot be replaced by $\mathcal{L}(M'/\tilde{G}_a) \cap K_{des} \subseteq \mathcal{L}(M/\tilde{G}_a) \cap K_{des}$ since there may exist different incomparable but local maximal solutions. By using different event orders for EventList, one may obtain different solutions.


(a) G


(b) S_P/G


(c) $\tilde{G}_a$

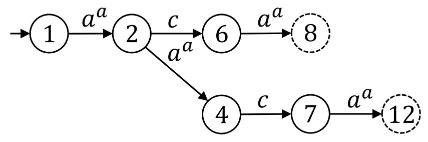

(d) $M/\tilde{G}_a$

Fig. 2: Examples for the online mitigation algorithm.

C. Illustrative Example

Finally, we explain the proposed mitigation strategy by an illustrative example.

Example 4: Let us consider system G shown in Figure 2(a) with $\Sigma_c = \{a, b, d, e\}$ and $\Sigma_o = \{c\}$. Suppose that the system is originally controlled by S_P to achieve the desirable specification $\mathcal{L}(S_P/G) = \overline{\{acde, aac\}}$ and S_P/G is shown as Fig. 2(b). Specifically, the supervisor disables event b initially and disables event a after observing c.

Now we assume that the vulnerable events set is $\Sigma_v = \{a, b\}$, defendable events set is $\Sigma_d = \{b\}$ and $X_{bad} = \{5, 13\}$ is the set of unsafe states. Clearly, under AE-attack, both unsafe state 5 and 13 can be reached. Furthermore, the strategy in [1] cannot prevent unsafe states.

To apply our online mitigation strategy, first, we construct DFA $\tilde{G}_a$ shown in Figure 2(c) with $\Sigma_{c,a} = \{b^a, d, e\}$. Note that the state space of $\tilde{G}_a$ is already partitioned as $X_D = \{1, 2, 4, 6, 7, 9, 10\}$, $X_T = \{3, 8, 11, 12\}$ and $X_F = \{5, 13\}$. Note that since state 3 can reach unsafe state 5 via uncontrollable event a^a, we have $X_F^+ = \{3, 5, 13\}$. Now we construct the mitigation strategy M using Algorithm 1. To start, we have $\alpha = \epsilon$ and $\hat{\mathcal{E}}(\alpha) = \{1\}$. We call procedure CHOOSEACTION($\{1\}$) as detailed in Algorithm 2. Initially, we have $\gamma \leftarrow \{c, a^a\}$ and we assume events in `EventList` are order by $[b^a, d, e]$ First, we test whether or not event b^a can be added to γ. However, by doing so, state 3 which is in X_F^+ will be reached; therefore, we delete it. Then we test events d and e in order and both events are not feasible within the unobservable reach. Therefore, we skip both and return $M(\epsilon) = \{c, a^a\}$, i.e., the mitigation module needs to defend event b initially. Next, when event c is observed, we have $\alpha = c$ and $\hat{\mathcal{E}}(\alpha) = \{6, 7\}$. Again, we call procedure CHOOSEACTION($\{6, 7\}$) to compute the current decision. First, we check event b^a, which is not feasible, and we skip it. Then by adding event d, states 13 will be reached and we delete d. For event e, although it is feasible from state 8, which is in the unobservable reach of $\hat{\mathcal{E}}(\alpha) = \{6, 7\}$, it is not feasible in X_D. Therefore, event e also needs to be skipped because it does not contributes to the desirable behavior although it is safe. Then we obtain the returned decision $M(c) = \{c, a^a\}$. The overall closed-loop system under the mitigate strategy M is shown in Figure 2(d). The closed-loop system sacrifices two desirable strings acd and $acde$, reaches two tolerable but not desirable states 8 and 12, but successfully avoids reaching unsafe states.

VI. CONCLUSION

In this paper, we considered the attack mitigation problem for a supervisory control system. Our contributions are summarized as follows. First, we introduced a general framework for attack mitigation that precisely distinguishes between controllable events and defendable events. Second, we formulated the attack mitigation problem as a tolerant control problem under partial-observation that aims to maximize the desirable behavior while minimizing tolerable but non-desirable behavior. The new formulation is more general and practical because the previous mitigation problem can easily be unsolvable. Finally, we provided an online solution to the attack mitigation problem. Our solution also generalizes the existing tolerant control problem from the full observation setting to the partial observation setting. Throughout this paper, we consider actuator enablement attack to state our result. In fact, our mitigation framework and solution strategy is generic and can also to applied to other types of attacks.

REFERENCES

[1] L. Carvalho, Y.-C. Wu, R. Kwong, and S. Lafortune, "Detection and mitigation of classes of attacks in supervisory control systems," *Automatica*, vol. 97, pp. 121–133, 2018.

[2] R. Meira-Góes, E. Kang, R. Kwong, and S. Lafortune, "Stealthy deception attacks for cyber-physical systems," in *IEEE Conference on Decision and Control (CDC)*, 2017, pp. 4224–4230.

[3] R. Meira-Góes, R. Kwong, and S. Lafortune, "Synthesis of sensor deception attacks for systems modeled as probabilistic automata," in *American Control Conference (ACC)*, 2019, pp. 5620–5626.

[4] R. Meira-Góes, H. Marchand, and S. Lafortune, "Towards resilient supervisors against sensor deception attacks," in *IEEE Conference on Decision and Control (CDC)*, 2019, pp. 5144–5149.

[5] P. Lima, L. Carvalho, and M. Moreira, "Detectable and undetectable network attack security of cyber-physical systems," in *International Workshop on Discrete Event Systems (WODES)*, 2018, pp. 179–185.

[6] R. Fritz, P. Schwarz, and P. Zhang, "Modeling of cyber attacks and a time guard detection for ICS based on discrete event systems," in *European Control Conference (ECC)*, 2019, pp. 4368–4373.

[7] R. Fritz and P. Zhang, "Modeling and detection of cyber attacks on discrete event systems," in *International Workshop on Discrete Event Systems (WODE)*, 2018, pp. 285–290.

[8] R. Su, "Supervisor synthesis to thwart cyber attack with bounded sensor reading alterations," *Automatica*, vol. 94, pp. 35–44, 2018.

[9] Y. Zhu, L. Lin, and R. Su, "Supervisor obfuscation against actuator enablement attack," in *European Control Conference (ECC)*, 2019, pp. 1760–1765.

[10] L. Lin, Y. Zhu, and R. Su, "Towards bounded synthesis of resilient supervisors against actuator attacks," in *IEEE Conference on Decision and Control (CDC)*, 2019, pp. 7659–7664.

[11] L. Lin, S. Thuijsman, Y. Zhu, S. Ware, R. Su, and M. Reniers, "Synthesis of supremal successful normal actuator attackers on normal supervisors," in *American Control Conference (ACC)*, 2019, pp. 5614–5619.

[12] Q. Zhang, Z. Li, C. Seatzu, and A. Giua, "Stealthy attacks for partially-observed discrete event systems," in *IEEE Conference on Emerging Technologies and Factory Automation (ETFA)*, 2018, pp. 1161–1164.

[13] C. Gao, C. Seatzu, Z. Li, and A. Giua, "Multiple attacks detection on discrete event systems," in *IEEE Conference on Systems, Man and Cybernetics (SMC)*, 2019, pp. 2352–2357.

[14] M. Wakaiki, P. Tabuada, and J. Hespanha, "Supervisory control of discrete-event systems under attacks," *Dynamic Games and Applications*, vol. 9, no. 4, pp. 965–983, 2019.

[15] Y. Wang and M. Pajic, "Supervisory control of discrete event systems in the presence of sensor and actuator attacks," in *IEEE Conference on Decision and Control (CDC)*, 2019, pp. 5350–5355.

[16] A. Khoumsi, "Sensor and actuator attacks of cyber-physical systems: A study based on supervisory control of discrete event systems," in *International Conference on Systems and Control (ICSC)*, 2019, pp. 176–182.

[17] A. Rashidinejad, B. Wetzels, M. Reniers, L. Lin, Y. Zhu, and R. Su, "Supervisory control of discrete-event systems under attacks: an overview and outlook," in *European Control Conference (ECC)*, 2019, pp. 1732–1739.

[18] D. Thorsley and D. Teneketzis, "Intrusion detection in controlled discrete event systems," in *IEEE Conference on Decision and Control (CDC)*, 2006, pp. 6047–6054.

[19] Y. Wang and M. Pajic, "Attack-resilient supervisory control of discrete-event systems," in *IEEE Conference on Decision and Control (CDC)*, 2019, pp. 2015–2020.

[20] A. Paoli and S. Lafortune, "Safe diagnosability for fault-tolerant supervision of discrete-event systems," *Automatica*, vol. 41, no. 8, pp. 1335–1347, 2005.

[21] S. Lafortune and F. Lin, "On tolerable and desirable behaviors in supervisory control of discrete event systems," *Discrete Event Dynamic Systems*, vol. 1, no. 1, pp. 61–92, 1991.

[22] N. Hadj-Alouane, S. Lafortune, and F. Lin, "Centralized and distributed algorithms for on-line synthesis of maximal control policies under partial observation," *Discrete Event Dynamic Systems*, vol. 6, no. 4, pp. 379–427, 1996.

[23] X. Yin and S. Lafortune, "A uniform approach for synthesizing property-enforcing supervisors for partially-observed discrete-event systems," *IEEE Transactions on Automatic Control*, vol. 61, no. 8, pp. 2140–2154, 2016.

[24] ——, "Synthesis of maximally permissive supervisors for partially-observed discrete-event systems," *IEEE Transactions on Automatic Control*, vol. 61, no. 5, pp. 1239–1254, 2016.

[25] ——, "Synthesis of maximally-permissive supervisors for the range control problem," *IEEE Transactions on Automatic Control*, vol. 62, no. 8, pp. 3914–3929, 2017.

[26] C. Cassandras and S. Lafortune, *Introduction to Discrete Event Systems*, 2nd ed. Springer, 2008.