Formulating Attacks with Supervisory Control ⋆

Ana Maria Mainhardt * **Andrew Wintenberg** **
Stéphane Lafortune ** **Anne-Kathrin Schmuck** *

* *Max Plank Institute for Software Systems Kaiserslautern, Germany*
** *University of Michigan, Ann Arbor, MI 48109 USA*

Abstract: We present a framework for the modelling and synthesis of sensor and actuator attacks on discrete event systems. Initially, we consider systems composed of a single plant and supervisor subject to an attack that can modify the supervisor's observations and control actions. The problem of designing stealthy attacks that can always inflict damage on the system is formulated as a standard supervisory control problem. Next, we show how to apply our framework to a decentralised system. Distributed attackers, one for each subsystem, are realised as nonconflicting supervisors that must coordinate to achieve their individual goals.

Keywords: Supervisory control and automata, Security, Sensor and actuator attacks

1. INTRODUCTION

We consider the problem of synthesizing distributed attacks that coordinate to disrupt their respective subsystem's behaviours while remaining undetected. First, we present a framework modelling a single attacker on a plant in closed-loop with a supervisor. We assume they can modify the supervisor's sensor readings from the plant – by its *sensor attack* component – and the control action emitted by the supervisor – by its *actuator attack* component. We show that these attackers can be realised as supervisors solving a basic supervisory control problem (Wonham and Cai, 2019). In the case of full observation, we can synthesise *supremal* solutions which identify all of the system's behaviours that are vulnerable to attack. We then extend this framework to the distributed setting with multiple subsystems, each subject to a different attacker. The implemented attackers coordinate by synchronization on common events. This framework is similar to that of Tai et al. (2023), which presents a corresponding *centralised* synthesis algorithm for distributed attacks as distributed supervisors satisfying global specifications. Our main contributions and key differences over the aforementioned work are twofold: *1. Our proposed framework models a rich set of attacks with a compact representation. 2. We apply* distributed *synthesis methods to design distributed attacks over decentralised plants with local specifications.*

To the first point, our framework models *preemption.* The sensor attacker may only be able to insert a fixed number of events that preempt the plant, after which the plant may execute an event or the insertions may continue. To the second point, we utilise the assume-guarantee algorithm of Mainhardt and Schmuck (2022) for distributed supervisor synthesis to design distributed attacks. This algorithm possesses a number of desirable qualities from the viewpoint of attackers. Notably, it maintains *information integrity*, as computations are performed locally with minimal exchange of local information between subsystems. This permits the consideration of attackers with distinct goals that do not wish to share their private knowledge beyond what is needed for coordination.

Our framework draws from the many existing DES models of sensor or deception attacks and actuator or enablement/disablement attacks, as summarised by Hadjicostis et al. (2022) [1]. We consider attacks that must remain covert, or stealthy, while always being able to achieve their goals, known as *for all* attacks. Supervisory control algorithms have been used extensively for attack synthesis; see Hadjicostis et al. (2022) and the references therein. Many such works model the system as a game between the supervisor and the attacker or the similar bipartization structure. Alternatively, we view the attacker as a controller or coordinator of decoupled plant and supervisor subsystems. This approach results in compact state space representations, which is critical for efficiency in synthesis for the distributed case.

We consider attacks which do not eavesdrop on the control actions output by the supervisor. Instead, they predict supervisor outputs using partial observation of its inputs. This approach avoids introducing events modelling attacker insertions and deletions, as well as supervisor control actions. Also, our parameterization of preemption generalizes the notion of preempting states in Zhang et al. (2022) and race conditions in Meira-Góes et al. (2020).

Example 1. Consider the manufacturing system depicted in Fig. 1. This system consists of a robotic arm that grabs items from a buffer and then drops them in one of two drop-off zones. We consider two attackers with distinct goals. The first attempts to force the arm to try to grab an item from the drop-off zone. The second attempts to force the buffer to overflow by having multiple items arrive at once. Using their model of the local subsystem, each attacker may independently find solutions achieving their goals; however, they may conflict when the subsystems are run in parallel in the actual system. This leads us to ask the following question: *Is it possible for attackers to cooperate*

⋆ Research supported in part by the US NSF under grant ECCS-2144416, and by the DFG Emmy Noether grant SCHM 3541/1-1.

[1] Due to strict page limitations, our bibliography is unusually brief.

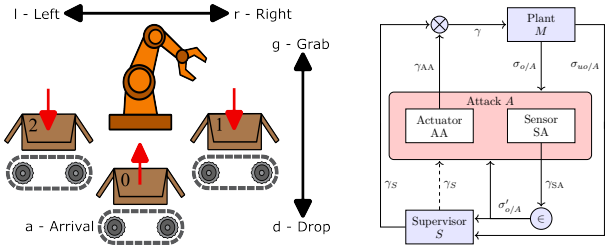

Fig. 1. Motivating manufacturing example. The dashed edge represents eavesdropping of the supervisor control action by the attacker, absent in our framework.

in designing nonconflicting solutions while exchanging a minimal amount of information? △

2. PRELIMINARIES

We assume familiarity with basic concepts related to strings, languages, and automata (Wonham and Cai, 2019). A deterministic (partial) finite automaton over an alphabet Σ is defined as a tuple $\mathbf{\Lambda} = (Q, \Sigma, \delta, q_0, Q_m)$ in the standard way and denoted by a boldface character. For a plant M represented by automaton $\mathbf{M}$ with alphabet $\Sigma = \Sigma_c \dot\cup \Sigma_{uc} = \Sigma_o \dot\cup \Sigma_{uo}$, a supervisor S is a map of control actions $\Gamma : \mathcal{L}(\mathbf{M}) \to 2^{\Sigma}$ such that, for all $w, w' \in \mathcal{L}(\mathbf{M})$, and for the natural projection $\mathrm{P} : \Sigma^* \to \Sigma_o^*$, we have that: (i) $\Sigma_{uc} \subseteq \Gamma(w)$, hence satisfying *controllability*, and (ii) $\Gamma(w) = \Gamma(w')$ if $\mathrm{P}(w) = \mathrm{P}(w')$, satisfying *observability* (Wonham and Cai, 2019). An automaton $\mathbf{S}$ *implements* Γ if, for all $\sigma \in \Sigma$, we have that $\sigma \in \Gamma(w)$ if and only if $w\sigma \in \mathcal{L}(\mathbf{S})$; we assume that transitions by events in Σ_{uo} only appear as selfloops, as they cannot be observed by the supervisor and, therefore, cannot change its control action. The generated and marked languages of the closed-loop behaviour of the supervised plant are the respective languages of the product $\mathbf{M} \,||\, \mathbf{S}$. We say that S satisfies a specification $\mathbf{K}$ over $\Sigma' \subseteq \Sigma$ if $\mathcal{L}_m(\mathbf{M} \,||\, \mathbf{S}) \subseteq \mathrm{P}^{-1}\,\mathcal{L}_m(\mathbf{K})$, with $\mathrm{P} : \Sigma^* \to \Sigma'^*$. Finally, we say that S is nonblocking if $\mathbf{M} \,||\, \mathbf{S}$ is nonblocking.

3. ATTACKER FORMULATION

First, we consider attacks on a single plant, M, in closed loop with a single supervisor, S, enforcing a nonblocking behaviour. In order to apply the theory of supervisory control, we view the attacker as a supervisor A with malicious objectives over an augmented plant formed by both the original M and supervisor S. The attacker updates its internal state based on observed events as input, while outputting control actions consisting of a set of events that depends on the current state.

The architecture of our attacks is depicted in Fig. 1. Here, an attack A is composed of a sensor attack, SA, and an actuator attack, AA. The latter tampers with the control actions of the supervisor, while the former alters the events perceived by the supervisor. In order to distinguish events that occur in M from the ones S observes under attack, we keep different copies of each event that SA can manipulate in the alphabets of the plant and of the supervisor, which are from now on denoted by Σ_M and Σ_S, respectively. The intersection of these sets consists of events that cannot be affected by SA and is denoted by Σ_{NSA}; it includes all events of the plant that are unobservable to the attacker, denoted by $\Sigma_{M,uo/A}$. Moreover, we assume that A cannot observe plant events unobservable by S, i.e., $\Sigma_{uo/S} \subseteq \Sigma_{M,uo/A} \subseteq \Sigma_{NSA}$. This implicitly means the attacker does not have access to more sensors than the supervisor does.

Based on the sequence of events observed by the attacker so far, SA outputs a control decision $\gamma_{SA} \subseteq \Sigma_S$. One event σ' from this set is then picked to be observed by S; this event is also observed by the attacker, because we assume SA would not insert an event that the attacker itself cannot detect. In this paper we consider that there is a nondeterministic mechanism that makes the choice of selecting σ' from γ_{SA}; this is analogous to the common assumption that the plant selects one event to be executed from the set enabled by the supervisor. We model SA this way to be able to represent all feasible edits it can do that lead to a successful attack. So the problem of finding the most permissive behaviour in supervisory control theory translates here to the synthesis of all possible strategies that satisfy the specifications for the attack.

The sequence of events the supervisor observes is then composed of both events executed by the plant that are not deleted by SA and events inserted by SA that did not occur in the plant. Based on this sequence, S outputs a control action γ_S, which is the set of events it decides to enable in the plant. Under attack, this action is affected by AA, which outputs its own control action γ_{AA}. Finally, the set of enabled events the plant receives, denoted here by γ, is then a function that depends both on γ_S and γ_{AA}.

Even though SA and AA are implemented separately, they are synthesised as a single attacker. Hence, both receive as inputs the events σ executed by the plant – as long as the attacker can observe them – and the events σ' picked from γ_{SA} to be sent to the supervisor. That is, the sequence composed of such events impact both γ_{SA} and γ_{AA}. Moreover, each possible control action of the supervisor, γ_S, is not explicitly represented as an event in our modelling framework, in contrast to Lin et al. (2020) for instance. Here, since the attacker has knowledge of the model of S, whether or not γ_S – and then also γ – can be inferred by the attacker depends on its observability settings when compared to the ones of the supervisor. E.g., if all the events observable by S are also observable by A, then we may consider the attacker to be *eavesdropping*, as it can exactly predict the supervisor's action γ_S, and hence, also γ; otherwise, this may not be the case.

Decoupling of events We decouple the alphabets of the plant and of the supervisor, Σ_M and Σ_S, in order to distinguish events executed by the plant from the ones observed by the supervisor. We have, in each one of these alphabets, distinct copies of each event that can be attacked by SA. The set of copies for the plant and for the supervisor are, respectively, denoted by $\Sigma_{M,SA}$ and $\Sigma_{S,SA}$. We use the subscripts, M and S, to indicate to which set an event σ_M or σ_S belongs. We do not keep copies of the events that cannot be attacked by SA, so they are in the intersection set $\Sigma_{NSA} = \Sigma_M \cap \Sigma_S$. Thus, we have that $\Sigma_M = \Sigma_{M,SA} \dot\cup \Sigma_{NSA}$, and $\Sigma_S = \Sigma_{S,SA} \dot\cup \Sigma_{NSA}$. To map the events from M to S, we define the isomorphism $\Theta_S : \Sigma_M \to \Sigma_S$, such that $\Theta_S(\sigma) = \sigma$, for all $\sigma \in \Sigma_{NSA}$, and $\Theta_S(\sigma_M) = \sigma_S$, for all $\sigma_M \in \Sigma_{M,SA}$. Note that $\Sigma_{S,SA} = \Theta_S(\Sigma_{M,SA})$. With a slight abuse of notation, this

mapping can be extended to strings and languages. We also define Θ_M as the inverse of Θ_S. We use the same letter with different subscripts to denote events that are the copy of each other; e.g., $\alpha_M = \Theta_M(\alpha_S)$ and $\alpha_S = \Theta_S(\alpha_M)$.

Supervised plant under attack The plant M is modelled by an automaton $\mathbf{M}$ over alphabet Σ_M, where all its states are marked. Under attack, we define the supervisor in terms of Σ_S, and not Σ_M, as follows. Its alphabet can be partitioned as $\Sigma_S = \Sigma_{c/S} \dot\cup \Sigma_{uc/S}$, where $\Sigma_{c/S}$ are its controllable events, and $\Sigma_{uc/S}$ its uncontrollable ones. Under partial observation, its alphabet can also be partitioned into its observable and unobservable events, $\Sigma_{o/S}$ and $\Sigma_{uo/S}$, respectively, and we assume that $\Sigma_{NSA} \supseteq \Sigma_{uo/S}$. The *control action* of the supervisor S is given by the map $\Gamma_S : \Theta_S\ \mathcal{L}(\mathbf{M}) \to 2^{\Sigma_S}$ implemented by a given automaton $\mathbf{S}$ over Σ_S, where all the states are marked.

Attacker A The attack we wish to synthesise is represented by an automaton $\mathbf{A}$ over Σ_A, which is the global alphabet of the attacked system. The behaviour of the attacked system is given by $\mathbf{A} \parallel \mathbf{G}$, where $\mathbf{G}$ is the *attack plant*, composed of the models of the plant, $\mathbf{M}$, and of the sensor and actuator attackers, respectively $\mathbf{G}_{SA}$ and $\mathbf{G}_{AA}$, which embed the structure of the supervisor $\mathbf{S}$. We denote by L_A the language accepted by the attacked system, i.e., $L_A = \mathcal{L}_m(\mathbf{A} \parallel \mathbf{G})$. We use the notation $P_\square$ for natural projections of the form $P_\square : \Sigma_A^* \to \Sigma_\square^*$, where $\square$ is any symbol such that $\Sigma_\square \subseteq \Sigma_A$.

Next, we introduce the attack constraints denoted by $\mathcal{A} = \big(\Sigma_{M,o/A}, \Sigma^{ins}_{M,SA}, \Sigma^{del}_{M,SA}, \Sigma^{ins}_{M,AA}, \Sigma^{del}_{M,AA}, n_{pre}\big)$, and define $\mathbf{G}_{SA}$ and $\mathbf{G}_{AA}$ reflecting such constraints. In Sec. 4, we properly define the alphabet Σ_A, the attack plant $\mathbf{G}$, and the attack specification $\mathbf{K}$ for damage and stealthiness. Finally, we show that $\mathbf{A}$ is equivalent to a (malicious) supervisor for this attack plant and this specification, with respect to the partition into controllable and uncontrollable events, denoted by $\Sigma_A = \Sigma_{c/A} \dot\cup \Sigma_{uc/A}$, and observable and unobservable events, $\Sigma_A = \Sigma_{o/A} \dot\cup \Sigma_{uo/A}$. From $\mathbf{A}$ we can then extract the control actions for the sensor and actuator attacker, γ_{SA} and γ_{AA}, respectively.

Deletion, replacement and insertion by SA If we ignore the constraints and requirements for A developed in the remainder of Sec. 3, an *all-out sensor attack* completely decouples the plant behaviour and supervisor observations; this attack realises the closed-loop behaviour over $\Sigma_M \cup \Sigma_S$ given by shuffle $\mathcal{L}(\mathbf{M}) \parallel \Sigma_S^*$. In practice, however, there may be limitations on how the sensor attacker can edit events, which we model as follows. A sensor deletion prevents an event executed by the plant to be observed by the supervisor. We denote the set of events that SA can delete by $\Sigma^{del}_{M,SA} \subseteq \Sigma_{M,SA}$, and its copy in Σ_S by $\Sigma^{del}_{S,SA} = \Theta_M(\Sigma^{del}_{M,SA}) \subseteq \Sigma_{S,SA}$. Thus, if an event $\alpha_M \in \Sigma^{del}_{M,SA}$ occurs in the plant, then SA may decide whether or not the supervisor observes its copy α_S before the execution of the next event by the plant, or before SA inserts other events. This insertion, on the other hand, happens whenever SA feeds to the supervisor an observation β_S when β_M was not the last event executed by the plant, or after another sensor insertion — since we assume there are no delays in the occurrence of an event and its observation by the supervisor. We denote the set of events that can be inserted by $\Sigma^{ins}_{S,SA} \subseteq \Sigma_{S,SA}$, and its copy in Σ_M by $\Sigma^{ins}_{M,SA} = \Theta_M(\Sigma^{ins}_{S,SA}) \subseteq \Sigma_{M,SA}$. A replacement consists of a deletion followed by an insertion.

We formally define these attacks as follows. Take any $w \in \overline{L_A}\backslash\{\epsilon\}$, any $p < w$, and any $s \in \Sigma^*$. For any $\alpha_M \in \Sigma^{del}_{M,SA}$ such that $w = p\alpha_M s$, we say α_M is *deleted* in w after $p\alpha_M$ if either $s = \epsilon$ and $w \in L_A$, or $s \neq \alpha_S t$ for all $t \in \Sigma^*$. For any $\beta_S \in \Sigma^{ins}_{S,SA}$, if $p\beta_S \leq w$ and $p \neq t\beta_M$ for all $t \in \Sigma^*$, then β_S is *inserted* in w after p. Finally, β_S *replaces* α_M after p in w (for $\alpha \neq \beta$) if α_M is deleted and β_S inserted after $p\alpha_M$.

Events in $\Sigma^{ins}_{S,SA} \cap \Sigma^{del}_{S,SA}$ impose no restriction, as they can be both inserted and deleted. Likewise, there is no restriction on events in Σ_{NSA} that are not attacked by SA, because they can only occur when feasible in the plant and are observed immediately by the supervisor. However, for events that can be exclusively inserted or deleted by SA, we model the following constraints. First, every event $\beta_M \in \Sigma^{ins}_{M,SA}\backslash\Sigma^{del}_{M,SA}$ should be immediately followed by its copy β_S; this is represented by $\mathbf{G}^{ins\backslash del}_{SA,\beta_M}$ in Fig. 2, together with a nonblockingness requirement expressed by its marking. Second, for every $\alpha_M \in \Sigma^{del}_{M,SA}\backslash\Sigma^{ins}_{M,SA}$, we have the automaton $\mathbf{G}^{del\backslash ins}_{SA,\alpha_M}$ given in Fig. 2, where we restrict the copy α_S to only occur immediately after α_M.

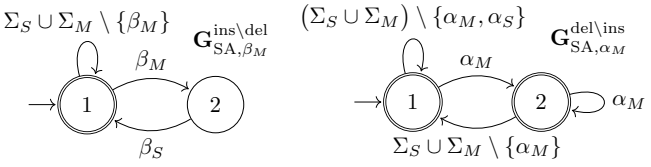

Fig. 2. Models of deletion and insertion constraints for SA.

Preemption of plant events by SA A sensor attack SA may be able to insert a sequence of events before the supervisor can issue a new control action that is subsequently executed by the plant. We refer to this ability as *preemption*. We suppose that after replacing an event, SA can only preempt the plant with a number n_{pre} of insertions. This constraint requires that plant events can be executed depending on the previous number of consecutive insertions. We could express this as state-dependent controllability, as in Wang and Cai (2009). However, in order to formulate our problem as one of basic supervisory control, we opt for capturing this constraint with the standard notion of controllability, by introducing the alphabet $\Sigma_{end} = \{end_c, end_{uc}\}$ to explicitly model the end of a sequence of sensor insertions. Its event end_c is controllable, representing that the sensor attack chooses to stop inserting. In contrast, the event $end_{uc} \in \Sigma_{end}$ is uncontrollable, representing the sequence of insertions is interrupted by a plant event.

We can then construct automata $\mathbf{G}^{pre}_{SA}(n_{pre})$ as in Fig. 3; we omit the input n_{pre} when it is clear from context. If we assume that SA can preempt forever, then we use the automaton at the left ($n_{pre} = \infty$), otherwise, the one at the right ($0 \leq n_{pre} < \infty$). In the latter, the transition from state 1 to 0 occurs with the event end_c, because we assume SA can always do replacement; moreover, from state 0 with events in Σ_{NSA} there is a transition to state 2, instead of 1, since these events can always be observed

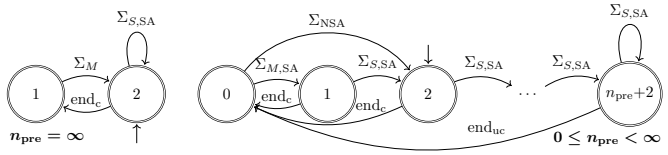

Fig. 3. Automata $\mathbf{G}^{\mathrm{pre}}_{\mathrm{SA}}(n_{\mathrm{pre}})$ modeling preemption. If $n_{\mathrm{pre}} = 0$, then states 2 and $n_{\mathrm{pre}} + 2$ are merged, the outgoing transition to 0 only occurs by $\mathrm{end_{uc}}$.

by the supervisor, i.e., they cannot be deleted and, hence, a subsequent insertion does not constitute a replacement.

Model for SA By taking the synchronous composition of $\mathbf{G}^{\mathrm{pre}}_{\mathrm{SA}}$ (Fig. 3) and the automata $\mathbf{G}^{\mathrm{ins}\backslash\mathrm{del}}_{\mathrm{SA},\beta_M}$ and $\mathbf{G}^{\mathrm{del}\backslash\mathrm{ins}}_{\mathrm{SA},\alpha_M}$ (Fig. 2), we obtain the model for the sensor attacker, $\mathbf{G}_{\mathrm{SA}}$. The alphabet for SA is then $\Sigma_{\mathrm{SA}} = \Sigma_M \cup \Sigma_S \cup \Sigma_{\mathrm{end}}$.

Example 2. For illustration, we use a simple model of the arm component in the manufacturing system of Example 1, given by the plant $\mathbf{M}^0$ depicted in Fig. 4. Its events are 'l' moves *left*, 'r' moves *right*, 'g' *grab* item, 'd' *drop* item. The indicated forbidden transitions of $\mathbf{M}^0$ are disabled by the supervisor $\mathbf{S}^0$ while the attacker $\mathbf{A}^0$ seeks to execute them to inflict damage.

We suppose this attacker can only insert the sensor event ℓ, i.e., $\Sigma^{\mathrm{ins},0}_{M,\mathrm{SA}} = \{\ell\}$, and can observe all events, i.e., $\Sigma^0_{M,\mathrm{o}/A} = \Sigma^0_M$. For $n_{\mathrm{pre}} = 1$, the model $\mathbf{G}^0_{\mathrm{SA}}$ is depicted in Fig. 5. Note that, after the plant event r, SA can only insert one event ℓ_S before the plant can interrupt it, which is represented by the uncontrollable event $\mathrm{end_{uc}}$. △

Now we introduce the remaining constraints from $\mathcal{A}$, and define the model $\mathbf{G}_{\mathrm{AA}}$ for the actuator attacker, AA. This component of the attacker acts by inserting events to, or deleting from, the current control action of the supervisor, γ_S, or from what the attacker can infer from it, in the case of partial observation. Denote by $\Sigma^{\mathrm{del}}_{M,\mathrm{AA}}$ and $\Sigma^{\mathrm{ins}}_{M,\mathrm{AA}}$, respectively, the set of events in Σ_M that can be deleted and inserted by AA; their intersection may be nonempty. We assume that $\Sigma_{\mathrm{NSA}} \cap \Sigma^{\mathrm{ins}}_{M,\mathrm{AA}} = \emptyset$, for if an event in this intersection is added by AA to the control action of the supervisor and subsequently executed by the plant, then SA cannot delete it from the supervisor's observation, so the attack would not be stealthy. Note that there are copies of the events from $\Sigma^{\mathrm{del}}_{M,\mathrm{AA}}$ in Σ_S only if they can also be attacked by SA.

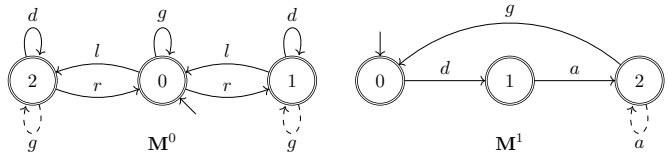

Fig. 4. Models of the arm and buffer subsystems with forbidden transitions indicated with dashes.

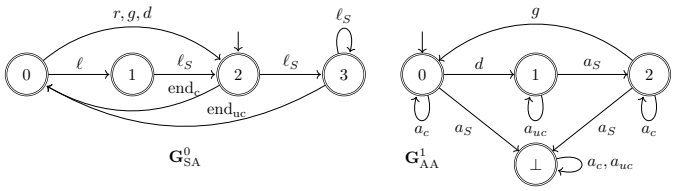

Fig. 5. The sensor attack model for the arm subsystem and actuator attack model for the buffer subsystem.

State-dependent controllability and renaming Events that can be inserted but not deleted by AA require special attention. When present in the control action of the supervisor, they cannot be deleted, in which case they should be considered as uncontrollable for the attacker. However, when enabled by the supervisor, the attacker may decide whether or not to insert them; hence, in this scenario, they are controllable. Similar to the issue of preemption for the sensor attack, rather than enforcing a state-dependent notion of controllability, we will split the problematic events into controllable and uncontrollable copies. Let us denote by Σ_{r} the set $\Sigma^{\mathrm{ins}}_{M,\mathrm{AA}} \backslash \Sigma^{\mathrm{del}}_{M,\mathrm{AA}}$, and introduce the alphabets $\Sigma_{\mathrm{r,c}} : \{\sigma_{\mathrm{r,c}} \,|\, \sigma_M \in \Sigma_{\mathrm{r}}\}$ and $\Sigma_{\mathrm{r,uc}} : \{\sigma_{\mathrm{r,uc}} \,|\, \sigma_M \in \Sigma_{\mathrm{r}}\}$. Then, we define the renaming map $\mathrm{r} : \Sigma_{\mathrm{r,c}} \cup \Sigma_{\mathrm{r,uc}} \to \Sigma_{\mathrm{r}}$ such that, for all $\sigma_{\mathrm{r,c}} \in \Sigma_{\mathrm{r,c}}$ and $\sigma_{\mathrm{r,uc}} \in \Sigma_{\mathrm{r,uc}}$, $r(\sigma_{\mathrm{r,c}}) = r(\sigma_{\mathrm{r,uc}}) = \sigma_M$.

Model for AA We construct the actuator attacker model from the given automaton $\mathbf{S}$; thus, we denote it as a function of the supervisor by $\mathbf{G}_{\mathrm{AA}}(S)$; when the context is clear, we omit the input S. The idea is that, at each state of $\mathbf{G}_{\mathrm{AA}}$, the transitions by events in Σ_M give us the control action γ_{AA}, which in turn depends on γ_S. Therefore, except for uncertainties arising from partial observation, we need to keep track of current state of the supervisor. Given supervisor $\mathbf{S} = (Q_S, \Sigma_S, \delta_S, q_{S_0}, Q_{S_m})$, we define the actuator attack model $\mathbf{G}_{\mathrm{AA}} = (Q_{\mathrm{AA}}, \Sigma_{\mathrm{AA}}, \delta_{\mathrm{AA}}, q_{\mathrm{AA}_0}, Q_{\mathrm{AA}})$, with states $Q_{AA} = Q_S \cup \{\perp\}$, and actuator attack alphabet $\Sigma_{\mathrm{AA}} = \big(\Sigma_M \backslash \Sigma_{\mathrm{r}}\big) \cup \Sigma_S \cup \Sigma_{\mathrm{r,c}} \cup \Sigma_{\mathrm{r,uc}} \subseteq \Sigma_A$, and such that, for all $q \in Q_{\mathrm{AA}}$, the function δ_{AA} is defined as follows.

(1) $\forall \sigma_S \in \Sigma_{S,\mathrm{SA}}. \delta_S(q, \sigma_S)! \Rightarrow \delta_{\mathrm{AA}}(q, \sigma_S) = \delta_S(q, \sigma_S)$, otherwise $\delta_{\mathrm{AA}}(q, \sigma_S) = \perp$.
(2) $\forall \sigma \in \Sigma_{\mathrm{NSA}} . \delta_S(q, \sigma)! \Rightarrow \delta_{\mathrm{AA}}(q, \sigma) = \delta_S(q, \sigma)$.
(3) $\forall \sigma_M \in \Sigma_{M,\mathrm{SA}} \backslash \Sigma^{\mathrm{ins}}_{M,\mathrm{AA}} . \delta_S(q, \sigma_S)! \Rightarrow \delta_{\mathrm{AA}}(q, \sigma_M) = q$.
(4) $\forall \sigma_M \in \Sigma^{\mathrm{ins}}_{M,\mathrm{AA}} \backslash \Sigma_{\mathrm{r}} . \delta_{\mathrm{AA}}(q, \sigma_M) = q$.
(5) $\forall \sigma_M \in \Sigma_{\mathrm{r}} . \delta_S(q, \sigma_S)! \Rightarrow \delta_{\mathrm{AA}}(q, \sigma_{\mathrm{r,uc}}) = q$, otherwise $\delta_{\mathrm{AA}}(q, \sigma_{\mathrm{r,c}}) = q$.

The transitions by events in Σ_S not only allows us to update the current state of the supervisor, but also to know what is its control action on this state. For that reason, we copy all the transitions from δ_S into δ_{AA} – see clauses (1) and (2). However, due to attacks, the supervisor may observe sequences that do not correspond to the domain of Γ_S; thus we add a state $\perp$ and, still in clause (1), complete the transition function with respect to events in $\Sigma_{S,\mathrm{SA}}$. We do not do so with respect to Σ_{NSA} because this set is also part of Σ_M: in each state of $\mathbf{G}_{\mathrm{AA}}$, the transitions by events in Σ_M give us the control action of AA, which cannot insert events in Σ_{NSA}, so if these transitions are not in $\mathbf{S}$, they also cannot be in $\mathbf{G}_{\mathrm{AA}}$.

The attacker does not need to tamper with every control action of the supervisor. Therefore, for each state in Q_S, and for every event in $\Sigma_{S,\mathrm{SA}}$ enabled there, we add in $\mathbf{G}_{\mathrm{AA}}$, through clause (3), a selfloop by its copy in $\Sigma_{M,\mathrm{SA}}$; we use selfloops since these copies cannot be observed by S, so they do not change its control action. Clause (4) represents that A can also insert events from $\Sigma^{\mathrm{ins}}_{M,\mathrm{AA}}$ even when their copies in Σ_S are not present in Γ_S. Events σ_M in Σ_{r} require special treatment, so we only deal with them in clause (5), as follows: when S is disabling their copies in $\Sigma_{S,\mathrm{SA}}$, we have a selfloop by the controllable version $\sigma_{\mathrm{r,c}}$,

representing the possibility of insertion; otherwise, we use the uncontrollable version $\sigma_{\mathrm{r,uc}}$.

Example 3. We model the buffer subsystem from Example 1 with the plant $\mathbf{M}^1$ depicted in Fig. 4. The events g and d for grabbing and dropping an item are common with the arm plant $\mathbf{M}^0$ from Example 2, while the event a denotes an item *arriving* to the buffer. This model assumes the arm starts having grabbed an item and that at most one item is present in the system at a time. The supervisor $\mathbf{S}^1$ and attacker $\mathbf{A}^1$ again seek to disable and execute the indicated forbidden transitions, respectively. This attacker can both delete the sensor event a as well as insert it as an actuator event, i.e, $\Sigma^{\mathrm{del},1}_{M,\mathrm{SA}} = \Sigma^{\mathrm{ins},1}_{M,\mathrm{AA}} = \{a\}$. In this case their actuator attack model $\mathbf{G}^1_{\mathrm{AA}}$ is depicted in Fig. 5, assuming all events are observable $\Sigma^1_{M,\mathrm{o}/A} = \Sigma^1_M$. As the attacker cannot disable the event a when it is enabled by the supervisor, the outgoing transition at state 1 in $\mathbf{G}^1_{\mathrm{AA}}$ is labelled by the uncontrollable event a_{uc}. This is in contrast to the event a at state 2 of S^1 that, while disabled by the supervisor, can be inserted by the actuator attacker, as represented by the controllable event a_c in $\mathbf{G}^1_{\mathrm{AA}}$. △

4. MONOLITHIC SYNTHESIS OF ATTACKER

The monolithic system under attack is described by the *attack plant* $\mathbf{G}$, which is constructed by composing the models of the plant under attack $\mathbf{M}$, the sensor attack $\mathbf{G}_{\mathrm{SA}}$, and the actuator attack $\mathbf{G}_{\mathrm{AA}}$, defined in Sec. 3. This composition should synchronise the events of $\Sigma_{\mathrm{r}} \subseteq \Sigma_M$ – present in the alphabets of both $\mathbf{M}$ and $\mathbf{G}_{\mathrm{SA}}$ – with their split copies $\Sigma_{\mathrm{r,c}}$ and $\Sigma_{\mathrm{r,uc}}$, used to model different controllability statuses in the actuator attack model $\mathbf{G}_{\mathrm{AA}}$.

In order to do so, let us finally define the alphabet of attacker as $\Sigma_A = \Sigma_{\mathrm{AA}} \cup \Sigma_{\mathrm{SA}} \backslash \Sigma_{\mathrm{r}}$, and introduce a modified composition operator $||^{\mathrm{r}}$, which is similar to standard synchronous composition, but using a more general map P_{r} in place of the natural projection. We define this map by $\mathrm{P}_{\mathrm{r}} : \Sigma^*_A \to \Sigma^*_{\mathrm{SA}}$, such that $\mathrm{P}_{\mathrm{r}}(\sigma) = \mathrm{r}(\sigma)$, if $\sigma \in \Sigma_{\mathrm{r,c}} \cup \Sigma_{\mathrm{r,uc}}$, and $\mathrm{P}_{\mathrm{r}}(\sigma) = \sigma$, otherwise; note that it can be naturally extended to strings and to languages. Next, for languages $L_1 \subseteq \Sigma^*_A$ and $L_{2,3} \subseteq \Sigma^*_{\mathrm{SA}}$, we define $||^{\mathrm{r}}$ such that $L_1 \,||^{\mathrm{r}}\, L_2 = L_2 \,||^{\mathrm{r}}\, L_1 = \mathrm{P}_{\mathrm{r}}^{-1}(L_2) \cap L_1$, and $L_2 \,||^{\mathrm{r}}\, L_3 = \mathrm{P}_{\mathrm{r}}^{-1}(L_2) \cap \mathrm{P}_{\mathrm{r}}^{-1}(L_3)$. Finally, this operation can be extended to automata as follows. For automata $\mathbf{\Lambda}_1$ over $\Sigma_1 \subseteq \Sigma_A$ or $\Sigma_1 \subseteq \Sigma_{\mathrm{SA}}$, and $\mathbf{\Lambda}_2$ over $\Sigma_2 \subseteq \Sigma_{\mathrm{SA}}$, $\mathbf{\Lambda} = \mathbf{\Lambda}_1 \,||^{\mathrm{r}}\, \mathbf{\Lambda}_2 = \mathbf{\Lambda}_2 \,||^{\mathrm{r}}\, \mathbf{\Lambda}_1$ is such that $\mathcal{L}(\mathbf{\Lambda}) = \mathcal{L}(\mathbf{\Lambda}_1) \,||^{\mathrm{r}}\, \mathcal{L}(\mathbf{\Lambda}_2)$ and $\mathcal{L}_m(\mathbf{\Lambda}) = \mathcal{L}_m(\mathbf{\Lambda}_1) \,||^{\mathrm{r}}\, \mathcal{L}_m(\mathbf{\Lambda}_2)$.

The attack plant used for the synthesis of A is then given by $\mathbf{G} = \mathbf{M} \,||^{\mathrm{r}}\, \mathbf{G}_{\mathrm{SA}} \,||^{\mathrm{r}}\, \mathbf{G}_{\mathrm{AA}}$ over Σ_A. In this plant, SA controls the supervisor copy of events it can edit, $\Sigma_{S,\mathrm{SA}}$, as well as the ending event $\mathrm{end}_{\mathrm{c}}$. Likewise, AA controls plant events that it can delete from the supervisor action, $\Sigma^{\mathrm{del}}_{M,\mathrm{AA}}$, as well as copies of events it can insert when they are not in the supervisor action, but not delete otherwise, $\Sigma_{\mathrm{r,c}}$. In addition to the plant events that are explicitly unobservable to the attacker, it can also not observe the end of insertion events in Σ_{end}, since consecutive end events would reveal the occurrence of unobservable plant events to the attack. This is not an issue if all plant events are observed by the attacker, in which case we can treat the end events as observable as well. Then in total

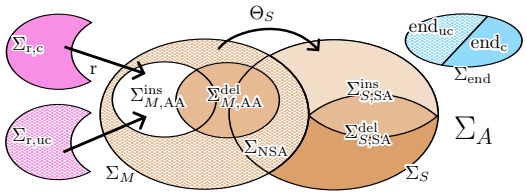

Fig. 6. The attack events Σ_A, with controllable events in solid colours and uncontrollable ones in patterns.

we define $\Sigma_{\mathrm{c}/A} = \Sigma_{S,\mathrm{SA}} \cup \{\mathrm{end}_{\mathrm{c}}\} \cup \Sigma^{\mathrm{del}}_{M,\mathrm{AA}} \cup \Sigma_{\mathrm{r,c}}$, and $\Sigma_{\mathrm{o}/A} = \Sigma_A \backslash \{\Sigma_{M,\mathrm{uo}/A} \cup \Sigma_{\mathrm{end}}\}$. The events of the attack and their controllability status is depicted in Fig. 6.

We notate by $L_{\mathrm{dam}} \in \mathcal{L}(\mathbf{M})$ the language whose strings correspond to damages the attacker A intends to cause to the plant M. This language is a specification for A and can be incorporated in its synthesis procedure by a trim automaton $\mathbf{K}_{\mathrm{dam}}$ such that $\mathcal{L}_m(\mathbf{K}_{\mathrm{dam}}) = L_{\mathrm{dam}}$. If, for all $s \in L_{\mathrm{dam}}$ and $s' \in \mathcal{L}(\mathbf{M}) \backslash L_{\mathrm{dam}}$, we have that s and s' do not correspond to the same state in $\mathbf{M}$, then we can omit $\mathbf{K}_{\mathrm{dam}}$ in the synthesis of A and, instead, define the marking of $\mathbf{M}$ such that $\mathcal{L}_m(\mathbf{M}) = L_{\mathrm{dam}}$. We then use the definition below, adapted to our context from Lin et al. (2020), to express whether A can always inflict damage.

Definition 1. (Damage-nonblocking). For any attacker A over any plant M and any supervisor S, and given any damage language L_{dam}, we say A is damage-nonblocking if $\mathbf{A} \,||\, \mathbf{G}$ is nonblocking and $\mathcal{L}_m(\mathbf{A} \,||\, \mathbf{G}) \subseteq L_{\mathrm{dam}}$.

Definition 2. (Stealthiness). For any attacker A over any plant M and any supervisor S, let $L_S = \mathrm{P}_S\, \mathcal{L}(\mathbf{A} \,||\, \mathbf{G})$ and $L_{\mathrm{stealth}} = \mathcal{L}(\mathbf{S}) \,||\, \Theta_S \mathcal{L}(\mathbf{M})$. Then A is *always stealthy* if $L_S \subseteq L_{\mathrm{stealth}}$.

Stealthiness is a safety condition, so it can be expressed by an automaton $\mathbf{K}_{\mathrm{stealth}}$ over Σ_S given as specification in the synthesis of the attacker. For A to be always stealthy, this automaton is the one that recognises and accepts the language L_{stealth}, and can be trivially obtained from $\mathbf{S}$ and $\mathbf{M}$. Combining damage-nonblocking and stealthiness, we consider the overall specification $\mathbf{K} = \mathbf{K}_{\mathrm{stealth}} \,||^{\mathrm{r}}\, \mathbf{K}_{\mathrm{dam}}$.

We can now state the attack synthesis problem as a supervisory control problem. Using the plant model $\mathbf{G}$ from Sec. 3 and specifications $\mathbf{K}$ attacks can be implemented as supervisors in the following result.

Theorem 1. For any plant M, supervisor S, and damage language L_{dam}, there is an always stealthy and damage-nonblocking attack A with constraints $\mathcal{A} = \big(\Sigma_{M,\mathrm{o}/A}, \Sigma^{\mathrm{ins}}_{M,\mathrm{SA}}, \Sigma^{\mathrm{del}}_{M,\mathrm{SA}}, \Sigma^{\mathrm{ins}}_{M,\mathrm{AA}}, \Sigma^{\mathrm{del}}_{M,\mathrm{AA}}, n_{\mathrm{pre}}\big)$ if and only if A is a nonblocking supervisor for the plant $\mathbf{G}$, specification $\mathbf{K}$, controllable events $\Sigma_{\mathrm{c}/A}$, and observable events $\Sigma_{\mathrm{o}/A}$ as defined previously.

Enforcing the controllability and observability settings for A, we synthesise a trim automaton $\mathbf{A}$ as a supervisor for $\mathbf{G}$ satisfying $\mathbf{K}$. The control actions for SA and AA after observing the string w are constructed from the attack action $\Gamma_A(w)$ as $\Gamma_{\mathrm{SA}}(w) = \Gamma_A(w) \cap \Sigma_{\mathrm{o}/S}$ and $\Gamma_{\mathrm{AA}}(w) = \mathrm{P}_{\mathrm{r}}(\Gamma_A(w)) \cap \Sigma_M$. The set of events $\Gamma(w)$ ultimately enabled at the plant M combines the actions of AA and S given by $\Gamma(w) = \Gamma_{\mathrm{AA}}(w) \cap \big(\Sigma^{\mathrm{ins}}_{M,\mathrm{AA}} \cup \Theta_M \Gamma_S(\mathrm{P}_S\, w)\big)$.

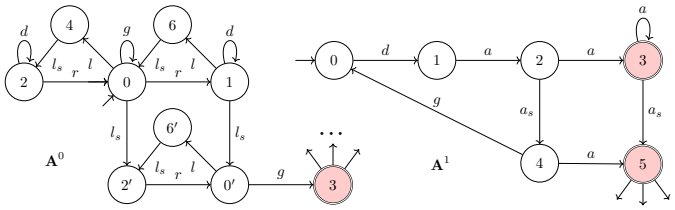

Fig. 7. Attacks on the arm and buffer subsystems. Ellipses represent behaviour after the attack goal is reached. The events end_c and end_uc have been projected out, and events renamed to simplify presentation.

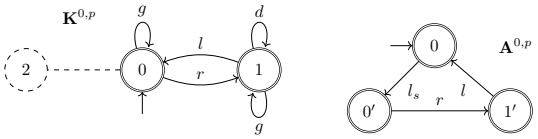

Fig. 8. The specification requiring the arm to never visit zone 2 and resulting attack on the arm subsystem.

Example 4. We now synthesise local attacks for the manufacturing system of Examples 2 and 3. First, we construct the attack plants $\mathbf{G}^0$ and $\mathbf{G}^1$ assuming no preemption and constraints given by $\mathcal{A}^0 = (\Sigma_M^0, \{\ell\}, \emptyset, \emptyset, \{d\}, 0)$ and $\mathcal{A}^1 = (\Sigma_M^1, \emptyset, \{a\}, \{a\}, \emptyset, 0)$. The attack specifications $\mathbf{K}^0$ and $\mathbf{K}^1$ capture stealthiness and damage achieved by executing forbidden transitions. Applying supervisory control synthesis with full observation, we obtain the supremal local attacks $\mathbf{A}^0$ and $\mathbf{A}^1$ depicted in Fig. 7. △

Example 5. While the local attacks from Example 4 can achieve their goals without preemption ($n_\text{pre} = 0$), this is not always the case. Suppose that the first attacker instead wanted to ensure the arm never moves to zone 2. We can verify that attacks only exist with preemption ($n_\text{pre} > 0$) such as the one depicted in Fig. 8. In words, the attacker must insert the event ℓ_S to mislead the supervisor into thinking the arm is at zone 2 so that it disables the event ℓ. This must be done before the arm has a chance to actually move left with ℓ into zone 2. △

5. DISTRIBUTED SYNTHESIS OF ATTACKS

We now consider a decentralised plant with a number of subsystems, each subject to a different attacker modelled with the architecture of Fig. 1 as in the centralised setting. For simplicity, we assume each local attacker has full observation of its subsystem's events. Then attacks can be found as solutions to a distributed supervisory control problem as in the following result.

Theorem 2. Given plants $\{M^i\}$, supervisors $\{S^i\}$, damage languages $\{L^i_\text{dam}\}$, and attacker constraints $\{\mathcal{A}^i\}$ for $i \in \{1, \ldots, n\}$, there are nonconflicting, stealthy and damage-nonblocking attacks $\{A^i\}$ if and only if they are nonconflicting supervisors for the plants $\{\mathbf{G}^i\}$, and the specifications $\{\mathbf{K}^i\}$, with local controllability requirements as constructed in Sec. 4. The attacks are nonconflicting if $||_{i=0}^n \overline{\mathcal{L}_m(\mathbf{A}^i \,||\, \mathbf{G}^i)} = \overline{||_{i=0}^n \mathcal{L}_m(\mathbf{A}^i \,||\, \mathbf{G}^i)}$.

To solve this problem for the case of two subsystems, we employ the assume-guarantee algorithm of Mainhardt and Schmuck (2022). Critically, this approach is distributed, requiring only local computations and exchange of information *during synthesis*. It iterates between *negotiation*, where attackers exchange the commonly observed behaviour of their solutions, and enforcing a local condition sufficient to ensure they are globally nonconflicting. We demonstrate this procedure with the following example.

Example 6. Recall the attacks $\mathbf{A}^0$ and $\mathbf{A}^1$ that were designed independently for the arm and buffer subsystems of the manufacturing system in Example 4. While they both achieve their goals locally, they conflict when executed in parallel, synchronizing on the common events $\{g, d\}$. This can be verified with a global conflict check. If the attack on the arm does not drop the held item with event d before achieving its goal, the buffer attack may deadlock at state 0 in $\mathbf{A}^1$, where no more items can arrive to overflow the buffer. Conversely, after achieving overflow at state 3 in $\mathbf{A}^1$, the buffer attack may continue to force more items to arrive with event a, thus preventing the arm attack to erroneously pickup an item to reach its state 3 in $\mathbf{A}^0$.

To avoid this possibility, both attackers are incentivised to coordinate in designing their attacks, for example with the assume-guarantee algorithm. They apply the first step of the algorithm, negotiation, to refine their solutions. While not guaranteed by negotiation, a global conflict check shows that these solutions are nonconflicting. As a consequence of Theorem 2 from Mainhardt and Schmuck (2022), these attacks are then guaranteed to be supremal. However, performing this global conflict check requires exchanging the attackers' private information. To avoid this, the attackers can apply the second step of the algorithm, enforcing a local condition sufficient for nonconflict. In this case, the resulting solutions are empty, indicating there is not enough shared information for the attackers to coordinate in this way. Instead, the first attacker A^0 can share information with A^1 about the arm moving left ℓ and its sensor attack on this event ℓ_S. Applying the synthesis algorithm again, now with the common events $\{g, d, \ell, \ell_S\}$, results in nonempty solutions that can be implemented to stealthily damage the system. △

REFERENCES

Hadjicostis, C.N., Lafortune, S., Lin, F., and Su, R. (2022). Cybersecurity and Supervisory Control: A Tutorial on Robust State Estimation, Attack Synthesis, and Resilient Control. In *IEEE CDC*, 3020–3040.

Lin, L., Zhu, Y., and Su, R. (2020). Synthesis of covert actuator attackers for free. *JDEDS*, 30, 561–577.

Mainhardt, A.M. and Schmuck, A.K. (2022). Assume-Guarantee Synthesis of Decentralised Supervisory Control. *IFAC WODES*, 55(28), 165–172.

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*.

Tai, R., Lin, L., Zhu, Y., and Su, R. (2023). Synthesis of Distributed Covert Sensor-Actuator Attackers. *IEEE TAC*, 1–15.

Wang, P. and Cai, K.Y. (2009). Supervisory control of discrete event systems with state-dependent controllability. *International Journal of Systems Science*, 40(4).

Wonham, W.M. and Cai, K. (2019). *Supervisory control of discrete-event systems*. Communications and Control Engineering. Springer.

Zhang, Q., Seatzu, C., Li, Z., and Giua, A. (2022). Selection of a stealthy and harmful attack function in discrete event systems. *Scientific Reports*, 12(1), 16302.