Synthesis of Stealthy Deception Attacks with Limited Resources

Uday Shankar
Student in School of Computer Science
Carnegie Mellon University
Pittsburgh, USA
us@cmu.edu

Advisor: Eunsuk Kang
Institute for Software Research
Carnegie Mellon University
Pittsburgh, USA
eskang@cmu.edu

***Abstract*—We study the security of Cyber-Physical Systems (CPS) with respect to attacks on the supervisory control layer. Specifically, we consider the sensory deception attacker model, originally proposed in [1], in which attackers aim to coerce the system into an unsafe state by altering the sensor readings received by the supervisor, while also concealing their presence from the supervisor. We modify this model by additionally placing a bound on the number of modifications made by the attacker. We present a method for the synthesis of such attackers, constructing and using an *Insertion Deletion Attack* (IDA) structure as our key technical tool.**

***Index Terms*—discrete event systems, supervisory control, cyber-physical systems, cyber-security, deception attacks**

I. Introduction

Cyber-Physical Systems (CPS) are characterized by the presence of computation that transforms input from physical sensors into physical effects. Typical examples include anti-lock braking systems in cars and automated control systems in power plants. As evidenced by such examples, CPS often appear in settings where safety is a vital consideration and correctness cannot be compromised upon. This has led to the development of an entire field focused on formal methods for verifying correctness properties of CPS [2].

Not all undesirable behavior in CPS is caused by flaws in the system design. There is also the possibility of an external agent interfering with the system with the intention of inducing some sort of malfunction. Several instances of such attacks on CPS have been reported, with the prime example being StuxNet [3]. These real-world examples necessitate a careful study of attacks on CPS.

This paper considers *sensory deception* attacks, a particular kind of attack on supervisor-controlled CPS. In such attacks, the attacker attempts to manipulate the supervisor into allowing the system to reach an unsafe state by interfering with the information flow from the system to the supervisor. The attacker additionally is required to remain *stealthy*, in the sense that the behavior of the attacker-compromised system should be indistinguishable from the normal behavior of the (uncompromised) system from the point of view of the supervisor. This attacker model was originally proposed in [1], and this paper is largely an extension of that work.

However, in this paper we consider attackers less powerful than those discussed in [1]. We do this by further restricting our attackers by placing a finite bound on the total number of actions they can take. This restriction can be used to model various real-life scenarios. For instance, we may want to construct minimally invasive attackers, simply because being minimally invasive is good practice for attackers who want to avoid detection. We may also want to model a situation in which certain kinds of attacker moves are impossible. For example, if the system and supervisor communicate via a wireless network in which the attacker has a foothold, the attacker may be able to send bogus information to the supervisor, but it might not be able to tamper with the legitimate information sent by the system.

We present a method of synthesis of these restricted attackers. The method revolves around the construction of a bipartite graph which allows the attacker to keep track of all knowable information about the states of the system and supervisor and deduce how different actions might affect these states. This graph is called an *Insertion-Deletion Attack* structure (IDA), and the method of its construction is similar to the construction of the same name in [1]. We additionally provide an implementation for the construction of the IDA given a system, supervisor, and other appropriate parameters.

The paper is organized as follows: in section II, we present the chosen CPS model, and define some important background concepts and notation. In section III, we formally define the aforementioned "restricted attacker." Section IV defines the construction of the IDA, and section V proves that this IDA serves as an appropriate search space containing precisely the kinds of attackers we are looking for. Finally, section VI discusses possible directions for future work.

II. Preliminaries

We model the system G as a deterministic finite automaton (DFA) with no marked states (X, Σ, δ, x_0), where

- X is the (finite) set of states.
- Σ is the (finite) set of events.
- $\delta : X \times \Sigma \to X$ is a partial function that describes how the system changes states as a response to the events.
- $x_0 \in X$ is the initial state.

Note that δ is only a partial function, despite the automaton being labeled "deterministic." This is different from the typical

definition of a DFA, but is the usual definition used in CPS work [2].

We define a function $\delta^* : X \times \Sigma^* \to X$ recursively, via

$$\begin{aligned} \delta^*(x, \varepsilon) &= x \\ \delta^*(x, se) &= \delta(\delta^*(x, s), e) \end{aligned} \tag{1}$$

and as is typical, we abuse notation by using δ when we mean δ^*. This is a harmless abuse, however, because δ and δ^* agree on their common domain of $X \times \Sigma$. Note that δ^* is still a partial function.

The *language generated* by G is defined in the usual manner:

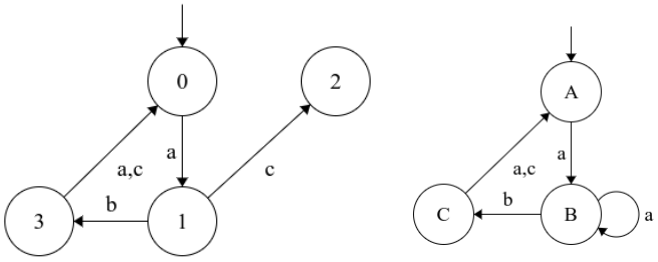
$$\mathcal{L}(G) = \{s \in \Sigma^* \mid \delta(x_0, s)!\} \tag{2}$$

where $\delta(x_0, s)!$ asserts that the partial function δ is defined on the argument (x_0, s). The set $\mathcal{L}(G)$ represents the "raw" behavior of the system G by itself. The system by itself may have some undesirable behavior. For instance, it may contain states deemed unsafe that are reachable from the initial state.

In order to prevent the system from reaching such unsafe states, we define the notion of a *supervisor*. Supervisors restrict the behavior of the system to a "safe subset" by observing the sequence of events executed by the system and issuing corresponding *control decisions*, which selectively enable or disable events from occurring. This leads to the definition of a supervisor as a (potentially partial)[1] function $S : \Sigma^* \to 2^\Sigma$, where $2^\Sigma = \mathcal{P}(\Sigma)$ is the set of all subsets of Σ. So, if the system has executed a sequence of events $s \in \Sigma^*$, the supervisor enables the set of events $S(s)$ and disables all others, meaning that the next event executed by the system must be a member of $S(s)$. However, in practice, supervisors are not this powerful. We consider two impediments to the supervisor:

- *Uncontrollable events* are events that the supervisor can never disable. Such events might exist in practice because of limited physical capability of the supervisor to disable such events. The presence of uncontrollable events induces a partition $\Sigma = \Sigma_\mathrm{c} \sqcup \Sigma_\mathrm{uc}$, where Σ_c is the set of controllable events (the ones that the supervisor has the capability to disable), and Σ_uc is the set of uncontrollable events.
- *Unobservable events* are events that are executed by the system, but which the supervisor cannot sense, due to e.g. a limited number of installed sensors. Such events are particularly painful for supervisors to deal with because their presence means that the supervisor cannot be sure of the system's state (more on this later). The presence of unobservable events induces a partition $\Sigma = \Sigma_\mathrm{o} \sqcup \Sigma_\mathrm{uo}$, where Σ_o is the set of observable events (the ones the supervisor can sense) and Σ_uo is the set of unobservable events.

[1]This function might be partial because we don't require it to be defined on sequences of events in Σ^* that are impossible under the control of the supervisor.



(a) System (b) Supervisor

Fig. 1: A system, along with a supervisor that ensures the system never enters the unsafe state 2. Example provided by the authors of [1].

Using these concepts, we can define a more general version of a supervisor called an *admissible partial observation* supervisor (with respect to $\Sigma_\mathrm{uc}, \Sigma_\mathrm{c}, \Sigma_\mathrm{uo}, \Sigma_\mathrm{o}$). Such a supervisor is defined as a function $S_P : \Sigma_\mathrm{o}^* \to 2^\Sigma$ with the property that every γ in the range of S_P satisfies $\Sigma_\mathrm{uc} \subseteq \gamma$. The domain of S_P is only Σ_o^* because the supervisor cannot sense, and hence cannot react, to the occurrence of unobservable events, and the second condition (called admissibility) forces the supervisor to never disable an uncontrollable event. Henceforth, we only work with admissible partial observation supervisors.

Now, we have all the pieces required to formally define the controlled behavior of a system G with respect to the supervisor S_P, denoted $\mathcal{L}(S_P/G)$. First, we define a projection function $P_o : \Sigma^* \to \Sigma_\mathrm{o}^*$, which simply deletes all unobservable events to give the supervisors point of view of the system's execution.

$$\begin{aligned} P_\mathrm{o}(\varepsilon) &= \varepsilon \\ P_\mathrm{o}(se) &= \begin{cases} P_\mathrm{o}(s)e & e \in \Sigma_\mathrm{o} \\ P_\mathrm{o}(s) & e \in \Sigma_\mathrm{uo} \end{cases} \end{aligned} \tag{3}$$

Now, we define $\mathcal{L}(S_P/G)$ recursively, in a manner that mirrors our earlier informal discussion.

$$\begin{aligned} &\varepsilon \in \mathcal{L}(S_P/G) \\ &s \in \mathcal{L}(S_P/G) \wedge e \in S_P(P_\mathrm{o}(s)) \wedge se \in \mathcal{L}(G) \\ &\quad \iff se \in \mathcal{L}(S_P/G) \end{aligned} \tag{4}$$

It is difficult to work directly with this abstract definition of a supervisor, so we instead work with the *realization* of a supervisor as a DFA $R = (Q, \Sigma, \mu, q_0)$. If we define

$$\Gamma_R(q) = \{e \in \Sigma \mid \mu(q, e)\} \tag{5}$$

as the *active event set* at a state $q \in Q$, then the control decisions are defined as follows.

$$S_P(s) = \Gamma_R(\mu(q_0, s)) \tag{6}$$

In order for this realization to be admissible, we require that $(\forall q \in Q)(\forall e \in \Sigma_\mathrm{uc})(\mu(q, e)!)$. This simply asserts that uncontrollable events are always active, and hence never disabled.

Example II.1. Consider the system G and supervisor R (constructed with $\Sigma = \Sigma_{\mathrm{o}} = \{a, b, c\}, \Sigma_{\mathrm{c}} = \{b, c\}$) depicted in figure 1. Observe that in the system, $\delta(0, ac) = 2$ and hence $ac \in \mathcal{L}(G)$ and state 2 is reachable in the raw behavior of the system. However, $ac \notin \mathcal{L}(S_P/G)$, because after reading a, the supervisor R is in state B, where the event c is disabled. In fact, it is not difficult to see that the system is in state 1 if and only if the supervisor is in state B. Since the only way for the system to reach state 2 is to be in state 1 and process the event c, this means that state 2 is unreachable in the system under the control of the supervisor.

With $\Sigma_{\mathrm{c}} = \{a, b\}$, the supervisor must immediately disable the event a from occurring to prevent the system from reaching state 2. If both a and c are uncontrollable, then it is impossible to design a supervisor that prevents the system from reaching state 2.

Lastly, we make a few definitions that will be useful in later parts of the paper. We define

$$UR_\gamma(S) = \{x \in X \mid (\exists y \in S)(\exists s \in \Sigma_{\mathrm{uo}}^* \cap \gamma) \\ (x = \delta(y, s))\} \tag{7}$$

as the unobservable reach of the set of states S under the control decision γ. This is useful for producing estimates for the state set of the system, as given a current estimate S and a control decision γ, it tells us all possible states the system could evolve to using only unobservable events, which are invisible to the supervisor. We also define

$$NX_e(S) = \{x \in X \mid (\exists y \in S)(x = \delta(y, e))\} \tag{8}$$

as a function which simply lifts the system's transition function δ to work on sets. Let

$$\Gamma = \{\Gamma_R(q) \mid q \in Q\} \tag{9}$$

be the set of all *control decisions* of the supervisor. Finally, we define a supervisor $\tilde{R}$ whose state set is $\tilde{Q} = Q \sqcup \{\texttt{dead}\}$. The idea is that the supervisor transitions to state `dead` when it detects abnormal behavior; we model this by defining the transition function of $\tilde{R}$ as $\tilde{\mu}(x, e) = \mu(x, e)$ whenever $\mu(x, e)$ is defined, and $\tilde{\mu}(x, e) = \texttt{dead}$ otherwise. Note that the control decisions of $\tilde{R}$ are not the same as the control decisions of R. We will refer to $\tilde{R}$ as the supervisor *augmented with attacker detection*.

III. Problem Statement

In a sensor deception attack, the attacker has the capability to tamper with the information flow from the system to the supervisor. To be a bit more specific, we assume the attacker has compromised a set $\Sigma_{\mathrm{a}} \subseteq \Sigma_{\mathrm{o}}$ of the events. Whenever the system emits an event in Σ_{a}, the attacker can choose to mask the occurrence of this event from the supervisor. The attacker also has the capability to generate fake events, making the supervisor think the system executed an event when in actuality nothing happened. These position of the attacker relative to the system, supervisor, and projection P_{o} is depicted in figure 2.

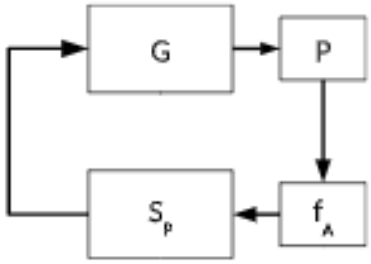


Fig. 2: Interaction of the system G, the attacker f_A and the supervisor S_P. P is the partial observation projection, denoted in this paper as P_{o}. Image from [1].

To formally define such an attacker, it is convenient to distinguish between inserted events, deleted events, and true system events. We define sets $\Sigma_{\mathrm{a}}^{\mathrm{i}} = \{e^i \mid e \in \Sigma_a\}$ and $\Sigma_{\mathrm{a}}^{\mathrm{d}} = \{e^d \mid e \in \Sigma_a\}$, which are copies of Σ_{a} with additional marking to label them as insertions or deletions. Note that $\Sigma_{\mathrm{a}}, \Sigma_{\mathrm{a}}^{\mathrm{i}}, \Sigma_{\mathrm{a}}^{\mathrm{d}}$ are pairwise disjoint. We also define $\Sigma_{\mathrm{a}}^{\mathrm{e}} = \Sigma_{\mathrm{a}}^{\mathrm{i}} \sqcup \Sigma_{\mathrm{a}}^{\mathrm{d}}$ are the set of edited events. With this, we can define an attacker as follows (definition taken from [1]).

Definition III.1. Given a system G and a subset $\Sigma_{\mathrm{a}} \subseteq \Sigma_{\mathrm{o}}$, an attacker is defined as a (partial) function $f_A : (\Sigma_{\mathrm{o}} \cup \Sigma_{\mathrm{a}}^{\mathrm{e}})^* \times (\Sigma_{\mathrm{o}} \sqcup \{\varepsilon\}) \to (\Sigma_{\mathrm{o}} \cup \Sigma_{\mathrm{a}}^{\mathrm{e}})^*$ satisfying the following constraints.

- $f_A(\varepsilon, \varepsilon) \in (\Sigma_{\mathrm{a}}^{\mathrm{i}})^*$.
- $f_A(s, e) \in \{e\}(\Sigma_{\mathrm{a}}^{\mathrm{i}})^*$ if $e \in \Sigma_{\mathrm{o}} \setminus \Sigma_{\mathrm{a}}$.
- $f_A(s, e) \in \{e, e^d\}(\Sigma_{\mathrm{a}}^{\mathrm{i}})^*$ if $e \in \Sigma_{\mathrm{a}}$.

The function f_A evaluated on argument (s, e) specifies the action taken by the attacker when the history of events is s, and the current event is e. The first constraint says that the initial modification of the attacker (before there is any history) must consist purely of inserted events. The second constraint says that if an event outside of the attacker's control occurs, the attacker has no choice but to let it through. However, the attacker can append a string of insertions after that event. The third constraint is the same as the second, except that the attacker has the option of deleting the current event, since the current event is compromised by the attacker.

It is often convenient to specify the attacker function in another way. As mentioned above, the attacker function f_A tells us the current modification, given the history and the current event. We can also define a function $\hat{f}_A$ that instead tells us the modified version of an entire string of observable events emitted by the system. This is easily defined recursively.

$$\begin{aligned} \hat{f}_A(\varepsilon) &= f_A(\varepsilon, \varepsilon) \\ \hat{f}_A(se) &= \hat{f}_A(s) f_A(s, e) \end{aligned} \tag{10}$$

This formulation of the attacker allows us to define an attacker that can only make a bounded number of actions. First, define projections $P_i : (\Sigma_{\mathrm{o}} \cup \Sigma_{\mathrm{a}}^{\mathrm{e}})^* \to (\Sigma_{\mathrm{a}}^{\mathrm{i}})^*$ and $P_d : (\Sigma_{\mathrm{o}} \cup \Sigma_{\mathrm{a}}^{\mathrm{e}})^* \to (\Sigma_{\mathrm{a}}^{\mathrm{d}})^*$ in the same manner as in (3).

Definition III.2. Given $m, n \in \mathbb{N}$, a system G, and a subset $\Sigma_{\mathrm{a}} \subseteq \Sigma_{\mathrm{o}}$, an m-insertion, n-deletion attacker (mInD

attacker) is defined as an attacker which additionally satisfies the following two constraints for any s where $\hat{f}_A(s)$ is defined

- $\left|P_i(\hat{f}_A(s))\right| \leq m$.
- $\left|P_d(\hat{f}_A(s))\right| \leq n$.

That is, a mInD attacker should never modify a string to contain more than m insertions or n deletions.

We can now formally define how an attacker interacts with the system and supervisor. This basically comes down to removing the annotations on the edited events in the appropriate way; for instance, the supervisor cannot differentiate between inserted events and truly occurring events, and it cannot even see the deleted events. To model this intuition, we define $P_e^S(e^i) = e$ and $P_e^S(e^d) = \varepsilon$ for each $e \in \Sigma_{\mathrm{a}}$, and $P_e^S(e) = e$ for any $e \in \Sigma_{\mathrm{o}}$ (and we extend P_e^S to strings in $(\Sigma_{\mathrm{o}} \cup \Sigma_{\mathrm{a}}^{\mathrm{e}})^*$ in the natural way, as we showed in (1)). Similarly, we define $P_e^G(e^i) = \varepsilon$ and $P_e^G(e^d) = e$ for each $e \in \Sigma_{\mathrm{a}}$ and $P_e^G(e) = e$ for $e \in \Sigma_{\mathrm{o}}$ to model the system's point of view. Finally, let $\mathcal{M}(e^i) = \mathcal{M}(e^d) = e$ for each $e \in \Sigma_{\mathrm{a}}$ and $\mathcal{M}(e) = e$ for $e \in \Sigma_{\mathrm{o}}$. $\mathcal{M}$ is simply a function which removes the annotations we placed on insertions and deletions.

Figure 2 now suggests how to define the interaction between the attacker and supervisor. The attacker and supervisor together effectively form a new supervisor for the system S_A, whose control decisions are defined as $S_A(s) = (S_P \circ P_e^S \circ \hat{f}_A)(s)$. This new supervisor has a corresponding controlled language $\mathcal{L}(S_A/G)$ that is defined as in (4).

Finally, we need to define the attacker's goal. In a real situation, the attacker might aim to cause damage to the system. This is often modeled as reaching an unsafe state in the system. So we designate a set of system states X_{crit} which are unsafe, and the attacker's goal is to manipulate the supervisor into allowing the system to reach a state in X_{crit}. The only requirement on X_{crit} is that it should not be possible to reach an unsafe state in the normal supervised system, when no attacker is present. That is, for all $s \in \mathcal{L}(S_P/G)$, we should have $\delta(x_0, s) \notin X_{\mathrm{crit}}$.

We now have all the tools we need to define a stealthy attacker.

Definition III.3. Given a system G, a supervisor S_P, and $\Sigma_{\mathrm{a}} \subseteq \Sigma_{\mathrm{o}}$, a stealthy attacker f_A is an attacker that satisfies the following conditions.

- For all $s \in \mathcal{L}(S_A/G)$, $\hat{f}_A(P_{\mathrm{o}}(s))!$.
- For all $s \in \mathcal{L}(S_A/G)$, $P_e^S(\hat{f}_A(P_{\mathrm{o}}(s))) \in P_{\mathrm{o}}(\mathcal{L}(S_P/G))$.

In addition, we say that the attack is strongly successful[2] if there exists $s \in \mathcal{L}(S_A/G)$ such that for every $t \in \mathcal{L}(S_A/G) \cap P_{\mathrm{o}}^{-1}(P_{\mathrm{o}}(s))$, $\delta(x_0, t) \in X_{\mathrm{crit}}$.

The first condition states that the attacker actually outputs an action on every string it could possibly observe. The second condition says that, from the supervisor's point of view, the attacked system is indistinguishable from the normal system. And the strongly successful condition states that there is some string s, where if the attacker ever sees $P_{\mathrm{o}}(s)$ occurring, the attacker is sure that the system has been compromised (regardless of how the actual sequence of events executed by the system might have interleaved unobservable events into $P_{\mathrm{o}}(s)$).

We can finally state the problem: given a system G, a supervisor R, a set of compromised events Σ_{a}, and bounds m and n on the number of allowed insertions and deletions, synthesize a strongly successful mInD attacker.

[2]A notion of a weakly successful attack is also defined in [1], but we won't need it here.

IV. Insertion-Deletion Attack Structure

An insertion-deletion attack structure (IDA) is a graph structure that captures all information about the system, supervisor, and attacker that can be known to the attacker at any given time. The labels on the edges are possible actions of the supervisor (control decisions) or the attacker (insertions, deletions, let event through). If an edge going from node a to node b has label x, then action x has the effect of transforming the states of the system, supervisor, and attacker from a to b. This IDA construction is an extension of the *bipartite transition structure* first presented in [4], and is closely related to the IDA constructed in [1]. We first present the formal definition, and then explain how it works. In the following, when $n \in \mathbb{N}$ we write $[n]$ to denote the set $\{0, 1, \ldots, n\}$.

Definition IV.1. An m-Insertion, n-Deletion Insertion-Deletion Attack structure (mInDIDA) A with respect to a system $G = (X, \Sigma, \delta, x_0)$, a set of compromised events Σ_a, a supervisor equipped with attacker detection $\tilde{R} = (\tilde{Q}, \Sigma, \tilde{\mu}, q_0)$, and bounds $m, n \in \mathbb{N}$ on the number of insertions and deletions allocated to the attacker, is a 7-tuple

$$A = (Q_S, Q_E, h_{SE}, h_{ES}, \Sigma, \Sigma_a^e, y_0)$$

where the components are defined as follows:

- $Q_S = 2^X \times \tilde{Q} \times [m+1] \times [n+1]$ is the set of S-states, where S stands for supervisor and where each S-state is of the form $y = (I_G(y), I_S(y), \mathsf{ins}(y), \mathsf{del}(y))$, where $I_G(y)$ and $I_S(y)$ denote the plant state estimate and the supervisor's state, and $\mathsf{ins}(y)$ and $\mathsf{del}(y)$ denote the number of insertion and deletion moves remaining.
- $Q_E = 2^X \times \tilde{Q} \times [m+1] \times [n+1]$ is the set of E-states where E stands for Environment; each E-state is of the form $z = (I_G(z), I_S(z), \mathsf{ins}(z), \mathsf{del}(z))$ where the components are defined in the same way as in the S-states case.
- $h_{SE} : Q_S \times \Gamma \to Q_E$ is the partial transition function from S-states to E-states, where $h(y, \gamma) = z$ if and only if all of the following constraints hold.
  - $I_G(z) = UR_\gamma(I_G(y))$
  - $I_S(z) = I_S(y)$
  - $\gamma = \Gamma_{\tilde{R}}(I_S(y))$
  - $\mathsf{ins}(z) = \mathsf{ins}(y)$
  - $\mathsf{del}(z) = \mathsf{del}(y)$
- $h_{ES} : Q_E \times (\Sigma_o \cup \Sigma_a^e) \to Q_S$ is the partial transition function from E-states to S states satisfying the following

constraints: for any $y \in Q_S, z \in Q_E, e \in \Sigma_o \cup \Sigma_a^e$, we have $h_{ES}(z,e) = y$ if and only if

$$y = \begin{cases} (NX_e(I_G(z)), \tilde{\mu}(I_S(z), e), \mathsf{ins}(z), \mathsf{del}(z)) \\ \quad \text{if } e \in \Gamma_R(I_S(z)) \cap \Gamma_G(I_G(z)) \\ (I_G(z), \tilde{\mu}(I_S(z), P_e^S(e)), \mathsf{ins}(z) - 1, \mathsf{del}(z)) \\ \quad \text{if } e \in \Sigma_a^i \\ \quad \text{and } \mathcal{M}(e) \in \Gamma_R(I_S(z)) \\ \quad \text{and } \mathsf{ins}(z) > 0 \\ (I_G(z), \tilde{\mu}(I_S(z), P_e(e)), \mathsf{ins}(z), \mathsf{del}(z) - 1) \\ \quad \text{if } e \in \Sigma_a^d \\ \quad \text{and } \mathcal{M}(e) \in \Gamma_R(I_S(z)) \cap \Gamma_G(I_G(z)) \\ \quad \text{and } \mathsf{del}(z) > 0 \end{cases}$$

- Σ is the set of events in G,
- Σ_a^e is the set of editable events.
- $y_0 \in Q_S$ is the initial S-state: $y_0 = (\{x_0\}, q_0, m, n)$.

Each state in a mInDIDA stores four things. The first two keep track of the set of possible states of the system the state of the supervisor. The second two keep track of the number of insertion moves and deletion moves that remain available to the attacker. The function h_{SE} transitions from supervisor to environment states by updating the set of possible states of the system according to the control decision provided by the supervisor. The function h_{ES} transitions from the environment back to the supervisor by considering all possible attacker actions. The first clause corresponds to a "let through event" - if an event is feasible both in the system and in the supervisor's current control decision, one option is for the attacker to let the event through without modification. The second clause corresponds to an "insertion" - any compromised event that is permissible according to the supervisor can be inserted, as long as we still have insertion moves remaining. The resulting state update changes the state of the supervisor accordingly, but not the system. since the event was fake. And the number of insertions we have available goes down by 1. The last clause handles event deletions and is similar.

In practice, a mInDIDA can be constructed via a BFS, starting from the initial state and growing every possible edge. An example mInDIDA is presented in figure 3.

V. Analysis

The first order of business is proving that all the information stored in the IDA is correct. To precisely define what we mean by this, we define the notion of an induced E-state.

Definition V.1. Given a mInDIDA A, we define the E state induced by a string $s \in (\Sigma_o \cup \Sigma_a^e)^*$ when starting in the E-state z recursively as follows.

$$IE(z, \varepsilon) = z$$

$$IE(z, se) = \begin{cases} h_{SE}(y, \Gamma_{\tilde{R}}(I_S(y))) & y = h_{ES}(IE(z,s), e) \\ \text{undefined} & \text{otherwise} \end{cases}$$

Intuitively, $IE(z,s)$ denotes the E-state reached in A upon starting from E-state z and taking the transitions given by the letters of s. We also define $IE(s) = IE(z_0, s)$, where $z_0 = h_{SE}(y_0, \Gamma_{\tilde{R}}(I_S(y_0)))$. This simply takes the "initial E-state" as the canonical starting point.

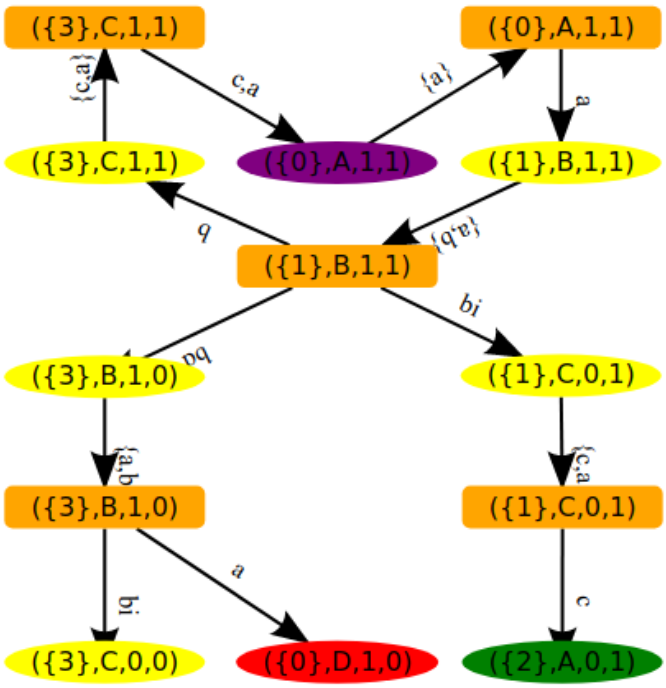


Fig. 3: A slightly abbreviated IDA constructed on the example system and supervisor with $m = n = 1$, $\Sigma_a = \{b\}$. Key: (S-state), [E−state], initial state, attacker detected, and system compromise, bi: insert event b, bd: delete b, no suffix: let-through. Image generated by our implementation.

The following so-called correctness lemma shows that the number of insertions and deletions consumed by a string s_A matches the number of insertions and deletions according to the state induced by s_A in the mInDIDA, thus proving that $\mathsf{ins}(z)$ and $\mathsf{del}(z)$ actually are equal to what they were intended to represent. Versions of this lemma can be stated and proved for the correctness of $I_G(z)$ and $I_S(z)$ as well, but the proofs are all similar and hence omitted.

Lemma V.1. *Given a system G, supervisor $\tilde{R}$, and a mInDIDA A with embedded attack function f_A, for any $s \in \mathcal{L}(S_A/G)$, setting $s_A = \hat{f}_A(P_o(s))$, we have $\mathsf{ins}(IE(s_A)) = m - |P_i(s_A)|$ and $\mathsf{del}(IE(s_A)) = n - |P_d(s_A)|$.*

Proof. By induction on $|s_A|$. We check only insertion, as deletion is identical.

- Base Case: If $|s_A| = 0$ then $s_A = \varepsilon$, so $|P_i(s_A)| = |\varepsilon| = 0 \leq m$. Also $IE(s_A) = z_0 = h_{SE}(y_0, \Gamma_{\tilde{R}}(I_S(y_0)))$. Since an SE transition doesn't change the insertion count, $\mathsf{ins}(z_0) = \mathsf{ins}(y_0) = m$, and we have the desired
$$\mathsf{ins}(IE(s_A)) = m = m - |P_i(s_A)|$$
- Induction Hypothesis: For any s_A of length at most k, if $|P_i(s_A)| \leq m$ and $|P_d(s_A)| \leq n$, then $\mathsf{ins}(IE(s_A)) =$

$m - |P_i(s_A)|$.

- Induction Step: Consider $s_A e \in \hat{f}_A(s)$ of length $k+1$, where $e \in \Sigma_o \cup \Sigma_a^e$. By the inductive hypothesis, $\mathsf{ins}(IE(s_A)) = m - |P_i(s_A)|$. Let $y = h_{ES}(IE(s_A), e)$. We compute

$$\mathsf{ins}(IE(s_A e)) = \mathsf{ins}(h_{SE}(y, \Gamma_{\tilde{R}}(I_S(y))))$$
$$= \mathsf{ins}(y) = \mathsf{ins}(h_{ES}(IE(s_A), e))$$

If $e \in \Sigma_o \cup \Sigma_a^d$, then $|P_i(s_A e)| = |P_i(s_A)|$, as e is not an insertion. By definition of h_{ES}, we have

$$\mathsf{ins}(IE(s_A e)) = \mathsf{ins}(h_{ES}(IE(s_A), e))$$
$$= \mathsf{ins}(IE(s_A)) = m - |P_i(s_A)|$$
$$= m - |P_i(s_A e)|$$

If $e \in \Sigma_a^i$, then $|P_i(s_A e)| = |P_i(s_A)| + 1$, as e is an insertion. By definition of h_{ES}, we have

$$\mathsf{ins}(IE(s_A e)) = \mathsf{ins}(h_{ES}(IE(s_A), e))$$
$$= \mathsf{ins}(IE(s_A)) - 1 = m - |P_i(s_A)| - 1$$
$$= m - (|P_i(s_A)| + 1) = m - |P_i(s_A e)|$$

□

These correctness lemmas allow us to refer to the information stored in a mInDIDA state and the "correct value" for that information that might be derived from a string leading to that state interchangeably.

Our goal now is to prove that the constructed mInDIDA contains precisely the set of all stealthy mInD attackers, in the following sense.

Definition V.2 (Embedded Attacker)**.** An attacker f_A is said to be embedded in a mInDIDA A if $(\forall s \in P_o(\mathcal{L}(S_A/G)))(\forall t \in \hat{f}_A(s))[IE(t)!]$. In words, f_A is embedded in A if every attack strategy given by f_A induces a path in the mInDIDA (starting from the initial state).

However, this is not quite true. In the mInDIDA from figure 3, there still exist red (attacker detected) states, so there is an embedded attacker that is not stealthy. Fortunately, removing these states is done simply using the BSCP algorithm [1] [2]. This algorithm, given a specification, trims the minimal number of states to prevent the attacker from reaching a red state. There is only one nontrivial modification that we make to this algorithm. We also remove E-states in the mInDIDA with only a single outgoing insertion edge. This is because at such states, the attacker is forced to win a race against the system.

Algorithm **??** essentially treats the mInDIDA as a *system* (whose event set is the set of all edge labels of the mInDIDA), and the goal is to meet the *specification*, namely avoiding red states. Since control decisions and non-attackable events are those actions which are uncontrollable from the point of view of the attacker, we set $E_{uc} = \Gamma \cup (\Sigma_o \setminus \Sigma_a)$. Running the algorithm produces a "pruned mInDIDA" which does not contain any red states *and* cannot uncontrollably lead to any red states[3]. This pruned mInDIDA now has the property we desire.

Algorithm 1 Modified BSCP

Require: A is a mInDIDA, A_{trim} is A with red states removed.
Step 1: Set $H_0 = (A_0, E, g_0, a_0) = A_{trim}$, $i = 0$, $E = \Gamma \cup \Sigma_o \cup \Sigma_a^e$, $E_{uc} = \Gamma \cup (\Sigma_o \setminus \Sigma_a)$.
Step 2.1: Set $A_i' = \{a \in A_i \mid \Gamma_A(a) \cap E_{uc} \subseteq \Gamma_{H_i}(a)\}$.
Step 2.2: Set $A_i^* = \{a \in A_i' \mid e \in \Gamma_A(a) \implies e \in \Gamma_{H_i}(a) \vee e^d \in \Gamma_{H_i}(a)\}$.
Step 2.3: Set $H_{i+1} = \mathrm{Trim}(A_i^*, E, g_i \restriction A_i^*, a_0)$.
Step 3: If $H_{i+1} = H_i$, stop. Else increment i and go to step 2.1.

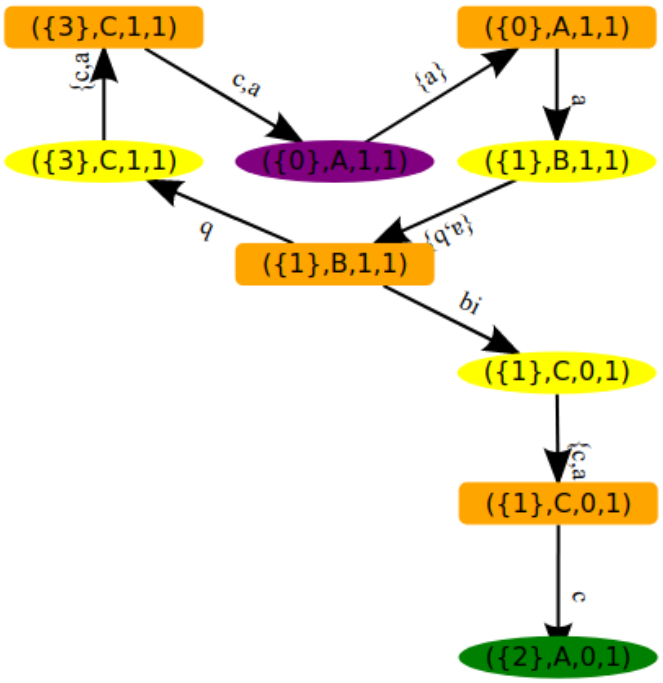


Fig. 4: The mInDIDA from figure 3 after the pruning algorithm is executed. Note the absence of red states. Observe that in this case, all nodes below the edge b_d incident to the state $(\{1\}, B, 1, 1)$ had to be removed, as if the attacker makes the move b_d, the attacker cannot do anything to prevent event $a \notin \Sigma_a$ from occurring and going into a red state. Key: (*S*-state), |E−*state*|, initial state, attacker detected, and system compromise, bi: insert event b, bd: delete b, no suffix: let-through. Image generated by our implementation.

Lemma V.2. *If a mInD attacker is stealthy, it is embedded in the pruned mInDIDA.*

Proof. If such an attacker f_A were not embedded in the pruned mInDIDA, then there exists strings $s \in \mathcal{L}(S_A/G), t_A \in \hat{f}_A(s)$

[3]For a more detailed treatment of the BSCP algorithm, see [2]

and an event $e \in \Sigma_o \cup \Sigma_a^e$ such that $IE(t_A)$ is defined but $IE(t_A e)$ is undefined. There are two ways this can happen.

1) $IE(t_A e)$ was undefined in the original $m\mathrm{I}n\mathrm{DIDA}$. This means that $h_{ES}(IE(t_A), e)$ is undefined. We split into cases based on whether e is an insertion, a deletion, or simple a let-through event.
   a) If e is a let-through event, then $h_{ES}(IE(t_A), e)$ is undefined if and only if $e \notin \Gamma_{\tilde{R}}(I_S(z)) \cap \Gamma_G(I_G(z))$. By our earlier lemma, this means that e either cannot occur in the system in its current state, or is disabled by the supervisor. This contradicts $s \in \mathcal{L}(S_A/G)$.
   b) If e is an insertion, then $h_{ES}(IE(t_A), e)$ is undefined if and only if $\mathcal{M}_e(e) \notin \Gamma_{\tilde{R}}(I_S(z))$ or $\mathsf{ins}(z) = 0$. The first violates stealthiness, since we are trying to insert an event that is disabled, and the second violates the fact that we are a $m\mathrm{I}n\mathrm{D}$ attacker.
   c) If e is a deletion, then $h_{ES}(IE(t_A), e)$ is undefined if and only if $\mathcal{M}_e(e) \notin \Gamma_{\tilde{R}}(I_S(z)) \cap \Gamma_G(I_G(z))$ or $\mathsf{del}(z) = 0$. The first violates $s \in \mathcal{L}(S_A/G)$, and the second violates the fact that we are a $m\mathrm{I}n\mathrm{D}$ attacker.
2) $IE(t_A e)$ is defined in the original $m\mathrm{I}n\mathrm{DIDA}$, but is removed by the pruning algorithm. The BSCP algorithm has the property that it returns a $m\mathrm{I}n\mathrm{DIDA}$ that is maximally permissive [2] under the conditions imposed by the specification A_{trim}. We only delete red states, which a stealthy attacker cannot every visit, and states where a race might happen, which an attacker should never visit to avoid racing with the system. Hence, if $IE(t_A e)$ is pruned by the algorithm, the attacker is either not stealthy or raced with the system, both of which result in a contradiction.

□

Lemma V.3. *If an attacker is embedded in the pruned $m\mathrm{I}n\mathrm{DIDA}$, it is a stealthy $m\mathrm{I}n\mathrm{D}$ attacker.*

Proof. Since the $m\mathrm{I}n\mathrm{DIDA}$ correctly keeps track of the number of insertions and deletions remaining, and this quantity is nonnegative at all states in the $m\mathrm{I}n\mathrm{IDA}$, any embedded attacker is guaranteed to be a $m\mathrm{I}n\mathrm{D}$ attacker. Since the pruned $m\mathrm{I}n\mathrm{DIDA}$ contains no red states, any embedded attacker is guaranteed to be stealthy. □

These two lemmas together tell us that the set of attackers embedded in the pruned $m\mathrm{I}n\mathrm{DIDA}$ is precisely the set of stealthy $m\mathrm{I}n\mathrm{D}$ attackers. Hence, it is possible to extract any of these desired attackers from the pruned $m\mathrm{I}n\mathrm{DIDA}$.

VI. Conclusions and Future Work

We took the model of an attacker from [1] and considered the problem of synthesizing an attacker of more limited capability, in the sense that the number of attacker moves is limited. When restricting the attacker to at most m insertion moves and at most n deletion moves, we constructed an IDA with a multiplicative overhead of at most $(m+1)(n+1)$ compared to the IDA presented in [1]. We also modified the implementation given by the authors of [1] to compute $m\mathrm{I}n\mathrm{DIDAs}$ for simple systems and supervisors.

One obvious generalization of the work in this paper is to assign each attacker move an arbitrary cost instead of just a cost of 1 for insertions and 1 for deletions, and then attempt to synthesize a minimum-cost attacker. We are also interested in investigating notions of attacker complexity completely distinct from having a cost associated to each move. Finally, we are looking for methods to avoid the construction of the entire IDA, which can grow exponentially in the size of the system under partial observation.

References

[1] R. M. Góes, E. Kang, R. Kwong, and S. Lafortune, "Stealthy deception attacks for cyber-physical systems," *2017 IEEE 56th Annual Conference on Decision and Control (CDC)*, pp. 4224–4230, 2017.

[2] C. G. Cassandras and S. Lafortune, *Introduction to Discrete Event Systems*. Springer Publishing Company, Incorporated, 2nd ed., 2010.

[3] J. P. Farwell and R. Rohozinski, "Stuxnet and the future of cyber war," *Survival*, vol. 53, no. 1, pp. 23–40, 2011.

[4] 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.