Dynamic Games and Applications (2019) 9:965–983
https://doi.org/10.1007/s13235-018-0285-3

Supervisory Control of Discrete-Event Systems Under Attacks

Masashi Wakaiki[1] · Paulo Tabuada[2] · João P. Hespanha[3]

Published online: 8 September 2018

Abstract

We consider a multi-adversary version of the supervisory control problem for discrete-event systems (DES), in which an adversary corrupts the observations available to the supervisor. The supervisor's goal is to enforce a specific language in spite of the opponent's actions and without knowing which adversary it is playing against. This problem is motivated by applications to computer security in which a cyber defense system must make decisions based on reports from sensors that may have been tampered with by an attacker. We start by showing that the problem has a solution if and only if the desired language is controllable (in the DES classical sense) and observable in a (novel) sense that takes the adversaries into account. For the particular case of attacks that insert symbols into or remove symbols from the sequence of sensor outputs, we show that testing the existence of a supervisor and building the supervisor can be done using tools developed for the classical DES supervisory control problem, by considering a family of automata with modified output maps, but without expanding the size of the state space and without incurring on exponential complexity on the number of attacks considered.

Keywords Supervisory control · Discrete-event systems · Game theory · Computer security

---

Submitted to Special Issue on Dynamic Games in Cyber Security. This work was supported by the JSPS KAKENHI Grant Number JP17K14699, the National Science Foundation Award No. 1705135, and the U.S. Office of Naval Research under MURI Grant No. N00014-16-1-2710.

✉ João P. Hespanha
hespanha@ucsb.edu

Masashi Wakaiki
wakaiki@ruby.kobe-u.ac.jp

Paulo Tabuada
tabuada@ee.ucla.edu

1 Graduate School of System Informatics, Kobe University, Kobe 657-8501, Japan

2 Department of Electrical Engineering, University of California, Los Angeles, CA 90095-1594, USA

3 Center for Control, Dynamical-Systems, and Computation, University of California, Santa Barbara, CA 93106-2560, USA

1 Introduction

Discrete-event systems (DESs) are non-deterministic transition systems defined over a typically finite state-space. The DESs supervisory control problem refers to the design of a feedback controller—called a *supervisor*—that restricts the set of possible sequences of transitions (typically represented by *strings* over an alphabet of transitions) to a desired set K. The supervisor's task is complicated by the fact that (i) only a subset of transitions can be inhibited (the so called "controllable" transitions) and (ii) the supervisor only has partial information about the state of the system, which it gathers by observing a string of "output symbols." This basic problem is motivated by a wide range of applications that include manufacturing systems, chemical batch plants, power grids, transportation systems, database management, communication protocols, logistics, and computer security. The latter is the key motivating application for the work reported here.

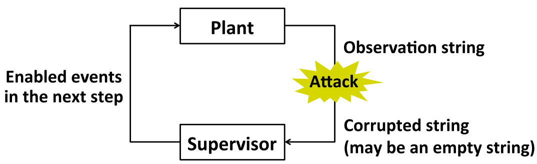
We consider a zero-sum multi-adversary version of the supervisory control problem, where one player (the supervisor) faces multiple adversaries with distinct action spaces and the supervisor needs to find a policy that wins against any of its possible adversaries, without knowing which one is the actual opponent. The supervisor loses if it accepts strings outside the desired set K or if it rejects strings within K. We consider a zero-sum game in which the adversary tries to make the supervisor lose by manipulating the string of output symbols that the supervisor uses to make decisions (see Fig. 1).

In computer security applications, the supervisor is typically responsible for enacting defense mechanisms, such as opening/closing firewalls, starting and stopping services, authorizing/deauthorizing users, and killing processes. Such decisions are based on observations collected by cyber-security sensors that log events like user authentication, network traffic, email activity, and access to services or files. Ultimately, the supervisors' goal is to allow users to perform all the tasks they are authorized to do, while preventing unauthorized access to resources and services. The zero-sum multi-adversary game is motivated by scenarios where a cyber attacker manipulates the supervisor's observations by tampering with one or more of the cyber-security sensors. A key challenge is that security mechanisms typically do not know if a particular sensor has been compromised so it needs to consider multiple alternatives for sensor manipulation, which explains the need for the multi-adversary model.

One of the paper's main results is a general necessary and sufficient condition for the multi-adversary game to have a "solution", i.e., for the existence of a supervisor that can win the game against any of the adversaries, without knowing which one it is playing against (Sect. 4). This necessary and sufficient condition is expressed in terms of a controllability condition, which also arises in the classical supervisory control setup and a novel observability condition that depends on how the adversaries can manipulate/attack the output string. The result is constructive in the sense that it provides an explicit formula for the supervisor, in case one exists, but this general formula is typically hard to use in practice.

The second part of the paper is focused on "output-symbol attacks", which correspond to games where each of the adversaries is restricted to attack a specific set of symbols.

Fig. 1 Closed-loop system under attacks



Specifically, each adversary is able to insert into or remove from the output string symbols from a given set. In the context of computer security, we can view such adversaries as attackers that infiltrate a security sensor and insert/remove entries from the corresponding security log. For output-symbol attacks, we show that testing observability and generating supervisors can be done using tools developed for classical supervisory control problems.

For output-symbol attacks, we show that testing the new notion of observability under attacks can be done by checking the classical notion of observability for a family of DESs that differ from each other only by their output maps (Sect. 5.1). These output maps are obtained from the original output map by considering all pairs of adversaries and *removing* from the output string every symbol that 2 adversaries could attack if they were allowed to combine their efforts (which they are not). This shows that observability under attacks can still be tested in polynomial time.

Also for output-symbol attacks, we show that a supervisor for the multi-adversary case can be constructed from a family of classical supervisors, each constructed for one of the classical DES supervisory control problems used for the observability test (Sect. 5.2). Since constructing classical supervisors has exponential worst-case complexity on the number of states, this procedure also has exponential worst-case complexity, but it has the desirable feature that it does not enlarge the state-space of the original system. This is in sharp contrasts with the alternative approach of reducing the multi-adversary supervisory control problem to a classical supervisory control problem, which generally requires expanding the state-space M times, where M is the number of adversaries.

Supervisory control theory provides a natural framework to address plant uncertainties and faults, either in a robust control context (e.g., [11,15,20]) or in a fault tolerant context (e.g., [12,16,19]). Several aspects of security have also been considered in the DES literature, including the stealthy deception attacks in [1,22] that aim at injecting false information without being detected by the controller; or the opacity of DESs in [5,14,21,29] and references there in, whose goal is to keep the system's secret behavior uncertain to outsiders. Intrusion detection in the DES framework has been investigated in [8,23,27], with the goal of guaranteeing the confidentiality and integrity of DESs.

The key novelty of the problem considered here is the focus on DES problems against adversaries that manipulate the observations available to the supervisor. By focusing on this type of attack, we obtain tests for the existence of a supervisor and algorithms to generate the supervisor that do not require an expansion of the state-space of the original (not attacked) DES.

DESs with non-deterministic (set-valued) observation maps have been studied in [26,30,31], where the problem formulation could be viewed has a version of the problem considered here but with a single adversary. The authors in [30] introduced the notion of lifting and synthesized supervisors by converting the original problem with non-deterministic observations to a problem in the lifted domain with deterministic observations. In [26,31], the so-called Mealy automata were used to deal with both state-dependent observations and non-deterministic outputs. The work reported here is inspired by the research on state estimation under sensor attacks developed in [3,6,18], where the authors consider a linear time invariant system with multiple outputs and an attacker that can manipulate a subset of the sensors that measure these outputs. The goal is to estimate the system's state and stabilize it using feedback control, without knowing which sensors have been compromised.

2 Background on Supervisory Control of Discrete-Event Systems

We start by recalling basic notation and definitions that are common in the Discrete-Event Systems (DES) literature.

For a finite set Σ of *events,* we denote by $|\Sigma|$ the number of elements of Σ and by Σ^* the set of all finite strings of elements of Σ, including the empty string ϵ. A subset L of Σ^* is called a *language over the alphabet* Σ and the *prefix closure* of L is the language

$$\bar{L} := \big\{u \in \Sigma^* : \exists v \in \Sigma^*,\ uv \in L\big\},$$

where uv denotes the concatenation of two strings in Σ^*. The language L is said to be *prefix closed* if $L = \bar{L}$. We define the concatenation of languages $L_1, L_2 \subset S^*$ by

$$L_1 L_2 := \{w_1 w_2 \in \Sigma^* : w_1 \in L_1,\ w_2 \in L_2\}.$$

Consider an *automaton* $G = (X, \Sigma, \xi, x_0)$, where X is the set of states, Σ the nonempty finite set of events, $\xi : X \times \Sigma \to X$ the transition mapping (a partially defined function), and $x_0 \in X$ the initial state. We write $\xi(x, \sigma)!$ to mean that $\xi(x, \sigma)$ is defined. The transition function ξ can be extended to a function $X \times \Sigma^* \to X$ according to the following inductive rules

$$\xi(x, \epsilon) := x, \qquad \forall x \in X$$

$$\xi(x, w\sigma) := \begin{cases} \xi(\xi(x, w), \sigma) & \text{if } \xi(x, w)! \text{ and } \xi(\xi(x, w), \sigma)! \\ \text{undefined} & \text{otherwise,} \end{cases} \qquad \forall x \in X,\ w \in \Sigma^*,\ \sigma \in \Sigma.$$

The *language generated by* G is then defined by

$$L(G) := \{w \in \Sigma^* : \xi(x_0, w)!\}.$$

Let Σ be a set of events that can be partitioned in two disjoint sets as $\Sigma = \Sigma_c \cup \Sigma_{uc}$, where Σ_c is called the *set of controlled events* and Σ_{uc} the *set of uncontrolled events*. For a language L defined on Σ, a prefix-closed set $K \subset L$ is said to be *controllable* if $K\Sigma_{uc} \cap L \subset K$.

Consider an *observation map* $P : \Sigma \to (\Delta \cup \{\epsilon\})$ that maps the set of events Σ into a set of *observations* Δ (augmented by the empty event ϵ). This observation map P can be extended to the map defined for strings of events according to $P(\epsilon) = \epsilon$ and

$$P(w\sigma) = P(w)P(\sigma), \quad \forall w \in \Sigma^*,\ \sigma \in \Sigma.$$

A prefix-closed language $K \subset L$ is P*-observable with respect to* L if

$$\ker P \subset \text{act}_{K \subset L}$$

where $\ker P$ denotes the relation on Σ^* defined by

$$\ker P := \big\{(w, \bar{w}) \in \Sigma^* \times \Sigma^* : P(w) = P(\bar{w})\big\}$$

and $\text{act}_{K \subset L}$ the binary relation on Σ^* defined by

$$\text{act}_{K \subset L} := \Big\{(w, \bar{w}) \in \Sigma^* \times \Sigma^* : w, \bar{w} \in K \Rightarrow \nexists \sigma \in \Sigma \text{ s.t. } [w\sigma \in K, \bar{w}\sigma \in L \backslash K] \text{ or } [w\sigma \in L \backslash K, \bar{w}\sigma \in K]\Big\}.$$

The key control design problem in DESs is to design a “supervisor” that modifies the original language L generated by the automaton to a desired language $K \subset L$, by judiciously disabling controllable events in Σ_c (and their associated transitions) based on the observations obtained through P, in a feedback fashion. The reader is referred to [2,13,28] for more details on DES models and key results.

3 Supervised Discrete-Event Systems Under Attacks

In this paper, we deviate from the classical DES model by introducing a set of adversaries $\mathcal{A}$ whose goal is to prevent the supervisor from achieving the desired language K. Each adversary $A \in \mathcal{A}$ can corrupt the string of output symbols $P(w)$, $w \in \Sigma^*$ in multiple (non-deterministic) ways, e.g., erasing and/or inserting specific output symbols (see Example 1 below). Our goal is thus to design a supervisor that can guarantee the desired language K regardless of (1) which $A \in \mathcal{A}$ is the actual attack, and (2) how the actual attack $A \in \mathcal{A}$ corrupts each string of output symbols $P(w)$, $w \in \Sigma^*$.

Formally, each adversary is a set-valued map $A : \Delta^* \to 2^{\Delta^*}$ that assigns to each string of output symbols $P(w)$, $w \in \Sigma^*$ a set $A\big(P(w)\big)$ of (possibly distorted) strings of symbols that the adversary can send to the supervisor, instead of the original string $P(w)$. A set-valued map is convenient because it enables us to consider multiple ways by which the attack A may corrupt a particular string $P(w)$. We call the map A an *observation attack* and the map $AP : \Sigma^* \to 2^{\Delta^*}$ obtained from the composition $AP := A \circ P$ the corresponding *attacked observation map*. The attack map $A_{\text{id}} : \Delta^* \to 2^{\Delta^*}$ that assigns to each string $y \in \Delta^*$ the set $\{y\}$ containing only the original output string y can be viewed as the absence of an attack.

A *P-supervisor for a language $L \subset \Sigma^*$ and an attack set $\mathcal{A}$* is a function $f : \bigcup_{A\in\mathcal{A}} AP(L) \to 2^{\Sigma}$. The supervisor is *valid (for the set of uncontrollable events Σ_{uc})* if

$$f(y) \in \Gamma, \quad \forall y \in \bigcup_{A\in\mathcal{A}} AP(L),$$

where $\Gamma := \{\gamma \subset \Sigma : \Sigma_{\text{uc}} \subset \gamma\}$ is the set of all subsets of Σ that contain all the uncontrollable symbols in Σ_{uc}. One should view $f(y)$ as the set of events that the supervisor enables after observing the (potentially distorted) string $y \in \Delta^*$. A valid supervisor is forced to always enable the uncontrolled symbols in Σ_{uc}.

By disabling symbols, a supervisor f effectively only "accepts" a subset of the strings in the original language L. However, which symbols are accepted depends on how the adversary actually distorts the observation strings. For a given supervisor f, the *maximal language $L^{\max}_{f,A}$ controlled by f under the attack $A \in \mathcal{A}$* is defined inductively by

$$\begin{cases} \epsilon \in L^{\max}_{f,A} \\ w\sigma \in L^{\max}_{f,A} \quad \Leftrightarrow \quad w \in L^{\max}_{f,A}, \quad w\sigma \in L, \quad \exists y \in AP(w) \text{ s.t. } \sigma \in f(y), \end{cases}$$

whereas the *minimal language $L^{\min}_{f,A}$ ($\subset L^{\max}_{f,A}$) controlled by f under the attack $A \in \mathcal{A}$* is defined inductively by

$$\begin{cases} \epsilon \in L^{\min}_{f,A} \\ w\sigma \in L^{\min}_{f,A} \quad \Leftrightarrow \quad w \in L^{\min}_{f,A}, \quad w\sigma \in L, \quad \forall y \in AP(w),\ \sigma \in f(y). \end{cases}$$

By construction $L^{\max}_{f,A}$ and $L^{\min}_{f,A}$ are prefix closed. One can conclude from these definitions that an adversary $A \in \mathcal{A}$ can distort the observations so that every string in the maximal language $L^{\max}_{f,A}$ is accepted by the supervisor f, but no string outside this language will be accepted. The same adversary is also able to cause the rejection of every string outside the minimal language $L^{\min}_{f,A}$, but is unable to cause the rejection of any string inside this language.

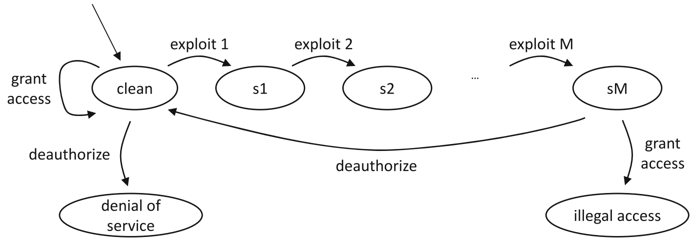


Fig. 2 Automaton representation of a multi-layered cyber attack to a computer system

In the absence of an attack (i.e., when $A = A_{\mathrm{id}}$), the maximal and minimal languages coincide and correspond to the classical notion of the *language L_f generated by the supervised system*. However, under attacks we can only be sure that $L_{f,A}^{\min} \subset L_{f,A}^{\max}$, typically with a strict inclusion. Motivated by the desire to construct supervisors that guarantee a specific language $K \subset L$, we say that f is a *solution to the supervision of $K \subset L$ under the attack set $\mathcal{A}$* if f is valid and

$$L_{f,A}^{\min} = L_{f,A}^{\max} = K, \quad \forall A \in \mathcal{A}.$$

This means that such f is a Stackelberg equilibrium policy for the supervisor (which we regard as the leader) and guarantees victory because, even if the supervisor advertises f as its policy, the adversary cannot force the rejection of any string in K (because $L_{f,A}^{\min} = K$) and cannot force the acceptance of any string outside K (because $L_{f,A}^{\max} = K$).

Example 1 (Multi-layer cyber attack to a computer system) Figure 2 shows an automaton representation of a multi-layer cyber attack to a computer system. Each automaton's state represents the interaction between a specific user and the system. This interaction starts in a "clean" state where legitimate users can request access to a secure service by providing appropriate credentials. However, through a sequence of exploits, a cyber attacker may get access to credentials that would give her access to the secure service. The goal is to design a supervisor that grants access to the service to legitimate users but not to the cyber attacker that made use of the exploits to obtain the needed credentials. Figure 2 shows a very simple sequential set of M exploits, but realistic scenarios are characterized by much more complicate networks of exploits.

In the example in Fig. 2, the set of events is

$$\Sigma = \Big\{\text{grant acess, deauthorize, exploit 1, exploit 2, } \dots \text{, exploit } M\Big\},$$

where the "exploits" are uncontrolled events, whereas "grant access" and "deauthorize" as controlled events:

$$\Sigma_{\mathrm{c}} = \Big\{\text{grant acess, deauthorize}\Big\}, \qquad \Sigma_{\mathrm{uc}} = \Big\{\text{exploit 1, exploit 2, } \dots \text{, exploit } M\Big\}.$$

We denote by $L \subset \Sigma^*$ the language G represented by the automaton described above and by $K_{\mathrm{safe}} \subset L$ the language that excludes from L all strings that include transitions to the "bad states" labeled with "denial of service" and "illegal access." Specifically, K_{safe} does not include any string that starts with "deauthorize" nor any string that includes the sequences "grant access, deauthorize"; "deauthorize, deauthorize"; or "exploit M, grant access."

While the "exploits" are uncontrolled events, we assume that the system has been instrumented with security logs that allow us to observe those transitions, which corresponds to the following observation map

$$\Delta = \Sigma, \qquad P(w) = w, \quad \forall w \in \Sigma^*. \tag{1}$$

In the absence of attacks (i.e., with $\mathcal{A} = \{A_{\mathrm{id}}\}$), the supervisor

$$f(w) = \begin{cases} \Sigma \backslash \text{“grant access”} & w \in \Sigma^* \, \Sigma_{\mathrm{uc}}^+, \\ \Sigma \backslash \text{“deauthorize”} & \text{otherwise,} \end{cases} \qquad \Sigma_{\mathrm{uc}}^+ := \Sigma_{\mathrm{uc}}^* \backslash \{\epsilon\},$$

disables “grant access” after strings that end with an exploit event and results in maximal and minimal languages that are both equal to the desired language: $L_{f,A}^{\max} = L_{f,A}^{\min} = K_{\mathrm{safe}}$.

We are interested in situations where the security systems that generate the logs may have been compromised and create false and/or remove legitimate observations. For simplicity, suppose that the observations corresponding to each of the M exploits are generated by M security systems and we suspect that one of them may have been compromised. We can model this scenario by considering the following set of $M + 1$ attacks:

$$\mathcal{A}_{\mathrm{log-attacks}} := \left\{A_{\mathrm{id}}, A_{\mathrm{exploit}\ 1}, A_{\mathrm{exploit}\ 2}, \ldots, A_{\mathrm{exploit}\ \mathrm{M}}\right\}, \tag{2}$$

where A_{id} corresponds to all security systems being good and each $A_{\mathrm{exploit}\ \mathrm{i}}$ corresponds to an attacker having compromised the security system that logs the occurrence of “exploit i”, which may therefore arbitrarily insert/remove the output symbols “exploit i” into/from the (correct) observation string $P(w)$, $w \in \Delta^*$. The key question addressed in this paper is whether or not there exists a solution to the supervision of the language K_{safe} under this more interesting attack set $\mathcal{A}_{\mathrm{log-attacks}}$. We shall see later that this is possible for $M \geq 3$, but not possible for $M \in \{1, 2\}$.

The paper [17] describes examples of the type of multi-layer attack considered in Example 1. One of these examples considers a scenario where an attacker wants to gain access to a database that requires root access at a particular Linux server. The server is behind a firewall and gaining access to it requires a sequence of exploits that include (1) exploiting a buffer overflow vulnerability in the Microsoft IIS Web Server to gain administrative privileges inside the firewall; (2) scanning the local network ports taking advantage of a misconfigured access control list in a Squid web proxy; (3) through this scan discovering and exploiting a vulnerability in a URL parsing function of the LICQ software to execute arbitrary code on a client’s computer; and finally (4) letting a user without administrative privileges gain them illegitimately through a local buffer overflow. This example would correspond to a simple sequence of $M = 4$ exploits in Fig. 1, but [17] discusses several alternative options to achieve the same goal by exploring these and other vulnerabilities. In fact, the examples considered in [17, Section 4] correspond to a network of exploits with over 300 nodes and many more transitions. The examples in [17] are quite specific in mapping exploits to known system vulnerabilities reported in the CVE database [4]. However, in practice, many system vulnerabilities are not known at the time cyber defense systems are designed so the type of analysis proposed in this paper should be based on automata that include exploits corresponding to *unknown software/hardware vulnerabilities.* E.g., one may include exploits like privilege escalation from user to administrative, even if no specific vulnerability to accomplish this is known for the system under consideration. Note that a system like [9] will be able to log the privilege escalation by a running process, even if the specific vulnerability that accomplishes this is unknown. The results in this paper seek to establish algorithms to process logs (like the ones generated by systems like [9]) that are robust to manipulation by an adversary. □

Remark 1 (Reduction to a supervisory control problem without attacks) It is often possible to reduce the problem of finding a solution to the supervision of a language K under an attack set $\mathcal{A}$ to a classical supervisory control problem without attacks. However, this is achieved by increasing the complexity of the original language L and the automaton G that generates it, which is problematic from a computational perspective.

Such a reduction is easy to illustrate for the problem described in Example 1, which could also be modeled by a DES without attacks by expanding the state-space of the automaton G. To do this, we would replicate M times the state of the automaton in Fig. 2, each replicate corresponding to one output-symbol attack $A_{\text{exploit i}}$, $i \in \{1, 2, \ldots, m\}$; and add a new initial state with unobservable transitions to the "clean" state of every replicate. These transitions from the new initial state define which attack $A \in \mathcal{A}_{\text{log}-\text{attacks}}$ will actually take place, but are not visible to the supervisor. The transitions within each one of the M replicates also need to be adjusted: taking, e.g., the replicate corresponding to the symbol "exploit 1", to emulate the insertion of this symbol we would add self-transitions at every state with the event "exploit 1", and to emulate the removal of this symbol, we would associate the original transition from "clean" to "s1" with a symbol that is not observable. This construction results in a new automaton G_{expanded} and an associated output map P_{expanded} with the property that any of its output strings $P_{\text{expanded}}(w)$, $w \in L(G_{\text{expanded}})$ could have been generated by the original automaton, under one of the attacks in $\mathcal{A}_{\text{log}-\text{attacks}}$. However, G_{expanded} has $O(M^2)$ states and transitions, as opposed to the $O(M)$ states and transitions in the original automaton G.

The reduction of supervisory control problem under attacks to a classical problem without attacks may be more complicated than what we have seen in this example. However, it will generally require some form of state replication of the original automaton $|\mathcal{A}|$ times, one for each possible attack map in the set $\mathcal{A}$, with initial unobservable transitions to each replicate. In addition, it will generally require the addition of further transitions within each replicate to model each of the attacks in $\mathcal{A}$. □

4 Existence of Supervisor Achieving Specifications

A necessary and sufficient condition for the existence of a solution to the supervision of K under $\mathcal{A}$ can be expressed in terms of a notion of observability that extends the conventional one presented in Sect. 2. Given an attack set $\mathcal{A}$, we say that a prefix-closed language $K \subset L$ is *P-observable for the attack set $\mathcal{A}$* if

$$R_{A,\bar{A}} \subset \text{act}_{K\subset L}, \qquad \forall A, \bar{A} \in \mathcal{A}, \tag{3}$$

where the relation $R_{A,\bar{A}}$ contains all pairs of strings that may result in attacked observation maps AP and $\bar{A}P$ with a common string of output symbols, i.e.,

$$R_{A,\bar{A}} := \big\{(w, \bar{w}) \in \Sigma^* \times \Sigma^* : AP(w) \cap \bar{A}P(\bar{w}) \neq \emptyset\big\}.$$

□

The P-observability condition (3) can be equivalently restated as requiring that, for all $w, \bar{w} \in K$,

$$\exists A, \bar{A} \in \mathcal{A} \text{ s.t. } AP(w) \cap \bar{A}P(\bar{w}) \neq \emptyset \\ \Rightarrow \quad \nexists \sigma \in \Sigma \text{ s.t. } [w\sigma \in K,\ \bar{w}\sigma \in L\backslash K] \text{ or } [w\sigma \in L\backslash K,\ \bar{w}\sigma \in K]. \tag{4}$$

or equivalently

$$\exists A, \bar{A} \text{ s.t. } AP(w) \cap \bar{A}P(\bar{w}) \neq \emptyset \\ \Rightarrow \quad \forall \sigma \in \Sigma : w\sigma \notin L \text{ or } \bar{w}\sigma \notin L \text{ or } w\sigma, \bar{w}\sigma \in K \text{ or } w\sigma, \bar{w}\sigma \in L\backslash K. \tag{5}$$

In words, observability means that we cannot find two attacks $A, \bar{A} \in \mathcal{A}$ that would result in the same observation $y \in \Delta^*$ for two strings $w, \bar{w} \in K$ such that $(w, w) \notin \mathrm{act}_{K \subset L}$, i.e., two $w, \bar{w} \in K$ such that one will transition to an element in K and the other to an element outside K, by the concatenation of the same symbol $\sigma \in \Sigma$.

The following theorem is the main result of this section. When the set of attacks $\mathcal{A}$ contains only A_{id}, it specializes to the classical result of [13].

Theorem 1 *For every nonempty prefix-closed set $K \subset L$ and every attack set $\mathcal{A}$:*

1. *there exists a solution f to the supervision of K under the attack set $\mathcal{A}$ if and only if K is controllable and P-observable for $\mathcal{A}$;*
2. *if K is controllable and P-observable for $\mathcal{A}$, the map $f : \bigcup_{A \in \mathcal{A}} AP(L) \to \Sigma$ defined by*

$$f(y) := \Sigma_{\mathrm{uc}} \cup \Big\{\sigma \in \Sigma_{\mathrm{c}} : \exists w \in K, A \in \mathcal{A} \text{ s.t. } [y \in AP(w),\ w\sigma \in K]\Big\},$$
$$\forall y \in \bigcup_{A \in \mathcal{A}} AP(L). \tag{6}$$

is a solution to the supervision of K under the attack set $\mathcal{A}$. □

Theorem 1 enables us to conclude that if K is controllable and P-observable for $\mathcal{A}$ then there exists a Stackelberg equilibrium policy for the leader (supervisor) that guarantees victory and Theorem 1 provides one such policy in (6). Conversely, if K is not controllable or not P-observable for $\mathcal{A}$, then there exists no policy for the supervisory that can guarantee victory for an adversary that knows the policy of the supervisory (in a Stackelberg sense) and has full information about the game. However, this result does not provide a solution to scenarios in which K is not controllable or not P-observable for $\mathcal{A}$, but the supervisor does not advertise its policy or the adversary does not have full information about the game. We conjecture that in such scenarios, equilibrium policies for the supervisor will be stochastic (either in a Nash or Stackelberg sense) and the value of the game will be a probability of victory in the (open) set $(0, 1)$.

Remark 2 In the proof of Theorem 1, we will see that to conclude that K is controllable, it suffices that $L^{\max}_{f,A} = K$ or $L^{\min}_{f,A} = K$ for some supervisory f and attack $A \in \mathcal{A}$. □

As in the classic supervisory control problem, the fact that controllability of K is a necessary condition for the existence of a supervisor is to be expected since, if $K\Sigma_{\mathrm{uc}} \cap L$ were not a subset of K, there would exist a string $\bar{w} \in K\Sigma_{\mathrm{uc}} \cap L$, $\bar{w} \notin K$ that should not be accepted (because $\bar{w} \notin K$) and yet $\bar{w}$ is the extension of a string that should be accepted (because it belongs to K) followed by an uncontrolled event in Σ_{uc}. Once we properly parse the meaning of P-observability for an attack set $\mathcal{A}$, the necessity of this condition is easy to understand: As noted above, when this condition does not hold, it is possible to find two attacks $A, \bar{A} \in \mathcal{A}$ that would result in the same observation $y \in \Delta^*$ for two distinct strings $w, \bar{w} \in K$, and yet w would transition to an element in K and $\bar{w}$ to an element outside K. The existence of such strings would mean that, upon observing y, the supervisor could not decide whether or not to enable the next transition. It turns out that these two conditions are also sufficient to guarantee that we can construct a P-supervisor f for that guarantees that $L^{\min}_{f,A} = L^{\max}_{f,A} = K$ for all $A \in \mathcal{A}$ and Theorem 1 provides such a supervisory. The following alternative characterization of observability under attacks will be used in the sufficiency proof.

Proposition 1 *Suppose that prefix-closed $K \subset L$ is controllable. Then K is P-observable for the set of attacks $\mathcal{A}$ if and only if for all $w, \bar{w} \in K$, $\sigma \in \Sigma_{\mathrm{c}}$, $A, \bar{A} \in \mathcal{A}$, the following statement holds:*

$$[AP(w) \cap \bar{A}P(\bar{w}) \neq \emptyset, \quad w\sigma \in K, \quad \bar{w}\sigma \in L] \quad \Rightarrow \quad \bar{w}\sigma \in K. \tag{7}$$

□

Proof of Proposition 1 To prove this result, we use the necessary and sufficient condition (4) for observability.

($\Rightarrow$) Suppose that K is P-observable for $\mathcal{A}$, and suppose that $w, \bar{w} \in K$, $\sigma \in \Sigma_{\mathrm{c}}$, $A, \bar{A} \in \mathcal{A}$ satisfy $AP(w) \cap \bar{A}P(\bar{w}) \neq \emptyset$, $w\sigma \in K$, and $\bar{w}\sigma \in L$. Then (4) leads to $\bar{w}\sigma \in K$.

($\Leftarrow$) Suppose that (7) holds for all $w, \bar{w} \in K$, $\sigma \in \Sigma_{\mathrm{c}}$, $A, \bar{A} \in \mathcal{A}$. Let $w, \bar{w} \in K$ satisfy $AP(w) \cap \bar{A}P(\bar{w}) \neq \emptyset$ for some $A, \bar{A} \in \mathcal{A}$. Since K is controllable, if $\sigma \in \Sigma_{\mathrm{uc}}$ and $w\sigma, \bar{w}\sigma \in L$, then $w\sigma, \bar{w}\sigma \in K$. Moreover, we see from (7) that, for all $\sigma \in \Sigma_{\mathrm{c}}$, if $w\sigma \in K$ and $\bar{w}\sigma \in L$, then $\bar{w}\sigma \in K$. Exchanging w and $\bar{w}$, we also have if $\bar{w}\sigma \in K$ and $w\sigma \in L$, then $w\sigma \in K$. Thus there does not exist $\sigma \in \Sigma$ such that $[w\sigma \in K,\ \bar{w}\sigma \in L\backslash K]$ or $[w\sigma \in L\backslash K,\ \bar{w}\sigma \in K]$. □

Proof of Theorem 1 To prove necessity in item 1, assume that there exists a valid P-supervisor f such that $L_{f,A}^{\min} = L_{f,A}^{\max} = K$ for all $A \in \mathcal{A}$. To prove that K is controllable, we (only) use the fact that $K = L_{f,A}^{\min}$ and pick some word $\bar{w} \in K\Sigma_{\mathrm{uc}} \cap L$. Such a word must be of the form $\bar{w} = w\sigma \in L$ such that $w \in K = L_{f,A}^{\min}$ for all $A \in \mathcal{A}$ and $\sigma \in \Sigma_{\mathrm{uc}}$. But then,

$$w \in L_{f,A}^{\min}, \quad w\sigma \in L, \quad \forall y \in AP(w),\ \sigma \in \Sigma_{\mathrm{uc}} \subset f(y).$$

The definition of $L_{f,A}^{\min}$ thus allows us to conclude that $\bar{w} = w\sigma \in L_{f,A}^{\min} = K$ for all $A \in \mathcal{A}$, which shows that $K\Sigma_{\mathrm{uc}} \cap L \subset K$ and therefore K is controllable.

To prove that K is P-observable, we use the fact that $K = L_{f,A}^{\min} = L_{f,A}^{\max}$ for all $A \in \mathcal{A}$ and pick a pair of words $w, \bar{w} \in K$ such that

$$\exists A, \bar{A} \text{ s.t. } AP(w) \cap \bar{A}P(\bar{w}) \neq \emptyset,$$

and an arbitrary symbol $\sigma \in \Sigma$ such that $w\sigma, \bar{w}\sigma \in L$. If $w\sigma \in K = L_{f,A}^{\min}$, then by the definition of $L_{f,A}^{\min}$, we must have

$$w \in L_{f,A}^{\min} = K, \quad w\sigma \in L, \quad \forall y \in AP(w),\ \sigma \in g(y).$$

Since $AP(w) \cap \bar{A}P(\bar{w}) \neq \emptyset$, we must then have

$$\bar{w} \in L_{f,\bar{A}}^{\max} = K, \quad \bar{w}\sigma \in L, \quad \exists y \in \bar{A}P(\bar{w}) \text{ s.t. } \sigma \in f(y)$$

and consequently $\bar{w}\sigma \in L_{f,\bar{A}}^{\max} = K$. Alternatively, if $w\sigma \in L\backslash K$, then $w\sigma \notin K = L_{f,A}^{\max}$ and we must have

$$w \in L_{f,A}^{\max} = K, \quad w\sigma \in L, \quad \forall y \in AP(w),\ \sigma \notin f(y).$$

Since $AP(w) \cap \bar{A}P(\bar{w}) \neq \emptyset$, we must then have

$$\bar{w} \in L_{f,\bar{A}}^{\min} = K, \quad \bar{w}\sigma \in L, \quad \exists y \in \bar{A}P(\bar{w}) \text{ s.t. } \sigma \notin f(y).$$

and consequently $\bar{w}\sigma \notin L_{f,\bar{A}}^{\min} = K$. This shows that (5) holds and therefore K is P-observable for $\mathcal{A}$.

To prove sufficiency in item 1 (and also the statement in item 2), pick the supervisor according to (6). We prove by induction on the word length that the supervisor f so defined satisfies $K = L^{\max}_{f,A} = L^{\min}_{f,A}$ for all $A \in \mathcal{A}$. The basis of induction is the empty string ϵ that belongs to $L^{\max}_{f,A}$ because of the definition of this set and belongs to K because this set is prefix-closed.

Suppose now that K, $L^{\max}_{f,A}$, and $L^{\min}_{f,A}$ have exactly the same words of length $n \geq 0$, and pick a word $\bar{w}\sigma \in L^{\max}_{f,A}$ of length $n+1$. Since $\bar{w} \in L^{\max}_{f,A}$ has length n, we know by the induction hypothesis that $\bar{w} \in K$. On the other hand, since $\bar{w}\sigma \in L^{\max}_{f,A}$, we must have

$$\bar{w} \in L^{\max}_{f,A}, \quad \bar{w}\sigma \in L, \quad \exists y \in AP(\bar{w}) \text{ s.t. } \sigma \in f(y).$$

If $\sigma \in \Sigma_{\mathrm{uc}}$, then we see from controllability that $\bar{w}\sigma \in K$. Let us next consider the case $\sigma \in \Sigma_{\mathrm{c}}$. By the definition of f, $\sigma \in f(y)$ must mean that

$$\exists w \in K, \bar{A} \in \mathcal{A} \quad \text{s.t.} \quad [y \in \bar{A}P(w),\ w\sigma \in K].$$

We therefore have

$$w, \bar{w} \in K, \quad AP(\bar{w}) \cap \bar{A}P(w) \neq \emptyset, \quad w\sigma \in K, \quad \bar{w}\sigma \in L.$$

Proposition 1 therefore shows that $\bar{w}\sigma \in K$. This shows that any word of length $n+1$ in $L^{\max}_{f,A}$ also belongs to K. Since $L^{\min}_{f,A} \subset L^{\max}_{f,A}$, it follows that any word of length $n+1$ in $L^{\min}_{f,A}$ also belongs to K.

Conversely, pick a word $\bar{w}\sigma \in K \subset L$ of length $n+1$. Since K is prefix closed, $\bar{w} \in K$, and by the induction hypothesis that $\bar{w} \in L^{\min}_{f,A}$. To obtain $\bar{w}\sigma \in L^{\min}_{f,A}$, we need to show that

$$\bar{w} \in L^{\min}_{f,A}, \quad \bar{w}\sigma \in L, \quad \forall y \in AP(\bar{w}),\ \sigma \in f(y).$$

The first statement is a consequence of the induction hypothesis (as discussed above). The second statement is a consequence of the fact that $\bar{w}\sigma \in K \subset L$. Regarding the third statement, if $\sigma \in \Sigma_{\mathrm{uc}}$, then $\sigma \in f(y)$ for all $y \in AP(\bar{w})$ by definition. In order to show that if $\sigma \in \Sigma_{\mathrm{c}}$, then $\sigma \in f(y)$ for all $y \in AP(\bar{w})$, we need to prove that

$$\forall y \in AP(\bar{w}), \quad \exists w \in K,\ \bar{A} \in \mathcal{A} \text{ s.t. } [y \in \bar{A}P(w),\ w\sigma \in K]. \tag{8}$$

This holds for the particular case $\bar{A} = A$, $w = \bar{w} \in K$. Thus any word of length $n+1$ in K also belongs to $L^{\min}_{f,A}$ and hence also to $L^{\max}_{f,A} \supset L^{\min}_{f,A}$, which completes the induction step. □

Remark 3 Similarly, we can show that

$$\bar{f}(y) := \Sigma_{\mathrm{uc}} \cup \Big\{\sigma \in \Sigma_{\mathrm{c}} : \forall w \in K, A \in \mathcal{A} \text{ s.t. } [y \in AP(w),\ w\sigma \in L] \ \Rightarrow\ w\sigma \in K\Big\},$$
$$\forall y \in \bigcup_{A \in \mathcal{A}} AP(L) \tag{9}$$

is also a solution to the supervision of K under the attack set $\mathcal{A}$. The supervisor f in (6) is said to be *permissive*, while $\bar{f}$ in (9) is *anti-permissive* [25,32]. □

5 Output-Symbol Attacks

Given a set of symbols $\alpha \subset \Delta$ in the observation alphabet, one can define an observation attack $A_\alpha : \Delta^* \to 2^{\Delta^*}$ that maps to each string $u \in \Delta^*$ the set of all strings $v \in \Delta^*$ that can be obtained from u by an arbitrary number of insertions or deletions of symbols in α. We say that A_α corresponds to an *attack on the output symbols in* α. In this context, it is convenient to also define the corresponding α*-removal observation map* $R_{\neg\alpha} : \Delta \to (\Delta \cup \{\epsilon\})$ by

$$R_{\neg\alpha}(t) = \begin{cases} \epsilon & t \in \alpha \\ t & t \notin \alpha. \end{cases}$$

The α-removal observation map can be extended to a map defined for strings of output symbols in the same way as observation maps P. This α-removal observation map allows us to define $A_\alpha : \Delta^* \to 2^{\Delta^*}$ as follows:

$$A_\alpha(u) = \big\{v \in \Delta^* : R_{\neg\alpha}(u) = R_{\neg\alpha}(v)\big\}. \tag{10}$$

Note that the absence of attack A_{id} corresponds to an empty set $\alpha = \emptyset$. This type of attacks are precisely the ones we found in Example 1.

5.1 Observability Test

The next result shows that for attacks on output symbols, one can test observability under attacks by checking regular observability (without attacks) for an appropriate set of output maps. This means that the observability tests developed for the non-attacked case [2,24] can be used to determine observability under output-symbol attacks.

Theorem 2 *For every nonempty prefix-closed set $K \subset L$ and attack set $\mathcal{A} = \{A_{\alpha_1}, A_{\alpha_2}, \dots, A_{\alpha_M}\}$ consisting of $M \geq 1$ observation attacks, K is P-observable for the set of attacks $\mathcal{A}$ if and only if K is $(R_{\neg\alpha} \circ P)$-observable (in the classical sense, i.e., without attacks) for every set $\alpha := \alpha_i \cup \alpha_j$, $\forall i, j \in \{1, 2, \dots, M\}$.*

In essence, Theorem 2 states that to have P-observability for a set of output-symbol attacks, we need to pick every possible pair of two attacks A_{α_i}, A_{α_j} and ask whether we would have "classical" observability if we were to remove all symbols affected by the two attacks. This result can be counter-intuitive because even though we assume that we only have one attack in $\mathcal{A}$, we need to protect consider the effect of pairs of attacks. If we know which attack in $\mathcal{A}$ we had to face, then it would suffice to erase from the output all symbols corresponding to that particular attack. The problem is that we do not know which attack we are facing and this forces us to have more "redundancy" in the sense that we need a stronger version of observability. In fact, we will see in Theorem 3, that we can construct a supervisor to solve this problem precisely by removing more symbols than those an attacker could control and then checking for consistency across the decisions made by "classical" supervisors that operate on reduced sets of symbols.

The following result provides the key step needed to prove Theorem 2.

Lemma 1 *Given any two sets $\alpha_1, \alpha_2 \subset \Delta$ and $\alpha := \alpha_1 \cup \alpha_2$, we have that*

$$(v, \bar{v}) \in \ker R_{\neg\alpha} \quad \Rightarrow \quad A_{\alpha_1}(v) \cap A_{\alpha_2}(\bar{v}) \neq \emptyset, \tag{11}$$

where $\ker R_{\neg\alpha} := \{(v, \bar{v}) \in \Delta^* \times \Delta^* : R_{\neg\alpha}(v) = R_{\neg\alpha}(\bar{v})\}$. □

Proof of Lemma 1 To prove this result, we must show that given two words $v, \bar{v} \in \Delta^*$ such that $R_{\neg\alpha}(v) = R_{\neg\alpha}(\bar{v})$, there exists a third word $y \in \Delta^*$ that belongs both to $A_{\alpha_1}(v)$ and $A_{\alpha_2}(\bar{v})$, and therefore

$$R_{\neg\alpha_1}(v) = R_{\neg\alpha_1}(y), \quad R_{\neg\alpha_2}(\bar{v}) = R_{\neg\alpha_2}(y). \tag{12}$$

The desired word y can be constructed through the following steps:

1. Start with the word $y_1 = R_{\neg\alpha_1}(v)$, which is obtained by removing from v all symbols in α_1. Since $R_{\neg\alpha} = R_{\neg\alpha_2} \circ R_{\neg\alpha_1}$, we have that
$$R_{\neg\alpha_2}(y_1) = R_{\neg\alpha_2}(R_{\neg\alpha_1}(v)) = R_{\neg\alpha}(v) = R_{\neg\alpha}(\bar{v}),$$
and therefore the words y_1 and $\bar{v}$ still only differ by symbols in α_1 and α_2.
2. Construct y_2 by adding to y_1 suitable symbols in α_1 so that $R_{\neg\alpha_2}(y_2) = R_{\neg\alpha_2}(\bar{v})$. This is possible because y_1 and $\bar{v}$ only differ by symbols in α_1 and α_2. To get $R_{\neg\alpha_2}(y_2) = R_{\neg\alpha_2}(\bar{v})$, we do not care about the symbols in α_2 so we just have to insert into y_1 the symbols in α_1 that appear in $\bar{v}$ (at the right locations).
3. By construction, $R_{\neg\alpha_1}(y_2) = R_{\neg\alpha_1}(y_1) = y_1 = R_{\neg\alpha_1}(v)$, and hence the original v and y_2 only differ by symbols in α_1. Since $R_{\neg\alpha_2}(y_2) = R_{\neg\alpha_2}(\bar{v})$, it follows that $\bar{v}$ and y_2 only differ by symbols in α_2. We therefore conclude that $y := y_2$ indeed satisfies (12) and hence belongs to both $A_{\alpha_1}(v)$ and $A_{\alpha_2}(\bar{v})$. □

Proof of Theorem 2 By definition, K is P-observable for the set of attacks $\mathcal{A}$ if and only if
$$R_{A_{\alpha_i}, A_{\alpha_j}} \subset \mathrm{act}_{K \subset L}, \qquad \forall i, j \in \{1, 2, \ldots, M\}.$$
Also by definition, K is $(R_{\neg\alpha} \circ P)$-observable for the set $\alpha := \alpha_i \cup \alpha_j$ if and only if
$$\ker(R_{\neg\alpha} \circ P) \subset \mathrm{act}_{K \subset L}.$$
To prove the results, it therefore suffices to show that, for every $i, j \in \{1, 2, \ldots, M\}$, we have that
$$R_{A_{\alpha_i}, A_{\alpha_j}} = \ker(R_{\neg\alpha} \circ P), \qquad \alpha := \alpha_i \cup \alpha_j.$$
To show that this equality holds, first pick a pair $(w, \bar{w}) \in R_{A_{\alpha_i}, A_{\alpha_j}}$, which means that there exists a word $y \in \Delta^*$ that belongs both to $A_{\alpha_i} P(w)$ and $A_{\alpha_j} P(\bar{w})$, and therefore
$$\begin{aligned} y \in A_{\alpha_i} P(w) \quad &\Leftrightarrow \quad R_{\neg\alpha_i}\big(P(w)\big) = R_{\neg\alpha_i}(y) \\ y \in A_{\alpha_j} P(\bar{w}) \quad &\Leftrightarrow \quad R_{\neg\alpha_j}\big(P(\bar{w})\big) = R_{\neg\alpha_j}(y) \end{aligned}$$
But then, since $\alpha := \alpha_i \cup \alpha_j$, we have that $R_{\neg\alpha} = R_{\neg\alpha_j} \circ R_{\neg\alpha_i} = R_{\neg\alpha_i} \circ R_{\neg\alpha_j}$, and consequently
$$\begin{aligned} R_{\neg\alpha}\big(P(w)\big) &= R_{\neg\alpha_j}\Big(R_{\neg\alpha_i}\big(P(w)\big)\Big) = R_{\neg\alpha_j}\big(R_{\neg\alpha_i}(y)\big) \\ &= R_{\neg\alpha_i}\big(R_{\neg\alpha_j}(y)\big) = R_{\neg\alpha_i}\Big(R_{\neg\alpha_j}\big(P(\bar{w})\big)\Big) = R_{\neg\alpha}\big(P(\bar{w})\big). \end{aligned}$$
We have thus shown that $R_{A_{\alpha_i}, A_{\alpha_j}} \subset \ker(R_{\neg\alpha} \circ P)$. To prove the reverse inclusion, pick a pair $(w, \bar{w}) \in \ker(R_{\neg\alpha} \circ P)$, which means that
$$R_{\neg\alpha}\big(P(w)\big) = R_{\neg\alpha}\big(P(\bar{w})\big),$$
and therefore $\big(P(w), P(\bar{w})\big) \in \ker R_{\neg\alpha}$. In conjunction with Lemma 1, this gives
$$A_{\alpha_i}\big(P(w)\big) \cap A_{\alpha_j}\big(P(\bar{w})\big) \neq \emptyset.$$
Therefore we have $(w, \bar{w}) \in R_{A_{\alpha_i}, A_{\alpha_j}}$. This shows that $\ker(R_{\neg\alpha} \circ P) \subset R_{A_{\alpha_i}, A_{\alpha_j}}$, which concludes the proof. □

Example 2 (Multi-layer cyber attack to a computer system (cont.)) In Example 1 we considered M attacks $A_{\text{exploit i}}$, $i \in \{1, \dots, M\}$, each corresponding to an adversary having compromised the security system that logs the occurrence of "exploit i", allowing it to arbitrarily insert/remove the output symbols "exploit i" into/from the observation string. Each of these is an output-symbol attack A_α, with the set α including a single output symbol "exploit i". Using the A_α-notation introduced above, we can thus write the attack set $\mathcal{A}_{\text{log}-\text{attacks}}$ in (2) as

$$\mathcal{A}_{\text{log}-\text{attacks}} := \left\{A_\emptyset, A_{\{\text{exploit 1}\}}, A_{\{\text{exploit 2}\}}, \dots, A_{\{\text{exploit M}\}}\right\}.$$

Theorems 1 and 2 can be used to confirm our previous assertion that there exists a solution to the supervision of the language K_{safe} under the attack set $\mathcal{A}_{\text{log}-\text{attacks}}$ if and only if $M \geq 3$. This is because K_{safe} is P-observable for the attack set $\mathcal{A}_{\text{log}-\text{attacks}}$ if and only if $M \geq 3$:

1. For $M = 1$, K_{safe} is not P-observable for the attack set $\mathcal{A}_{\text{log}-\text{attacks}} := \left\{A_\emptyset, A_{\{\text{exploit 1}\}}\right\}$ because K_{safe} is not $(R_{\neg\{\text{exploit 1}\}} \circ P)$-observable. This is straightforward to conclude from Theorem 2 because if we remove from the output strings all symbols "exploit 1", the supervisor cannot distinguish between the "clean" and "s1" states.
2. For $M = 2$, K_{safe} is still not P-observable for the attack set $\mathcal{A}_{\text{log}-\text{attacks}} := \left\{A_\emptyset, A_{\{\text{exploit 1}\}}, A_{\{\text{exploit 2}\}}\right\}$ because K_{safe} is not $(R_{\neg\{\text{exploit 1,exploit 2}\}} \circ P)$-observable. Again, this is straightforward to conclude from Theorem 2 because if we remove from the output strings all symbols "exploit 1" and "exploit 2", the supervisor cannot distinguish between the "clean", "s1", "s2" states.
   This result may seem counter-intuitive because none of the attacks in $\mathcal{A}_{\text{log}-\text{attacks}}$ is actually able to insert/remove both symbols "exploit 1" and "exploit 2" and yet the test involves a system where we removed both symbols. To understand this apparent paradox, suppose that the supervisor observes the output string

   $$\{\text{grant access, exploit 1}\}.$$

   This output string could have resulted from two scenarios:

   (a) The event sequence is

   $$\{\text{grant access, exploit 1, exploit 2}\},$$

   but an output-symbol attack $A_{\{\text{exploit 2}\}}$ erased the symbol "exploit 2".

   (b) The event sequence is

   $$\{\text{grant access}\},$$

   but an output-symbol attack $A_{\{\text{exploit 1}\}}$ inserted the symbol "exploit 1".

   The existence of these two options is problematic because in the former case "grant access" must be disabled, whereas in latter case it must be enabled.
3. For $M \geq 3$, K_{safe} becomes P-observable for the attack set

   $$\mathcal{A}_{\text{log}-\text{attacks}} := \left\{A_\emptyset, A_{\{\text{exploit 1}\}}, \dots, A_{\{\text{exploit M}\}}\right\}$$

   because K_{safe} is $(R_{\neg\{\text{exploit i,exploit i}\}} \circ P)$-observable, for every $i, j \in \{1, 2, \dots, M\}$. Again, this is straightforward to conclude from Theorem 2 because even if we remove from the output strings all pairs of symbols "exploit i" and "exploit j", there will still remain a third symbol "exploit k" $k \neq i, k \neq j$ in the output string that allows the supervisor to deduce a transition out of the "clean" state. □

Remark 4 (Complexity) As noted in Remark 1, it is often possible to reduce the problem of finding a solution to the supervision under an attack set $\mathcal{A}$ to a classical supervisory control problem without attacks, at the expense of having to expand the state of the original automaton, essentially by replicating each state $M := |\mathcal{A}|$ times. For problems in which the complexity of testing observability is linear in the number of states of the original automaton, this reduction may be computationally more efficient than using the result in Theorem 2, which would require testing the observability of $\binom{M}{2} = O(M^2)$ systems.

The simple structure of Example 2 allowed us to prove observability for every $M \geq 3$ through a formal argument. However, more complicated networks of exploits typically require the use of computational tests for observability. If we had to perform a computational test for Example 2, Theorem 2 would require testing the observability of $\binom{M}{2} = (M^2+M)/2$ distinct systems, each with $O(M)$ states and transitions. For this problem, one can show that each of these tests would have a worst-case computational complexity $O(M^3)$, using the algorithm in [2, Section 3.7.3]. This would lead to a worst-case complexity $O(M^5)$. Alternatively, the expansion described in Remark 1 would lead to a single observability test for a system with $O(M^2)$ states and transitions. Even with more states, the observability test for this system based on the algorithm in [2, Section 3.7.3] would still only have a worst-case complexity $O(M^4)$. However, while sometimes attractive to *test observability*, we shall see shortly that the expansion described in Remark 1 generally results in much higher worst-case complexity for the *design of the supervisor*. $\square$

5.2 Supervisor Design

The following result shows that for attacks on output symbols, one can construct a solution to the supervision of $K \subset L$ under attacks by appropriately combining a set of classical supervisors, each designed for an appropriately defined output map without attacks. This allows us to reuse classical supervisor-design approaches and tools [2,7,10].

Theorem 3 *For a given nonempty prefix-closed set $K \subset L$ and attack set $\mathcal{A} = \{A_{\alpha_1}, A_{\alpha_2}, \dots, A_{\alpha_M}\}$ consisting of $M \geq 1$ observation attacks, assume that K is controllable and P-observable for $\mathcal{A}$. In view of Theorems 1 and 2, this means that, for every $i, j \in \{1, 2, \dots, M\}$ there exists a valid supervisor f_{ij} that generate the language K for the observation map $P_{ij} := R_{\neg(\alpha_i \cup \alpha_j)} \circ P$ (in the classical sense, i.e., without attacks). In this case, the following supervisor is a solution to the supervision of K under the attack set $\mathcal{A}$:*

$$f(y) := \bigcup_{i=1}^{M} f^i(y), \qquad f^i(y) := \bigcap_{j=1}^{M} \tilde{f}_{ij}(y), \qquad \forall y \in \bigcup_{i=1}^{M} A_{\alpha_i} P(L),$$

where

$$\tilde{f}_{ij}(y) := \begin{cases} f_{ij}(R_{\neg(\alpha_i \cup \alpha_j)}(y)) & y \in A_{\alpha_i} P(L) \cup A_{\alpha_j} P(L) \\ \Sigma_{\mathrm{uc}} & y \notin A_{\alpha_i} P(L) \cup A_{\alpha_j} P(L). \end{cases} \tag{13}$$

$\square$

Remark 5 As we shall see in the proof of Theorem 3, the lower branch of (13) can be set equal to any subset of Σ (possibly y-dependent) that contains Σ_{uc}. $\square$

Proof of Theorem 3 Since every f_{ij} is a valid supervisor, all the sets $f_{ij}(y)$ and $\tilde{f}_{ij}(y)$ contain Σ_{uc} and consequently so does $f(y)$, which proves that f is a valid supervisor. To complete the proof, it thus suffices to show that

$$L_{f,A_{\alpha_k}}^{\min} \supset K \supset L_{f,A_{\alpha_k}}^{\max}, \quad \forall A_{\alpha_k} \in \mathcal{A}.$$

First we prove $L_{f,A_{\alpha_k}}^{\min} \supset K$ by showing that every word w in K also belongs to $L_{f,A_{\alpha_k}}^{\min}$. We do this prove by induction on word length n. The basis of induction is trivially true because the empty string belongs to $L_{f,A_{\alpha_k}}^{\min}$ by construction. Suppose now that we have established that every word $w \in K$ of length n also belongs to $L_{f,A_{\alpha_k}}^{\min}$ and take an arbitrary word $w\sigma \in K$ of length $n+1$. Since K is prefix closed, we have that $w \in K$ and by the induction hypothesis that $w \in L_{f,A_{\alpha_k}}^{\min}$. In view of the definition of the minimal language, to prove that $w\sigma \in L_{f,A_{\alpha_k}}^{\min}$ we need to show that $\sigma \in f(y)$ for all $y \in A_{\alpha_k}P(w)$. To accomplish this, we pick an arbitrary string $y \in A_{\alpha_k}P(w)$ and note that, because every f_{kj} generates the language K for the observation map $P_{kj} := R_{\neg(\alpha_k \cup \alpha_j)} \circ P$, we have that

$$w\sigma \in K \quad \Leftrightarrow \quad w \in K,\ w\sigma \in L,\ \sigma \in f_{kj}\big(R_{\neg(\alpha_k \cup \alpha_j)}\big(P(w)\big)\big), \quad \forall j \in \{1, \dots, M\}. \tag{14}$$

Since $y \in A_{\alpha_k}P(w)$, we conclude from (10) that

$$R_{\neg(\alpha_k \cup \alpha_j)}(y) = R_{\neg(\alpha_k \cup \alpha_j)}\big(P(w)\big). \tag{15}$$

Moreover, we now that $w\sigma \in K$, therefore (14) and (15) imply that

$$\sigma \in f_{kj}\big(R_{\neg(\alpha_k \cup \alpha_j)}\big(P(w)\big)\big) = f_{kj}\big(R_{\neg(\alpha_k \cup \alpha_j)}(y)\big) = \tilde{f}_{kj}(y), \quad \forall j \in \{1, \dots, M\}.$$

We have thus shown that

$$\sigma \in f^k(y) = \bigcap_{j=1}^{M} \tilde{f}_{kj}(y) \subset f(y).$$

This confirms that $w\sigma \in L_{f,A_{\alpha_k}}^{\min}$, which completes the induction step.

We show next that $L_{f,A_{\alpha_k}}^{\max} \subset K$, also using an induction argument. As before, the basis of induction is trivial so to establish the induction step, we suppose that we have established that every word $w \in L_{f,A_{\alpha_k}}^{\max}$ of length n also belongs to K and take an arbitrary word $w\sigma \in L_{f,A_{\alpha_k}}^{\max}$ of length $n+1$. By the induction hypothesis, we have that $w \in K$ and to prove the induction step, we need to show that $w\sigma \in K$. From the construction of $L_{f,A_{\alpha_k}}^{\max}$, we known that $w\sigma \in L_{f,A_{\alpha_k}}^{\max}$ implies that $w \in L_{f,A_{\alpha_k}}^{\max}$ and

$$\exists y \in A_{\alpha_k}P(w) \quad \text{s.t.} \quad \sigma \in f(y) := \bigcup_{i=1}^{M} f^i(y)$$

and therefore

$$\begin{aligned} &\exists y \in A_{\alpha_k}P(w),\ i \in \{1, \dots, M\} \\ &\text{s.t.} \quad \sigma \in f^i(y) := \bigcap_{j=1}^{M} \tilde{f}_{ij}(y) \subset \tilde{f}_{kj}(y) = f_{kj}\big(R_{\neg(\alpha_k \cup \alpha_j)}(y)\big). \end{aligned}$$

Also here (15) holds and we conclude that

$$\sigma \in f_{kj}\big(R_{\neg(\alpha_k \cup \alpha_j)}(y)\big) = f_{kj}\big(R_{\neg(\alpha_k \cup \alpha_j)}\big(P(w)\big)\big).$$

Again, because f_{kj} generates the language K for the observation map $P_{kj} := R_{\neg(\alpha_k \cup \alpha_j)} \circ P$, the equivalence in (14) holds. Since we have shown that all the conditions in the right-hand side of (14) hold, we conclude that $w\sigma \in K$, which completes the proof of the induction step. □

Example 3 (Multi-layer cyber attack to a computer system (cont.)) To construct a supervisor for the problem described in Examples 1–2 using Theorem 3 we need valid supervisors f_{ij} that generate the language K_{safe} for the observation maps $P_{ij} := R_{\neg(\alpha_i \cup \alpha_j)} \circ P$ for all $\alpha_i, \alpha_j \in \mathcal{A}_{\text{log-attacks}}$. For $M \geq 3$, all these supervisors can be the same and defined by

$$f_{ij}(y) = \begin{cases} \Sigma_{\text{uc}} \cup \{\text{grant access}\} & T(y) \text{ does not contain any “exploit” symbol} \\ \Sigma_{\text{uc}} \cup \{\text{deauthorize}\} & T(y) \text{ contains some “exploit” symbols,} \end{cases}$$

where $T(y)$ denotes the last output symbols in the string y, starting right after the last “grant access” or “deauthorize” symbols, or from the beginning of y if it does not contain such symbols. Essentially $T(y)$ contains the output symbols starting from the last time that the system was in the “clean” state. These supervisors enforce the language K_{safe} because if $T(y)$ does not contain any “exploit” symbol for the output map $P_{ij} := R_{\neg(\alpha_i \cup \alpha_j)} \circ P$, then we must be in the “clean” state (in case P_{ij} did not remove any “exploit” symbol), in the “s1” state (in case P_{ij} removed one “exploit” symbols), or in the “s2” state (in case P_{ij} removed two “exploit” symbols). In either case, it is okay to enable the “grant access” transition and disable the “deauthorize” transition. On the other hand, if $T(y)$ does contain any “exploit” symbol, then we must be in one of the “si” states and it is safe to enable the “deauthorize” transition and disable the “grant access” transition.

For these supervisors f_{ij}, we have

$$\tilde{f}_{ij}(y) = \begin{cases} \Sigma_{\text{uc}} \cup \{\text{grant access}\} & T(y) \text{ does not contain any “exploit” symbol other than} \\ & \quad \text{those in } \alpha_i \cup \alpha_j, \\ \Sigma_{\text{uc}} \cup \{\text{deauthorize}\} & T(y) \text{ contains some “exploit” symbols other those in } \alpha_i \cup \alpha_j, \end{cases}$$

$$f^i(y) = \begin{cases} \Sigma_{\text{uc}} \cup \{\text{grant access}\} & T(y) \text{ does not contain any “exploit” symbol, other than} \\ & \quad \text{that in } \alpha_i, \\ \Sigma_{\text{uc}} & T(y) \text{ contains one or more copies of the same “exploit”} \\ & \quad \text{symbol, not in } \alpha_i, \\ \Sigma_{\text{uc}} \cup \{\text{deauthorize}\} & T(y) \text{ contains one or more copies of two distinct “exploit”} \\ & \quad \text{symbols not in } \alpha_i, \end{cases}$$

$$f(y) = \begin{cases} \Sigma_{\text{uc}} \cup \{\text{grant access}\} & T(y) \text{ does not contain any “exploit” symbol,} \\ \Sigma_{\text{uc}} \cup \{\text{grant access}\} & T(y) \text{ contains one or more copies of a single “exploit”} \\ & \quad \text{symbol,} \\ \Sigma_{\text{uc}} \cup \{\text{deauthorize}\} & T(y) \text{ contains one or more copies of two distinct “exploit”} \\ & \quad \text{symbols.} \end{cases}$$

The implementation of this supervisor could be done with a finite state machine with $M + 1$ symbols, one state would correspond to “no exploit symbol observed in $T(y)$” and the remaining M states would be use to memorize the index of the 1st exploit symbol observed in $T(y)$. □

Remark 6 (Complexity) While testing observability has polynomial complexity in the number of states of the *plant automaton* G representing the original language L and on the number of states of the *specification automaton* G_K representing the desired language K, the number of states of the supervisor is typically exponential in the number of states of the plant automaton [2,24]. To apply Theorem 3, we thus need to design $O(|\mathcal{A}|^2)$ supervisors, each with worst-case exponential complexity in the number of states of G, leading to a complexity of $O(|\mathcal{A}|^2 \mathrm{e}^{|X|})$, where $|X|$ denotes the number of states of G. If we were to use instead the reduction to a supervisory control problem without attacks discussed in Remark 1, we would need to solve a single supervisor design problem. However, this problem would have worst-case exponential complexity on the number of states of an expanded automaton, leading to a much worse worst-case complexity of $O(\mathrm{e}^{|\mathcal{A}||X|})$. □

6 Conclusion

Motivated by computer security applications, we introduced a multi-adversary version of the DESs supervisory control problem, where the supervisor is asked to enforce a desired language based on measurements that have been corrupted by one of several potential opponents, not knowing which is the actual opponent. For the particular case of output-symbol attacks, we have shown that testing observability and constructing a supervisor that solves this multi-adversary problem can be done using tools developed for the classical supervisory control problem, without expanding the state-space of the original plant automaton.

Important direction for future research include the development of efficient computational techniques for adversaries that use attacks more general than output-symbol attacks and investigating distributed solutions to the problem considered here. The latter is especially interesting for scenarios with distributed cyber-security sensors. As noted above, this paper also does not provide results for scenarios in which the controllability and observability conditions do not hold. We conjecture that, in the absence of observability under attacks, saddle-point policies for the attacker/supervisory that maximize/minimize the probability that the supervisor will make a mistake will be mixed, but we do not know of efficient algorithms to compute such policies.

References

1. Amin S, Litrico X, Sastry S, Bayen AM (2013) Cyber security of water SCADA systems-part I: analysis and experimentation of stealthy deception attacks. IEEE Trans Control Syst Technol 21:1963–1970
2. Cassandras CG, Lafortune S (2008) Introduction to discrete event systems, 2nd edn. Springer, Berlin
3. Chong MS, Wakaiki M, Hespanha JP (2015) Observability of linear systems under adversarial attacks. In: Proceedings of the 2015 American control conference
4. Corporation TM (2018) Common vulnerabilities and exposures (CVE) list. https://cve.mitre.org. Accessed 1 Oct 2017
5. Dubreil J, Darondeau P, Marchand H (2010) Supervisory control for opacity. IEEE Trans Autom Control 55:1089–1100
6. Fawzi H, Tabuada P, Diggavi S (2014) Secure estimation and control for cyber-physical systems under adversarial attacks. IEEE Trans Autom Control 59:1454–1467
7. Feng L, Wonham W (2006) TCT: a computation tool for supervisory control synthesis. In: 8th international workshop on discrete event systems, pp 388–389
8. Hubballi N, Biswas S, Roopa S, Ratti R, Nandi S (2011) LAN attack detection using discrete event systems. ISA Trans 50:119–130

9. Ji Y, Lee S, Downing E, Wang W, Fazzini M, Kim T, Orso A, Lee W (2017) Rain: refinable attack investigation with on-demand inter-process information flow tracking. In: Proceedings of the 2017 ACM SIGSAC conference on computer and communications security. ACM, pp 377–390
10. Lafortune S, Ricker L (2014) Desuma2. https://wiki.eecs.umich.edu/desuma/. Accessed 1 Oct 2017
11. Lin F (1993) Robust and adaptive supervisory control of discrete event systems. IEEE Trans Automt Control 38:1848–1852
12. Paoli A, Sartini M, Lafortune S (2011) Active fault tolerant control of discrete event systems using online diagnostics. Automatica 47:639–649
13. Ramadge PJ, Wonham WM (1989) The control of discrete event systems. Proc IEEE 77:81–98
14. Saboori A, Hadjicostis CN (2012) Opacity-enforcing supervisory strategies via state estimator constructions. IEEE Trans Autom Control 57:1155–1165
15. Saboori A, Zad SH (2006) Robust nonblocking supervisory control of discrete-event systems under partial observation. Syst Control Lett 55:839–848
16. Sánchez AM, Montoya FJ (2006) Safe supervisory control under observability failure. Discrete Event Dyn Syst Theory Appl 16:493–525
17. Sheyner O, Wing J (2004) Tools for generating and analyzing attack graphs. In: de Boer FS, Bonsangue MM, Graf S, de Roever WP (eds) Formal methods for components and objects: second international symposium, FMCO 2003, Leiden, The Netherlands, November 4–7, 2003. Revised Lectures, no. 3188 in Lecture Notes on Computer Science. Springer, Berlin, pp. 344–371
18. Shoukry Y, Tabuada P (2016) Event-triggered state observers for sparse noise/attacks. IEEE Trans Autom Control 61(8):2079–2091
19. Shu S, Lin F (2014) Fault-tolerant control for safety of discrete-event systems. IEEE Trans Autom Sci Eng 11:78–89
20. Takai S (2000) Robust supervisory control of a class of timed discrete event systems under partial observation. Syst Control Lett 39:267–273
21. Takai S, Oka Y (2008) A formula for the supremal controllable and opaque sublanguage arising in supervisory control. SICE J Control Meas Syst Integr 1:307–311
22. Teixeira A, Shames I, Sandberg H, Johansson KH (2015) A secure control framework for resource-limited adversaries. Automatica 51:135–148
23. Thorsley D, Teneketzis D (2006) Intrusion detection in controlled discrete event systems. In: Proceedings of the 45th conference on decision and control
24. Tsitsiklis JN (1989) On the control of discrete-event dynamical systems. Math Control Signals Syst 2:96–107
25. Ushio T, Takai S (2009) Supervisory control of discrete event systems modeled by Mealy automata with nondeterministic output functions. In: Proceedings of the 2009 American control conference
26. Ushio T, Takai S (2016) Nonblocking supervisory control of discrete event systems modeled by Mealy automata with nondeterministic output functions. IEEE Trans Autom Control 61(3):799–804
27. Whittaker SJ, Zulkernine M, Rudie K (2008) Toward incorporating discrete-event systems in secure software development. In: Proceedings of ARES'08
28. Wonham WM (2010) Supervisory control of discrete-event systems. http://www.se.wtb.tue.nl/_media/wonham/wonham_scdes2010.pdf. Accessed 1 Oct 2017
29. Wu YC, Lafortune S (2014) Synthesis of insertion functions for enforcement of opacity security properties. Automatica 50:1336–1348
30. Xu S, Kumar R (2009) Discrete event control under nondeterministic partial observation. In: Proceedings of IEEE CASE'09
31. Yin X (2017) Supervisor synthesis for Mealy automata with output functions: a model transformation approach. IEEE Trans Autom Control 62(5):2576–2581
32. Yoo TS, Lafortune S (2002) A general architecture for decentralized supervisory control of discrete-event systems. Discrete Event Dyn Syst Theory Appl 12:335–377