Automatica 50 (2014) 1336–1348

Contents lists available at ScienceDirect

Automatica

journal homepage: www.elsevier.com/locate/automatica

Synthesis of insertion functions for enforcement of opacity security properties☆

Yi-Chin Wu [1], Stéphane Lafortune

Department of EECS, University of Michigan, Ann Arbor, United States

ARTICLE INFO

Article history:
Received 19 May 2013
Received in revised form
23 October 2013
Accepted 12 February 2014
Available online 17 March 2014

Keywords:
Discrete event systems
Opacity

ABSTRACT

Opacity is a confidentiality property that characterizes whether a "secret" of a system can be inferred by an outside observer called an "intruder". In this paper, we consider the problem of enforcing opacity in systems modeled as partially-observed finite-state automata. We propose a novel enforcement mechanism based on the use of insertion functions. An insertion function is a monitoring interface at the output of the system that changes the system's output behavior by inserting additional observable events. We define the property of "i-enforceability" that an insertion function needs to satisfy in order to enforce opacity. I-enforceability captures an insertion function's ability to respond to every system's observed behavior and to output only modified behaviors that look like existing non-secret behaviors. Given an insertion function, we provide an algorithm that verifies whether it is i-enforcing. More generally, given an opacity notion, we determine whether it is i-enforceable or not by constructing a structure called the "All Insertion Structure" (AIS). The AIS enumerates all i-enforcing insertion functions in a compact state transition structure. If a given opacity notion has been verified to be i-enforceable, we show how to use the AIS to synthesize an i-enforcing insertion function.

1. Introduction

Cybersecurity is an increasingly important issue as computers and networks are integrated into every aspect of our lives. Ranging from stealing personal identification information to infiltrating national websites to conduct espionage, attackers in cyberspace target a wide range of systems. In this research, we study an important cybersecurity property called "opacity", based on the control theory for Discrete Event Systems (DES). Opacity characterizes whether some secret information of a system can be inferred by outside observers. Introduced in the computer science community (Mazaré, 2003), opacity has become an active research topic in DES, as this class of dynamic systems provides suitable formal models and analytical techniques for investigating opacity (Bryans, Koutny, Mazaré, & Ryan, 2008; Bryans, Koutny, & Ryan, 2005; Saboori & Hadjicostis, 2007).

The ingredients of the DES formulation of an opacity problem are: (1) the system has a *secret*; (2) the system is only partially observable; (3) the *intruder* is an observer who has full knowledge of the system structure. The secret of the system is *opaque* if for every behavior relevant to the secret, there is an observationally-equivalent behavior that is not relevant to the secret. For simplicity, we call the former "secret behavior" and the latter "non-secret behavior". When opacity holds, the intruder is not sure if the secret or the non-secret has occurred. The system is guaranteed the "plausible deniability" of the secret.

The secret of the system can be defined by any representation in the given DES model, such as states and languages. With different representations of the secret, various opacity notions have been introduced in the literature. For example, the secret can be defined in terms of a sublanguage, the current state, the initial state, a sequence of K states, or a set of initial–final state pairs. Opacity notions corresponding to these various cases have been investigated using different DES modeling formalisms such as Petri nets, labeled transition systems, and automata; see e.g., (Bryans et al., 2008, 2005; Cassez, Dubreil, & Marchand, 2012; Lin, 2011; Saboori & Hadjicostis, 2007; Wu & Lafortune, 2013).

In this paper, we consider opacity problems in DES modeled as finite-state automata (FSA). Given an opacity notion, methods for verifying if the secret is opaque or not have been investigated in Cassez et al. (2012), Lin (2011), Saboori and Hadjicostis (2008), and Wu and Lafortune (2013). When a secret is not opaque, the

☆ This work was partially supported by the NSF Expeditions in Computing project ExCAPE: Expeditions in Computer Augmented Program Engineering (grant CCF-1138860). The material in this paper was presented at the 51st IEEE Conference on Decision and Control (CDC 2012), December 10–13, 2012, Maui, Hawaii, USA. This paper was recommended for publication in revised form by Associate Editor Bart De Schutter under the direction of Editor Ian R. Petersen.

E-mail addresses: ycwu@umich.edu (Y.-C. Wu), stephane@umich.edu (S. Lafortune).

[1] Tel.: +1 7348468150; fax: +1 734 763 8041.

http://dx.doi.org/10.1016/j.automatica.2014.02.038
0005-1098/

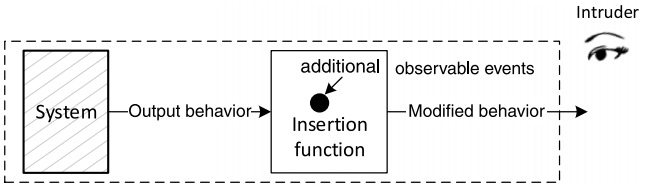

Fig. 1. The insertion mechanism.

ensuing question is: How can we enforce the secret to be opaque? The focus of this paper is the development of an enforcement mechanism for opacity notions. Many prior studies have designed the minimally-restrictive opacity-enforcing supervisory controller based on the supervisory control theory of DES; see, e.g., (Dubreil, Darondeau, & Marchand, 2010; Saboori & Hadjicostis, 2012; Takai & Oka, 2011). The system behavior under supervisory control is restricted such that a behavior is disabled by feedback control if it is going to reveal the secret. While the secret under this controller is guaranteed to be opaque, this approach does not apply to situations where the system must execute its full behavior. Another approach to enforce opacity notions is to use a dynamic observer, which dynamically modifies the observability of every system event (Cassez et al., 2012). Unlike a supervisory controller, a dynamic observer allows the full system behavior, but it also erases some information that was to be output. Such erasure could create "new" observed strings that would not be seen in the original system (under a static observable projection). This is not desirable because it reveals clues about the defense model to the intruder. Another enforcement approach that allows the full system behavior is the runtime enforcement mechanism in Falcone and Marchand (2013). This enforcement mechanism employs delays when outputting executions to enforce K-step opacity. However, this method applies only to secrets for which time duration is of concern.

There are many application areas where the above enforcement mechanisms are not suitable. For example, in location-based services, neither restricting users' query behaviors nor erasing the location information in queries is desirable. Also, to capture location privacy for users' home locations, we may need the notion of initial-state opacity, for which delaying queries as in Falcone and Marchand (2013) is not practical. We propose a new enforcement mechanism that overcomes these limitations, based on the use of insertion functions at run-time. As shown in Fig. 1, an insertion function is a monitoring interface placed at the output of the system. It receives an output behavior from the system, inserts an additional observable string if necessary in order to prevent the system from revealing the secret, and outputs the modified behavior. The intruder is assumed to have no knowledge of the insertion function at the outset. But by knowing the system structure, the intruder may learn the existence of the insertion function from observing the modified output. This would occur if for instance the insertion function inserts random events. Hence, our goal is to synthesize insertion functions such that they protect the secret behavior and never reveal to the intruder that an insertion function has been implemented.

Specifically, every modified output needs to replicate some original non-secret behavior; moreover, insertion functions should not interact with the system and must allow all system behaviors. It is the combination of these two requirements that makes the design of insertion functions challenging. An insertion can only be made if it assures protection of both the current and the future system output. We characterize the requirements as the *i-enforceability* property. Given an opacity notion, it is called *i-enforceable* if there exists an i-enforcing insertion function.

The first question we consider is whether a given insertion function is i-enforcing. For this purpose, we construct an equivalent automaton that captures the modified system and use it to verify i-enforceability of the function. Then, the next question we address is whether there exists an i-enforcing insertion function that enforces the secret of a given system to be opaque. We provide an algorithmic procedure that verifies i-enforceability of a given opacity notion. The algorithm is based on a structure called the "All Insertion Structure" (AIS), which enumerates in a compact manner all i-enforcing insertion functions. To verify if opacity is i-enforceable, it suffices to determine if the AIS is the empty structure or not. Furthermore, if opacity is i-enforceable, we show how to use the AIS to *synthesize* an i-enforcing insertion function. All these algorithms are general enough so that they apply to four opacity notions: current-state opacity, initial-state opacity, language-based opacity, and initial-and-final-state opacity.

Other works in the computer science literature have also used insertion functions to enforce security properties; see e.g., (Ligatti, Bauer, & Walker, 2005; Schneider, 2000). However, the class of security policies considered does not include opacity. To the best of our knowledge, our work is the first to address opacity enforcement using insertion functions.

The remaining sections of this paper are organized as follows. Section 2 introduces the system model and relevant definitions. Section 3 reviews the basics of the opacity problem. Section 4 formally defines insertion functions and the i-enforcing property. In Section 5, we present our algorithm for verifying if a given insertion function is i-enforcing. In Section 6, we show the 4-stage construction of the All Insertion Structure (AIS) and verify i-enforceability of a given opacity notion using the AIS. Section 7 presents the synthesis of an i-enforcing insertion function. The complexity of the AIS is analyzed in Section 8. Section 9 discusses opacity enforcement by insertion in the context of opaque communications. Section 10 discusses how intruder's knowledge of the insertion function affects our results. Finally, Section 11 concludes the paper. A preliminary and partial version of this paper was presented in the conference paper (Wu & Lafortune, 2012). Specifically, the materials in Sections 5 and 8–10 are new; the algorithms in Section 6 are improved; full proofs of the theorems and examples omitted in Wu and Lafortune (2012) are presented; and additional explanation and discussion have been added.

2. Preliminaries

Automata models

We consider opacity problems in DES systems modeled as finite-state automata. An automaton $G = (X, E, f, X_0)$ has a set of states X, a set of events E, a deterministic state transition function $f : X \times E \to X$, and a set of initial-states X_0. In opacity problems, the initial state need not be known *a priori* by the intruder and thus we include a set of initial states X_0 in the definition of G. The language generated by G is the system behavior that is defined by $\mathscr{L}(G, X_0) := \{t \in E^* : (\exists i \in X_0)[f(i, t) \text{ is defined}]\}$. For simplicity, we will use $\mathscr{L}(G)$ if the set of initial states is clearly defined. In general, the system is partially observable. Hence, the event set is partitioned into an observable set E_o and an unobservable set E_{uo}. Given an event $e \in E$, its observation is the output of the natural projection $P : E \to E_o$ such that $P(e) = e$ if $e \in E_o$ and $P(e) = \varepsilon$ if $e \in E_{uo} \cup \{\varepsilon\}$ where ε is the empty string. With this definition at hand, projection P is extended from $E \to E_o$ to $P : E^* \to E_o^*$ in a recursive manner: $P(te) = P(t)P(e)$ where $t \in E^*$ and $e \in E$.

Inserted event set E_i

In our enforcement mechanism, the insertion function can insert any event in E_o. Such an inserted event looks identical to a system observable event. However, for the purpose of discussion, we want to clearly distinguish between inserted events and observable events. Thus, we define a set of inserted events E_i, where an