Computer Languages, Systems & Structures 54 (2018) 273–296

Contents lists available at ScienceDirect

Computer Languages, Systems & Structures

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

On transparent value-sensitive run-time monitoring for information flow policies

Fatemeh Imanimehr, Mehran S. Fallah*

Amirkabir University of Technology (Tehran Polytechnic), Iran

ARTICLE INFO

Article history:
Received 9 February 2018
Revised 13 June 2018
Accepted 11 July 2018
Available online 18 July 2018

Keywords:
Corrective enforcement
Information flow policies
Run-time monitoring
Transparency
Value sensitivity

ABSTRACT

Run-time monitoring proves to be a successful mechanism for enforcing information flow policies. The main challenge, however, is to achieve transparency which generally demands that monitors should make as minimal changes to program executions as possible. We investigate the level of transparency a monitor can attain when it uses static and dynamic information about possible values of program variables. To study such value-sensitive monitors, we consider two paradigms of corrective enforcement that indeed formulate the ultimate transparency. Then, we propose a number of purely dynamic and hybrid value-sensitive monitors for some known noninterference policies. Although value sensitivity leads to more transparent monitors, it can hardly provide the ultimate transparency. This motivates us to give partial orders reflecting the level of transparency a monitor may achieve and to locate monitors on the proposed partial orders. It is shown that hybrid value-sensitive monitors can correctively enforce so-called termination-insensitive noninterference only if they can compute the set of possible values of certain variables. We also prove that such an ideal monitor is the only hybrid monitor, in the large class of monitors identified in this paper, that can be more transparent than purely dynamic monitors.

1. Introduction

Run-time monitors enforce security policies by observing and transforming the executions of target programs. An execution is defined as a possibly infinite sequence of actions a program takes at run time. Monitors may change the actions in an execution of the target merely on the basis of the actions already observed in that execution. A monitor may additionally analyze the code of the program being monitored to obtain information about its other related executions. Monitors of the former class are called purely dynamic monitors and the ones of the latter are known as hybrid monitors.

Many policies, such as access control policies, can simply be defined as a set of valid (secure) executions. Such policies are called properties. A monitor is said to be sound for a property if it transforms any program execution into a secure execution. It is also called transparent if the transformed execution is as similar to the original execution as possible. There exist different interpretations of transparency. The paradigm of effective enforcement, for example, does not care about insecure executions and demands that secure executions should remain unchanged [1]. The paradigm of corrective enforcement, on the contrary, stipulates that any execution must undergo minimal changes no matter it is valid or not [2].

* Corresponding author at: Amirkabir University of Technology (Tehran Polytechnic), Iran.
E-mail addresses: imanimehr@aut.ac.ir (F. Imanimehr), msfallah@aut.ac.ir (M.S. Fallah).

https://doi.org/10.1016/j.cl.2018.07.003
1477-8424/

An information flow policy, which is usually formulated as a noninterference policy, generally states that the runs of a program must lead to the same behavior in the view of public observers whenever the inputs to the program only differ in what is not observable to public, i.e., in secret or private inputs. Thus, information flow policies are not properties of individual executions and demand specific relations among possible executions of a program. In fact, information flow policies are hyperproperties [3] and specify a family of secure programs, where a program is defined as a computable set of executions. As a result, a monitor can naturally be defined to be sound for an information flow policy if it produces a secure set of executions in response to the set of possible executions of every target program. Arriving at a promising interpretation of transparency for such policies, however, is a demanding challenge. Here, an individual execution of a program cannot be classified as a secure or an insecure execution, and consequently, transparency may be formulated as an appropriate relation on programs reflecting the fact that monitors must make as minimal changes to the set of possible executions of any given program as possible.

Some attempts have been made at generalizing the concept of transparent monitoring so that it can be applied to information flow policies. The effective enforcement of an information flow policy, for example, is defined as demanding that the executions of a secure program remain unchanged [4]. The corrective enforcement of some noninterference policies, e.g., the policies defined in [5], is also formulated as requiring the monitor to preserve all those executions of the target program that are compatible with all other executions of that program [6]—the authors of [7] refer to this formulation of transparency as true transparency. Here, an execution σ of a program is said to be compatible with all other executions of that program, or a compatible execution of the program for short, if the two-element set $\{\sigma, \sigma'\}$ satisfies the given information flow policy P for any execution σ' of the program. In this way, the monitor should arrive at some kind of ultimate transparency. We call such an enforcement paradigm corrective$_{\sqsubseteq_P^C}$ enforcement, where C is the abbreviation for Compatible and $\sqsubseteq_P^C$ is a relation on programs reflecting the fact that the monitor should preserve all compatible executions of the target. It is evident that every execution of a secure program is a compatible execution of that program. Thus, any policy P that is correctively$_{\sqsubseteq_P^C}$ enforceable can also be effectively enforced.

Another paradigm of corrective enforcement, which we introduce in this paper for the first time, is corrective$_{\sqsubseteq_P^S}$ enforcement—the superscript S stands for Supreme. This enforcement paradigm stipulates that the set of program executions preserved by the monitor must be maximal in the sense that adding any other original execution of the target to this set leads to an insecure set of executions. A monitor that correctively$_{\sqsubseteq_P^S}$ enforces a policy also correctively$_{\sqsubseteq_P^C}$ enforces that policy. In fact, such a monitor must preserve all compatible executions of the target. Otherwise, adding a missing compatible execution to the set of what preserved by the monitor does not lead to an insecure set of executions.

It has been shown that run-time monitors may be able to effectively enforce information flow policies only if the monitor is provided with the exact set of possible executions of the target [4]. Thus, without such an exact knowledge, monitors cannot correctively enforce information flow policies. It follows that to compare the capacity of different classes of monitors for enforcing noninterference policies, we should define partial orders (lattices) over the set of (classes of) monitors. The proposed partial orders indeed reflect the level of transparency a monitor can achieve. The monitors enforcing information flow policies in the paradigms of corrective$_{\sqsubseteq_P^C}$ and corrective$_{\sqsubseteq_P^S}$ enforcement should be maximals of the set of monitors ordered by the proposed partial orders.

The next problem is to ascertain how monitors can attain a specific level of transparency represented by a point on the proposed partial orders. Consider programs that output values on private and public channels. A purely dynamic monitor blocks an execution when the program is going to take certain actions. It may also be flow-sensitive and change the security classes of variables at run time. To be more transparent, a flow-sensitive purely dynamic monitor may only be allowed to block the commands that output secret values on public channels [8], in which case it may be called a purely flow-sensitive monitor. Such a monitor, however, cannot enforce information flow policies. To resolve this problem, a number of flow-sensitive purely dynamic monitors have been proposed that, in addition to output commands, are also able to block some other actions taken by the target [9–13]. As purely dynamic monitors have no information about other possible executions of the target, they must act conservatively and unavoidably block some compatible executions of target programs. They also preserve some incompatible executions of the target in the secure set of executions they produce. Thus, a plausible upper bound of transparency for purely dynamic monitors is the one defined by corrective$_{\sqsubseteq_P^S}$ enforcement. Consequently, to compare the transparency of purely dynamic monitors in enforcing a given policy P, we must give a partial order on classes of monitors whose maximal is the class of monitors that correctively$_{\sqsubseteq_P^S}$ enforce P.

Hybrid monitors tackle the root of the problem of purely flow-sensitive monitors. They indeed analyze the code of the target to upgrade certain variables that occur in untaken branches of the code [14]. By blocking the commands that output secret values on public channels, hybrid monitors achieve soundness. The analysis of untaken branches of the code enables hybrid monitors to preserve many of the compatible executions of the target that are blocked by purely dynamic monitors. Hybrid monitors, however, may block many of the incompatible executions preserved by purely dynamic monitors. Thus, the main possible advantage of hybrid monitors over purely dynamic monitors is in preserving a larger set of compatible executions of target programs. This motivates us to study the transparency of hybrid monitors through a partial order reflecting how a hybrid monitor is able to preserve the compatible executions of the target. The maximum of the set of classes of monitors ordered in this way is the class of monitors that correctively$_{\sqsubseteq_P^C}$ enforce a given policy P.