Cost-Aware Runtime Enforcement of Security Policies*

Peter Drábik, Fabio Martinelli, and Charles Morisset

IIT-CNR, Security Group
Via Giuseppe Moruzzi 1, 56124 Pisa, Italy
firstname.lastname@iit.cnr.it

Abstract. In runtime enforcement of security policies, the classic requirements on monitors in order to enforce a security policy are soundness and transparency. However, there are many monitors that successfully pass this specification but they differ in complexity of both their implementation and the output they produce. In order to distinguish and compare these monitors we propose to associate cost with enforcement.

We present a framework where the cost of enforcement of a trace is determined by the cost of operations the monitor uses to edit the trace. We explore cost-based order relations on sound monitors. We investigate cost-optimality of monitors which allows considering the most cost-efficient monitors that soundly enforce a property.

Keywords: runtime enforcement, cost, constructible monitors.

1 Introduction

An enforcement mechanism is a program in charge of controlling the actions of a target over a system, such that the sequences of actions submitted to the system satisfy a security policy. For instance, a security policy can state that the user of a database cannot execute a request to remove a table she does not own, or that an application downloaded onto a mobile operating system cannot modify the core functionality of the system. An enforcement mechanism can therefore be seen as a monitor between a target, seen as a black-box, and a system, such that only correct sequences of actions are executed by the system.

In the context of runtime enforcement, the classic requirements on monitors in order to enforce a security policy are soundness, i.e. producing only correct output, and transparency, i.e. acting invisibly on correct input [1]. However, there are many monitors that successfully pass this specification but differ in complexity of both their implementation and the output they produce for incorrect input [2]. In order to distinguish and compare these monitors we propose to associate cost with enforcement.

For a motivating example consider a museum which has a policy that no child should be inside without the presence of a guard. A monitor processing

* This research was supported by the EU FP7-ICT project NESSoS under the grant agreement n. 256980.

A. Jøsang, P. Samarati, and M. Petrocchi (Eds.): STM 2012, LNCS 7783, pp. 1–16, 2013.

the queue of visitors is requested a decision upon seeing a child in the case that no guard is inside the museum. Such an input trace is considered incorrect and the responsibility of the monitor is to turn it to a correct one. There are many ways to do it, for example to refuse the entrance to any further visitor, or a less drastic one to refuse the entrance to that child. The current theory of runtime enforcement does not offer any way to compare these two solutions and say which one is better. In this paper we argue that cost can be a convenient tool for doing so. For example, cost could be associated with the lost gain from the tickets not sold, which would favour the second solution.

In this work we present a framework for considering the editing cost of enforcement. First we adopt a high level view on enforcement, where we see runtime monitors as functions. In real life, we can construct only monitors that upon processing an action have to decide only based on the current action and the past trace. Following this intuition, we introduce a class of constructible monitors, which are monitors that at each step choose performing one of the specified operations on the input action: either to accept it, suppress it or to substitute or insert a sequence of actions. Moreover, such a monitor in each step outputs a non-empty sequence of symbols. We argue that this concept of monitor is close to the nature of runtime enforcement.

For a constructible monitor, the cost of enforcement of a trace is determined by the cost of operations the monitor uses to edit the trace. The cost enables to compare monitors and we introduce two cost-based order relations on sound constructible monitors, one based on pointwise comparison for each trace and another one on the expected cost for a set of traces. The minimality in this order relations allows us to reason about cost-optimal monitors. Moreover, we prove that (under some reasonable assumptions) each cost-optimal monitor is transparent, which justifies the classic concept of transparency [1].

We believe that our approach can be useful to a security designer in three ways. Firstly, it allows her to calculate the actual cost of enforcing a given policy with a given monitor. Secondly, it provides a way of comparing two correct monitors to choose the more cost-efficient one. Finally, the "optimal" monitor(s) can be defined, together with the minimal expected cost for enforcing a policy.

The article is structured as follows. In Section 2 we recast the topic of runtime enforcement in the framework of functions, after which in Section 3 we introduce the class of constructible monitors. For these we develop the notion of cost of enforcement in Section 4 and investigate cost-optimal monitors in Section 5. Finally we relate our work with the literature and discuss possible extensions of our work in Section 6.

2 Runtime Enforcement

As we said in the introduction, a monitor is responsible for enforcing a policy over a system. We consider the three following entities: the *target* is an active entity sending actions to the system; the *system* is a passive entity waiting for actions from the target, and executing them; the *monitor* is an entity between

the target and the system, receiving actions from the target and sending actions to the system. We now introduce the notion of traces, together with some usual notations, and then we express the notion of policy and monitor.

Actions, traces. The monitor observes the *actions* of the target, we denote $\mathcal{A}$ the set of security-relevant ones. Let $\mathcal{A}^*$ denote the set of all finite sequences over $\mathcal{A}$. A *trace* is a finite sequence of actions, that is the set of traces $\mathcal{T} \subseteq \mathcal{A}^*$. For the purposes of runtime monitoring, which acts on all prefixes of the traces we assume that $\mathcal{T}$ is prefix-closed. In this work we consider only security policies over finite traces. This is a choice made by many approaches in the field of enforcement, e.g. [1,3,4,2]. The treatment of infinite traces is left for future work. By σ we refer to a trace and by ϵ we refer to the empty trace. We write $\sigma;\tau$ to denote the concatenation of two traces. By $\sigma^{<k}$ we denote the prefix of σ constituted of the first $k-1$ actions, while σ^k means the k-th action of the trace, where the initial action is σ^0. Lastly, by $\preceq$ we denote the relation of being a prefix of a trace.

Properties. Following most existing approaches in the context of run-time enforcement, we consider security policies defined in terms of individual executions of the program. A *security property* is a computable predicate $P \subseteq \mathcal{T}$ and a trace σ that satisfies the property P is called *correct* (denoted as $P(\sigma)$), and a trace that does not satisfy the property is called *incorrect* (denoted as $\neg P(\sigma)$). In this paper we focus on reasonable properties, those which always hold for the empty trace: $P(\epsilon)$. Note that this is a classic restriction made in the field of runtime enforcement [1]. Among reasonable properties, the category of *safety properties* has been particularly studied. Intuitively, with a safety property, an incorrect trace cannot be extended into a correct one. Formally a reasonable property $P \subseteq \mathcal{T}$ is a safety property iff $\forall \sigma_1, \sigma_2 \in \mathcal{T}\ \neg P(\sigma_1) \wedge \sigma_1 \preceq \sigma_2 \Rightarrow \neg P(\sigma_2)$ [5,6].

Monitors. We choose in this work to model the runtime *monitor* as a function on sequences of actions. While this is a view directly adopted by later works on runtime monitoring, such as by Bielova and Massacci [2], it is present indirectly in the original approach by Ligatti et al. [1], since the operational semantics of the security automaton takes the full trace as an input and, after possibly many multi-steps transitions, outputs another trace. So, a monitor processes the input trace action by action and produces the output trace: $M : \mathcal{T} \to \mathcal{T}$.

Enforcement – soundness and transparency. We recall the classic notions of soundness and transparency of enforcement. Soundness implies that all outputs of the monitor must obey the security property. Transparency requires that monitors operate invisibly on executions that satisfy the property already.

Definition 1. *A monitor* $M : \mathcal{T} \to \mathcal{T}$ soundly *enforces property* P *iff* $\forall \sigma \in \mathcal{T} : P(M(\sigma))$. M transparently *enforces* P *iff* $\forall \sigma \in \mathcal{T} : (P(\sigma) \Rightarrow M(\sigma) = \sigma)$.

If the property P is understood from the context, we might omit it and say that monitor M is sound and transparent, respectively.

Table 1. Monitors

mon.	description	sound	transp.
M_0	insert no guard, suppress all	✓	×
M_1	insert no guard, suppress all children	✓	×
M_2	insert no guard, suppress children when no guard is in	✓	✓
M_3	insert a guard before the whole trace	✓	×
M_4	insert a guard upon seeing the first child when no guard is in	✓	✓
M_5	insert a guard upon seeing the first child	✓	×
M_6	insert a guard before each child	✓	×
M_7	insert no guard, let everybody in	×	✓

Running Example. *The security policy in a museum distinguishes between two types of visitors: adults and children. While adults can enter on their own, children can be inside only in a presence of a guard. In this setting, the (abstract) target "produces" sequences of visitors and guards as a trace of actions from $\mathcal{A}$, where the possible action can be the entering of an adult* $\mathtt{a}$*, a child* $\mathtt{c}$ *or a guard* $\mathtt{g}$*.*

In particular, we consider the set of traces $\mathcal{T} \subseteq \mathcal{A}^*$ which consists of all traces of finite length over $\mathcal{A}$. For instance the trace $\mathtt{acc}$ *means that the first to enter is an adult and then two children and* $\mathtt{agcc}$ *says that the children are preceded by a guard.*

The security policy is expressed as a property P_M on traces, where $P_M(\sigma) \Leftrightarrow (\sigma^k = \mathtt{c} \Rightarrow \exists i < k \text{ s.t. } \sigma^i = \mathtt{g})$. We can see that $\neg P_M(\mathtt{acc})$ but $P_M(\mathtt{agcc})$. Note that P_M is a safety property, as an incorrect trace cannot be extended to a correct one.

In Table 1, we list some approaches to enforcement and monitors that implement them. For example consider the monitor M_2, that implements the enforcement mechanism which lets enter all adults, but lets in children only when there are guards inside. We have that $M_2(\mathtt{acc}) = \mathtt{a}$ but $\mathtt{agcc}$ *is left unaltered by M_2. It is easy to see that M_2 soundly and transparently enforces P_M. Another approach could be not to let in anyone, i.e. a monitor M_0 such that $M_0(\sigma) = \epsilon$ for all $\sigma \in \mathcal{T}$, which is sound for P_M but clearly not transparent.*

3 Constructible Monitors

Traditionally, a monitor can be any function from traces to traces. However, as we can see in the example, in real life we can construct only monitors that upon processing an action have to decide only based on the current action and the past trace. Following this intuition, in this section we define a class of monitors that we consider constructible. We believe that such a view accurately reflects the nature of runtime monitoring.

3.1 General Definition

Intuitively, a constructible monitor is a monitor which for each input action, based on the seen past trace, takes a decision and outputs a non-empty sequence of actions.

That is, since a monitor is a function from traces to traces, the output of the monitor needs to be defined for each prefix of a trace. Moreover, with each subsequent action, the output of the monitor is strictly incremental.

Definition 2. *A monitor $M : \mathcal{T} \to \mathcal{T}$ is considered to be* constructible*, if for each $\sigma \in \mathcal{T}$ and for all $0 \leqslant i < |\sigma|$ we have $M(\sigma^{<i}) \prec M(\sigma^{<i+1})$.*

We can see that the definition corresponds to the intuitive specification above. Indeed, for each seen action the output of the monitor is only based on that action and the preceding ones. At the time of the decision the monitor is in possession of no information about the future of the trace. Moreover, the requirement of a non-empty output at all times excludes postponing the decision by outputting an empty trace.

Class of constructible monitors. We define the class of constructible monitors $\mathsf{CM} = \{M : \mathcal{T} \to \mathcal{T} \mid M \text{ is constructible}\}$.

We observe, that not all monitors are constructible. We illustrate this fact on our running example. Consider a monitor M defined so that if there is at most one child in the whole trace it must be suppressed, and in all other cases a guard is inserted at the beginning of the trace. Therefore, we have for example $M(\texttt{ca}) = \texttt{\_a}$ and $M(\texttt{cc}) = \texttt{gcc}$ where the action _ signalises an action suppression. We can see that monitor M does not satisfy the definition of constructibility, because $M(\texttt{c})$ cannot be a non-empty prefix of both _a and gcc since there is none. Intuitively, the reason for M not being constructible because at seeing the first child, the monitor needs to decide whether to suppress it or send a guard before, but at the point of the decision there is no way of knowing if more children will come.

Dealing with suppression. As we can see, the output at each step of the enforcement can be arbitrary, but not empty. Therefore, suppressing an action in the sense of removing it completely from the trace is not possible by a constructible monitor. The reason is that it is necessary to signalise the suppression to the system. We assume that a system understands such a signal and even requires it. This function can be for example performed by a special wait action (_), which in the set of actions. Otherwise _ is considered a normal action, which means that there is no restriction on its use or its presence in the output of the target.

In the example above, the interaction of the target (the queue of visitors) and the monitor is synchronous. Therefore the suppression of a child occupies a visitor's turn, and that is why we must signalise such an event in the output trace by a _ action. It corresponds to the fact that in that particular turn no one will enter the museum. The system (the museum) must be able to handle the wait action. Similarly the target could produce such an action to model a situation where at some turn there is no one at the entrance. Such a scenario is a valid one and the monitor should be able to deal with it. Such a case is represented by the _ in the input trace.

Power of constructible monitors. Under the assumptions of soundness and transparency, the power of constructible monitors is equal to the one of Schneider's security automata [7] and to precise enforcement of edit automata of Ligatti et al. [1].

Lemma 1. *Given any property P, there exists a monitor M in* CM *such that M enforces soundly and transparently P iff P is a safety property.*

The proof of this lemma is analogous to the case of precise enforcement in edit automata in Ligatti et al. [1].

3.2 Specifying Constructible Monitors

In each step of runtime monitoring, a constructible monitor produces an increment of the output trace.

A convenient way to build constructible monitors is by specifying how to obtain this piece of output based on the past trace and the current action.

We abstract the decision process into a *selector* function $f : \mathcal{T} \times \mathcal{A} \to Ops$, where Ops is a set of atomic operations. Given a trace σ and an action a, $f(\sigma, a)$ stands for the decision taken for a considering that the previous trace is σ[1].

In a second step, we define the semantics of each atomic operation over traces. This decomposition allows us to make explicit the power of editing of the monitor.

For instance, if we were to consider the set $Ops = \{acc\}$, where acc stands for the acceptance of an action, then any monitor would be bound to accept every trace, since no other operation is available. In the following, we consider the $Ops = \{acc, sub(\tau), sup, ins(\tau)\}$, where $sub(\tau)$ stands for the substitution of an action by a non-empty sequence $\tau \in \mathcal{A}^+$, sup for the suppression of an action and $ins(\tau)$ for the insertion of a non-empty sequence $\tau \in \mathcal{A}^+$.

The semantics of the atomic operations is given with the function $opsem : Ops \times \mathcal{A} \to \mathcal{A}^+$. In the following, we consider the semantics given by:

$$opsem(op, a) = \begin{cases} a & \text{if } op = acc \\ \tau & \text{if } op = sub(\tau) \\ \_ & \text{if } op = sup \\ \tau; a & \text{if } op = ins(\tau) \end{cases}$$

Note that the semantics function $opsem$ respects the requirement from the definition of constructible monitors and always returns a non-empty trace increment.

Finally, given a selector f, we define the monitor $M_f : \mathcal{T} \to \mathcal{T}$ as the concatenation of $opsem(f(\sigma^{<i}, \sigma^i), \sigma^i)$, for any $0 \leqslant i < |\sigma|$.

The class of monitors that are constructible according to Definition 2 and the class of monitors definable through selectors coincide. It can be easily checked

[1] To some extent, a selector can be seen as an analogue to the functions δ, γ and ω used for edit-automata [1] with the exception that the action cannot be suppressed "silently".

that each monitor with a selector is constructible. For the other direction, note that for any constructible monitor $M : \mathcal{T} \to \mathcal{T}$, there exists at least one selector f such that $M_f(\sigma) = M(\sigma)$ for all $\sigma \in \mathcal{T}$. We denote the following selector for M as f_M.

Let us denote for a $\tau \in \mathcal{A}^*$ and $a \in \mathcal{A}$ as $\mathit{tracedif}(\tau, a)$ the difference between what M outputs for traces τ and $\tau; a$, i.e. $M(\tau); \mathit{tracedif}(\tau, a) = M(\tau; a)$. Then

$$f_M(\tau, a) = \begin{cases} \mathit{acc} & \text{if } \mathit{tracedif}(\tau, a) = a \text{ and } a \neq \_ \\ \mathit{sup} & \text{if } \mathit{tracedif}(\tau, a) = \_ \\ \mathit{ins}(\tau') & \text{if } \mathit{tracedif}(\tau, a) = \tau'; a \text{ for some } \tau' \in \mathcal{A}^+ \\ \mathit{sub}(\mathit{tracedif}(\tau, a)) & \text{otherwise} \end{cases}$$

Note that f_M is only one of the selectors f for M such that $M_f(\sigma) = M(\sigma)$ for all $\sigma \in \mathcal{T}$ and there exist others, for example $f'_M = \mathit{sub}(\mathit{tracedif}(\tau, a))$. Even if the semantics of these two monitors is the same, we will show in the next section that their cost of enforcement might differ. More importantly, the fact that any constructible monitor can be defined through a selector function enables us to concentrate in the rest of the paper only on constructible monitors with a given selector.

Running Example. *Actually, all the monitors M_0 through M_7 described in the previous section are constructible, provided that in the case of suppression they produce the _ action included in $\mathcal{A}$. This also illustrates that when one wants to specify a runtime-monitor, intuitively always reasons in terms of constructible monitors.*

For example monitor M_2 can be specified by using the selector f_2 as follows:

$$f_2(\tau, a) = \begin{cases} \mathit{sup} & \mathit{if}\ a = \mathtt{c}\ \mathit{and}\ \mathtt{g} \notin \tau \\ \mathit{acc} & \mathit{otherwise} \end{cases}$$

4 Cost of Enforcement

In the previous section, we have introduced a simple and intuitive way to build a monitor, using a selector. A monitor can therefore be seen as a function that to a trace output by the target associates a sequence of atomic operations. In this section we introduce the idea that enforcing a property, and in particular modifying the trace input by the target, should come with a cost. In other words, we want to make explicit the cost for the monitor to transform the input trace into the output trace.

Hence, we first introduce a basic cost-model, based on cost of atomic operations, which we extend to the cost of enforcing a single trace, and a set of traces. The cost enables to compare monitors and we introduce a cost-based order relation on sound monitors for a property.

4.1 Cost

Cost domain. The cost of transformation is expressed by a value – the smaller the value, the better. We consider values from the cost domain $(\mathbb{C}, \leqslant, \bot, \top, +)$ such that $\leqslant$ is a well-founded total order, $\bot$ is the minimal element of $\leqslant$, $\top$ the maximal element and $+$ is a cost aggregation function, such that that $c \leqslant c + c'$ and $c' \leqslant c + c'$. Moreover, $\bot$ and $\top$ are the neutral and absorbing elements for $+$, respectively. In general, we use $\bot$ to denote the absence of cost, and $\top$ to denote an unreachable cost. For instance, a possible cost domain is $(\mathbb{R}^+, \leqslant, 0, \infty, +)$.

Operation costs. In the same way that we define the semantics of each atomic operation in *Ops* for each action in $\mathcal{A}$ with the function *opsem* in Section 3, we associate cost with each operation and each action with the function *opcost* : $Ops \times \mathcal{A} \to \mathbb{C}$.

Running Example. *We want to express the fact that denying the entrance to the museum to an adult or to a guard costs 4, to a child costs 3 and inserting a guard costs 5. Given an atomic operation op and an action a, we therefore define the function opcost as:*

$$opcost(op, a) = \begin{cases} 0 & \text{if } op = acc \text{ or } (op = sup \text{ and } a = \_) \\ 4 & \text{if } op = sup \text{ and } a \in \{\mathsf{a}, \mathsf{g}\} \\ 3 & \text{if } op = sup \text{ and } a = \mathsf{c} \\ 5 & \text{if } op = ins(\mathsf{g}) \\ \infty & \text{otherwise} \end{cases}$$

To some extent, associating an atomic operation with an infinite cost could be used to denote that some operations are not possible. For instance, the idea of observable actions introduced in [8], on which a monitor cannot stop, can be modelled as associating an infinite cost with their suppression. We leave the characterisation of enforceable policies based on the cost of their enforcement for future work.

Cost of editing a trace. Given a monitor built using a selector, the total cost of enforcement of this monitor for a given trace can be calculated as the sum of the cost of each atomic operation.

Definition 3. *Given a selector f, a monitor M_f built using this selector and a trace σ in $\mathcal{T}$, we define the* cost function $\mathsf{cost}_{M_f} : \mathcal{T} \to \mathbb{C}$ *as:*

$$\mathsf{cost}_{M_f}(\sigma) = \sum_{0 \leqslant i < |\sigma|} opcost(f(\sigma^{<i}, \sigma^i), \sigma^i)$$

We remark that different sequences of operations might cost differently even if they produce the same output. For example for general actions a and b and c, the transformation from ac to abc can be done either by accepting a and inserting b upon seeing c, or by substituting a by ab and accepting c. In general these two

Table 2. Enforcement

σ	M_0		M_1		M_2		M_3		M_4		M_5		M_6		M_7	
ϵ	ϵ	0	ϵ	0	ϵ	0	ϵ	0	ϵ	0	ϵ	0	ϵ	0	ϵ	0
_a	__	4	_a	0	_a	0	g_a	5	_a	0	_a	0	_a	0	_a	0
aa	__	8	aa	0	aa	0	gaa	5	aa	0	aa	0	aa	0	aa	0
cc	__	6	__	6	__	6	gcc	5	gcc	5	gcc	5	gcgc	10	cc	0
ca	__	7	_a	3	_a	3	gca	5	gca	5	gca	5	gca	5	ca	0
ga	__	8	ga	0	ga	0	gga	5	ga	0	ga	0	ga	0	ga	0
gc	__	7	g_	3	gc	0	ggc	5	gc	0	ggc	5	ggc	5	gc	0
aaaa	____	16	aaaa	0	aaaa	0	gaaaa	5	aaaa	0	aaaa	0	aaaa	0	aaaa	0
cccc	____	12	____	12	____	12	gcccc	5	gcccc	5	gcccc	5	gcgcgcgc	20	cccc	0
caaa	____	15	_aaa	3	_aaa	3	gcaaa	5	gcaaa	5	gcaaa	5	gcaaa	5	caaa	0
gaaa	____	16	gaaa	0	gaaa	0	ggaaa	5	gaaa	0	gaaa	0	gaaa	0	gaaa	0
gccc	____	13	g___	9	gccc	0	ggccc	5	gccc	0	ggccc	5	ggcgcgc	15	gccc	0

sequences can have different cost. Therefore, what is important for assigning the cost is the intent of the modeller when creating the selector function for the monitor.

Running Example. *In Table 2 we can see the result of enforcement of monitors M_0 through M_7 on several traces, together with the cost.*

Expected cost of editing a set of traces. Sometimes a particular subset of the set of all traces produced by the target is of interest. For such a set, the expected cost of enforcement can be defined. We define a function $expcost_M : \mathcal{P}(\mathcal{T}) \to \mathbb{C}$, that for a set of traces T computes the *expected cost* of editing of traces in T.

Definition 4. *Let T be a set of traces, f be a selector function and $opcost : Ops \to \mathbb{C}$ a function assigning cost to operations. The function $expcost_{M_f} : \mathcal{P}(\mathcal{T}) \to \mathbb{C}$ is defined as follows*

$$expcost_{M_f}(T) = \frac{\sum_{\sigma \in T} \mathsf{cost}_{M_f}(\sigma)}{|T|}.$$

Running Example. *Suppose that the maximum number of visitors that can come in one day is 30, that is $T = \mathcal{T}^{30}$. Then $expcost_{M_2}(T) = 2.99$.*

Intuitively, if T were associated with a probability distribution, we could extend $expcost_M$ be weighting the cost of each trace by its probability. We leave this extension for future work and we consider that all traces are equiprobable.

4.2 Cost-Based Comparison of Monitors

The interest of comparing monitors is to provide the security designer with a way of choosing a more efficient monitor which guarantees the correct functionality. That is why in this section we concentrate only on monitors that produce correct output at all times, i.e. on sound monitors. Note that transparency is intrinsically related to cost, so we drop it at this point of the paper to establish its relation to cost in the following section.

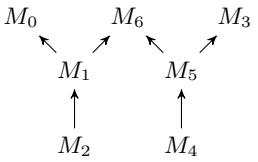

Fig. 1. Pointwise order between monitors

Pointwise comparison. We start with a natural way of comparing monitors – the pointwise comparison. Clearly, if for each trace a monitor spends less for enforcement than another one, it can be considered more cost-efficient.

Definition 5. *Let M and M' be monitors from* CM*. We define the relation* more cost-efficient *for monitors as follows:*

$$M \sqsubseteq M' \text{ iff } \forall \sigma \in \mathcal{T} : \mathsf{cost}_M(\sigma) \leqslant \mathsf{cost}_{M'}(\sigma).$$

Running Example. *The monitor M_2, which lets enter all adults but lets in children only when there are guards inside is assigned cost as follows: $\mathsf{cost}_{M_2}(\sigma) = opcost(sup, \mathsf{c}) \cdot n_k$, where n_k is the number of* c*s not preceded by any* g *in σ.*

As for monitor M_0, which prevents anyone from entering the museum, the cost is $\mathsf{cost}_{M_0}(\sigma) = opcost(sup, \mathsf{c}) \cdot \#_{\mathsf{c}}(\sigma) + opcost(sup, \mathsf{a}) \cdot \#_{\mathsf{a}}(\sigma) + opcost(sup, \mathsf{g}) \cdot \#_{\mathsf{g}}(\sigma)$.

Intuitively, monitor M_2 should be more cost-efficient than the monitor M_0. In fact, none of these two monitors introduces any guards, and each child that is prevented from entering by M_2 is prevented also by M_0. Since this is true for each trace in $\mathcal{T}$, we have that $M_2 \sqsubseteq M_0$. The order between all sound monitors M_0 through M_6 according to $\sqsubseteq$ can be seen on Figure 1. For instance, we can see that in order to soundly enforce P_M there is no point in choosing M_0 since it is less cost-efficient than M_2 on each trace. Note that both M_2 and M_4 are more cost-efficient than the others, but they are incomparable.

Comparison by expected cost. In practice, it is quite difficult to obtain a monitor that is pointwise more cost-efficient than another monitor. The reason is, that frequently one monitor can be more cost-efficient on a set of traces, but less cost-efficient on another. In such a case the two monitors are incomparable according to $\sqsubseteq$. That is why we introduce another order relation, a comparison based on the expected cost which can serve as a distinguishing criterion.

Definition 6. *We define the relation* globally more cost-efficient *for monitors M and M' on the set of traces T as follows: $M \trianglelefteq^T M'$ iff $expcost_M(T) \leqslant expcost_{M'}(T)$.*

If we take as T the set $\mathcal{T}^n$ of all fixed-length traces of length n from $\mathcal{T}$, we write $\trianglelefteq^{\mathcal{T}^n}$ as $\trianglelefteq^n$.

It is easy to see that the order relation $\trianglelefteq^T$ is total, that is each two constructible monitors are comparable.

Running Example. *We are interested in comparing monitors M_2 and M_4, which are incomparable according to $\sqsubseteq$.*

It is easy to see[2] *, that for short traces, i.e. small n monitor M_2, is more efficient. For long traces instead, M_4 becomes more convenient. This is due to the fact that when the trace to come is long enough, then inserting a guard is more efficient, because on average enough children will arrive to justify the cost for the guard. If the trace is not long enough, it is convenient to suppress all children.*

It is possible to derive the n^* where this shift occurs. Note, that it is possible to combine these two strategies and obtain a monitor that subsumes both these monitors. More details on this matter, along with a table containing actual expected costs (Table 3), are included in the following section.

Relationship between pointwise and global order relations. It is clear that the pointwise order between two monitors implies the global order between them. That is if $M \sqsubseteq M'$ then $M \trianglelefteq^T M'$ for any T. The reason is that if the cost is smaller for each trace in a set, then for sure also the expected cost, which is the average of the cost on that set, is better. In general, the converse implication does not hold, in particular there can be two monitors, and one monitor is better on one trace in T, the other monitor is better on another trace in T, but the first maintains a better expected (average) cost on T.

5 Cost-Optimal Constructible Monitors

There are many sound and transparent constructible monitors for any safety property. The security designer might want to choose the one that minimises the cost of enforcement, that is in a way the best out of the monitors that perform their task well. In this section we consider sound cost-optimal monitors. Transparency is not required, in fact it is derived as cost-optimality in one of the special cases.

We associate the notion of cost-optimality with the minimality in the cost-based order relation. We call cost-optimal monitors the elements for which no element is smaller.

Definition 7. *We say that monitor M is* $\sqsubseteq$-optimal *(*$\trianglelefteq^T$-optimal*) iff there is no $M' \in \mathsf{CM}$ such that $M' \sqsubset M$ ($M' \triangleleft^T M$).*

5.1 Cost-Optimality for $\sqsubseteq$

Consider, for a property P, the set of monitors that soundly enforce P. Now since $\sqsubseteq$ is a preorder on CM, on this set there exist minimal elements. That

[2] Formulas for computing the expected cost of monitors M_2 and M_4 can be found in the technical report [9].

means that for each property P there always exists at least one cost-optimal sound constructible monitor.

We call a monitor *absolutely* $\sqsubseteq$-optimal, if it the smallest element in CM w.r.t. $\sqsubseteq$, i.e. it is more cost-efficient than any other constructible monitor. It is worth noting, that in general there is no absolutely $\sqsubseteq$-optimal monitor, since there can be more than one incomparable minimal elements.

Running Example. *As can be seen on Figure 1, both monitors M_2 and M_4 are more cost-efficient that the rest of monitors in our example. It can be shown that for a general set of traces $\mathcal{T}$, there is no monitor that is strictly better than M_2 or M_4. Therefore both of them are $\sqsubseteq$-optimal. However, M_2 and M_4 are incomparable as we showed in the previous section. It follows that there is no absolutely cost-optimal monitor to enforce the museum security policy.*

Transparency as cost-optimality. In the beginning of this section we decided to relax the classic requirements of enforcement by dropping transparency. Now we will motivate this choice by showing, that under assumptions very reasonable in practice, transparency is implied by $\sqsubseteq$-optimality. In other words, it is enough to concentrate on cost-optimal monitors in order to obtain transparent ones. In particular, let us assume that *acc* is the single cheapest operation. Then for safety properties by assuming soundness, cost-optimality implies transparency. Note that we only consider safety properties P, as constructible monitors only can soundly and transparently enforce these properties.

Lemma 2. *Let P be a safety property. Let opcost be such that $opcost(acc, x)$ is strictly less than $opcost(op, x)$ for all other operations op. If M_f is sound for P, then if it is $\sqsubseteq$-optimal then it is transparent for P.*[3]

We remark, that the converse implication does not hold, i.e. transparency does not imply cost-optimality. The reason is that transparency does not talk about incorrect traces. It can be the case that even a sound and transparent monitor can be subsumed by another sound and transparent monitor with lower cost on incorrect traces.

Running Example. *We can see, that monitors M_2 and M_4 are transparent. On the other hand, all other monitors modify the trace* $\mathtt{gc}$*, which is correct, and therefore are not transparent.*
It is worth noting, that if suppression was the cheapest operation, a trivial monitor that suppresses everything is the absolutely $\sqsubseteq$-optimal one for any property P (even beyond safety). This suggests that having suppression as the cheapest operation is a degenerate case since it favours security by blocking in the sense "what is shut off is secure".

5.2 Cost-Optimality for $\trianglelefteq^{\boldsymbol{T}}$

As we have seen, the pointwise order does not allow for determining the cost-optimal monitor that is more cost-efficient than all others. However, the criterion

[3] The proof can be found in the technical report [9].

of expected cost can be of help in establishing cost-based relationship between pointwise incomparable monitors. Now we show how it can be useful for finding the best monitor.

Since $\trianglelefteq^T$ is a total order on CM, there is always a minimal element, which corresponds to the smallest element, i.e., the $\trianglelefteq^T$-optimality coincides with the absolute $\trianglelefteq^T$-optimality. Note that each $\trianglelefteq^T$-optimal monitor must be also $\sqsubseteq$-optimal which can be readily seen by reasoning by contradiction.

A natural question is how to find or construct a cost-optimal monitor for a set of traces $\mathcal{T}^n$. In the general case of an arbitrary cost setting, the cost-optimal monitor exists, since the cost-induced order is a preorder on CM, but it might not be trivial to find it. A very non-efficient way of finding $\trianglelefteq^n$-optimal monitor would be to enumerate all the monitors (there is a finite number of them for a $\mathcal{T}^n$) and choose the one with the lowest expected cost. We leave the investigation of more efficient ways to find such monitors for future work. Nevertheless, in some cases the $\trianglelefteq^n$-optimal monitors can be specified directly, as is in the case of our running example.

Running Example. *Note that out of M_0 through M_7 the only candidates for $\trianglelefteq^n$-optimality are M_2 and M_4. Following the reasoning from the previous section, there are lengths n for which M_2 is more cost-efficient than M_4 on $\mathcal{T}^n$ and others on which it is the other way round. Actually, none of these two monitors is $\trianglelefteq^n$-optimal. A cost-optimal monitor combines the strategies of these two monitors.*

Let us define a monitor M^* such that knowing that traces have a fixed length n, upon seeing the first child it inserts a g unless it is no longer convenient for the expected cost because on average not enough children can enter any more. Recall that n^* is the borderline length between "short" and "long" traces from the previous section. We can compute n^* as the smallest n that satisfies $expcost_{M_2}(\mathcal{T}^n) > expcost_{M_4}(\mathcal{T}^n)$. Thus if there remains a trace of $m \geqslant n^*$ to be seen (including the current action), it is still better to insert a guard, but if $m < n^*$, then instead it is better to suppress all children. In our case $n^* = 6$.

So the monitor M^* is defined through the selector function f^* as follows

$$f^*(\tau, x) = \begin{cases} ins(\mathsf{g}) & \text{if } x = \mathsf{c} \text{ and } \mathsf{g} \notin \tau \text{ and } n - |\tau| \geqslant 6 \\ sup & \text{if } x = \mathsf{c} \text{ and } \mathsf{g} \notin \tau \text{ and } n - |\tau| < 6 \\ acc & \text{otherwise} \end{cases}$$

We show that M^* has the optimal expected cost on traces of length n, i.e. $\forall M \in \mathsf{CM} : expcost_{M^*}(\mathcal{T}^n) \leqslant expcost_M(\mathcal{T}^n)$. Proof: consider a trace σ of length n. On each prefix of σ that does not contain c, it can be easily seen that an optimal monitor M' should accept all actions. When a prefix of σ ends with a c, an optimal monitor should accept if a g is already in the prefix. Any other action would lead to a bigger expected cost. If there is no g in the prefix, then the optimal monitor should, based on the average number of cs in all the possible extensions of length n of the current prefix, insert g iff it costs less than the expected cost of suppression of all the cs. This is exactly the way in which M^* works, and thus it is cost-optimal with respect to $\trianglelefteq^n$.

Table 3. Expected costs of M_2, M_4 and M^*

T	$\mathcal{T}^1$	$\mathcal{T}^2$	$\mathcal{T}^3$	$\mathcal{T}^4$	$\mathcal{T}^5$	$\mathcal{T}^6$	$\mathcal{T}^7$	$\mathcal{T}^8$
$expcost_{M_2}(T)$	0.75	1.31	1.73	2.05	2.29	2.47	2.60	2.70
$expcost_{M_4}(T)$	1.25	1.88	2.19	2.34	2.42	2.46	2.48	2.49
$expcost_{M^*}(T)$	0.75	1.31	1.73	2.05	2.29	2.39	2.45	2.47

The expected cost of monitor M^* can be seen in Table 3. On traces shorter that 6, it has expected cost exactly as M_2, and it is less than that of M_4. Starting from traces of length 6, M^* outperforms both M_2 and M_4. The difference between the three monitors is illustrated on several traces from $\mathcal{T}^8$ as follows: if a child comes at the first turn, i.e. $\sigma =$ `caaaaaaa`*, then $M_2(\sigma) =$* `_aaaaaaa`*, $M_4(\sigma) =$* `gcaaaaaaa` *and M^* acts as M_4 because it is convenient to put a guard since on average enough children will come to pay off for the cost of the guard. If the first child comes at the last turn instead, , i.e. $\sigma =$* `aaaaaaac`*, it is no longer convenient to send in a guard and M^* acts as M_2 on σ thus $M^*(\sigma) =$* `aaaaaaa_`*.*

It is also worth noting that since the trace length n is a part of the definition of M^*, a monitor for $n = n'$ and $n = n' + 1$ behave differently. In particular, if a monitor M^* is $\trianglelefteq^{n'}$-optimal, it will not necessarily be $\trianglelefteq^{n'+1}$-optimal. We also remark, that since the constructibility of M^* depends vitally on the prior knowledge of the trace length by the monitor, if the security expert defining the monitor is not in possession of the information about the length of traces on which the monitor will be applied, such a monitor cannot be constructed. This issue is worth further investigation and is left as future work.

6 Conclusions

Discussion and related work. Since Schneider's seminal work [7], runtime enforcement of policy using security automata has been a well-studied subject in the literature, such as [1,10,3,11]. We do not detail here all these approaches, and we refer to [12] for an extensive survey. However, to the best of our knowledge, our approach is the first one to deal with the problem of cost of enforcement.

Recent research has argued that the original definition of effective enforcement [1] is inadequate because it does not sufficiently constrain the behaviour of the monitor when it is faced with a possible violation of the security policy [12]. Researchers have revisited the notion of enforcement by a monitor have proposed alternative ones.

Bielova and Massacci [2] propose to apply a distance metrics known from string analysis in order to capture the similarity between traces. With help of this distance metrics they propose a new requirement on the enforcement mechanism, called predictability, which is based on the following principle "for each illegal trace – if it is close enough to a legal one, it should be projected close enough to that trace". They leave the problem of characterising the class of properties that can be enforced by a predictable enforcement mechanism as open. Our proposal,

using cost-optimal monitors, restated using similar vocabulary would be "for each illegal trace – (at all times) it should be projected onto the closest legal trace" where the distance is given by the cost of operations used for editing the trace. We prove, that for each safety property there is a cost-optimal monitor that soundly enforces this property.

Another attempt at a more restrictive concept of enforcement is the corrective enforcement of Khoury et al. [13]. Their approach is to group together related sequences into equivalence classes, and then limit the monitor so that it can only return an output sequence equivalent to the input. Moreover they use a partial order on sequences to obtain better results than with equivalence. The drawback of their approach is that restrictions on the treatment of incorrect sequences must be defined explicitly by the user for each property.

Edit automata [1] have also been identified as being too powerful, because they can use buffering and make the enforcement decisions with information about entire trace at its disposal. Bielova and Massacci [4] investigate subclasses of automata as delayed automata, all-or-nothing automata and Ligatti automata. We believe that our paradigm of constructive monitors is close to the practical nature of runtime enforcement, as we oblige the monitor to decide at each step only based on the information it can obtain from the current action and the past trace. To certain point this is similar to the concept of precise enforcement of Ligatti et al. [1] and therefore it comes as no surprise that this class of monitors enforces soundly and transparently only safety properties.

Our model of constructible monitor bears similarities to the MRA model of Ligatti and Reddy [3], which obliges the monitor to returns a result to the target application before seeing the next action it wishes to execute. Our model, instead, requires that the monitor outputs a non-empty trace before seeing the next action, but does not provide any feedback to the target.

Future work. It would clearly be interesting to leverage the fact that our cost model is parameterized to consider more practical operations inspired by real scenarios, for instance a complex treatment of buffering in our cost framework, with positive/negative costs. In addition, a general approach for constructing the optimal monitor for a cost model would be worth investigating, for instance by using recent work linking decision theory and access control [14].

Furthermore, our framework paves the way for investigations about quantitative aspects of runtime enforcement [15]. The present is only a first attempt to consider cost in the context of monitoring, where all traces are seen as equiprobable. A straightforward extension could be done when in possession of probabilistic information about traces, which would represent a more accurate notion of expected cost. Other dimensions to be treated in a quantitative manner include considering impact/benefit of traces and penalty for not respecting the policy.

Conclusion. In this paper we have presented a framework where the cost of enforcement is determined by the cost of operations used by monitor. The classic requirements on enforcement of security policies, soundness and transparency, leave a margin for specifying the behaviour of the monitor on incorrect inputs.

We argue that the cost of enforcement can serve as a distinguishing criterion for sound monitors for safety policies. We investigate cost-optimality of monitors and show that under reasonable assumptions it justifies the concept of transparency. We demonstrate the approach on a case study of a museum security policy and show how to find the cost-optimal monitor to enforce the policy.

References

1. Ligatti, J., Bauer, L., Walker, D.: Edit automata: Enforcement mechanisms for run-time security policies. International Journal of Information Security 4(1-2), 2–16 (2005)
2. Bielova, N., Massacci, F.: Predictability of enforcement. In: Erlingsson, Ú., Wieringa, R., Zannone, N. (eds.) ESSoS 2011. LNCS, vol. 6542, pp. 73–86. Springer, Heidelberg (2011)
3. Ligatti, J., Reddy, S.: A theory of runtime enforcement, with results. In: Gritzalis, D., Preneel, B., Theoharidou, M. (eds.) ESORICS 2010. LNCS, vol. 6345, pp. 87–100. Springer, Heidelberg (2010)
4. Bielova, N., Massacci, F.: Do you really mean what you actually enforced? IJIS, 1–16 (2011)
5. Alpern, B., Schneider, F.B.: Recognizing safety and liveness. Distributed Computing 2(3), 117–126 (1987)
6. Lamport, L.: Proving the correctness of multiprocess programs. IEEE Trans. Software Eng. 3(2), 125–143 (1977)
7. Schneider, F.B.: Enforceable security policies. ACM Trans. Inf. Syst. Secur. 3, 30–50 (2000)
8. Basin, D., Jugé, V., Klaedtke, F., Zălinescu, E.: Enforceable security policies revisited. In: Degano, P., Guttman, J.D. (eds.) Principles of Security and Trust. LNCS, vol. 7215, pp. 309–328. Springer, Heidelberg (2012)
9. Drábik, P., Martinelli, F., Morisset, C.: Cost-aware runtime enforcement of security policies. Technical Report TR-11-2012, IIT-CNR (2012)
10. Ligatti, J., Bauer, L., Walker, D.: Run-time enforcement of nonsafety policies. ACM Transactions on Information and System Security 12(3), 1–41 (2009)
11. Fong, P.W.L.: Access control by tracking shallow execution history. In: IEEE Symposium on Security and Privacy, pp. 43–55. IEEE Computer Society (2004)
12. Khoury, R., Tawbi, N.: Which security policies are enforceable by runtime monitors? a survey. Computer Science Review 6(1), 27–45 (2012)
13. Khoury, R., Tawbi, N.: Using equivalence relations for corrective enforcement of security policies. In: Kotenko, I., Skormin, V. (eds.) MMM-ACNS 2010. LNCS, vol. 6258, pp. 139–154. Springer, Heidelberg (2010)
14. Martinelli, F., Morisset, C.: Quantitative access control with partially-observable Markov decision processes. In: Proceedings of CODASPY 2012, pp. 169–180. ACM (2012)
15. Martinelli, F., Matteucci, I., Morisset, C.: From qualitative to quantitative enforcement of security policy. In: Kotenko, I., Skormin, V. (eds.) MMM-ACNS 2012. LNCS, vol. 7531, pp. 22–35. Springer, Heidelberg (2012)