A Quantitative Approach for Inexact Enforcement of Security Policies*

Peter Drábik, Fabio Martinelli, and Charles Morisset

IIT-CNR, Security Group
Via Giuseppe Moruzzi 1, 56124 Pisa, Italy
{peter.drabik,fabio.martinelli,charles.morisset}@iit.cnr.it

Abstract. A run-time enforcement mechanism is a program in charge of ensuring that all the traces of a system satisfy a given security policy. Following Schneider's seminal work, there have been several approaches defining what kind of policies can be automatically enforced, and in particular, non-safety properties cannot be correctly and transparently enforced. In this paper, we first propose to build an enforcement mechanism using an abstract notion of selector. We then propose to quantify the *inexact* enforcement of a non-safety property by an enforcement mechanism, by considering both the traces leading to a non-secure output by this mechanism and the secure traces not output, thus formalizing an intuitive notion of security/usability tradeoff. Finally, we refine this notion when probabilistic and quantitative information is known about the traces. We illustrate all the different concepts with a running example, representing an abstract policy dealing with emergency situations.

Keywords: Runtime Enforcement, Safety, Security/Usability Tradeoff.

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 functionalities 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 *secure* sequences of actions are executed by the system. There have been several important approaches aiming at understanding and characterizing what kind of policies are enforceable, and we can cite in particular the seminal work of Schneider [18] and the one of Ligatti *et al.* [15]. In the former, Schneider showed that if a policy is not a safety property (*i.e.*, a non-secure trace might be extended to a secure one), then it

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

can only be enforced by a "mechanism that uses more information than would be available only from observing the steps of a target's execution". In the latter, Ligatti *et al.* introduce, among others, the edit-automaton, and prove that any security property (over finite traces), including non-safety ones, can be enforced by a monitor built from an edit-automaton.

Indeed, an edit-automaton is powerful enough to *buffer* the entire trace, and to only release the actions submitted by the target when it is certain that they do not violate the policy. However, in many cases, buffering is not a desirable option, for instance when a non-safety policy is built to deal with critical situations, where a non-secure action needs to be performed immediately, and the control might only come afterwards. Such situations can typically occur in the context of healthcare policies [1,5]. Moreover, since a monitor usually needs to output an action to the system whenever the target submits one, buffering the trace would lead to violate the *transparency*[1] of the monitor.

In this paper, we therefore start from the observation that since non-safety policies cannot be exactly enforced (*i.e.*, correctly and transparently), then a monitor can only enforce *inexactly* a non-safety policy. We formalize this concept by characterizing the non-secure traces output by a monitor together with the secure traces not output. By quantifying over these traces, we show that not all inexact monitors are alike, and that it is possible to define a *strategy* to adopt. Intuitively, this strategy corresponds to defining a *security/usability*[2] *tradeoff.*

The main contributions of this paper are three-fold. Firstly, we present a simple framework for runtime monitors with no editing power using a general concept of *selector* (which can be seen as a generalization of a security-automaton) and we define the notion of n-safety policy, which, intuitively, is a policy allowing for at most n non secure steps. Secondly, we introduce the concept of inexact enforcement with the sets $\mathbf{C}$ and $\mathbf{T}$, which, intuitively, contain the traces violating the correctness of the monitor and those violating its transparency, respectively, and we illustrate it on a detailed running example, inspired by a basic break-glass healthcare policy. In particular, we observe that in a basic setting, it might be better to go for an "all-or-nothing" approach. Finally, we present different levels of quantification over these sets, in particular using probabilistic and quantitative information about traces, paving the way towards a formal risk-aware run-time enforcement.

Outline. The rest of this paper is structured as follows: after presenting some related approaches, we introduce in Section 2 the basic concepts such as trace, security property, monitor and enforcement. In Section 3, we define selectors, and we show how to build a monitor. In Section 4, we define the notion of n-safety

[1] We adopt here a strict notion of transparency, which states that a correct input trace should not be edited at all, even if the result of the editing is somehow transparent. This is the reason why we do not consider expressive editing power, such as insertion or suppression, as they are not helpful with respect to strict transparency.

[2] We use here the notion usability in its general meaning, *i.e.*, stating that a system can be used, rather than as the ability to realize user requirements, as it can sometimes be found in security.

property, together with the one of n-selector. In Section 5, we characterize the inexact enforcement of a property, and we define the notion of security/usability tradeoff. In Section 6, we consider the case where the monitor knows the probability of each possible extension. Finally, in Section 7, we introduce the concept of quantitative policy, and we transform the previous tradeoff into the notion of risk exposure, after which we conclude and detail our future work.

Related Work. There has been a significant bulk of work on enforcement mechanisms, especially about their theoretical foundations. We mention here only the closest to our work, and we refer the reader to [12] for a comprehensive survey.

As already mentioned Schneider formalized Execution Monitoring (EM) mechanisms as security automata in [18]. Later, Ligatti *et al.* [15], starting from this initial model, defined a series of automata with increasing power that act as trace consumers and producers. The main novelty was the capability to alter the trace produced by the target, by suppressing, inserting and editing actions. The most powerful class is called edit automata. The semantics of the enforcement mechanisms was still given in terms of big step-semantics for automata. Later work [16] has shown how to enforce non-safety properties, also proposing a notion of enforcement up to equivalence among traces. In [3], Basin *et al.* consider the case where some actions are uncontrollable (*i.e.*, cannot be stopped), and define what policies can then be enforced, by using a Deterministic Turing Machine to model a monitor.

After these different approaches characterizing what properties one can enforce with a specific mechanism, a slightly different strand of work started to consider what happens when limiting the editing power of the edit-automata, already recognized to be too powerful. We can mention here some work considering a limited capability for trace history recording, e.g., shallow history in [9] and limited memory in [19]. The former considers automata that only record the set of relevant actions that happened in the past, while the latter allows only a limited knowledge. Basically, these approaches are limiting the visibility over the past trace, while we are interested here in investigating a limited visibility over the future trace. Some approaches also try to go beyond the traditional runtime enforcement model, such as the definition of facets values [2], where a variable can take different values according to the security level of the user, or the Secure Multi Execution framework [6], where a program is executed multiple times, with different levels of security, thus being able to consider a wider range of security policies, such as non-interference ones.

Another recent class of work is mainly interested in what behaviour an enforcement mechanism should have when the trace is not secure, since traditional definitions do not specify this case. In [4], the authors use a notion of distance among traces, thus expressing that if a trace is not secure, it should be edited to a secure trace close to the non-secure one.

2 Basic Definitions

As we said in the introduction, a monitor is responsible for enforcing a policy over a *system*, and we identify 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.

2.1 Actions and Traces

Let $\mathcal{A}$ be the set of actions that the target can output, such that $\mathcal{A}$ contains the special action $\ominus$, indicating that the target has finished its execution, and does not output any other actions. We consider here traces consisting of an unbounded number of actions, and we say that a trace is *consistent* when any action $\ominus$ is followed only by $\ominus$ actions. We write $\mathcal{T}$ for the set of all finite, unbounded, consistent traces, and given a natural number n, abusing the notation, we write $\mathcal{T}^n$ for the subset of $\mathcal{T}$ consisting of all traces of length n. For the sake of exposition, we consider that any trace σ can always be extended by adding any finite number of $\ominus$ actions, and conversely that the actions $\ominus$ at the end of a trace can always be removed. Given a trace $\sigma \in \mathcal{T}$, we write $|\sigma|$ for the length of σ (potentially including $\ominus$ actions at the end), and we write ϵ for the empty trace.

Given a trace σ, we write σ^i for the action located at the i-th index of σ, starting from 0. Note that we consider that σ^i is always defined, and returns $\ominus$ if i is greater than the length of σ. We extend this notation to write $\sigma^{i<j}$ for the sub-trace of σ between the locations i and $j-1$. For instance, given the trace $\sigma = abcde$, we have $\sigma^{1<3} = bc$. When the first index is equal to 0, we write $\sigma^{0<j} = \sigma^{<j}$, which corresponds to the prefix of length j of the trace. In particular, we have $\sigma^{<0} = \epsilon$.

Given two traces σ and τ in $\mathcal{T}$, we write $\sigma;\tau$ (or $\sigma\tau$ when no confusion can arise) for the concatenation of σ with τ, we write $\sigma \preceq \tau$ to denote that σ is a prefix of τ. Note that we consider the prefix relation to be built modulo the $\ominus$ actions[3] at the end of the trace, in other words, if $\sigma \preceq \tau$, then $\sigma\ominus \preceq \tau$.

2.2 Security Policy and Property

In general, a security policy is expressed over sets of traces, describing all the secure runs of the system. For instance, a security policy could state that the system cannot execute the same trace twice. However, such a policy cannot be enforced by run-time monitoring, since it is usually considered that the monitor can only see the current trace submitted by the target. Following most existing approaches in the context of run-time enforcement, we only consider here security policies that can be defined using a security *property*, which is a computable predicate $\mathcal{P} : \mathcal{T} \to \mathbb{B}$.

[3] In general, we consider the $\ominus$ actions at the end of a trace as non-significant actions, and we use them mostly to avoid checking the length of a trace every time we want to access an action of a trace.

Moreover, we consider here only *reasonable* properties [16], which, in addition to being defined as computable predicates, also always hold over the empty trace, so that a system cannot start in a non-secure state. We therefore exclude liveness properties that require that a particular action is always executed. The reason we impose this constraint is to ensure that a monitor can always output a secure trace. Note that many access control policies (*e.g.*, the Access Matrix [13], the Bell-Padula model [14], the Role-based model [8], etc), can be expressed as security properties, since they do not depend on previous executions of the target, but only on the current state of the system.

Among reasonable properties, the category of *safety* properties has been particularly studied. Intuitively, with a safety property, a non-secure trace cannot be extended into a secure one. More formally:

Definition 1. *A reasonable property $\mathcal{P}$ is a safety property if, and only if:*

$$\forall \sigma_1, \sigma_2 \in \mathcal{T} \ \neg\mathcal{P}(\sigma_1) \land \sigma_1 \preceq \sigma_2 \Rightarrow \neg\mathcal{P}(\sigma_2)$$

Running Example. For instance, consider the set of actions $\mathcal{A}_e = \{\mathsf{b}, \mathsf{d}, \mathsf{s}\}$, where b stands for a *benign* action, d for a *dangerous* action and s for a *supervision* action. We want to express that the target can execute any benign action, but must execute first a supervision action before executing a dangerous action, which is given by the following property $\mathcal{P}_e$, which is clearly a safety property:

$$\mathcal{P}_e(\sigma) \Leftrightarrow (\forall i \ \sigma^i = \mathsf{d} \Rightarrow \exists j < i \ \sigma^j = \mathsf{s}).$$

We introduce in Section 4.1 the concept of *n-safety property*, which, roughly speaking, is a property allowing at most n “bad” steps. One of the problems we try to address in this paper is the enforcement of such properties.

2.3 Monitor

A monitor can be represented as a function $E : \mathcal{T} \to \mathcal{T}$, that is, a function taking a trace as an input from the target, and returning another trace to the system. As we said in Section 1, we are interested here in monitors that have no editing power over the trace, only a truncating one. In other words, the trace output by the monitor is either exactly the one input by the target, or its prefix. Intuitively, we want a monitor to be both *secure* (*i.e.*, that the monitor only output secure traces), and *usable* (*i.e.*, that the monitor does not truncate secure traces), which we express with the traditional notions of correct enforcement, transparent enforcement, and exact enforcement (for both correct and transparent) defined as follows.

Definition 2. *An enforcement mechanism $E : \mathcal{T} \to \mathcal{T}$ is said to:*

– *be* conservative *if, and only if, for any trace $\sigma \in \mathcal{T}$, we have $E(\sigma) \preceq \sigma$;*
– correctly *enforce a property $\mathcal{P}$ if, and only if, for any trace $\sigma \in \mathcal{T}$, we have $\mathcal{P}(E(\sigma))$,*

- transparently *enforce a property $\mathcal{P}$ if, and only if, for any trace $\sigma \in \mathcal{T}$ such that $\mathcal{P}(\sigma)$, we have $E(\sigma) = \sigma$.*
- exactly *enforce a property $\mathcal{P}$, which we denote as $E \vdash \mathcal{P}$ if, and only if, E is conservative, and correctly and transparently enforces $\mathcal{P}$,*

It is worth observing that even though we do not consider editing power here, for any reasonable property, there exists a monitor enforcing exactly this property: since the monitor has access to the whole trace, it can simply check whether the security property holds for the trace, and output it in the positive case, or output ϵ otherwise. In other words, the ability to edit a trace is not necessary to enforce exactly a reasonable property, knowing the future of the trace is enough.

We now present a framework based on the notion of *selector*, which is a simple function taking the past trace, the action to control, and decides whether the action should be kept or the trace stopped. To some extent, the role of a selector is close to the one of a security automaton [15]. A monitor is then defined by calling a selector for each action submitted by the target, thus making explicit the fact that a monitor must make a decision at each step, and cannot simply wait for the rest of the trace. We consider the notion n-selectors in Section 4. This framework allows us to tackle the problem of inexact enforcement in Section 5.

3 Selector-Based Monitor

We introduce our framework by revisiting the definition of the class EM (for Execution Monitoring), defined by Schneider [18] as excluding "mechanisms that use more information than would be available only from observing the steps of a target's execution", which we characterize by defining the concept of selector, which embodies the fact that a monitor must make a decision at each step using only past information.

3.1 Selector

A *selector* is a function $F : \mathcal{T} \times \mathcal{A} \to \mathbb{B}$, such that given a trace σ and an action a, $F(\sigma, a)$ is true if and only if the action a should be kept after having executed the trace σ. Moreover, we consider only selectors stating that if an action should not be kept, then any following action should also not be kept. We say that a monitor satisfying this constraint is *consistent*, which is formally specified as:

$$\forall \sigma \in \mathcal{T}\ \forall i\ F(\sigma^{<i}, \sigma^i) = \mathit{false} \Rightarrow \forall j > i\ F(\sigma^{<j}, \sigma^j) = \mathit{false}$$

For instance, the selectors F_{false} and F_{true}, which return always *false* and always *true*, respectively, are trivially consistent. Clearly, this notion of consistency is related to the one of safety. Indeed, given a computable property $\mathcal{P}$, the selector $F_{\mathcal{P}}$, as defined in Equation (1), is clearly consistent when $\mathcal{P}$ is a safety property.

$$F_{\mathcal{P}}(\sigma, a) = \begin{cases} \mathit{true} & \text{if } \mathcal{P}(\sigma; a) \\ \mathit{false} & \text{otherwise} \end{cases} \tag{1}$$

Intuitively, the selector $F_{\mathcal{P}}$ is analogous to a truncation automaton [15].

3.2 From Selector to Monitor

Given a selector, we can straight-forwardly define a monitor by calling the selector at each step. More formally, given a selector $F : \mathcal{T} \times \mathcal{A} \to \mathbb{B}$, we define the monitor $E_F : \mathcal{T} \to \mathcal{T}$ as follows:

$$\forall i \in \mathbb{N}\ E_F(\sigma)^i = \begin{cases} \sigma^i & \text{if } F(\sigma^{<i}, \sigma^i) = \mathit{true} \\ \ominus & \text{otherwise} \end{cases}$$

Clearly, if F is a consistent selector, for any consistent trace σ, $E_F(\sigma)$ is also consistent.For instance, from the two selectors F_{false} and F_{true} described above, we can define the monitors[4] E_{false} and E_{true}, which respectively always outputs the empty trace and always outputs the input trace. It is worth noting that both monitors are conservative, that E_{false} enforces correctly any property and that E_{true} enforces transparently any property. In other words, it is always trivial to build a correct monitor, and it is always trivial to build a transparent monitor, but in general, such monitors are not exact.

When $\mathcal{P}$ is a safety property, we can however always build the monitor $E_{\mathcal{P}}$ from the selector $F_{\mathcal{P}}$ defined in Equation 1, and we can prove that $E_{\mathcal{P}}$ enforces exactly $\mathcal{P}$ (all proofs can be found in the companion report [7]).

Lemma 1. *For any safety property $\mathcal{P}$, $E_{\mathcal{P}} \vdash \mathcal{P}$.*

Running Example. For instance, consider the trace bbsddb. Following the definition of $\mathcal{P}_e$, we have $E_{\mathcal{P}_e}(\mathsf{bbsddb}) = \mathsf{bbsddb}$, because a supervision action is present before any dangerous action. Now, consider the trace bbdsdb: we have $E_{\mathcal{P}_e}(\mathsf{bbdsdb}) = \mathsf{bb}\ominus\ominus\ominus\ominus = \mathsf{bb}\ominus$. Indeed, even though the second dangerous action is preceded by a supervision action, the first one is not, and since the $E_{\mathcal{P}_e}$ is consistent, every action after the first dangerous action included is transformed into the $\ominus$ action.

We can now formally express and prove that, as stated by Schneider [18] "if the set of executions for a security policy is not a safety property, then an enforcement mechanism from EM does not exist for this policy". Moreover, since we do not consider real-time availability problems or infinite memory, we can even prove the converse, as stated in Proposition 1 (which is simply the reformulation of Schneider's result in our framework).

Proposition 1. *Given a property $\mathcal{P}$, there exists an enforcement mechanism E in* EM *such that $E \vdash \mathcal{P}$ if and only if $\mathcal{P}$ is a safety property.*

This result, although expected, nonetheless narrows down enforceable properties to safety properties, and does not provide a way to enforce non-safety properties, such as those existing in emergency contexts. We present in the following section such properties, and we show in Section 5 how to enforce them inexactly.

[4] For the sake of clarity, when the name of the selector comes with a subscript, such as F_x, when no ambiguity can arise, we write directly E_x for the corresponding monitor instead of E_{F_x}.

4 *n*-safety Properties and *n*-selectors

In this section, we first introduce the concept of n-safety properties, which are properties where a maximum of n non-secure steps is allowed. We then define n-selectors, which are selectors that can use the following n steps to make a decision about an action. Intuitively, we introduce n-selectors to characterize the selection process with complete information about the future, and we use them when we have incomplete or imperfect information (Section 5.2).

4.1 *n*-safety

Although many security policies are safety properties, there also exist some properties which do not satisfy the safety condition, such as some liveness properties, as stated in Section 2.2, or policies including obligations requiring a given action to be executed after a particular trace, otherwise the entire trace would be non-secure. For instance, it might be authorized to create a temporary copy of a file, as long as this copy is deleted after usage. Another example concerns break-glass policies [1,5], usually defined in healthcare environments, where some violations of the property can be tolerated during an emergency situation, as long as an audit is performed afterwards.

In order to define the notion of n-safety, we first introduce the one of *critical trace*: given a trace σ and a property $\mathcal{P}$, a trace τ is said to be critical if, and only if, extending σ with τ can lead to a secure trace even though each step in τ is non-secure. More formally, we define the predicate $\mathsf{D}_{\mathcal{P}}$ as follows:

$$\forall \sigma, \tau \in \mathcal{T}\ \mathsf{D}_{\mathcal{P}}(\sigma, \tau) \Leftrightarrow \forall \tau' \preceq \tau\ (\tau' \neq \epsilon \Rightarrow\ \neg\mathcal{P}(\sigma;\tau')) \wedge \exists \tau'' \in \mathcal{T}\ \mathcal{P}(\sigma;\tau;\tau'')$$

Note that in the above definition, we implicitly assume that $\sigma;\tau$ belongs to $\mathcal{T}$. Clearly, this is not necessarily the case, for instance if σ ends with $\ominus$. In order to be rigorous, we would need to define the set $\mathcal{T}(\sigma)$, as the set of possible extensions from σ. However, for the sake of readability, we assume this constraint to be implicit, and notationally speaking, we use τ to indicate extensions rather than full traces.

We then define a n-safety property as a property holding for the empty trace and admitting critical traces whose length are at most n.

Definition 3. *A property $\mathcal{P}$ is an n-safety property if, and only if*

$$\forall \sigma, \tau \in \mathcal{T}\ D_{\mathcal{P}}(\sigma, \tau) \Rightarrow |\tau| \leqslant n$$

Clearly, given $m \leqslant n$, a m-safety property is also a n-safety property.

Running Example. One of the limitations of the running example is that it requires the target to first ask for supervision before being able to perform the dangerous operation. In some contexts, such as healthcare, it might not be possible to ask for a supervision first, and it might be needed to do the dangerous action first, and then only to ask for the supervisor to verify that everything was correct. For the sake of this example, let us introduce a new "critical" action c, such that when executing c, the supervision can be done at most k steps *after* the action.

For instance, for $k = 2$, the trace cb is critical for the trace b (or any other secure trace that does not contain s), since bcbs is secure while bcbb is not. However, cbb is not critical, since it is not possible to extend cbb to a secure trace. We can then define the property $\mathcal{P}_{k,e}$, which extends $\mathcal{P}_e$ by tolerating at most k non-secure steps, as:

$$\mathcal{P}_{k,e}(\sigma) \Leftrightarrow \mathcal{P}_e(\sigma) \vee (\forall i\ \sigma^i = \mathsf{c} \Rightarrow \exists (j \leqslant i + k)\ \sigma^j = \mathsf{s})$$

The property $\mathcal{P}_{k,e}$ is a combination of a safety property, $\mathcal{P}_e$, and of a k-safety property, and therefore is also a k-safety property.

4.2 n-selector

An *n-selector* is a function $F_n : \mathcal{T} \times \mathcal{A} \times \mathcal{T}^n \to \mathbb{B}$, such that for any n, for any traces σ and τ such that $|\tau| = n$, and any action a, $F_n(\sigma, a, \tau)$ indicates whether the action a should be kept knowing that the past trace is σ and that the future execution is τ. Intuitively, an n-selector is a selector with a visibility of n steps over the future trace, and it follows that n is equal to 0, an n-selector is equivalent to a selector. Let us observe that since it is always possible to ignore the end of a trace, given a selector F_n, it is always possible to build an equivalent selector F_m when $n \leqslant m$. For instance, the selectors F_{false} and F_{true} defined in Section 3.1 are also n-selectors, for any n.

Given a property $\mathcal{P}$, we can define the n-selector $F_{n,\mathcal{P}}$ in a similar fashion than the selector $F_{\mathcal{P}}$:

$$F_{n,\mathcal{P}}(\sigma, a, \tau) = \begin{cases} \mathit{true} & \text{if there exists } \tau' \preceq \tau\ \mathcal{P}(\sigma; a; \tau') \\ \mathit{false} & \text{otherwise} \end{cases} \tag{2}$$

For any n-safety property, the selector $F_{n,\mathcal{P}}$ is consistent, since if for some trace σ and index i, we have $F_{n,\mathcal{P}}(\sigma^{<i}, \sigma^i, \sigma^{i+1<i+n}) = \mathit{false}$, then by definition of n-safety, we know that any extension of σ^{i+n} is non-secure.

We show in [7] that n-selectors can be directly used to enforce exactly n-safety properties. However, in order to use an n-selector, one needs to be able to produce the n following steps of the trace, and although it could be possible in some contexts (for instance when the entire input trace is known in advance, or by buffering the n following steps, such as done by an edit-automaton [15]), in general the monitor does not have access to this information. For instance, in the running example, buffering the trace would defeat the purpose of the n-safety property, that is to accept time-critical actions in the absence of supervision. We study in the next section the problem of *inexact* enforcement.

5 Inexact Enforcement of n-safety Properties

An n-safety property can only be enforced *inexactly* (*i.e.*, either not correctly or not transparently, or both) by a mechanism in EM (i.e. with no precise knowledge about the future). We address in this section the problem of inexact enforcement, and we propose to quantify the necessary security/usability tradeoff.

5.1 Security/Usability Tradeoff

In order to quantify the non-correctness of a monitor, given a monitor E and a property $\mathcal{P}$, we introduce the set $\mathbf{C}_{\langle E,\mathcal{P}\rangle}$, which represents all traces for which the monitor outputs a non-secure trace. Similarly, the quantification of the non-transparency is done using the set $\mathbf{T}_{\langle E,\mathcal{P}\rangle}$, which represents the secure traces that the monitor does not output as they are. More formally, we have:

$$\mathbf{C}_{\langle E,\mathcal{P}\rangle} = \{\sigma \in \mathcal{T} \mid \neg\mathcal{P}(E(\sigma))\} \qquad \mathbf{T}_{\langle E,\mathcal{P}\rangle} = \{\sigma \in \mathcal{T} \mid \mathcal{P}(\sigma) \wedge E(\sigma) \neq \sigma\}.$$

When E enforces exactly $\mathcal{P}$, we clearly have $\mathbf{C}_{\langle E,\mathcal{P}\rangle} = \mathbf{T}_{\langle E,\mathcal{P}\rangle} = \emptyset$. Conversely, since an n-safety property cannot be enforced exactly using a selector, at least one of these two sets is necessarily non empty.

We showed in Section 3.2 that it is always trivial to define a monitor that enforces a property either correctly or transparently, with the monitors E_{false} and E_{true}, respectively. It follows that minimizing one set without considering the other is trivial, and thus we are interested here in minimizing both sets together. In other words, we want to establish a tradeoff between security (*i.e.*, correctness) and usability (*i.e.*, transparency), when both cannot be achieved simultaneously. Intuitively, the bigger the set $\mathbf{C}_{\langle E,\mathcal{P}\rangle}$ is, the less secure E is, while the bigger the set $\mathbf{T}_{\langle E,\mathcal{P}\rangle}$, the less usable E is.

5.2 Building a Selector from an n-selector

Given a trace σ and an action a, we can define a selector F from the n-selector $F_{n,\mathcal{P}}$ defined in Section 4.2. In order to illustrate the general approach, let us first consider a very simple case, and let us assume that only two traces τ_1 and τ_2 of length n are possible after $\sigma; a$ (meaning that the whole trace is either $\sigma; a; \tau_1$ or $\sigma; a; \tau_2$). Three cases might occur:

– $F_{n,\mathcal{P}}(\sigma, a, \tau_1) = F_{n,\mathcal{P}}(\sigma, a, \tau_2) = \mathit{true}$, and we can define $F(\sigma, a) = \mathit{true}$;
– $F_{n,\mathcal{P}}(\sigma, a, \tau_1) = F_{n,\mathcal{P}}(\sigma, a, \tau_2) = \mathit{false}$, and we can define $F(\sigma, a) = \mathit{false}$;
– $F_{n,\mathcal{P}}(\sigma, a, \tau_1) = \mathit{true}$ and $F_{n,\mathcal{P}}(\sigma, a, \tau_2) = \mathit{false}$ (the converse case being equivalent), and in this case, the selector F has to make a choice: either to stop and not output the secure trace $\sigma; a; \tau_1$, or to continue and output the non-secure trace $\sigma; a; \tau_2$ (or a non-secure prefix of it).

The first two cases are rather straight-forward, and the difficulty clearly lies in the third case. In general, this choice has to be made with respect to all possible extensions of length n, and not only τ_1 and τ_2. The crucial point is to make a decision for a trace σ and an action a such that $\mathcal{P}(\sigma)$ holds and $\mathcal{P}(\sigma; a)$ does not, and when there exists a trace τ such that $F_{n,\mathcal{P}}(\sigma, a, \tau) = \mathit{true}$. Indeed, at this point, the selector is left with two choices: either accepting a and taking the risk to output an non-secure trace, for instance if the target stops right after a or if the trace τ is not output, or stopping at this point, and therefore not outputting the secure trace $\sigma; a; \tau$.

Since at this level of formalism, we only compare two traces based on the fact that they are secure or not (we consider in the following sections the probability and the impact of a trace), we propose here a general selector F_r, specifying that the number non-secure extensions must not exceed a given ratio r of the total number of traces:

$$F_r(\sigma, a) = \begin{cases} \mathit{true} & \text{if there exists } \tau \in \mathcal{T}^n \text{ such that } F_{n,\mathcal{P}}(\sigma, a, \tau) = \mathit{true} \\ & \quad \text{and } |\{\tau \in \mathcal{T}^n \mid F_{n,\mathcal{P}}(\sigma, a, \tau) = \mathit{false}\}| \leqslant r\,|\mathcal{T}^n| \\ \mathit{false} & \text{otherwise} \end{cases}$$

In particular, the selector F_0 only accepts an action when it is certain that every possible extension is secure, while the selector F_1 accepts actions as long as there is a possibility for a secure extension. Note that given any $0 \leqslant r \leqslant 1$, if a trace is secure at each step, then F_r accepts it, and if a trace is irremediably bad (*i.e.*, there is no possibility to extend it to a secure trace), then F_r stops the trace.

5.3 From Local Ratio to Global Ratio

The local ratio r defined in the previous section acts as "worst-case scenario", since in any case, the selector F_r stops a trace whenever the proportion of non-secure extensions is above this ratio. However, from a global perspective, we are mostly interested in quantifying the proportion of non-secure traces output and of secure traces not output from *all* possible traces. We thus define $\mathbf{QC}_{\langle E,\mathcal{P},k\rangle}$ and $\mathbf{QT}_{\langle E,\mathcal{P},k\rangle}$ as the proportion of the number of traces of length k in $\mathbf{C}_{\langle E,\mathcal{P}\rangle}$ and in $\mathbf{T}_{\langle E,\mathcal{P}\rangle}$ to all the traces in $\mathcal{T}^k$, respectively.

To some extent, these two values can be seen as *utility* functions of a monitor with respect to a property, and therefore establishing the security/usability tradeoff consists in providing a minimization strategy for these utility functions. For instance, a very simple strategy could be to minimize the sum $(\mathbf{QC}_{\langle E,\mathcal{P},k\rangle} + \mathbf{QT}_{\langle E,\mathcal{P},k\rangle})$, or to define a threshold on $\mathbf{QC}_{\langle E,\mathcal{P},k\rangle}$ and to minimize $\mathbf{QT}_{\langle E,\mathcal{P},k\rangle}$.

The local ratio r can be therefore seen as a pessimistic approach, and it is straight-forward to observe that $\mathbf{QC}_{\langle E_r,\mathcal{P},k\rangle} \leqslant r$. However, as we illustrate in the next section, it is possible to have a situation where r is much greater than $\mathbf{QC}_{\langle E_r,\mathcal{P},k\rangle}$, and in general, given a specific strategy over $(\mathbf{QC}_{\langle E_r,\mathcal{P},k\rangle}, \mathbf{QT}_{\langle E_r,\mathcal{P},k\rangle})$, it is not trivial to define an r that satisfies it.

Indeed, the selector only makes a decision based on the relative number of possible non-secure extensions, which is not necessarily representative of the global number of non-secure traces. In other words, the local ratio r should be considered as a parameter of the selector specifying which behaviour to adopt when facing a critical trace, rather than a global measure of inexactitude. Let us nonetheless observe that for any specific ratio, given a fixed trace length k, we can derive the values $\mathbf{QC}_{\langle E_r,\mathcal{P},k\rangle}$ and $\mathbf{QT}_{\langle E_r,\mathcal{P},k\rangle}$. In other words, the required ratio r can be *inferred* from the strategy, as we illustrate in the next section. We believe the problem of calculating the optimal ratio r for any strategy is a challenging one, and we plan to address it in future work.

Table 1. Tradeoff values for $\mathcal{P}_{2,e}$ and $k = 4$

E ratio	E_{false}	E_{r_1} $0 \leqslant r_1 < 2/3$	E_{r_2} $2/3 \leqslant r_2 < 16/21$	E_{r_3} $16/21 \leqslant r_3 \leqslant 1$	E_{true}
$\mathbf{QC}\langle\ \rangle$	0	0	0.25	0.21	0.54
$\mathbf{QT}\langle\ \rangle$	0.45	0.11	0.04	0	0

5.4 Running Example

Let us consider the 2-safety property $\mathcal{P}_{2,e}$, as defined in Section 4.1, and a trace length of 4. We recall that a trace consists of any sequence of actions in $\{\mathsf{b}, \mathsf{d}, \mathsf{s}, \mathsf{c}\}$, possibly followed by $\ominus$ actions. It follows that $|\mathcal{T}^4| = \frac{4^5-1}{3} = 341$, with the following break-down: there are 112 non-secure traces caused by a d not preceded by an s (e.g., $\mathsf{bd}\ominus\ominus$); 73 non-secure traces caused by a c not followed by an s in the next two steps (e.g., $\mathsf{cbb}\ominus$); 12 secure traces with exactly two non-secure steps (e.g., bcbs); 27 secure traces with exactly one non-secure step (e.g., csdb) and 117 secure traces without non-secure step (e.g., $\mathsf{sdc}\ominus$).

The monitor E_{true} outputs all 185 non-secure traces and all secure traces, and therefore we have $\mathbf{QC}\langle E_{true}, \mathcal{P}_{2,e}, 4\rangle \approx 0.54$ and $\mathbf{QT}\langle E_{true}, \mathcal{P}_{2,e}, 4\rangle = 0$. Conversely, the monitor E_{false} does not output any non-secure trace, but only output the secure trace ϵ, meaning that $\mathbf{QC}\langle E_{false}, \mathcal{P}_{2,e}, 4\rangle = 0$ and $\mathbf{QT}\langle E_{false}, \mathcal{P}_{2,e}, 4\rangle \approx 0.45$. Intuitively, those two monitors should act as "worst cases", meaning that defining a monitor worse than them is pointless.

Now, let us identify the different possible ratios r, and the corresponding tradeoff values. Let σ be a trace that does not contain s and the action c, then there are 14 traces τ out of 21 of length 2 such that $F_{2,\mathcal{P}_{2,e}}(\sigma, \mathsf{c}, \tau) = \mathit{false}$. Hence, for any ratio $0 \leqslant r_1 < 2/3$, $F_{r_1}(\sigma, \mathsf{c}) = \mathit{false}$, which means that E_{r_1} does not accept any trace with a non-secure step. On the other hand, if the ratio is above $2/3$, the action c is accepted. Now, given the trace $\sigma; \mathsf{c}$, where σ does not contain s, and an action a either equal to b or to c, then there are 16 traces τ out of 21 of length 2 such that $F_{2,\mathcal{P}_{2,e}}(\sigma; \mathsf{c}, a, \tau) = \mathit{false}$. It follows that given a ratio $2/3 \leqslant r_2 < 16/21$, $F_{r_2}(\sigma; \mathsf{c}, a, \tau) = \mathit{false}$. Finally, it is easy to see that if the ratio is above $16/21$, then the selector only stops when all future traces are non-secure. We summarize these results in Table 1.

Interestingly, we can observe that the monitor E_{r_2} is always worse than E_{r_3}, which is due to the fact that since F_{r_2} always accepts the first c, but stops if the following action is not an s, then it outputs a non-secure trace in this case, even for the correct input traces. In other words, it means that without further information about the probability or the impact of a trace, an "all-or-nothing" approach should be adopted: either all secure traces should be output, with E_{r_3}, or none, with E_{r_1}.

6 Probabilistic Enforcement

In the previous section, we implicitly consider any two traces have the same probability to occur. However, in some cases, it is possible to quantify the *likelihood* of a particular trace to happen, typically when the probabilistic behaviour of the target is provided. For instance, consider the running example: it could be possible to determine the likelihood of a particular nurse to ask for supervision after executing a dangerous action, by statistically analyzing the past activity of this nurse, or by using a pre-established level of trust for this nurse. In general, let us assume that there exists a *conditional probability function* $\psi(\tau|\sigma)$, indicating the probability of the trace τ to occur after the trace σ. We require such a function to be a probability distribution for traces of same length, that is, $\sum_{\tau \in \mathcal{T}^n}(\tau \mid \sigma) = 1$, for any n and any σ.

The security/usability tradeoff can now be expressed with $\mathbf{PC}\langle E,\mathcal{P},k\rangle$, which represents the accumulated probability for the target to output a trace of length n such that the monitor outputs a non-secure trace, and $\mathbf{PT}\langle E,\mathcal{P},k\rangle$, which represents the accumulated probability for the target to output a secure trace of length n such that the monitor does not output the same trace. More formally, given a monitor E, an n-safety property $\mathcal{P}$ and trace length k, we define:

$$\mathbf{PC}\langle E,\mathcal{P},k\rangle = \sum_{\sigma \in \mathcal{T}^k} \{\psi(\sigma|\epsilon) \mid \sigma \in \mathbf{C}\langle E,\mathcal{P}\rangle\}$$

$$\mathbf{PT}\langle E,\mathcal{P},k\rangle = \sum_{\sigma \in \mathcal{T}^k} \{\psi(\sigma|\epsilon) \mid \sigma \in \mathbf{T}\langle E,\mathcal{P}\rangle\}$$

The ratio-based selector F_r defined in Section 5.3 can therefore be refined to a probabilistic ratio-based selector F_{p_c}, such that given a trace σ, an action a and a probability threshold p_c, F_{p_c} is defined by:

$$F_{p_c}(\sigma, a) = \begin{cases} \mathit{true} & \text{if: - there exists } \tau \in \mathcal{T}^n \text{ such that } F_{n,\mathcal{P}}(\sigma, a, \tau) = \mathit{true}; \\ & \quad\text{- } \psi(\tau|\sigma; a) \neq 0; \text{ and} \\ & \quad\text{- } \sum_{\tau \in \mathcal{T}^n}\{\psi(\tau|\sigma; a) \mid F_{n,\mathcal{P}}(\sigma, a, \tau) = \mathit{false}\} \leqslant p_c \\ \mathit{false} & \text{otherwise} \end{cases}$$

Following a similar reasoning to the one for F_r, we can see that $\mathbf{PC}\langle E_{p_c},\mathcal{P},k\rangle \leqslant p_c$, since at each step, the monitor only keep accepting actions if the aggregated probability of reaching a bad trace is lower than p_c.

7 Quantitative Enforcement

In some context, it can be argued that outputting a particular non-secure trace does not have the same *impact* as outputting another one. For instance, in the running example, the impact of the trace dddd is intuitively much higher than that of b⊖⊖⊖. In other words, quantifying only over the sets $\mathbf{C}\langle E,\mathcal{P}\rangle$ and $\mathbf{T}\langle E,\mathcal{P}\rangle$ might not be enough, even with the probability of traces. In this section, we consider that in addition to the security property, we are provided with a *quantitative property*, associating each trace with a specific impact.

Given a set of traces $\mathcal{T}$ and a domain of values $(\mathbb{V}, \leq, +, -)$, a *quantitative property* is a function $Q : \mathcal{T} \to \mathbb{V}$. For the sake of generality, we assume that both $\mathcal{P}$ and Q coexist, as they serve a different purpose: $\mathcal{P}$ indicates which traces are secure, while Q indicates the impact of each trace. For instance, Q might denote the financial cost on the system to accept a trace, which could be in some instances smaller for a non-secure trace than for a secure one.

Given an n-safety quantitative property $(\mathcal{P}, Q)$, an enforcement mechanism $E : \mathcal{T} \to \mathcal{T}$ and a fixed length of traces k, we define $\mathbf{RC}\langle E,\mathcal{P},k\rangle$, which corresponds to the risk created by accepting non-secure traces, and therefore corresponds to a notion of *damage*, and $\mathbf{RT}\langle E,\mathcal{P},k\rangle$, which corresponds to the risk of not accepting some secure traces, and therefore rather corresponds to a notion of *loss*:

$$\mathbf{RC}\langle E,\mathcal{P},Q,k\rangle = \sum_{\sigma\in\mathcal{T}^k} \{\psi(\sigma|\epsilon)\cdot Q(E(\sigma)) \mid \sigma \in \mathbf{C}\langle E,\mathcal{P}\rangle\}$$

$$\mathbf{RT}\langle E,\mathcal{P},Q,k\rangle = \sum_{\sigma\in\mathcal{T}^k} \{\psi(\sigma|\epsilon)\cdot (Q(\sigma) - Q(E(\sigma)) \mid \sigma \in \mathbf{T}\langle E,\mathcal{P}\rangle\}.$$

It is worth observing that using the monitor E_{false} leads to $\mathbf{RC}\langle E_{false},\mathcal{P},k\rangle = 0$, which corresponds to the intuition that refusing every trace does not create any risk in terms of security, while for the monitor E_{true}, we have $\mathbf{RT}\langle E_{true},\mathcal{P},k\rangle = 0$, which corresponds to the intuition that accepting every trace does not create a risk of loss of usability. Let us also observe that if $\mathcal{P}$ is a safety property, then $\mathcal{P}$ can be exactly enforced by the monitor $E_{\mathcal{P}}$, with $\mathbf{RC}\langle E_{\mathcal{P}},\mathcal{P},k\rangle = \mathbf{RC}\langle E_{\mathcal{P}},\mathcal{P},k\rangle = 0$. This observation is consistent with the intuition that if the future can be ignored, then the monitor does not need to expose itself to any risk.

In general, given an n-safety property $\mathcal{P}$ and a risk threshold λ, indicating the risk limit a monitor can take, we can define the selector F_λ, such that given a trace σ and an action a:

$$F_\lambda(\sigma, a) = \begin{cases} \mathit{true} & \text{if } \sum_{\tau\in\mathcal{T}^n}\{\psi(\tau|\sigma;a)\cdot Q(\sigma;a;\tau) \mid F_{n,\mathcal{P}}(\sigma,a,\tau) = \mathit{false}\} \leq \lambda \\ \mathit{false} & \text{otherwise} \end{cases}$$

Following a similar reasoning to the one for F_{p_c}, we can see that $\mathbf{RC}\langle E_\lambda,\mathcal{P},k\rangle \leq \lambda$, since at each step, the monitor only keep accepting actions if the aggregated probability of reaching a bad trace is lower than λ.

Running Example. We measure the impact of a trace by the number of non-secure actions executed by the system: we add 1 for each d not preceded by s, and 1 for each c neither preceded by s nor followed by s in the two following steps. For instance, $Q_e(\mathsf{ddsd}) = 2$, and $Q_e(\mathsf{cbbs}) = 1$. Note that in this case $\mathbf{RT}\langle E,\mathcal{P}_{2,e},Q_e,4\rangle$ roughly measures the impact of secure traces that have been output as non-secure traces by E. We present in Table 2 the different thresholds experimentally obtained for the parameter λ, assuming all traces are equiprobable.

Two results are particularly interesting to observe. Firstly, the monitor E_{λ_2}, which corresponds to the monitor E_{r_2}, has a lower impact than the monitor E_{λ_4}, which corresponds to the monitor E_{r_3}. In other words, when the impact of traces is taken into account, then an approach "all-or-nothing" is not necessarily the best, and it might be worth stopping in a non-secure state, even though

Table 2. Tradeoff values for $\mathcal{P}_{2,e}$, Q_e, $k = 4$ and equiprobable traces

E	E_{false}	E_{λ_1}	E_{λ_2}	E_{λ_3}	E_{λ_4}	E_{true}
ratio		$0 \leqslant \lambda_1 < 1.34$	$1.34 \leqslant \lambda_2 < 1.48$	$1.48 \leqslant \lambda_3 \leqslant 2.1$	$2.1 \leqslant \lambda_4$	
QC	0	0	0.25	0.23	0.21	0.54
QT	0.45	0.11	0.04	0.02	0	0
RC	0	0	0.25	0.23	0.28	1.09
RT	0	0	0.04	0.02	0	0

there still is a possibility to find a secure extension. Secondly, the monitor E_{λ_3} is created, and the only difference with E_{λ_2} is that E_{λ_3} accepts the first non-secure c, and accepts the following action only if it is a b or a s. In other words, E_{λ_3} is willing to take the risk of the first non-secure c, but not of a second one.

8 Conclusion

In this paper, we have presented the problem of inexact enforcement of security policies, and of non-safety properties in particular. We have formalized the concept of security/usability tradeoff by quantifying over the set of non-secure traces output by a monitor and the set of secure traces not output. When probabilistic knowledge about the future trace is available, we can define the notion of probability of failure. Moreover, when quantitative information is also available, then it is possible to define the risk exposure of a monitor. We have illustrated that the best approach to adopt when dealing with critical traces depends on whether quantitative information is available or not, which reinforces the idea that security policies need to include a quantitative aspect [11].

The ultimate goal, the construction of the optimal monitor that minimizes the risks or its approximations, is an ongoing and future work. Verification techniques such as probabilistic model checking [10] might be a useful tool, since they enable analysis of quantitative properties of probabilistic models. An interesting approach is to consider the tradeoff as some utility functions that need to optimized, such as it is done in [17], where the authors model access control systems as Markov Decision Processes, thus leading towards a general notion of utility-based security. We also need to validate the applicability of our approach for real-world scenarios, and to understand whether the calculations can be done at runtime, or should rather be done as an analysis of a security property. Finally, we would like to increase the expressiveness of monitors, in order to consider editing of a bad trace, which extends the range of the choices to analyze at run-time.

References

1. Ardagna, C.A., De Capitani di Vimercati, S., Grandison, T., Jajodia, S., Samarati, P.: Regulating Exceptions in Healthcare Using Policy Spaces. In: Atluri, V. (ed.) DAS 2008. LNCS, vol. 5094, pp. 254–267. Springer, Heidelberg (2008)

2. Austin, T.H., Flanagan, C.: Multiple facets for dynamic information flow. In: Proceedings of POPL 2012, pp. 165–178. ACM, New York (2012)
3. Basin, D., Jugé, V., Klaedtke, F., Zălinescu, E.: Enforceable Security Policies Revisited. In: Degano, P., Guttman, J.D. (eds.) POST 2012. LNCS, vol. 7215, pp. 309–328. Springer, Heidelberg (2012)
4. 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)
5. Brucker, A.D., Petritsch, H.: Extending access control models with break-glass. In: Proceedings of SACMAT 2009, pp. 197–206. ACM, New York (2009)
6. Devriese, D., Piessens, F.: Noninterference through secure multi-execution. In: Proceedings of the 2010 IEEE Symposium on Security and Privacy, SP 2010, pp. 109–124. IEEE Computer Society, Washington, DC (2010)
7. Drábik, P., Martinelli, F., Morisset, C.: A quantitative approach for the inexact enforcement of security policies. Technical Report TR-07-2012, IIT-CNR (2012)
8. Ferraiolo, D.F., Kuhn, D.R.: Role-based access control. In: Proceedings of the 15th National Computer Security Conference, pp. 554–563 (1992)
9. Fong, P.W.L.: Access control by tracking shallow execution history. In: Proceedings of Security and Privacy, pp. 1–13 (2004)
10. Forejt, V., Kwiatkowska, M., Norman, G., Parker, D.: Automated Verification Techniques for Probabilistic Systems. In: Bernardo, M., Issarny, V. (eds.) SFM 2011. LNCS, vol. 6659, pp. 53–113. Springer, Heidelberg (2011)
11. Kephart, J.: The utility of utility: Policies for self-managing systems. In: Proceedings of POLICY 2011, Pisa, Italy. IEEE Computer Society (2011)
12. Khoury, R., Tawbi, N.: Which security policies are enforceable by runtime monitors? a survey. Computer Science Review 6(1), 27–45 (2012)
13. Lampson, B.: Protection. In: Proceedings of the 5th Annual Princeton Conference on Information Sciences and Systems, pp. 437–443. Princeton University (1971)
14. LaPadula, L., Bell, D.: Secure Computer Systems: A Mathematical Model. Journal of Computer Security 4, 239–263 (1996)
15. Ligatti, J., Bauer, L., Walker, D.: Edit automata: Enforcement mechanisms for run-time security policies. Journal of Information Security 4(1-2), 2–16 (2005)
16. Ligatti, J., Bauer, L., Walker, D.: Run-time enforcement of nonsafety policies. ACM Transactions on Information and System Security 12(3), 1–41 (2009)
17. Martinelli, F., Morisset, C.: Quantitative access control with partially-observable markov decision processes. In: Proceedings of ACM CODASPY 2012, pp. 169–180. ACM, New York (2012)
18. Schneider, F.B.: Enforceable security policies. ACM Trans. Inf. Syst. Secur. 3, 30–50 (2000)
19. Talhi, C., Tawbi, N., Debbabi, M.: Execution monitoring enforcement under memory-limitation constraints. Information and Computation 206(2-4), 158–184 (2008)