Computer-Aided Generation of Enforcement Mechanisms for Error-Tolerant Policies

Nataliia Bielova   Fabio Massacci
Department of Computer and Information Science
University of Trento, Italy
Email: lastname@disi.unitn.it

Abstract—The basic tenet of security management when actions violate policies is that the former must be forbidden or amended. This requires to specify precisely all possible exceptions and corrections to the default workflow.

In many practical e-health business processes this is not feasible: the default clinical or administrative protocol is simple and well understood by clinicians but the precise codification of all possible amendable errors into the policy would transform it from a straight-line to an unreadable spaghetti-graph.

In this paper we propose a more practical alternative: the clinician only specifies the default protocol and marks for each protocol step the venial errors and their possible corrections. Given a global bound on the amount of errors in a trace that can be tolerated for each workflow execution, we can automatically generate an edit-automata that can provably enforce the policy with a sufficient degree of predictability (a policy metric for error correction).

We illustrate our approach with a concrete e-health workflow from the Italian region of Lombardy.

Keywords-security; e-health; error correction; automated policy generation

I. INTRODUCTION

One of the challenge in policy specification and enforcement is the trade-off between writing simple policies and enforcing complex run-time behaviors. This is particularly important for workflows in which human actors must interact with the policy enforcement mechanism. A paradigmatic example that we consider in this paper is the (relatively simple) case of drug dispensation workflows at a hospital.

From the perspective of the run-time monitor in charge of enforcing the compliance of the actual workflow with the health and privacy regulations, the richer the policy the better. During the executions doctors, nurses, and pharmacists will not be disturbed in their primary mission (delivering the right drug to the right patient) because an insignificant deviation from the default workflow has taken place. Run-time exceptions and the consequent additional workload needed to complete the execution distract users and convince them that "The system doesn't work" (or, worse, "The system doesn't trust us"). If such disruptions occurs too often users will increasingly try bypass the run-time monitor.

From the perspective of policy management, detailed policies are difficult to write, are difficult to check for consistencies and, de facto impossible to communicate to the end users. So, the simpler the policy the better. Back to our case, a simple drug dispensation protocol for highly sensitive drugs (HIV or chronic-related drugs) is, in its essence, a linear sequence of steps (with few loops for stock replenishing). This is easily understood by doctors and nurses. However each and every step might be subject to a number of exceptions or common minor errors (for example closing a window instead of pressing the button done). Detailing and representing all these exceptional steps in a graphical form would make the protocol unreadable.

In our domain (we discuss more details of the requirements of the hospital workflow in [1]) there is a further difficulty: our users would definitely insist that there is only one *policy*, i.e. the "official" protocol workflow. There is no such a thing as a policy including all exceptions (this would require validation by the risk manager and a number of responsible for the pharmacy and dispensation process). However, the *system* can (should) be flexible and users could specify which and how many deviations could be tolerated by the system or which actions should be amended.

A. Our Contribution

In order to exit from this impasse, we build upon the works of automatic policy generation and run-time enforcement to propose a semi-automatic way to generate enforcement mechanisms that can tolerate up to k errors given a "default" workflow and a specification of a simple list of errors and possibly their corrections.

As the underlying enforcement mechanism we use *edit automata* [2] as they are based on a strong theory for runtime enforcement, and they effectively enforce all renewable policies. The edit automata can transform the actions that do not comply with the policy (*bad* actions) in many different ways to produce *good* actions. We would like to generate a subclass of edit automata that perform this transformation in a well-defined way.

This subclass extends the classical default-deny policy, considering the type and number of deviations from the policy that the edit automaton can allow or amend. In contrast to the classical approach, where the runtime enforcer is constructed from the given policy, we summarize our idea in Figure 1.

(a) Mechanism enforces the policy.

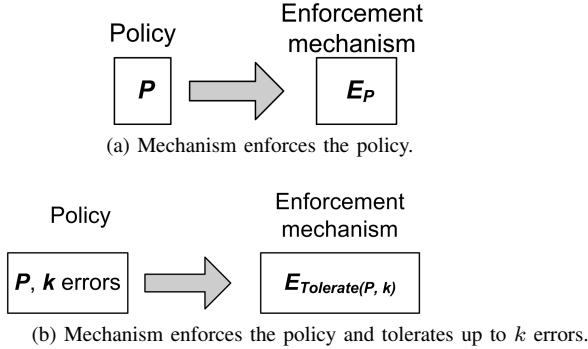(b) Mechanism enforces the policy and tolerates up to k errors.

Figure 1: Original idea and our idea of constructing an enforcement mechanism.

Later in the paper we will propose a class of errors called *venial errors*. They are actions that are not explicitly "allowed" in the security policy, however, they are not harmful.

Given a policy P and a maximum number of errors/deviations k, we construct a subclass of edit automata that can provably enforce the given policy by tolerating up to k errors and whose behavior is predictable (i.e. deviates minimally from the intended behavior of the users as we defined it in our previous work [1]).

Section II describes a concrete case study from the Hospital S. Raffaele of Milano (HSR) that shows an example of an application and presents the intuitive notion of venial errors. Then, we introduce the basic notions of enforcement monitors, provide the formal definition of venial errors and present the properties of enforcement mechanism that can tolerate them in Sections III- IV. Section V presents an automatic construction of enforcement mechanism. We conclude the paper in Section VI.

II. RUNNING EXAMPLE

The drug dispensation process is a high level business process for a hospital. As one example, there is a process called File F that allows refunding of the drugs for specific critical and chronic diseases that should be done by the public authority. As another example, if the doctor is prescribing a specific drug for the research program purposes (i.e. the patient has been enrolled for the clinical trial for the testing of that drug), the reimbursement should be done by the clinical trial funds.

Drug dispensation process involves human participants as well as IT technologies. Some tasks are human activities without any interaction with IT system (e.g. all patients tasks, or delivering drugs from stock to patient (physically) by doctor or nurse). Hence, some of the activities may be done in a different way than described in the default process, and we will make a precise distinction between the actual execution of the process and the official default process. The described model of drug dispensation assures that if all the activities adhere to it, they comply with the encoded rules (e.g. File F process).

The fragment of the drug dispensation process that we consider in this paper is the *drug selection*. The main steps of this subprocess are listed below (in brackets the abbreviation used in the rest of the paper):

1) The doctor fills list of drugs for his patient and selects one drug (Dis).
2) If the drug is highly sensitive, reviewing therapeutical notes is needed. They will be shown to the doctor that has to review them (Tnn; Rtn). Otherwise therapeutical notes can be emitted (TnNn).
3) The system checks drug's submission to Research program and in case the drug is registered (Dr) shows the notification to the doctor. Then the doctor should insert the research protocol number (Irpn), a number of the protocol according to which the drug can be given to the patient. If the drug is not registered to Research program, then the doctor skips this step (DNr).
4) The doctor performs other actions needed for the drug prescription (Dpres).

The part of process we present has been simplified with respect to its real description but it can happen that the actions of the process execution do not correspond to the policy. In any practical environment there is always a trade-off between the needs of the users and actual executions they are allowed to perform. Strict enforcement might be too costly from an organizational perspective.

We have an aside but important note on the terminology used in this paper. Most papers on enforcement mechanisms use the words "violation" rather than "error". We will use them interchangeably. In the course of our many interactions with HSR, it has become apparent that the word "error" is preferred for psychological reasons. The term "violation" implies that a doctor would deliberately ignore the steps of the protocol and this implies for the end users a deliberate mistrust in their behavior by the evil security department. Of course doctors (as any user) could misbehave but in order to gain acceptance of the mechanisms it is preferable to present enforcement as a way to support honest users rather than to deter malicious users.

A. Venial and Amendable Errors

Coming back to our example of drug selection process, consider the action of reviewing the therapeutical notes by the doctor. While such notes are important (as they contain information about allergies, unwanted interactions etc.) they are normally updated very rarely and for frequently used drugs doctors might "forget" to actually review them and just skip them by closing the window of drug prescription instead of clicking on the "Done" button. In this scenario, therapeutical notes play the same role of click-through software installation agreements: which system administrator reads the n-th Microsoft software installation agreement?

Table I: A policy and possible errors

Policy	Error	No. of errors	Error type	Correction
Sequences satisfying the description of the process from running example	Instead of reviewing therapeutical notes close the window	k errors per day	Venial	No correction
	Research protocol number is not inserted		Can be corrected	Insert special number for audit

From the point of view of the medical process this can be considered a *venial error*: we can tolerate few deviations in which the logs showed that the doctor clicked ignore rather than reviewed some of the most commonly prescribed drugs.

On the other hand, the doctor should not be allowed to violate the policy systematically, nor we want to over-complicate the definition of the policy with all possible ways to treat venial errors. From a usability perspective we would just like to have a high level view, for example allowing to close the window (Ctw) instead of reviewing the therapeutical notes (Rtn) k times per day.

The enforcement mechanism should do the rest automatically. It should allow the user to make "almost" correct actions only this limited number of times and only if his errors are venial. We show Table I that can be made by an expert in the application domain saying which errors can be allowed and what number of times.

The second example is inserting the research protocol number in the protocol window. A doctor has to complete this step in order to proceed with the drug selection. After she fills in this number, the reimbursement of the drug will be done by the clinical trial funds. For all the drugs that are not for research the reimbursement later on is done by the public authority as described in the File F procedure. However, it might happen that the doctor skips the insertion of this number. In this case the drug reimbursement process will be done by the public authority which can not be considered a venial error since the whole reimbursement process for this drug will be wrong (reimbursement will be done by a wrong party). The enforcement mechanism would therefore need to generate an alert and "correct" the wrong step i.e. inserting another special number that later will be used during the audit[1].

In practice, different doctors can prescribe different drugs to different patients and we would like to avoid that a local infringement of the policy (e.g. a doctor forgot to click "I have reviewed the therapeutical notes") does not hang the entire process while keeping the overall policy as a whole.

Notice that we need the concept of a global policy and we cannot spawn an enforcement monitor for each doctor and patient pair. At first the notion of venial errors would be trivial: in the individual prescription process there is at most one venial error that could be made. Second and

foremost, the hospital is liable as a whole if too many errors are present. If all doctors are allowed one venial error in the individual prescription the hospital might end up with all process without therapeutic notes checklist and thus the venial error would become a systematic error leading to potential lawsuits.

So we want to define how the executions where "something locally bad may happen" can be enforced by tolerating errors. The actions, which are neither venial, nor amendable, are always present and cannot be fixed in practice. For example, when the process involves the interactions of an organization with another one; for instance we can refer to the cases of outsourcing services or to the cases in which some actions are done by external parties.

III. BACKGROUND AND NOTATION

The set of observable process actions is denoted by Σ and a set of possible actions to be executed is T. A *trace* is a finite or infinite sequence of actions; the set of all finite sequences over Σ is denoted by Σ^*, the set of infinite sequences is Σ^ω, and the set of all sequences is Σ^∞. By σ we refer to a trace and by $\cdot$ we refer to an empty trace. We write $\sigma; \tau$ to denote concatenation of two sequences, where σ must be finite. We denote the length of the trace σ by $|\sigma|$.

A trace consisting of actions requested for execution is a *tentative execution*. A runtime monitor $E : \Sigma^\infty \to T^\infty$ transforms tentative executions into sequences of actions that will be finally executed.

A *security policy* is a set of traces $P \subseteq \Sigma^\infty$. A policy P is a *security property* if there exists a predicate $\widehat{P}$ over the traces, such that $\forall \sigma \in \Sigma^\infty : \widehat{P}(\sigma) \Leftrightarrow \sigma \in P$. So we will use interchangeably the policy P with its corresponding predicate $\widehat{P}$. The trace σ that satisfies the property $\widehat{P}$ is called *good*, and the trace that does not satisfy the property is called *bad*.

In practice the policy is given implicitly by describing the workflow corresponding to good executions.

Example 1: The security policy P is made by the traces

- SimpleRun = Dis; TnNn; DNr; Dpres,
- NoteRun = Dis; Tnn; Rtn; DNr; Dpres,
- ResearchRun = Dis; TnNn; Dr; Irpn; Dpres,
- NoteResearchRun = Dis; Tnn; Rtn; Dr; Irpn; Dpres

and their closure under concatenation: for every $\sigma, \sigma' \in P$: $\sigma; \sigma' \in P$. An empty trace NoRun also satisfies the policy.

Example 2: Let us now make some examples of bad traces with respect to the policy P. The doctors might forget

---

[1]In the real implementation systems are not allowed to automatically perform certain actions as the final liability must stay with a human, however they can support the human by suggesting the relevant correction.
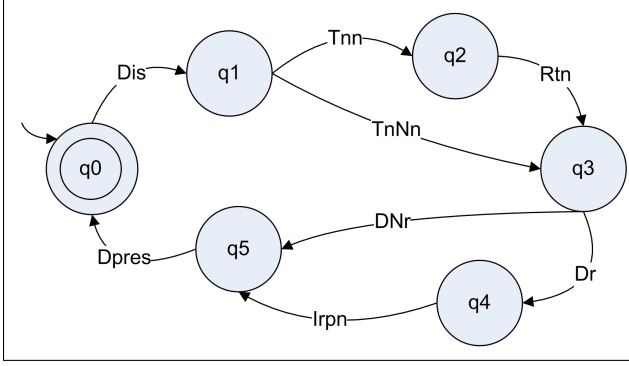
Figure 2: Finite state automaton representing a policy of drug selection subprocess

Abbreviations

Dis = Drug is selected
Tnn = Therapeutical notes needed
Rtn = Review therapeutical notes
TnNn = therapeutical notes not needed
Dr = Drug is for research
Irpn = Insert research protocol number
DNr = Drug is not for research
Dpres = Other drug selection actions

to click the "I have read the Therapeutical Note" button and rather close the window (Ctw). A similar event could happen for the step in which research protocol number is not inserted (Cpw) but the protocol window is closed. We list them below:

- CloseNote = Dis; Tnn; Ctw; Dr; Irpn; Dpres,
- CloseProt = Dis; Tnn; Rtn; Dr; Cpw; Dpres,
- CloseNoteProt = Dis; Tnn; Ctw; Dr; Cpw; Dpres

Figure 2 presents the security policy as a usual finite-state automaton, that we call a *Policy automaton* $A^P = \langle \Sigma, Q, q_0, \delta, F \rangle$. Σ is finite nonempty set of security-relevant actions, Q is a finite set of states, $q_0 \in Q$ is an initial state, $\delta : Q \times \Sigma \to Q$ is a labeled partial transition function, $F \subseteq Q$ is a set of accepting states. We will write $q \xrightarrow{a} q'$ whenever $\delta^P(q, a) = q'$. A finite run *is accepting* if the first state of the run is an initial state and the last state of the run is an accepting state. Good executions of the workflow are all the accepting runs of the Policy automaton.

An enforcement mechanism $E : \Sigma^\infty \to T^\infty$ is a sequence transformer, in this paper we consider a particular model of it, called *edit automaton* [3], [2]. Edit automata have a power of transforming sequences of actions by inserting actions and suppressing them. We present our own definition of this automaton. Intuitively, we have just simplified the original notions by enucleating the notions of output and memory and always forced the enforcement mechanism to progress in the processing of the input. Our actions can then be shown to be identical to a combinations of the atomic actions (read symbol but no output, output symbol but don't read input) from [2] on every non-diverging computation[2].

Definition 3.1 (Edit automata): An *edit automaton* E is a 5-tuple of the form $\langle Q, q_0, \delta, \gamma_o, \gamma_k \rangle$ with respect to some

---

[2]A diverging computation is a computation where the edit automaton will run forever without reading any input while keeping outputting data. While it was theoretically useful in [2] the very idea that an enforcement mechanism could possibly produce output without any input stimulus turned out a difficult sell to normal users. In contrast, the idea that the enforcement mechanism could spend a lot of time in order to process an input and eventually report a long sequence of follow-up actions was considered impractical but understandable.

system with actions set Σ. Q specifies the possible states, and $q_0 \in Q$ is the initial state. The total function $\delta : (Q \times \Sigma) \to Q$ specifies the transition function; the total function $\gamma_o : (Q \times \Sigma^* \times \Sigma) \to \Sigma^*$ defines the output of the transition according to the current state, the current input action and the memory; the total function $\gamma_k : (Q \times \Sigma^* \times \Sigma) \to \Sigma^*$ defines the updated memory after committing the transition.

In order for the enforcement mechanism to be effective all functions δ, γ_k and γ_o should be computable. In practice they should require polynomial if not constant time.

The function γ_o defines an output of the automaton produced at one transition and the function γ_k defines the memory containing the actions that are proceeded by the automaton but not output yet.

Definition 3.2 (Run of an Edit automaton): Let $A = \langle Q, q_0, \delta, \gamma_o, \gamma_k \rangle$ be an edit automaton. A *run* of A on an *input* sequence of actions $\sigma = a_1; a_2; \ldots$ is a sequence of pairs $\langle (q_0, \epsilon), (q_1, \sigma_1^k), (q_2, \sigma_2^k), \ldots \rangle$ such that $q_{i+1} = \delta(q_i, a_{i+1})$ and $\sigma_{i+1}^k = \gamma_k(q_i, \sigma_i^k, a_{i+1})$. The *output* of A on input σ is sequence of actions $\sigma^o = \sigma_1^o; \sigma_2^o; \ldots$ such that $\sigma_{i+1}^o = \gamma_o(q_i, \sigma_i^k, a_{i+1})$.

We use a notation $q, \sigma \overset{a|\sigma^o}{\leadsto} q', \sigma'$ to represent one transition in the edit automaton from the state q with current memory σ on input action a. As a result, a new state is $q' = \delta(q, a)$, an output is $\sigma^o = \gamma_o(q, \sigma, a)$ and the updated memory is $\sigma' = \gamma_k(q, \sigma, a)$.

We will have several types of transitions in our automata:

a) $q, \sigma \overset{a|\sigma;b}{\leadsto} q', \cdot$ outputs the memory followed by some action b and empties it afterwards;

b) $q, \sigma \overset{a|\cdot}{\leadsto} q', \sigma; b$ adds some action b to the memory and does not output anything.

We use action b different from an input action a because a can potentially be a deviation and b can be its correction.

Soundness and transparency are two basic properties that were originally proposed in [3]. Soundness means that output of enforcement mechanism should be always good.

Definition 3.3: An enforcement mechanism E is *sound*

for a policy $\widehat{P}$ over action set Σ iff

$$\forall \sigma \in \Sigma^*. \, \widehat{P}(E(\sigma))$$

Transparency means that good executions should not be changed by the enforcement mechanism.

Definition 3.4: An enforcement mechanism E is *transparent* for a policy $\widehat{P}$ over the action set Σ iff

$$\forall \sigma \in \Sigma^*. \, \widehat{P}(\sigma) \Rightarrow E(\sigma) = \sigma$$

We also use an additional evaluation of an enforcement mechanism called *predictability*. This notion was proposed in [1] and means that every trace that is close to a good trace is mapped by an enforcement mechanism into a trace close to the same good trace. This notion is based on metrics and, among the possible metrics discussed in [1], we will use one which has been used for dictionary searches. However, we first formally introduce the concept of predictability.

Definition 3.5: An enforcement mechanism E is *predictable within* ε if for every trace $\sigma_P \in P$ and every $\nu \geq \varepsilon$, there exists a $\exists \delta > 0$ such that for all $\sigma \in \Sigma^*$ the following holds:

$$d(\sigma, \sigma_P) \leq \delta \Rightarrow d'(E(\sigma), E(\sigma_P)) \leq \nu$$

Informally, it says that for every good trace there always exists a radius δ, such that all the traces within this radius are mapped into the circle with radius ε from this trace. In this definition d and d' are some metrics.

Definition 3.6: A *metric*[3] on a set S is a function $d : S \times S \to \mathbb{R} \cup \{\infty\}$ such that

a) $d(\sigma, \tau) \geq 0$,
b) $d(\sigma, \tau) = 0$ if and only if $\sigma = \tau$,
c) symmetry: $d(\sigma, \tau) = d(\tau, \sigma)$,
d) triangular inequality: $d(\sigma, \tau) \leq d(\sigma, \sigma') + d(\sigma', \tau)$.

We have already argued in [1] that the metric that counts the number of replaced actions is more palatable for the end users than, for example, the metric that counts insertions.

Our end users insist that there is only one official policy (the simple default one). An enforcement mechanism could be entitled only to correcte small errors without changing the protocol used by the operators (such as patient identification, patient consent and blood sampling before blood transfusion). If the doctor forgot to fill one field in the form, the mechanism can help her by inserting a default value. On the other hand, insertions of new steps by the monitor to compensate a bad event are guarded with suspicion because a different protocol might have different medical or legal consequences and those can only be inserted by human who will be held accountable for unexpected consequences.

---

[3]Notions of metrics and metric spaces are adapted from [4], [5], [6].

Table III: Examples of errors and their characteristics

Deviation e	Expected action $ex(e)$	Correction $c(e)$
e	a	e
e	a	a
e	a	b

Definition 3.7: The *replacing distance* between two finite traces is a total function $d_R : \Sigma^* \times \Sigma^* \to \mathbb{N} \cup \{\infty\}$, s.t.

$$d_R(a\sigma, b\sigma') = \begin{cases} 0 & \text{if } a\sigma = \cdot \text{ and } b\sigma' = \cdot \\ \infty & \text{if } a\sigma = \cdot \text{ xor } b\sigma' = \cdot \\ d_R(\sigma, \sigma') & \text{if } a = b \\ 1 + d_R(\sigma, \sigma') & \text{if } a \neq b \end{cases}$$

Distance d_R counts the number of replacements to tell traces apart. If the traces have different length, the distance is ∞ (as they should belong to different protocols).

IV. Enforcement for Error-Tolerant Policies

We propose to distinguish between several types of errors: venial errors, some other group of errors that are possible to fix and fatal errors. An enforcement mechanism is able to fix or tolerate errors that are not fatal.

For every possible deviation/error from the legal behavior we propose the following table with the following functions. A function $ex(e)$ defines an expected action in the legal trace when an error e occurred and a function $c(e)$ defines a correction for an error e. Assuming that action e is an error and action a is an expected action of the policy, there are few alternatives summarized in Table III.

Correction $c(e) = e$ (line 1) means that e is a venial error and can be tolerated by the enforcement mechanism. A case when action a is an expected action and $c(e) = a$ (line 2) means that e is not a venial error, but it is possible to fix it to a legal action. The third case $c(e) = b$ (where $a \neq b$) means that e is not a venial error, but there is a possible correction that is by itself a venial error.

Let us come back to the running example from Section II. Table II contains traces that can be slightly changed and then be accepted by the end users as good traces. It can be done in two ways: by allowing a venial error to occur (meaning $c(a) = a$) and by correcting it ($c(a) \neq a$). Venial errors can occur instead of the action defined by the policy and are "harmless", so in our example venial error is to close the therapeutical notes window (Ctw) instead of reviewing the notes (Rtn), formally $ex(\mathsf{Ctw}) = \mathsf{Rtn}$ and $c(\mathsf{Ctw}) = \mathsf{Ctw}$. In sequence 1 of Table II the user makes this venial error and so our tolerant enforcement mechanism does not change the tentative execution of the user.

If an error is not venial, it should be corrected. We show such an example in sequence 2. Closing the protocol number window (Cpw) instead of inserting the research protocol number (Irpn) is an error that should be corrected, so we can replace this action by inserting the special number for the

Table II: Almost bad traces that can be corrected

No	Trace	Expected enforcement
1	Dis; Tnn; Ctw; Dr; Irpn; Dpres	Dis; Tnn; Ctw; Dr; Irpn; Dpres
2	Dis; Tnn; Rtn; Dr; Cpw; Dpres	Dis; Tnn; Rtn; Dr; InA; Dpres
3	Dis; Tnn; Ctw; Dr; Cpw; Dpres	Dis; Tnn; Ctw; Dr; InA; Dpres

audit (InA), formally $ex(\mathsf{Cpw}) = \mathsf{Irpn}$, $c(\mathsf{Cpw}) = \mathsf{InA}$ and InA is a venial error. When performing the reimbursement of the drug, this number will mean that the drug is for research but the protocol number inserted will be some default number. By doing so we assure that the drug reimbursement will be done by a correct party (clinical trial funds). Sequence 3 contains both types of errors: a venial error of closing the therapeutical notes window (Ctw) instead of reviewing these notes (Rtn) and an error of closing the protocol number window (Cpw) instead of inserting the research protocol number (Irpn). Both of the errors should be corrected.

We propose a metric based on the replacing distance that counts a number of venial errors.

Definition 4.1: The *replacing distance with venial errors* between two finite traces is a total function $d_R^v : \Sigma^* \times \Sigma^* \to \mathbb{N} \cup \{\infty\}$, such that

$$d_R^v(a\sigma, b\sigma') = \begin{cases} 0 & \text{if } a\sigma = \cdot \text{ and } b\sigma' = \cdot \\ d_R^v(\sigma, \sigma') & \text{if } a = b \\ 1 + d_R^v(\sigma, \sigma') & \text{if } a \neq b \text{ and } ex(a) = b \\ \infty & \text{otherwise} \end{cases}$$

Except for venial errors, and errors that can be corrected, there are other errors that we call *fatal*.

V. CONSTRUCTION

In this section we present a construction of an enforcement mechanism in a form of edit automaton. The input is a default security policy P, a maximum number of venial errors k and two functions: function ex that defines an expected action for a given bad action and function c that defines a correction of this action. In the rest of the paper we will denote the input by $\mathcal{I} = \langle P, k, c, ex \rangle$. The output is an edit automaton that transforms the bad executions of the system by tolerating up to k errors.

Assuming that a given policy P is represented as a Policy automaton $A^P = \langle \Sigma, Q^P, q_0^P, \delta^P, F^P \rangle$, we construct an edit automaton $E = \langle Q, q_0, \delta, \gamma_o, \gamma_k \rangle$. Its states have the form $(q, q_F)^{\sharp s, p}$, where q is a state of a Policy automaton, q_F is the last accepting state before reaching q in some run, s is a number of venial errors so far (we write s to denote the measure of soundness) and p is a number of corrections made to the original execution (we write p to denote the measure of precision).

The idea behind the following construction is that all the good executions are not changed and all the bad executions are corrected if they contain up to k venial errors. Executions that have more than k venial errors are not amendable, so they are halted as soon as $(k+1)$st error arrives.

The set of states of edit automaton is $Q = \{(q, q_F)^{\sharp s, p} | q \in Q^P, q_F \in F^P, 0 \le s \le k\} \cup \{(q_\perp, q_F)^{\sharp k, 0} | q_F \in F^P\}$ and an initial state is $q_0 = (q_0^P, q_0^P)^{\sharp 0, 0}$. We define the semantics of the enforcement mechanism in Figure 3. The transition of a Policy automaton is represented by $q \xrightarrow{a} q'$, and a transition of edit automaton is represented by $q, \sigma \overset{a|\sigma^o}{\rightsquigarrow} q', \sigma'$ (defined above). If a tentative execution corresponds to the policy the edit automaton "copies" the Policy automaton. If the sequence is accepted, it resets the counter of errors [GOOD-OUT]. If the sequence is not accepted, the automaton keeps counting the errors [GOOD-WAIT].

Otherwise a tentative execution does not correspond to the policy. So, we check whether more venial errors are allowed ($s < k$) and whether there exists a transition starting at the state q on the expected action $ex(a)$ that does not have to be corrected ($c(a) = a$ in [VENIAL-OUT] and [VENIAL-WAIT]). If such action has to be corrected, we use the $c(a)$ function to define an appropriate correction (rules [CORRECT-OUT] and [CORRECT-WAIT]). If none of the previous cases holds, we check whether this action can initiate a new good iteration from the last visited accepting state q_F (rules [ITERATION-OUT] and [ITERATION-WAIT]). If there is no new iteration, we reach an error state [ERROR].

Even though this construction seems to be standard, it automatically generates an enforcement mechanism from the policy P, number of errors k and functions that define expected actions and corrections.

Lemma 5.1: Given inputs $\mathcal{I} = \langle P, k, c, ex \rangle$, an enforcement mechanism E with the semantics from Figure 3 is transparent: $\forall \sigma \in \Sigma^* : \widehat{P}(\sigma) \Rightarrow E(\sigma) = \sigma$

Proof: A sequence σ satisfies the policy, hence there is an accepting run in the policy automaton A^P. Therefore, only rules [GOOD-WAIT] and [GOOD-OUT] are used and the whole sequence σ is kept in the memory and is output upon reaching an accepting state of the Policy automaton. Hence, E is transparent. ∎

Lemma 5.2: Given inputs $\mathcal{I} = \langle P, k, c, ex \rangle$, an enforcement mechanism E with the semantics from Figure 3 for every iteration $\sigma_P \in P$ is such that

$$\forall \sigma \in \Sigma^* : (d_R^v(\sigma, \sigma_P) \le k \Rightarrow d_R^v(E(\sigma), E(\sigma_P)) \le k)$$

Proof: From the definition of replacing distance with venial errors we have that $d_R^v(\sigma, \sigma_P) \le k$ means $|\sigma| = |\sigma_P|$ and among all indices $0 < i \le |\sigma|$ there are up to k indices $i_1, ... i_n$ ($n \le k$) such that for all $0 < j \le n$: $\sigma[i_j] \neq \sigma_P[i_j]$ and $ex(\sigma[i_j]) = \sigma_P[i_j]$. For all the other indices l we have $\sigma[l] = \sigma_P[l]$.

$$\text{GOOD-OUT}\ \frac{q \xrightarrow{a} q' \qquad q' \in F^P}{(q, q_F)^{\sharp s,p}, \sigma \overset{a|\sigma;a}{\rightsquigarrow} (q^P, q^P)^{\sharp 0,0}, \cdot} \qquad\qquad \text{GOOD-WAIT}\ \frac{q \xrightarrow{a} q' \qquad q' \notin F^P}{(q, q_F)^{\sharp s,p}, \sigma \overset{a|\cdot}{\rightsquigarrow} (q', q')^{\sharp s,p}, \sigma;a}$$

$$\text{VENIAL-OUT}\ \frac{q \xrightarrow{e} \bot \qquad q \xrightarrow{ex(e)} q' \qquad q' \in F^P \qquad c(e) = e \qquad s < k}{(q, q_F)^{\sharp s,p}, \sigma \overset{e|\sigma;e}{\rightsquigarrow} (q', q')^{\sharp 0,0}, \cdot}$$

$$\text{VENIAL-WAIT}\ \frac{q \xrightarrow{e} \bot \qquad q \xrightarrow{ex(e)} q' \qquad q' \notin F^P \qquad c(e) = e \qquad s < k}{(q, q_F)^{\sharp s,p}, \sigma \overset{e|\cdot}{\rightsquigarrow} (q', q')^{\sharp s+1,p}, \sigma;e}$$

$$\text{CORRECT-OUT}\ \frac{q \xrightarrow{e} \bot \qquad q \xrightarrow{ex(e)} q' \qquad q' \in F^P \qquad c(e) \neq e \qquad s < k}{(q, q_F)^{\sharp s,p}, \sigma \overset{e|\sigma;c(e)}{\rightsquigarrow} (q', q')^{\sharp 0,0}, \cdot}$$

$$\text{if } c(e) = ex(e) \text{ then } s' = s \text{ else } s' = s + 1$$

$$\text{CORRECT-WAIT}\ \frac{q \xrightarrow{e} \bot \qquad q \xrightarrow{ex(e)} q' \qquad q' \notin F^P \qquad c(e) \neq e \qquad s < k}{(q, q_F)^{\sharp s,p}, \sigma \overset{e|\cdot}{\rightsquigarrow} (q', q')^{\sharp s',p+1}, \sigma;c(e)}$$

$$\text{ITERATION-OUT}\ \frac{q \xrightarrow{a} \bot \quad q_F \xrightarrow{a} q' \quad q' \in F^P}{(q, q_F)^{\sharp s,p}, \sigma \overset{a|a}{\rightsquigarrow} (q', q')^{\sharp 0,0}, \cdot} \qquad \text{ITERATION-WAIT}\ \frac{q \xrightarrow{a} \bot \quad q_F \xrightarrow{a} q' \quad q' \notin F^P}{(q, q_F)^{\sharp s,p}, \sigma \overset{a|\cdot}{\rightsquigarrow} (q', q')^{\sharp 0,0}, a}$$

$$\text{ERROR}\ \frac{Otherwise}{(q, q_F)^{\sharp s,p}, \sigma \overset{e|\cdot}{\rightsquigarrow} (q_\bot, q_F)^{\sharp k,0}, \cdot}$$

Figure 3: Semantics for enforcement mechanism derived from the policy automaton.

Therefore, for the actions $\sigma[i_1], ..., \sigma[i_n]$ one of the rules [VENIAL-OUT], [VENIAL-WAIT], [CORRECT-OUT] or [CORRECT-WAIT] holds. Since $ex(\sigma[i_j]) = \sigma_P[i_j]$, there is always a transition on $ex(\sigma[i_j])$ in the Policy automaton and it's always true that $s < k$ since there are only up to k actions in σ that are different from σ_P.

For all the other indices l different from $i_1, ..., i_n$ we have $\sigma[l] = \sigma_P[l]$. Hence, only rules [GOOD-OUT] or [GOOD-WAIT] will be used.

So all the rules, where the action of σ is transformed to another action, will be applied up to k times. Therefore, for every iteration σ_P (that by definition brings a Policy automaton to an accepting state) every sequence σ such that $d_R^v(\sigma, \sigma_P) \leq k$ will be transformed to a sequence $E(\sigma)$, such that $d_R^v(\sigma, E(\sigma_P)) \leq k$. Then, according to Lemma 5.1, $E(\sigma_P) = \sigma_P$, therefore theorem is proven. $\blacksquare$

Example 3: Suppose $k = 2$. The sequence $\sigma = $ CloseNote = Dis; Tnn; Ctw; Dr; Irpn; Dpres contains only one venial error: it has a closing window action Ctw instead of reviewing therapeutical notes Rtn. We have defined that $ex(\text{Ctw}) = $ Rtn and $c(\text{Ctw}) = $ Ctw. Hence when we compare it to the good sequence $\sigma_P = $ NoteResearchRun = Dis; Tnn; Rtn; Dr; Irpn; Dpres, we have $d_R^v(\sigma, \sigma_P) = d_R^v(\text{CloseNote}, \text{NoteResearchRun}) = 1$. Now since the number of errors is less than $k = 2$ then by outputting the input sequence without changes we have $E(\text{CloseNote}) = \text{CloseNote}$ and $d_R^v(E(\sigma), E(\sigma_P)) = d_R(\text{CloseNote}, \text{NoteResearchRun}) = 1$.

Theorem 5.1: Given inputs $\mathcal{I} = \langle P, k, c, ex \rangle$, an enforcement mechanism E with the semantics from Figure 3

- is transparent: $\forall \sigma \in \Sigma^* : \widehat{P}(\sigma) \Rightarrow E(\sigma) = \sigma$,
- is predictable within k – for every iteration $\sigma_P \in P$: $\forall \nu \geq k : \exists \delta > 0 : \forall \sigma \in \Sigma^* : (d_R^v(\sigma, \sigma_P) \leq \delta \Rightarrow d_R^v(E(\sigma), E(\sigma_P)) \leq \nu)$

Proof: The given mechanism is transparent according to Lemma 5.1, it is predictable since for every $\nu \geq k$ there exists $\delta = k$ such that $\forall \sigma \in \Sigma^* : (d_R^v(\sigma, \sigma_P) \leq k \Rightarrow d_R^v(E(\sigma), E(\sigma_P)) \leq k \leq \nu)$ according to Lemma 5.2 $\blacksquare$

VI. Conclusions

Runtime enforcement is a common mechanism for ensuring that executions adhere to constraints specified by a security policy. It is based on two simple ideas: the enforcement mechanism should leave good executions without changes and make sure that the bad ones got amended. From the theory side, the characterization of enforcement mechanisms like security automata or edit automata has been provided [2], [7], [8]. However, most of the theories do not distinguish what happens when an execution is actually bad (the practical case). In our previous work [9] we proposed to suppress the bad parts of execution, however even this is not enough. From the practical side, as a rule the users mostly concerned about achieving their goals: successfully executing their processes with the least amount of interruptions possible.

To the best of our knowledge, the only *automatic* constructions of the enforcement mechanisms from the given policies provide the mechanisms that either stop the whole execution [2], [10] or delete a part of it [9], [11]. They can be too restrictive for the users. We need to deal with errors in a way that is more flexible than current methods but also more principled than just hacking the reaction to errors in the monitor's implementation. Another alternative would be automatic generation of the policy, as done in [12], but as we have explained this would not resonate with our users.

In this paper we proposed to address this problem by emphasizing a distinctive type of users' infringements: venial errors. These errors are simply tolerated in critical situations. There are other bad actions that can be fixed in real life, they are errors that can be corrected. There is always yet another kind of actions called observable actions, that cannot be changed but only observed.

In order to avoid what auditors call systematic errors we limit the number of errors that the enforcement mechanism can tolerate and automatically constructs a runtime enforcer for a given security policy that tolerates the given number of errors.

References

[1] N. Bielova and F. Massacci, "Predictability of enforcement," in *Proceedings of the International Symposium on Engineering Secure Software and Systems 2011*, 2011, to appear.

[2] J. Ligatti, L. Bauer, and D. Walker, "Run-time enforcement of nonsafety policies," *ACM Transactions on Information and System Security*, vol. 12, no. 3, pp. 1–41, 2009.

[3] L. Bauer, J. Ligatti, and D. Walker, "Edit automata: Enforcement mechanisms for run-time security policies," *International Journal of Information Security*, vol. 4, no. 1-2, pp. 2–16, 2005.

[4] A. Arkhangel'skii and L. Pontryagin, *General topology I : basic concepts and constructions, dimension theory*. Springer-Verlag, 1990.

[5] D. Cohn, *Measure Theory*. Birkhauser, 1980.

[6] S. Matthews, "Partial metric topology," in *Proceedings of the 8th Summer Conference, Queen's College*, vol. 728. Annals of the New York Academy of Sciences, 1994, pp. 183–197.

[7] K. W. Hamlen, G. Morrisett, and F. B. Schneider, "Computability classes for enforcement mechanisms," *ACM Transactions on Programming Languages and Systems*, vol. 28, no. 1, pp. 175–205, 2006.

[8] C. Talhi, N. Tawbi, and M. Debbabi, "Execution monitoring enforcement under memory-limitation constraints," *Information and Computation*, vol. 206, no. 2-4, pp. 158–184, 2007.

[9] N. Bielova, F. Massacci, and A. Micheletti, "Towards practical enforcement theories," in *Proceedings of The 14th Nordic Conference on Secure IT Systems*, ser. Lecture Notes in Computer Science, vol. 5838. Springer-Verlag Heidelberg, 2009, pp. 239–254.

[10] N. Bielova and F. Massacci, "Do you really mean what you actually enforced?" in *Proceedings of the 5th International Workshop on Formal Aspects in Security and Trust*, vol. 5491. Springer-Verlag Heidelberg, 2008, pp. 287–301.

[11] Y. Falcone, J.-C. Fernandez, and L. Mounier, "Synthesizing enforcement monitors wrt. the safety-progress classification of properties," in *Proceedings of the Fourth International Conference on Information Systems Security (ICISS'08)*. Springer-Verlag, 2008, pp. 41–55.

[12] C. Jason, G. Lewin, Y. Gottlieb, R. Chadha, S. Li, A. Poylisher, S. Newman, and R. Lo, "On automated policy generation for mobile ad hoc networks." IEEE Computer Society Press, 2007.