

Towards Practical Enforcement Theories: Extended Abstract*

Nataliia Bielova Fabio Massacci
DISI - University of Trento, Italy,
surname@disi.unitn.it

Abstract

Run-time enforcement is based on two simple ideas: the enforcement mechanism should leave good traces untouched and make sure bad traces got amended. From the theory side, this leads to a number of papers [3, 4, 6] characterizing precisely the nature of good traces that can be captured by a security policy and thus enforced by a specific mechanism. Unfortunately, those theories do not distinguish what happens when a trace actually turns out to be bad (the practical case). The theory only says that the outcome should be “good” but not how far should we change the original input.

In this paper we propose a notion of enforcement that revises the notion of good traces in terms of repeated transactions. We start discussing what happens to bad actions and how we can characterize an enforcement mechanism in terms of how it deals with the bad actions.

1 Motivation

We have been motivated by the real life example of the drug selection process, which is running in the hospital. For the sake of brevity, we simplify the original process. It starts as soon as drug is requested and its execution is acceptable if it has the structure (a) or (b): (a): Request Drug, Drug is for Research, Insert Research Protocol Number (RPN), Drug is Available; (b): Request Drug, Drug is Not for Research, Drug is Available. So every time when the drug is for research (the patient taking this drug is under some research medical program) the RPN must be inserted. Hence the execution (c): Request Drug, Drug is for Research, Drug is Available is not valid.

The classical enforcement mechanisms assume that as soon as some restricted operation happens, either the process must be immediately stopped [5] or the mechanism should wait until the process becomes valid again by itself [4]. We want to show that truncating the process or waiting until it becomes acceptable again is not always the best enforcement. For example, let’s assume the mechanism that stops the whole drug dispensation process because once the RPN was not inserted when the drug was for research. Then, no health personnel can do any actions for drug dispensation until this number is inserted. We believe that this behavior is not the one expected by the users of the system. Therefore, in this paper we want to define the right notion of enforcement, which is more expected by the users in case when the process execution does not correspond to the security policy and which kind of policies or properties we can actually enforce.

2 Repeatable property

There are several classes of properties mentioned in the classical security papers. The property “nothing bad ever happens” is called *safety* property and formally defined by Lamport [1]. Additional to safety properties, there are *liveness* properties that claim that “something good finally happens during any execution”. However, except for safety and liveness properties there are other more general properties, which allow executions to alternate between satisfying and violating security property. In [4] authors also present the notion of *renewal* property and every decidable renewal property can be enforced by Ligatti automata (a particular kind of edit automata [2]). These automata will always output only the longest valid prefix of the input and in this way enforce the renewal property.

*Research partly supported by the Project EU-FP7-IP-MASTER.

However, in the example of drug selection outputting the longest valid prefix is not always an expected correction. For example, the process execution is (b) followed by (c) followed by (b). Obviously, the user would expect a system that is able to enforce his properties to allow the first (b) to execute, then delete the (c) invalid part and allow another (b) part to execute as well. By this example we show that there is a class of properties, for which corresponding set of traces consists of repeatable independent executions ((b) is good, (c) is bad, (b) is good). Therefore we define this kind of properties as *repeatable properties* (we assume that empty sequence is always valid: $\hat{P}(\epsilon)$). By \mathcal{A} we denote the set of observable actions, \mathcal{A}^* is a set of all finite sequences, \hat{P} is a predicate that defines whether the sequence valid or invalid.

Definition 2.1 Property \hat{P} is a repeatable property if and only if the following holds:

$$\forall \sigma, \sigma' \in \mathcal{A}^*. \hat{P}(\sigma) \wedge \hat{P}(\sigma') \implies \hat{P}(\sigma; \sigma') \quad (1)$$

Considering this definition of the repeatable property we also have shown the relation to the renewal, liveness and safety properties.

3 Repeatable enforcement mechanism

Traditionally an enforcement mechanism should satisfy (at least) two properties: soundness and transparency. Soundness says that all the output of the enforcement mechanism should be legal. An enforcement mechanism obeys transparency if all the good sequences are output without changes. Enforcement mechanism provides “effective=enforcement” [4] if it obeys the properties of soundness and transparency. The definition of effective=enforcement is enough if we deal only with valid input. However, in many practical cases this is highly unsatisfactory: what distinguishes an enforcement mechanism is not what happens when traces are good (nothing should happen) but the precise detail of how bad traces are converted into good ones. To this extent soundness only says they should be made legal. However, none of our references from the hospital will consider an “effective=enforcement” mechanism one that will stop drug selection process because the doctor failed to insert the research protocol number. Therefore, sequences are not wholly good or wholly bad. They are composed by fragments that can be good or bad. The removal of a bad fragment from an otherwise good sequence can make it good.

First we define the type of enforcement made by a Ligatti automaton. In [2] it was proved that Ligatti automaton provides some particular type of effective enforcement that always outputs the longest valid prefix. Then we propose the notion of *repeatable enforcement* that is an extension of the Ligatti automaton enforcement capabilities. The major difference is enforcement of bad sequences. Repeatable enforcement mechanism is *able to remove the bad parts* of the sequence, and this is possible for the repeatable property, where the parts of the sequence are independent.

In our work we have defined the notion of repeatable property and showed its relation to the classical safety, liveness and renewal properties. We have also defined the repeatable enforcement which is more expected by the users than longest valid prefix enforcement and also proposed the algorithm to construct the edit automaton that can actually provide this enforcement for given repeatable property.

References

- [1] M. W. Alford, J.-P. Ansart, G. Hommel, L. Lamport, B. Liskov, G. P. Mullery, and F. B. Schneider. Distributed systems: Methods and tools for specification, an advanced course. In *Advanced Course: Distributed Systems*, volume 190 of *Lecture Notes in Computer Science*. Springer, 1985.
- [2] 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*, volume 5491, pages 287–301. Springer-Verlag Heidelberg, 2008.
- [3] K.W. Hamlen, G. Morrisett, and F.B. Schneider. Certified in-lined reference monitoring on .net. In *Proceedings of the 2006 workshop on Programming Language and analysis for security*, pages 7–16. ACM Press, 2006.
- [4] J. Ligatti, L. Bauer, and D. Walker. Run-time enforcement of nonsafety policies. *ACM Transactions on Information and System Security*, 12(3):1–41, 2009.
- [5] F.B. Schneider. Enforceable security policies. *ACM Transactions on Information and System Security*, 3(1):30–50, 2000.
- [6] C. Talhi, N. Tawbi, and M. Debbabi. Execution monitoring enforcement under memory-limitation constraints. *Information and Computation*, 206(2-4):158–184, 2007.