The X-CREATE framework: a comparison of XACML policy testing strategies

Antonia Bertolino, Said Daoudagh, Francesca Lonetti and Eda Marchetti
Istituto di Scienza e Tecnologie dell'Informazione "A. Faedo", CNR
via G. Moruzzi, 1 - 56124 Pisa, Italy
{*firstname.lastname*}*@isti.cnr.it*

Keywords: XACML; policy testing; XACML requests derivation.

Abstract: The specification of access control policies with the XACML language could be an error prone process, so a testing is usually the solution for increasing the confidence on the policy itself. In this paper, we compare two methodologies for deriving test cases for policy testing, i.e. XACML requests, that are implemented in the X-CREATE tool. We consider a simple combinatorial strategy and a XML-based approach (XPT) which exploit policy values and the XACML Context Schema. A stopping criterion for the test cases generation is also provided and used for the comparison of the strategies in terms of fault detection effectiveness.

1 Introduction

XACML has become the de facto standard for specifying policies for access control decisions in many application domains such as Service Oriented Architectures (SOAs) and Peer-to-Peer (P2P) systems. Access control policies need to be carefully designed and tested to protect data from unauthorized access. A common approach for testing XACML policies is the derivation of test inputs (XACML requests) that are used for probing the XACML policy implementation engine, called the PDP (Policy Decision Point), and checking the PDP's responses against the expected ones. However the XACML requests generation is a particular process that could require a quite big effort to be manually managed, due to the complexity of XACML requests.

In (Bertolino et al., 2010) we proposed a *XPT* testing strategy for deriving test inputs. It exploits the XACML Context Schema representing the format of the XACML requests and the policy values combinations defining the XACML policy functionalities. We observed a higher or similar effectiveness of the proposed approach with respect to that of the existing ones. This is due to the higher structural variability of the derived requests by the *XPT* strategy. This higher variability is also a limitation of the *XPT* strategy since it sets a too high upper bound to the test set. Thus the idea of this paper: the definition of a more effective stopping criterion in the automated XACML requests generation based on the coverage of the input domain of a XACML policy. For this we limit the number of generated requests to the number of possible combinations of the values of the subject, resource, action and environment of the XACML policy. In (Bertolino et al., 2012) we detailed a simple testing strategy, called *Simple Combinatorial*, targeting the proposed stopping criterion. In this paper we present a comparison of its effectiveness with that of the *XPT* testing strategy proposed in (Bertolino et al., 2010). Experimental results confirm that the higher structural variability of the derived requests improves the effectiveness of the *XPT* strategy and that the coverage of the input domain of a XACML policy represents a suitable upper bound for the stopping criterion of the *XPT* strategy and the *Simple Combinatorial* one.

The rest of this paper is structured as follows. Section 2 briefly presents the background and some similar works. Section 3 illustrates the motivations of the proposed approach. In Section 4 we provide a brief description of the two strategies. Section 5 reports the experimental results. Finally, Section 6 concludes the paper and gives related discussions.

2 Background and Related work

XACML (OASIS, 2005) is a platform-independent XML based standard language designed by the Organization for the Advancement of Structured Information Standards (OASIS). The root of

all XACML policies is a *Policy* or a *PolicySet*. A *PolicySet* can contain other *Policies* or *PolicySets*. A *Policy* consists of a *Target*, a set of *Rules* and a *Rule combining algorithm*. The *Target* specifies the *Subjects*, the *Resources*, the *Actions* and the *Environments* on which a policy can be applied. If a request satisfies the target of the policy, then the set of rules of the policy is checked, otherwise the policy is skipped without examining its rules. A *Rule* is the basic element of a policy. It is composed by a *Target*, that is similar to the policy target and specifies the constraints of the requests to which the rule is applicable. The heart of most rules is a *Condition* that is a boolean function evaluated when the rule is applicable to a request. The result of the condition evaluation is the rule effect (*Permit* or *Deny*) if the condition is evaluated to be true, *NotApplicable* otherwise. If an error occurs during the application of a policy to the request, *Indeterminate* is returned as decision. More than one rule in a policy may be applicable to a given request. The *rule combining algorithm* specifies the approach to be adopted to compute the decision result of a policy containing rules with conflicting effects. The access decision is given by considering all attribute values describing the subjects, the resources, the actions and the environments of an access request and comparing them with the attribute values of a policy.

Some existing approaches consider the policy values in the test cases derivation. In particular, (Martin and Xie, 2006) presents the Targen tool that derives the set of requests satisfying all the possible combinations of truth values of the attribute id-value pairs found in the subject, resource, and action sections of each target included in the policy under test. A testing strategy implemented into X-CREATE framework (Bertolino et al., 2010) exploits the potentiality of the XACML Context schema defining the format of the test inputs, and also applies combinatorial approaches to the policy values. A different approach is provided by Cirg (Martin and Xie, 2007a) that is able to exploit change-impact analysis for test cases generation starting from policies specification. In particular, it integrates the Margrave tool (Fisler et al., 2005) which performs change-impact analysis so to reach high policy structural coverage. Other approaches for policy testing are based on representation of policy implied behavior by means of models (Traon et al., 2007). Usually these approaches provide methodologies or tools for automatically generating abstract test cases that have to be then refined into concrete requests for being executed.

A different research direction focuses on the development of engines for testing XACML policy. In particular, the authors of (Liu et al., 2011) propose a XACML Policy Evaluation Engine that is faster and more scalable than commonly used Sun PDP (Sun Microsystems, 2006).

3 Motivation and Research Questions

The testing scenario considered in this paper is focused on XACML policies and is represented in Figure 1. We suppose that the policy developer needs to check the correctness of a policy specification. The actors of this scenario include: a Test Generator, which takes as input the XACML Context Schema and the Policy specification and applies the set of available test strategies to generate XACML requests; a PDP which executes the requests and provides the responses, and the Policy developer which collects the responses and checks their correctness. In this scenario the PDP is considered correct, as highlighted by the cockade, and the SUT is the policy specification.

In (Bertolino et al., 2010) we discussed about the effectiveness of a test strategy based on a XML-based methodology, called *XPT*, in terms of fault detection. In particular, we provided a comparison of the performance of the *XPT* testing strategy with that of the Targen tool (Martin and Xie, 2006) that represents the most similar existing approach to the *XPT* testing strategy. Observing the obtained results we deduced that the effectiveness of the *XPT* testing strategy is comparable with, or higher than, that provided by the test suites derived by Targen tool.

As evidenced by the analysis of the results of that experiment, the most important advantage of the XPT-based testing is the structure variability of the derived requests: i.e., a request may include more than one subject or resource or action, or environment entity. This feature is especially important for testing policies or rules in which the access decision involves simultaneously more than one subject or resource or action or environment. However, the possibility of having a high variability in the structure of requests is also the main limitation of the *XPT* testing strategy. As described in (Bertolino et al., 2010), the total amount of different requests that could be generated by varying the structure only of the XACML Context Schema was MAXREQ = 118098 [1], which is extremely high for any kind of policy specification. Even if in the *XPT* testing strategy are considered some n-wise approaches for ordering and selecting the instances that maximize the fault detection ca-

[1] This number refers to XACML 2.0 Context Schema.

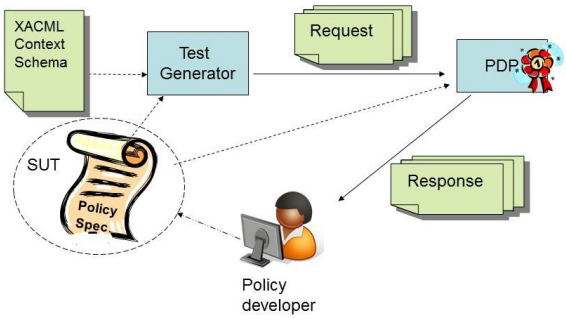

Figure 1: Testing Scenario

pability, a generic stopping criterion is missing. In particular, once the intermediate requests are generated, by construction they are filled with values taken from the tested policy, but no guarantee is provided that the values of the subjects, resources and actions of the policy are all covered if an arbitrary number of requests is derived.

For this, we introduce in this paper the following stopping criterion: generate as many requests as the number of possible combinations of the values of the subjects, resources, actions and environment of the XACML policy. This criterion focuses on the coverage of the input domain of a XACML policy since the policy input domain is represented by the combinations of its values. Following this new stopping criterion, as natural consequence, we defined a new strategy (Bertolino et al., 2012) for requests generation: derive a request for each combination of the subject, resource, action and environment values of the policy. We called this new strategy *Simple Combinatorial*.

We compared the effectiveness of the *Simple Combinatorial* with that of the *XPT* testing strategy by answering to the following research questions:

TSEff: Adopting the proposed stopping criterion, is the fault detection of the *Simple Combinatorial* strategy similar to that of the *XPT-based* one?

TSDecr: Is it possible to reduce the test suites maintaining the same level of fault detection?

We provide in Section 5 an experimental evaluation and discuss about the obtained results.

4 Testing strategies comparison

For aim of completeness, in this section we provide a brief description of the two testing strategies remarking their main advantages and limitations.

4.1 *XPT* strategy

The *XPT* strategy as presented in (Bertolino et al., 2010) consists of three main steps: intermediate-request generation; policy-under-test analysis; request values assignment.

In the *XPT* strategy, given the XACML Context Schema, a set of conforming XML instances is generated by applying a variant of the Category Partition (CP) method (Ostrand and Balcer, 1988) and traditional boundary conditions. In particular, the occurrences declared for each element in the schema are analyzed and, applying a boundary condition strategy, the border values (`minOccurs` and `maxOccurs`) to be considered for the instances generation are derived.

Combining the occurrence values assigned to each element, the set of intermediate instances are generated. In such manner, given the Context Schema, up to $3^Y * 2^Z$ intermediate instances can be derived, where Y is the number of schema elements with unbounded cardinality, and Z is the number of elements having [0,1] cardinality.

In particular, in the XACML 2.0 Context Schema, only for the part concerning the requests specification, there are 10 elements with unbounded occurrence and 1 having [0,1] cardinality. The other elements have cardinality 1. Thus, the application of *XPT* to the XACML 2.0 Context Schema (only for the part concerning the requests specification), will generate a maximum number of $3^{10} * 2^1 = 118098$ structurally different intermediate requests. This huge number of generated intermediate requests is obviously unmanageable for testing purposes. X-CREATE gives to the tester the possibility to choose the generation of a lower number of requests. Those requests are selected starting by a set of intermediate instances, derived by applying the well-known pair-wise approach (Cohen et al., 1997), that has been shown to be effective in picking a test subset with good fault detection capability.

Once derived the set of intermediate requests, the values of elements and attributes of the policy under test are used so to obtain executable and meaningful final requests. In particular *subject entities*, *resource entities*, *action entities* and *environment entities* are generated (for more details about the entities generation procedure see (Bertolino et al., 2010)) and they are used for filling the intermediate requests.

The combinations of *subject entities*, *resource entities*, *action entities* and *environment entities* are used to fill the values of elements and attributes of the subject, resource, action and environment of an intermediate instance respectively. Specifically, depending on the number of intermediate requests to complete,

the values entities are selected by applying an incremental combination approach (Cohen et al., 1997; Pretschner et al., 2008) so that all the possible combinations of the values of *subject entities*, *resource entities*, *action entities* and *environment entities* can be considered.

Taking then one by one the combinations of entities values the set of the intermediate requests is completely filled according to the following considerations:

- take the values defined for *subject entity*, *resource entity*, *action entity* and *environment entity* of each combination and use them to fill the values of elements and attributes of the subject, resource, action and environment of an intermediate instance respectively;
- if all the combinations of *subject entity*, *resource entity*, *action entity* and *environment entity* have been already used for filling a subset of intermediate requests, then start again by selecting the *subject entity*, *resource entity*, *action entity* and *environment entity* combinations in the same order till the completion of the intermediate requests set;
- in the case in which the structure of an intermediate request requires more than one entity in the subject, resource, action and environment, then the necessary entities are randomly taken from the available ones avoiding duplicates.

4.2 Simple combinatorial strategy

In this section we describe the *Simple Combinatorial* (Bertolino et al., 2012) strategy that has been implemented into the X-CREATE framework.

In this strategy, we apply a combinatorial approach to the policy values. We define the SubjectSet, ResourceSet, ActionSet, and EnvironmentSet sets as described in (Bertolino et al., 2010), considering also random entities for robustness and negative testing purposes. In particular, we define a *subject entity* as a combination of the values of elements and attributes of the SubjectSet set, and similarly the *resource entity*, the *action entity* and the *environment entity* as a combination of the values of the elements and attributes of the ResourceSet, ActionSet, and EnvironmentSet respectively.

Then, we generate all combinations of subject entities, resource entities, action entities and environment entities contained in these sets in the following way:

- First, we apply the pair-wise combination and we obtain the *PW* set
- Then, we apply the three-wise combination and we obtain the *TW* set
- Finally, we apply the four-wise combination and we obtain the *FW* set

These sets have the following inclusion propriety $PW \subseteq TW \subseteq FW$.

For eliminating duplicated combinations we consider the following set of combinations: *PW* called *Pairwise*, $TW \setminus PW$ called *Threewise* and $FW \setminus (TW \cup PW)$ called *Fourwise*. For each combination included in the above sets, we generate a simple request containing the entities of that combination. The derived requests are first those obtained using the combinations of the *Pairwise* set, then those ones using the combinations of the *Threewise* set and finally those using the combinations of the *Fourwise* set. In this way, we try to generate a test suite guaranteeing a coverage first of all pairs, then of all triples and finally of all quadruples of values entities derived by the policy. The maximum number of requests derived by this strategy is equal to the cardinality of the *FW* set. This number represents the stopping criterion for the proposed testing strategy. However, by the X-CREATE framework the user can choose a number of requests lower than the maximum one. The main advantage of the proposed strategy is that it is simple and achieves the coverage of the policy input domain represented by the policy values combinations.

5 Experimental Results

In this section, we discuss about the effectiveness of the *Simple Combinatorial* strategy and the *XPT* one in terms of fault-detection capability. The test suites of the two test strategies were derived by the tool X-CREATE. For the comparison we used (see column 1 in Table 1) three policies presented in (Martin and Xie, 2006) (specifically *demo-5*, *demo-11*, *demo-26*) and some real policies taken from the employability and health care services implemented in the EC FP7 TAS3 project (TAS3 Project, 2011).

We applied mutation analysis (DeMillo et al., 1978) to introduce faults into the policies and consequently assess the quality of a test suite in terms of fault detection. We generated the mutants set using the mutation operators for XACML policies indicated in (Martin and Xie, 2007b) that include: insert syntactic faults into the policy and rule target elements and condition elements; changing logical constructs; emulate semantic faults.

Table 1 in the second column, shows the total amount of mutants obtained for the policies. The

Table 1: Mutant-kill ratios achieved by test suites of *Simple Combinatorial* and *XPT*

		Simple Combinatorial				**XPT**			
		TSEff		TSDecr		TSEff		TSDecr	
policy	# Mut	# Req	Mut Kill %	# Req	Mut Kill %	# Req	Mut Kill %	# Req	Mut Kill %
demo-5	23	84	100 %	56	100 %	84	95.65 %	35	95.65 %
demo-11	22	40	95.45 %	13	95.45 %	40	95.45 %	18	95.45 %
demo-26	17	16	58.82 %	6	58.82 %	16	94.11 %	9	94.11 %
read-patient	25	30	40 %	2	40 %	30	40 %	13	40 %
university-admin-1	20	80	50 %	2	50 %	80	50 %	20	50 %
student-application-2	15	32	60 %	2	60 %	32	6 %	5	6 %
university-admin-3	20	76	50 %	2	50%	76	50 %	22	50 %

mutants have been used for answering the Research Questions of Section 3 and labeled **TSEff** and **TSDecr**. Experimental answers to these questions are described below.

RQ TSEff. Applying the *Simple Combinatorial* methodology we obtain seven test suites having cardinality as reported in Table 1 third column. For a fair comparison we generated the same number of requests for each policy also using the *XPT* test strategy. All the test suites obtained by the application of *Simple Combinatorial* and *XPT* one have been executed on the associated policies and on their mutants. The Sun XACML engine has been used for the experiment (Sun Microsystems, 2006). For each request, if the responses obtained by the execution of the request on the policy and one of its mutants are different, then the mutant policy is considered killed. For each policy under test, Table 1 reports the percentage of mutants killed using *Simple Combinatorial* strategy (4th column), and *XPT* strategy (8th column).

Observing the obtained results we can deduce that the effectiveness of the *XPT* derived test suites is similar to that of the test suites derived by the *Simple Combinatorial* strategy. Note that there are cases in which *XPT* has a better (see *demo-26*) and a lower (see *student-application-2*) performance than the *Simple Combinatorial*. A deep analysis of these anomalous situations showed that the test suites derived by the *Simple Combinatorial* strategy were not able to detect situations where the access decision of the policy rules depends concurrently on the values of more than one subject or resource or action or environment entity (this is the better performance of *XPT* for *demo-26*). On the contrary by construction the request derived by the *Simple Combinatorial* strategy contains almost a subject, a resource, an action and an environment entity. This can be a point of strength when the policies are very simple and the satisfiability of the policy rules depends on the combinations of a single subject, resource, action and environment entity as in the case of *student-application-2*. In this case, the high variability of the *XPT* requests prevents to have a good performance of the *XPT* strategy if a stopping criterion is assumed. This is because having many intermediate requests, those that allow for a good fault detection go beyond the limit given by the stopping criterion of the *Simple Combinatorial* strategy.

RQ TSDecr. The next point we analyzed was the possibility of reducing the test suites of the two test strategies maintaining the same level of fault detection. Thus, wherever the test suite reduction was applicable, starting from the first request ahead, following the order in which the requests have been generated, we incrementally derived the score of mutants killed of the two test sets till we reached the values reported in the 4th and 8th columns of Table 1. The cardinality of the subsets are in 5th and 9th columns of the Table 1. As shown, for the *Simple Combinatorial* strategy the maximum reachable percentage is reached with a few number of requests for all policies, while for *XPT* strategy an higher number of requests is needed. Thus the considered stopping criterion is a good upper bound assuring a good fault detection effectiveness with a manageable low number of requests.

6 Conclusions and Discussion

In this paper we proposed a comparison of two testing strategies implemented into X-CREATE framework, named *XPT* strategy and *Simple Combinatorial*. In addition, we proposed a stopping criterion for the automated generation of the test inputs for the XACML policy, focused on the coverage of the policy input domain. The preliminary conclusions we can draw from this initial evaluation are:

- A good fault detection percentage of the *XPT* testing strategy due to the variability of the structures of the generated requests, which could result in improved effectiveness of the derived test suites when the satisfiability of the policy rules depends on more than one subject, resource, action and en-

vironment.

- It is possible to reduce the number of requests for both strategies keeping the same test effectiveness. That means that the introduced stopping criterion is a good upper bound. However, further criteria for test reduction could be conceived.
- The high variability of the *XPT* strategy can limit its performance when policies are very simple and the stopping criterion of *Simple Combinatorial* strategy is assumed. For this, it is needed further study for achieving a trade-off between the structure variability and the cardinality of the test suite. Preliminary results about this have been presented in (Bertolino et al., 2012).

Note that, the percentage of mutants killed by the test suite derived by *Simple Combinatorial* strategy is the maximum reachable. As it was conceived, it is not possible to include additional test cases to the test suite and consequently to get higher value of fault detection.

Of course such conclusions must be taken in light of the threats to validity of the performed experiment. We need to make larger experiments to generalize the statement as well as consider further mutation operators than those of (Martin and Xie, 2007b).

From the performed analysis we noticed an impact of the policy specification on the effectiveness of the derived test suite. Thus, we would like to investigate other methodologies for requests generation taking into account this.

In particular, a limitation of *Simple Combinatorial* strategy was that it is not able to detect situations where the satisfiability of the policy rules depends simultaneously on the values of more than one entity. We would like to force by construction the requests derived by this strategy to contain all the possible combinations of more than one subject, resource, action and environment entity. In this way, the number of requests increases exponentially and could be soon comparable to the maximum number of requests obtained by the *XPT* testing strategy, i.e MAXREQ introduced in Section 3.

As a future work, we plan to investigate about the comparison between *XPT* approach and this new test inputs derivation proposal in terms of fault detection effectiveness.

ACKNOWLEDGEMENTS

This work has been partially funded by the Network of Excellence on Engineering Secure Future Internet Software Services and Systems (NESSoS) FP7 Project contract n. 256980. We also thank the EC FP7 TAS^3 (Trusted Architecture for Securely Shared Services) project for providing us with XACML policies.

REFERENCES

Bertolino, A., Lonetti, F., Daoudagh, S., and Marchetti, E. (2012). Automatic XACML requests generation for policy testing. submitted to The Third International Workshop on Security Testing 2012.

Bertolino, A., Lonetti, F., and Marchetti, E. (2010). Systematic XACML Request Generation for Testing Purposes. In *Proc. of 36th EUROMICRO Conference on Software Engineering and Advanced Applications (SEAA)*, pages 3 –11.

Cohen, D. M., Dalal, S. R., Fredman, M. L., and Patton, G. C. (1997). The AETG system: An approach to testing based on combinatiorial design. *IEEE Trans. on Soft. Eng.*, 23(7):437–444.

DeMillo, R., Lipton, R., and Sayward, F. (1978). Hints on test data selection: Help for the practicing programmer. *Computer*, 11(4):34–41.

Fisler, K., Krishnamurthi, S., Meyerovich, L., and Tschantz, M. (2005). Verification and change-impact analysis of access-control policies. In *Proc. of ICSE*, pages 196–205.

Liu, A. X., Chen, F., Hwang, J., and Xie, T. (2011). Designing fast and scalable xacml policy evaluation engines. *IEEE Transactions on Computers*, 60(12):1802–1817.

Martin, E. and Xie, T. (2006). Automated test generation for access control policies. In *Supplemental Proc. of ISSRE*.

Martin, E. and Xie, T. (2007a). Automated test generation for access control policies via change-impact analysis. In *Proc. of Third International Workshop on Software Engineering for Secure Systems (SESS)*, pages 5–12.

Martin, E. and Xie, T. (2007b). A fault model and mutation testing of access control policies. In *Proc. of WWW*, pages 667–676.

OASIS (1 Feb 2005). eXtensible Access Control Markup Language (XACML) Version 2.0. http://docs.oasis-open.org/xacml/2.0/access_control-xacml-2.0-core-spec-os.pdf.

Ostrand, T. J. and Balcer, M. J. (1988). The category-partition method for specifying and generating functional tests. *Commun. ACM*, 31(6):676–686.

Pretschner, A., Mouelhi, T., and Le Traon, Y. (2008). Model-based tests for access control policies. In *Proc. of ICST*, pages 338–347.

Sun Microsystems (2006). Sun's XACML Implementation. http://sunxacml.sourceforge.net/.

TAS3 Project (2011). Trusted Architecture for Securely Shared Services. http://www.tas3.eu/.

Traon, Y., Mouelhi, T., and Baudry, B. (2007). Testing security policies: going beyond functional testing. In *Proc. of ISSRE*, pages 93–102.