Testing Access Control Policies against Intended Access Rights

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

ABSTRACT

Access Control Policies are used to specify who can access which resource under which conditions, and ensuring their correctness is vital to prevent security breaches. As access control policies can be complex and error-prone, we propose an original framework that supports the validation of the implemented policies (specified in the standard XACML notation) against the intended rights, which can be informally expressed, e.g. in tabular form. The framework relies on well-known software testing technology, such as mutation and combinatorial techniques. The paper presents the implemented environment and an application example.

CCS Concepts

•Security and privacy → Software security engineering; •Software and its engineering → Software testing and debugging;

Keywords

Access Control Rights; XACML Language; Software Testing

1. INTRODUCTION

Security is among the top most pressing concerns of both developers and consumers of modern software systems. In today's highly connected and pervasive software-intensive systems, preventing unauthorized, erroneous or even malicious usage of critical resources becomes imperative. Thus, secure software engineering relies on sophisticated control and protection mechanisms to ensure the proper functioning of the software system at each level and against any potential threat. Among security mechanisms, a critical role is played by access control systems, which aim at ensuring that: i) only the intended subjects can access the protected data; and ii) these subjects get only the permission levels required to accomplish their tasks and no more. To this purpose, access control systems involve three related activities, namely *identification* of the user or service requesting an access, *authentication* of the declared identity, usually by means of credentials, and finally *authorization* of authenticated users to perform a set of allowed operations. Authorization mechanisms are typically based on *access control policies* that rule the level of confidentiality of data, the procedures for managing data and resources, and the classification of resources into category sets yielding different security requirements. Access control policies must be specified and verified with great accuracy, as any error or overlook could result either in forbidding due access rights, or worse in authorizing accesses that should be denied, thus jeopardizing the security of the protected data.

XACML (eXtensible Access Control Markup Language) [15] is a widely used standard language for specifying access control policies. It is an XML-based language conceived with interoperability, extensibility, and distribution in mind, thus enabling the specification of very complex rules. However, such advantages are paid in terms of complexity and verbosity: writing XACML policies is hard and may be deceiving, as *inconsistencies could arise between the security requirements the policy authors intend to specify and those that the policies actually state.* Indeed, an XACML policy could hide some combinations of access rules that were not foreseen by the policy author and that establish access rights not reflecting the actually intended rights. This can easily happen for instance when some modifications are introduced in a long complex policy or when the policy is obtained from the integration of more policies coming from different organizations. For this reason the policy authors, or more in general any stakeholder using the policy (generically referred in the following as the policy validators), need to accurately test the access rights resulting from the coded XACML policies against the actual intended behavior in granting/denying accesses. These intents could be easily represented with a model or expressed into natural language or by means of a table specifying the access control rights.

This work shows how common software testing techniques can be effectively applied to the testing of access control policies. In particular, the main contribution of this work is a generic framework, called TXPAINT (Testing XACML Policy Against INTentions), for testing the compliance of an XACML policy to intended access rights or discovering

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from Permissions@acm.org.

SAC 2016, April 04-08, 2016, Pisa, Italy
©2016 ACM. ISBN 978-1-4503-3739-7/16/04…$15.00
DOI: http://dx.doi.org/10.1145/2851613.2851829

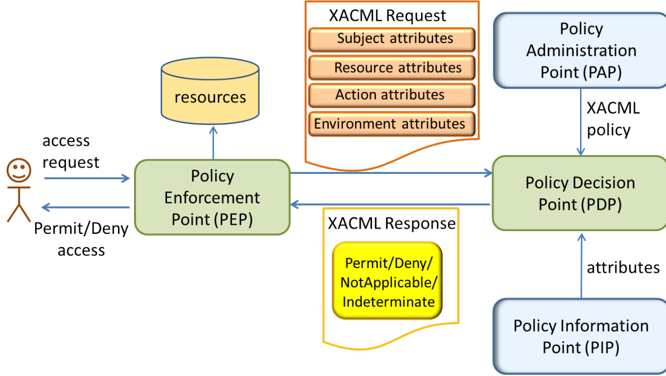

Figure 1: Access Control System Architecture

possible inconsistencies.

TXPAINT adopts two well-known testing techniques, i.e., combinatorial [13] and mutation testing [8], and provides support for generating appropriate test inputs (i.e., requests of access) able to test the constraints, permissions and prohibitions defined in the policy. The framework also provides support to locate the elements involved in the policy under test that are the causes of detected inconsistencies.

In the rest of the paper we first briefly introduce some needed background concepts, then in Section 3 we present an overview of the approach as well as its implementation. In Section 4 a case study illustrates how the approach is applied. Finally, the related work and conclusions sections conclude the paper.

2. BACKGROUND

The proposed framework relies on XACML policy and software testing techniques. In the following we provide a brief background to make the paper self-contained.

2.1 XACML based Access Control System

XACML [15] is a platform-independent XML-based language for the specification of access control policies. The main purpose of an XACML policy is to define the constraints that a subject needs to comply with for accessing a resource and doing an action in a given environment.

Briefly, an XACML policy has a tree structure whose main elements are: *PolicySet*, *Policy*, *Rule*, *Target* and *Condition*. The *PolicySet* includes one or more policies. A *Policy* contains a *Target* and one or more rules. The *Target* specifies a set of constraints on *attributes* of a given request. Typical categories of *attributes* are the *subject*, *resource*, *action* and *environment*, but users may define their own categories as needed. The *Rule* specifies a *Target* and a *Condition* containing one or more boolean functions. If the *Condition* evaluates to true, then the Rule's *Effect* (a value of *Permit* or *Deny*) is returned, otherwise a *NotApplicable* decision is formulated. If an error occurs during the evaluation of a policy against a request, *Indeterminate* is returned. The *PolicyCombiningAlgorithm* and the *RuleCombiningAlgorithm* define how to combine the results from multiple policies and rules respectively in order to derive a single access result. Figure 3(a) sketches an example of an XACML policy, whose details are illustrated later (Section *Application Example*). The main actors in the XACML domain are shown in Figure 1: the Policy Administration Point (PAP) is the system entity in charge of managing the policies; the Policy Enforcement Point (PEP), usually embedded into an application system, receives the access request in its native format, constructs an XACML request and sends it to the Policy Decision Point (PDP); the Policy Information Point (PIP) provides the PDP with the values of subject, resource, action and environment attributes; the PDP evaluates the policy against the request and returns the response, including the authorization decision, to the PEP.

2.2 Software Testing Techniques

In this section we present the testing techniques more closely related to our work.

In *Mutation Testing*, small syntactic faults, aimed at simulating true programmer's mistakes, are seeded in the original program code. A set of faulty programs, called mutants, is obtained, each containing one fault only. The main purpose of mutation testing is to assess the adequacy of a test suite. Each test case is executed on the original program and all its mutants. If the mutant's output is different from the original program's one, the mutant is said to be killed. We refer to [8] for an extensive survey of software mutation testing.

Combinatorial Testing aims at detecting failures triggered by the interactions among the input parameters of the software under test. Specifically, most failures are caused by the joint combinatorial effect of two factors, whereas the failures caused by the combination of three or more factors progressively decrease. In a combinatorial test plan, all combinations of parameters up to a certain level are considered. For instance, in pairwise testing, for every pair of parameters, every pair of values of these parameters must be tested at least once. The work in [13] provides a survey of combinatorial testing research.

Fuzz Testing involves generating random or semivalid data and submitting them in defined input fields or parameters (files, network protocols, API calls, and other targets) in an attempt to break the program and find bugs. Fuzzing is useful in identifying the presence of common vulnerabilities in data handling, and the results of fuzzing can be exploited by an attacker to crash or hijack the program using a vulnerable data field. We refer to [16] for a survey of fuzzing techniques and tools.

3. TESTING FRAMEWORK

The specification of XACML policies is a challenging task. Although some tools exist that help the user to check the syntactic correctness of the defined policy with respect to the standard XACML context [17], to the best of our knowledge the only facility that allows the partial verification of access control policy properties against a formal access control policy model is ACPT [7, 14]. However, there is no comprehensive facility that allows a policy author or validator to check the compliance of an XACML policy against the intended access control rights from which the policy originates. To address this issue, we introduce TXPAINT, a generic and

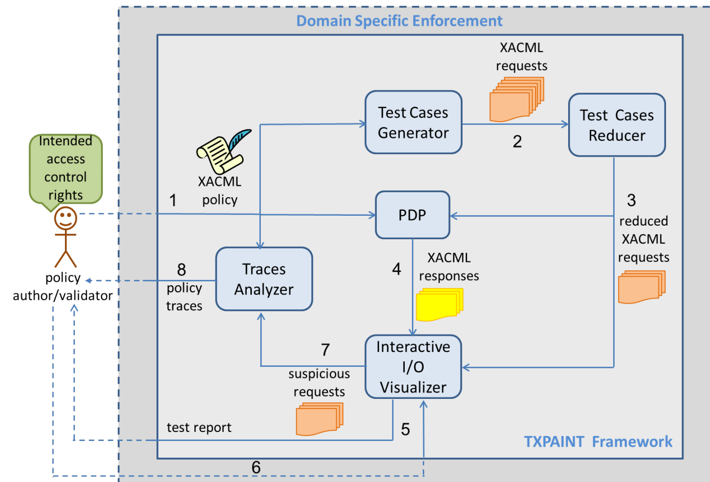

Figure 2: Overview of the approach

extensible framework supporting users during the testing of XACML policies against the intended access control rights.

Figure 2 presents an overview of the approach. The first step (in the left part of the figure) is the definition of the intended access rights. These could have different degrees of formalization: from a high-level formal model to an informal list of intended rights, expressed for instance in a table containing the access constraints. Such access control rights are translated into an XACML policy (step 1 of Figure 2). Depending on the formalization used, the implementation of this step could vary from a manual derivation, to the application of automated model2code transformation facilities. The translation of the access control intents into an XACML policy is part of the `Domain Specific Enforcement` (the dark grey part of Figure 2) and is out of the scope of the proposed testing framework (the light grey part of the same figure). The derived XACML policy is the processable input of TXPAINT and is used by the `Test Case Generator` component to derive a set of test cases (called XACML requests, step 2 in the figure). For this, different combinatorial strategies can be used, focused on the testing of functional aspects, constraints, permissions and prohibitions of the policy. In specific circumstances, for instance in case of regression testing or for timing/effort constraints, the number of the derived test cases should be reduced so to obtain a manageable and effective test suite. Different criteria for test cases reduction, such as fault detection effectiveness or filtering by one or more request values, are provided by the `Test Case Reducer` component.

The (reduced) test suite (step 3 of Figure 2) is then executed on the `PDP` component that derives the XACML response associated to each request (step 4 of Figure 2). The request/response pairs obtained by the XACML policy execution are collected and visualized in tabular form by the `Interactive I/O Visualizer` component. Eventually, the `Domain Specific Enforcement` is in charge of the (manual or automated) translation of the derived request/response pairs back into the same language used for the specification of the access control intents.

The request/response pairs are represented into a test report visualized by the framework (step 5 of Figure 2), which provides a valid help for checking the compliance of the XACML policy against the intended access control rights and highlighting possible inconsistencies. In case of inconsistencies, it is possible to select the subset of XACML requests that originated the inconsistent request/response pairs, called *suspicious requests* in Figure 2 (step 6, 7), and ask TXPAINT to locate the policy elements that caused a PDP response different from the expected one. This task is performed by the `Traces Analyzer` component, which evaluates the XACML policy against a given request and visualizes the sequence of policy elements and values (named policy traces in Figure 2, step 8) involved in the policy evaluation. Using the visualized trace, the policy author/validator can more easily identify the needed corrections to the XACML policy or even revise the original access control intents.

TXPAINT also provides regression testing facilities in order to assess that possible modifications to the intended access control rights and/or to the derived XACML policy do not invalidate other access control intents not addressed by the change. In particular, a previously derived test suite can be reduced by means of the `Test Case Reducer` component, and re-executed on the new derived XACML policy.

We have developed a reference implementation of the above described conceptual framework [1]. The components are described in the following. For the PDP component we rely on Balana XACML Implementation [2].

Test Cases Generator.

The *Test Case Generator* component proposed in TXPAINT is based on a combinatorial technique taken from the literature [2]. Specifically, it derives an XACML request for each of the possible combinations of the subject, resource, action and environment values taken from the policy. In addition, it

[1]The tool is available at http://labse.isti.cnr.it/tools/txpaint

[2]https://github.com/wso2/balana

allows the generation of XACML requests having more than one subject, resource, action and environment values. This latter feature specifically targets the policy rules in which the effect is simultaneously dependent on more than one constraint [2]. The number of subjects, resources, actions and environments of each request is automatically established according to the complexity of the policy structure. This component, similarly to fuzz testing, allows also to derive invalid data, namely combinations of subjects, resources, actions and environments that include random values or are not allowed by the policy.

Test Cases Reducer.

In case of a high number of policy values, the *Test Cases Generator* can derive a large set of requests. Consequently, the execution of the requests, the derivation and above all the analysis of the requests/responses pairs (i.e., the test oracle) could be a time and effort consuming task. To address this issue, the current implementation of this component provides two test cases reduction methodologies: a mutation based approach and a request/response values based filter. The first performs the following steps: i) derives a set of faulty policies (or mutants) taking into account common syntactic and semantic faults able to reveal inconsistencies between the XACML policy and the intended access control rights. The used mutation operators are those of [1, 11, 12]; ii) analyzes the results of test cases execution on the mutants themselves; and iii) applies a greedy heuristic for selecting the test cases. This heuristic guarantees that the reduced test suite keeps the same level of test effectiveness of the entire test suite or a user defined effectiveness. For implementing this mutation based approach, the *Test Cases Reducer* component integrates and adapts the XACMUT tool [1]. The main risk of mutation based test suite reduction is to exclude some requests that are redundant in terms of fault detection effectiveness, but that include combinations of subject, resource, action and environment values that could be useful to discover access control violations. The request/response values based filter selects the XACML requests according to a combination of the subjects, resources, actions, environments values and the expected authorization right. This reduction approach allows to focus the testing activity on the access control constraints that are considered the most critical, and excludes all the other ones. The risk of this reduction approach is to exclude some requests not involving the selected values combinations but that could be useful to discover access control violations. To mitigate the risks of either test cases reduction methodology, a combined use of them is supported and advised. Alternatively, test suite reduction approaches leveraging fast constraint solver as in [3] could also be integrated.

Interactive I/O Visualizer.

The *Interactive I/O Visualizer* represents the orchestrator of the proposed testing framework and provides the main interface for interacting with it. This component allows the policy tester to perform the following main steps: i) select an input XACML policy and the associated test suite if this has been already derived; ii) generate the XACML requests and get a visualization of their values (subject, resource, action and environment) ordered by the XACML request identifier; iii) reduce the test suite using either the mutation based approach, or the request/response values based filter, or both; iv) execute the XACML requests on the selected policy; v) get a visualization of the test report, namely the values of the executed requests (subject, resource, action and environment categories) and the associated authorizations rights, ordered by the XACML request identifier; vi) filter the executed requests according to either their values, or the associated authorization right, or a combination of them; vii) validate that the combination of subject, resource, action, environment values of the request and the associated authorization right is consistent with the access control intents associated to the XACML policy; viii) require the analysis of the XACML policy using as input the suspicious XACML requests generating the inconsistencies with the intended access control rights. This component represents a valid help for the analysis of the test results since by means of the filter facility it allows to split them into meaningful subsets. Indeed, having a view of the test results localized to specific combinations of values of the requests and the associated authorization rights allows the user to focus on only a few access control constraints at a time, and interactively validate all the results.

Traces Analyzer.

This component takes as input the XACML policy and the set of suspicious requests. For each suspicious request, it generates an XACML policy trace containing the policy elements (target, rules and combining algorithm) and the values that have been evaluated for deriving the associated authorization decision. Since the output of this component is specified in the XACML language, the use of the facility provided by this component is optional and suggested when the policy author has a background on the XACML formalism. To make the analysis of the traces by the side of the policy author/validator easier, the `Domain Specific Enforcement` could be in charge of translating (manually or automatically) them back into the same formalization adopted for the access control intents.

4. APPLICATION EXAMPLE

As an application example of the proposed testing framework we describe the steps performed for testing a real XACML policy against the associated access control intents, both released for the pilot application of the TAS3 project (www.tas3.eu). The access control intents, which are made publicly available from one academic partner of the project (the Nottingham University), specify in tabular form the access rights to a service, called MatchingService, by the side of the users having different roles in the University (student, placement coordinator and others). Specifically, as showed in Table 1, these intents specify that: i) the students, registered to only one course (Spanish, German, Italian or French), can access the functionalities of the MatchingService; ii) the students who are not registered to any course as well as the placement coordinator of the Nottingham University and all the other users cannot access the functionalities of the MatchingService. The implementation of the access rights

Table 1: Intended access rights

Subject	Action	Resource	Result
Nottingham/Student/Spanish	Get Match	MatchingService	Pass
Nottingham/Student/German	Get Match	MatchingService	Pass
Nottingham/Student/Italian	Get Match	MatchingService	Pass
Nottingham/Student/French	Get Match	MatchingService	Pass
Nottingham/Student	Get Match	MatchingService	Deny
Nottingham/Placement Coordinator	Get Match	MatchingService	Deny
All Others	GetMatch	MatchingService	Deny
Any Client	Any Other Actions	MatchingService	Deny

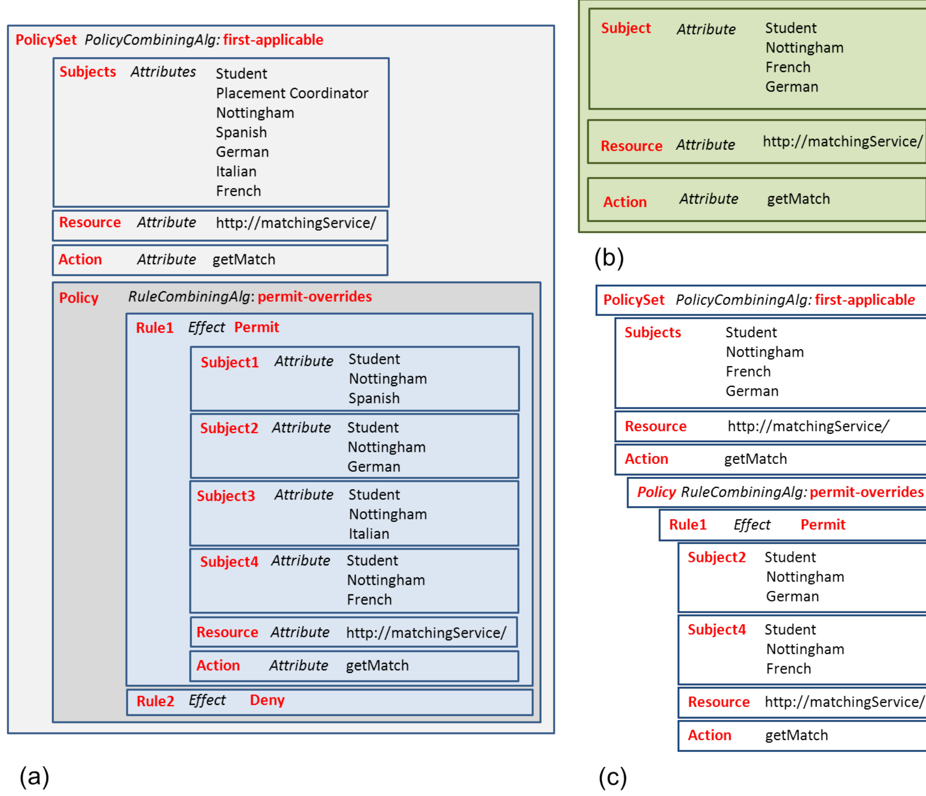

Figure 3: (a) An XACML policy and (b) a suspicious request and (c) a derived policy trace

of Table 1 is represented by the XACML policy sketched in Figure 3(a). This policy contains two rules: the former allows the access to the MatchingService by the side of students who are registered to either the Spanish, or the German, or the Italian, or the French course; the latter denies the access to all the other users.

We have used TXPAINT to check the compliance of the XACML policy of Figure 3(a) against the intended access rights of Table 1. The first step of this testing activity has been the generation of the XACML requests starting from the XACML policy. Specifically, 127 XACML requests have been derived using the combinatorial-based test cases generation strategy provided by the framework. Since the number of generated XACML requests was quite large to be manually checked against the access control intents associated to the policy, the original test suite has been reduced using the mutation-based approach. Specifically, using a greedy heuristic, the XACML requests able to guarantee the same level of test effectiveness of the original test suite (48 %) have been selected obtaining a reduced set of 31 XACML requests, corresponding to a reduction of 76% of the original test suite. The reduced XACML requests have been executed against the XACML policy on the XACML PDP, and the corresponding XACML responses containing the access results have been collected. To better manage the visualization of the values of the executed requests and the associated authorizations rights, a request/response values based filter has been applied. Specifically, to focus only on the permission rights, the executed requests have been filtered by *Permit* authorization value obtaining a set of 9 XACML requests. These requests and the associated permission rights represented the test report provided by TXPAINT. The values of the 9 requests and the associated permission rights

have been checked against those of the intended rights specified in Table 1, revealing an inconsistency in five requests that have been marked as suspicious.

In particular, the first suspicious request has an associated Permit right and contains a subject with the following values: *Student*, *Nottingham*, *French* and *German*. This means that an access request to the MatchingService by the side of a student who is registered to both *French* and *German* courses is allowed. This is not consistent with a strict interpretation of the access control intents of Table 1 specifying that in order to access the functionalities of the MatchingService a student must be registered to only one of those two courses. The other four suspicious requests have an associated Permit right and contain the following values: *Student*, *Nottingham*, *Placement Coordinator* and the name of a course (*German*, *Spanish*, *French* or *Italian*). This means that an access request to the MatchingService by the side of a user of the University, who has at the same time two roles, namely student (registered to a course) and placement coordinator is allowed. This is not consistent with the access control intents of Table 1 specifying that only the users of University who have the student role and are registered to only one course (Spanish, German, Italian or French) can access the functionalities of the MatchingService whereas the users of University who have the placement coordinator role are not allowed.

To better localize the parts of the XACML policy generating the inconsistencies described before, the five suspicious requests have been used for deriving five policy traces. As an example, the first suspicious request and the policy trace generated are showed in Figure 3(b) and (c) respectively. The presented trace shows the elements of the XACML policy and the associated subject, resource and action values that are involved in the evaluation of the first suspicious request. As showed in Figure 3(c), the first suspicious request triggers the evaluation of: i) the PolicySet and the associated combining algorithm; ii) the Policy and the associated combining algorithm; iii) the Rule1 and the associated Permit effect. The trace contains the values of subject, resource and action of the PolicySet and of Rule1 matching those of the request. The analysis of this trace evidenced that the request containing the values *Student*, *Nottingham*, *French* and *German* is able to match the values of 2 subjects of Rule1 (Subject 2 and Subject 4) representing a user registered to two different courses. As said, this part of the policy is inconsistent with the intended access rights. The table specifies that a student must be registered to one of the four courses in order to access to the MatchingService. To overcome this inconsistency a possible solution could be to add a uniqueness constraint on the course value in Rule1, thus forcing a Deny verdict if more than one subject matches. Otherwise, it is possible that the real intent was that a student must be registered to at least one course (and not only one), in which case the XACML policy is indeed correct and the policy validator could realize from the analysis of the trace that the table of intended access rights needs to be revised. Similarly, the analysis of the remaining traces evidenced the necessity of adding in the policy a uniqueness constraint also on the subject role value (student or placement coordinator).

5. RELATED WORK

In the context of access control systems, some proposals address mutation techniques to assess the fault detection effectiveness of test sets for security policies. The work in [11] defines a fault model and a set of mutation operators for XACML access control policies. The authors of [12] define a generic metamodel able to express various rule-based security policy formalisms (R-BAC, OrBAC), and introduce a set of mutation operators that can be applied to all rule-based formalisms. The mutation tool integrated in TXPAINT is that of [1] that includes and enhances the mutation operators of [11] and [12] addressing specific faults of the XACML language and providing derivation of XACML mutation operators and their application to XACML policies.

Concerning the testing of access control policies, combinatorial approaches have been proven to be effective in the automated generation of the test cases (access requests). The Targen tool [9] generates test inputs using combinatorial coverage of the truth values of independent clauses of XACML policy values. This combinatorial test derivation approach exhibits a better performance compared with a random test generation technique in terms of structural coverage of the policy and fault detection capability [9]. The test generation strategy integrated in TXPAINT relies on combinatorial approaches of the subject, resource, action and environment values taken from the XACML policy. The main advantage of this approach with respect to that of Targen is the higher structural variability of the derived test inputs able to guarantee the coverage of the input domain of the XACML policy.

Model-based testing has been applied to XACML policies [18]. The approach proposed in [18] focuses on the representation of policy implied behavior by means of models consisting of a role hierarchy, a permission hierarchy, and a context hierarchy, and on the adoption of combinatorial testing strategies for test cases derivation. Differently from our proposal, it focuses on automatic test code generation from the models for deriving test targets, i.e., combinations of roles, permissions and contexts.

More closely related works to our proposal rely on analysis of policy specifications aimed at verifying that they properly reflect some intended properties. They use different verification techniques, such as model-checking [19] or SAT solvers [6]. A tool for policy analysis is Margrave [4], which represents policies as Multi-Terminal Binary Decision Diagrams (MTBDDs) and can answer queries about policy properties. Cirg [10] tool integrates Margrave [4] and exploits change-impact analysis for test cases generation starting from the policy specification. Based on Margrave [4], the authors of [5] automatically synthesize concrete properties for static policy verification and perform conformance checking of an XACML policy against these properties. Finally, the Access Control Policy Tool (ACPT) [7] performs syntactic and semantic verification of access control policies and provides many functionalities including property checking for access control policy models through symbolic model checker, test suite generation based on combinatorial approaches and XACML policy generation as output of verified model.

All the above works dealing with policy verification aim to ascertain that the policies specifications are correct against some specified criterion. Differently from them, TXPAINT addresses a complementary problem, namely checking that the XACML policies (which are in principle assumed to be correct) are compliant with the intended access control rights, specified in a notation that is independent from any formal language adopted by the previous verification approaches. Therefore, an advantage of our proposal is that access rights can be also specified in a semi-structured natural language. The proposed framework leverages some XACML-based testing strategies for generating appropriate test cases which are able to test functional aspects, constraints, permissions and prohibitions of the XACML policy. The execution of these test cases and the associated results are exploited for checking the compliance of the XACML policy implied behavior with the intended access control rights.

6. CONCLUSIONS

We presented the TXPAINT framework that applies well-known software testing techniques for the purpose of validating the compliance of a coded XACML policy against the intended access rights. We have illustrated the application of the framework to a real-world case study that shows how accurate testing of security mechanisms is needed to gain confidence in the data protection solutions.

Concerning threats to validity of the presented experiment, three aspects can be considered: i) the used mutation operators; ii) the test set; iii) the used access control intents and the derived XACML policy. Because test reduction focuses on fault detection effectiveness, the set of utilized mutation operators may influence the reported results. It could be that a different choice of mutation operators might have provided different effectiveness results. To reduce this risk, the presented study employs different sets of mutation operators [1, 11, 12] addressing all the main faults of the XACML policy. Another threat of our proposal concerns the employed test set. We used that derived by the combinatorial approach of [2], but it is likely that other combinatorial-based test generation approaches may produce different results. However, [2] represents the current state of the art in XACML test generation tools. Finally, the presented results relate to a simple and real set of access control intents and their associated XACML policy, more complex XACML policies could reveal different results. However, we are confident that the framework yields the greatest benefits when applied for testing longer and complex access policies, as those that are obtained from the merging of several XACML specifications.

We are currently working to include in the framework more sophisticated debugging techniques as well as enhanced versioning and logging facilities for recording and comparing the execution results of different policy versions. We also plan to integrate in the framework further test suite reduction mechanisms based on XACML policy coverage criteria.

7. REFERENCES

[1] A. Bertolino, S. Daoudagh, F. Lonetti, and E. Marchetti. XACMUT: XACML 2.0 Mutants Generator. In *Proc. of 8th International Workshop on Mutation Analysis*, pages 28–33, 2013.

[2] A. Bertolino, S. Daoudagh, F. Lonetti, E. Marchetti, and L. Schilders. Automated testing of eXtensible Access Control Markup Language-based access control systems. *IET Software*, 7(4):203–212, 2013.

[3] J. Campos and R. Abreu. Leveraging a constraint solver for minimizing test suites. In *Proc. of QSIC*, pages 253–259, 2013.

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

[5] V. Hu, E. Martin, J. Hwang, and T. Xie. Conformance Checking of Access Control Policies Specified in XACML. In *Proc. of COMPSAC*, volume 2, pages 275–280, 2007.

[6] G. Hughes and T. Bultan. Automated verification of access control policies using a SAT solver. *Int. J. Softw. Tools Technol. Transf.*, 10:503–520, 2008.

[7] J. Hwang, T. Xie, V. Hu, and M. Altunay. ACPT: A Tool for Modeling and Verifying Access Control Policies. In *International Symposium on Policies for Distributed Systems and Networks*, pages 40–43, 2010.

[8] Y. Jia and M. Harman. An analysis and survey of the development of mutation testing. *IEEE Transactions on Software Engineering*, 37(5):649–678, 2011.

[9] E. Martin. Automated test generation for access control policies. In *Proc. of 21st SIGPLAN Symposium on Object-oriented programming systems, languages, and applications*, pages 752–753, 2006.

[10] E. Martin and T. Xie. Automated test generation for access control policies via change-impact analysis. In *Proc. of SESS*, pages 5–12, 2007.

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

[12] T. Mouelhi, F. Fleurey, and B. Baudry. A generic metamodel for security policies mutation. In *Proc. of ICST Workshop*, pages 278–286, 2008.

[13] C. Nie and H. Leung. A survey of combinatorial testing. *ACM Comput. Surv.*, 43(2):1–29, 2011.

[14] NIST. ACCESS CONTROL POLICY TOOL (ACPT). http://csrc.nist.gov/groups/SNS/acpt/, February 2015.

[15] OASIS. eXtensible Access Control Markup Language (XACML). http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-os-en.pdf, January 2013.

[16] M. Sutton, A. Greene, and P. Amini. *Fuzzing: brute force vulnerability discovery.* Pearson Education, 2007.

[17] Syncro Soft. oXygen XML editor, 2015.

[18] D. Xu, L. Thomas, M. Kent, T. Mouelhi, and Y. Le Traon. A Model-based Approach to Automated Testing of Access Control Policies. In *Proc. of the 17th SACMAT*, pages 209–218, 2012.

[19] N. Zhang, M. Ryan, and D. Guelev. Evaluating access control policies through model checking. In *Information Security*, volume 3650 of *LNCS*, pages 446–460. 2005.