Information and Software Technology 58 (2015) 355–372

Contents lists available at ScienceDirect

Information and Software Technology

journal homepage: www.elsevier.com/locate/infsof

Similarity testing for access control

Antonia Bertolino [a], Said Daoudagh [a], Donia El Kateb [b], Christopher Henard [b], Yves Le Traon [b], Francesca Lonetti [a], Eda Marchetti [a], Tejeddine Mouelhi [b,*], Mike Papadakis [b]

[a] Istituto di Scienza e Tecnologie dell'Informazione "A. Faedo", Consiglio Nazionale delle Ricerche via G. Moruzzi 1, 56124 Pisa, Italy
[b] Interdisciplinary Research Centre, SnT, University of Luxembourg, Luxembourg

ARTICLE INFO

Article history:
Received 4 November 2013
Received in revised form 4 July 2014
Accepted 6 July 2014
Available online 21 July 2014

Keywords:
Similarity
Test prioritization
Security policies

ABSTRACT

Context: Access control is among the most important security mechanisms, and XACML is the de facto standard for specifying, storing and deploying access control policies. Since it is critical that enforced policies are correct, policy testing must be performed in an effective way to identify potential security flaws and bugs. In practice, exhaustive testing is impossible due to budget constraints. Therefore the tests need to be prioritized so that resources are focused on their most relevant subset.
Objective: This paper tackles the issue of access control test prioritization. It proposes a new approach for access control test prioritization that relies on similarity.
Method: The approach has been applied to several policies and the results have been compared to random prioritization (as a baseline). To assess the different prioritization criteria, we use mutation analysis and compute the mutation scores reached by each criterion. This helps assessing the rate of fault detection.
Results: The empirical results indicate that our proposed approach is effective and its rate of fault detection is higher than that of random prioritization.
Conclusion: We conclude that prioritization of access control test cases can be usefully based on similarity criteria.

1. Introduction

Modern networked systems must be equipped with security services that provide adequate protection to users and companies in a relatively open environment. Several approaches and infrastructures, e.g., [1,2] have been recently proposed for the delivery of adaptive dynamic services that can provide seamless connectivity while preserving privacy and confidentiality of personal and critical data.

Security is achieved through appropriate mechanisms that guarantee the confidentiality, integrity and availability (the so-called CIA triad) of on-line data. Among security mechanisms, one important component is the *access control system*, which mediates all requests of access to protected data. Access control ensures that only the intended, i.e., authorized users can access the data, and that these intended users are only given the level of access required to accomplish their tasks. In short, the access control system replies to an authorization request with a permit/deny decision that is typically based on predefined *security policies*. Any fault in the access control system could lead to security flaws, resulting in either denial of accesses that should be allowed, or, even worse, allowance of accesses to non authorized users. Thus, it is important to perform a careful verification and validation of such system.

XACML [3] is the de facto standard for specifying, storing and deploying access control policies. However, the process of XACML policy specification can be error-prone due to the language complexity. Several approaches have been proposed to automate the generation of XACML tests, including Targen [4] and X-CREATE [5]. A common drawback of existing tools is that they produce a huge number of tests. For evident limitations of testing budget and time, it is generally impossible to run all those tests and check that the results are correct (this latter step is usually done manually). Therefore means to identify, among the many generated tests, those ones that deserve higher priority are crucial. This paper focuses on this specific issue, namely, on *XACML test prioritization*.

Test prioritization has been widely investigated in the field of software testing: it aims at defining a test execution order

[*] Corresponding author.
E-mail addresses: antonia.bertolino@isti.cnr.it (A. Bertolino), said.daoudagh@isti.cnr.it (S. Daoudagh), donia.elkateb@uni.lu (D. El Kateb), christopher.henard@uni.lu (C. Henard), yves.letraon@uni.lu (Y. Le Traon), francesca.lonetti@isti.cnr.it (F. Lonetti), eda.marchetti@isti.cnr.it (E. Marchetti), tejeddine.mouelhi@uni.lu (T. Mouelhi), mike.papadakis@uni.lu (M. Papadakis).

http://dx.doi.org/10.1016/j.infsof.2014.07.003
0950-5849/© 2014 Elsevier B.V. All rights reserved.

according to some criteria (e.g., coverage, fault detection rate), so that those tests that have a higher priority are executed before the ones having a lower priority. Several proposals include approaches for test prioritization in the context of regression testing [6,7]. We clarify that in this paper we do not address prioritization techniques expressly for regression testing; more in general we aim at deriving a test execution order for a given test suite.

In [8], several test prioritization techniques are used to increase the fault detection rate of test suites. More recent results [9] still confirm the effectiveness of test case prioritization based on fault detection rate and show the flexibility of the approach for application in different contexts. However, as demonstrated in [10], no prioritization metric is the best one for any system: indeed, the performance of the prioritization approach varies according to the considered application and could depend on the evaluated test suites. Another proposal addresses time-constrained test prioritization in the context of integer programming [11].

An approach that is currently considered very promising is based on the notion of test similarity, e.g., [12]: the intuition behind similarity-based prioritization is that when resources are limited and only a subset of test cases within a large test suite can be executed, then it is convenient to start from those that are the most dissimilar according to a predefined distance function.

In this paper, we propose to adapt *similarity-based prioritization to order XACML test cases*. To do this, we need to capture and specify what is a suitable notion of distance between XACML requests. To the best of our knowledge, our approach is the first attempt to introduce a prioritization strategy in XACML access control systems. The approach has been implemented into a tool called SIMTAC (SIMilarity Testing for Access-Control) that is publicly available for download.[1]

To evaluate the proposed prioritization strategy for the testing of XACML access control systems, we consider the fault detection rate criterion. In particular, we rely on mutation analysis to inject faults into the XACML policy, and challenge the ordered tests to detect the faults seeded in the policy itself. The goal is to end up with XACML tests ordered in a way that enables to quickly reach a high mutation score.

The contributions of this paper include:

- the introduction of the first test prioritization technique for XACML access control systems;
- the definition of two XACML similarity metrics, a simple one independent of the XACML policy, and another exploiting the XACML policy specification;
- an empirical study that compares different alternative techniques to prioritize XACML requests (on six policies of various complexity) for assessing our proposed technique.

The remainder of this paper is organized as follows. Section 2 introduces the XACML language and how XACML test cases are generated. Section 3 motivates this work while Section 4 presents our new test similarity-based prioritization approach. Then, Section 5 shows the empirical evaluation of the proposed approach, followed by Section 6 that discusses threats to validity. Finally, Section 7 presents the related work and Section 8 concludes the paper, also hinting at future work.

2. Background

This section introduces the background behind the proposed approach. Specifically, we first present the XACML language and an XACML policy example. Then we focus on XACML requests generation and provide a short description of a combinatorial testing strategy used for deriving the test suites adopted in the empirical validation.

2.1. XACML language

XACML [3] is a de facto standardized specification language that defines access control policies and access control decision requests/responses in an XML format. An XACML policy defines the access control requirements of a protected system. An access control request aims at accessing a protected resource in a given system whose access is regulated by a security policy. The request is evaluated against the policy and the access is granted or denied.

The main components of an access control systems architecture are the Policy Enforcement Point (PEP) and the Policy Decision Point (PDP). A PEP intercepts a user's request, transforms it into an XACML format and transmits it to the PDP. As showed in Fig. 1, the PDP evaluates the request against the XACML policy and returns the access response (Permit/Deny/NotApplicable/Indeterminate).

In a simplified vision an XACML policy has a hierarchical structure: at the top level there is the *policy set*, which can contain in turn one (or more) *policy set*(s) or *policy* elements. A *policy set* (a *policy*) consists of a target, a set of rules and a rule combining algorithm. The target specifies the subjects, resources, actions and environments on which a policy can be applied. If a request satisfies the target of the *policy set* (*policy*), then the set of rules of the *policy set* (*policy*) is checked, else the *policy set* (*policy*) is skipped. A rule is composed by: a target, which specifies the constraints of the request that are applicable to the rule; a condition, which is a boolean function evaluated when the request is applicable to the rule. If the condition is evaluated to true, the result of the rule evaluation is the rule effect (*Permit* or *Deny*), otherwise a *NotApplicable* result is given. If an error occurs during the application of a request to the policy, *Indeterminate* is returned. The rule combining algorithm specifies the approach to be adopted to compute the decision result of a policy when more than one rule may be applicable to a given request. For instance, the *permit-overrides* algorithm specifies that Permit takes the precedence regardless of the result of evaluating any of the other rules in the combination, then it returns Permit if there is a rule that is evaluated to Permit, otherwise it returns Deny if there is at least a rule that is evaluated to Deny and all other rules are evaluated to NotApplicable. If there is an error in the evaluation of a rule with Permit effect and the other policy rules with Permit effect are not applicable, the Indeterminate result is given. The access decision is given by considering all attribute and element values describing the subject, resource, action and environment of an access request and comparing them with the attribute and element values of the policy.

Listing 1 illustrates an XACML policy with two rules. The first rule (lines 33–67) states that a student can borrow and return books from the library. The second rule (lines 68–96) states that a professor is authorized to buy books for the library.

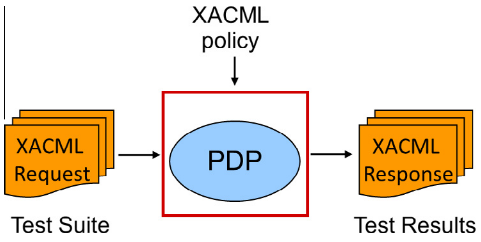


Fig. 1. XACML policy evaluation.

[1] A release of the SIMTAC tool is available at http://labse.isti.cnr.it/tools/simtac.

```
<PolicySet xmlns="xacml:2.0:policy:schema:os"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 PolicyCombiningAlgId="first-applicable" PolicySetId="LibrarySet">
    <!--                THE POLICY SET TARGET                              -->

 <Target>
  <Resources>
   <Resource>
    <ResourceMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
     <AttributeValue DataType="string">Book</AttributeValue>
     <ResourceAttributeDesignator AttributeId="resource-id" DataType="string"/>
    </ResourceMatch>
   </Resource>
  </Resources>
 </Target>
 <Policy PolicyId="Library" RuleCombiningAlgId="first-applicable">

     <!--                THE POLICY TARGET                              -->

  <Target>
   <Subjects>
    <Subject>
     <SubjectMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
      <AttributeValue DataType="string">Student</AttributeValue>
      <SubjectAttributeDesignator AttributeId="subject-id" DataType="string"/>
     </SubjectMatch>
    </Subject>
   </Subjects>
  </Target>

  <!--                THE POLICY RULES                              -->

  <Rule Effect="Permit" RuleId="Rule1">

      <!--             RULE 1 TARGET: SUBJECTS, RESOURCES AND ACTIONS                   -->

   <Target>
    <Subjects> <Subject>
      <SubjectMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
       <AttributeValue DataType="string">Student</AttributeValue>
       <SubjectAttributeDesignator AttributeId="subject-id" DataType="string"/>
      </SubjectMatch>
     </Subject>
    </Subjects>
    <Resources><Resource>
      <ResourceMatch
       MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
       <AttributeValue DataType="string">Book</AttributeValue>
       <ResourceAttributeDesignator AttributeId="resource-id" DataType="string"/>
      </ResourceMatch>
     </Resource>
    </Resources>
    <Actions><Action>
      <ActionMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
       <AttributeValue DataType="string">Borrow</AttributeValue>
       <ActionAttributeDesignator AttributeId="action-id" DataType="string"/>
      </ActionMatch>
     </Action>
     <Action>
      <ActionMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
       <AttributeValue DataType="string">Return</AttributeValue>
       <ActionAttributeDesignator AttributeId="action-id" DataType="string"/>
      </ActionMatch>
     </Action>
    </Actions>
   </Target>
  </Rule>
  <Rule Effect="Permit" RuleId="Rule2">

      <!--             RULE 2 TARGET: SUBJECTS, RESOURCES AND ACTIONS                   -->

   <Target>
    <Subjects><Subject>
      <SubjectMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
       <AttributeValue DataType="string">Professor</AttributeValue>
       <SubjectAttributeDesignator AttributeId="subject-id" DataType="string"/>
      </SubjectMatch>
     </Subject>
    </Subjects>
    <Resources><Resource>
      <ResourceMatch
       MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
       <AttributeValue DataType="string">Book</AttributeValue>
       <ResourceAttributeDesignator AttributeId="resource-id" DataType="string"/>
      </ResourceMatch>
     </Resource>
    </Resources>
    <Actions> <Action>
      <ActionMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
       <AttributeValue DataType="string">Buy</AttributeValue>
       <ActionAttributeDesignator AttributeId="action-id" DataType="string"/>
      </ActionMatch>
     </Action>
    </Actions>
   </Target>
  </Rule>
 </Policy>
</PolicySet>
```

Listing 1. XACML policy example.

```
<?xml version="1.0" encoding="UTF-8"?>
<Request>
    <Subject>
        <Attribute AttributeId="subject-id" DataType="XMLSchema#string">
            <AttributeValue>Student</AttributeValue>
        </Attribute>
    </Subject>
    <Resource>
        <Attribute AttributeId="resource-id"
            DataType="XMLSchema#string">
            <AttributeValue>Book</AttributeValue>
        </Attribute>
    </Resource>
    <Action>
        <Attribute AttributeId="action-id"
            DataType="XMLSchema#string">
            <AttributeValue>Borrow</AttributeValue>
        </Attribute>
    </Action>
    <Environment/>
</Request>
```

Listing 2. XACML request example.

An XACML request is composed of four elements: a subject, a resource, an action and an environment. The values and types of these four elements should be among the values and types defined by the policy rules or targets. Testing an XACML policy involves generating a set of requests to be evaluated based on the policy. The responses to these requests are then checked against the expected decisions. The next section presents a strategy for automatically generating the XACML requests.

2.2. Test cases generation

A critical issue in testing XACML access control systems is the generation of an effective test suite.

Listing 2 presents an example of an XACML request of a student asking to borrow a book from the library. In detail the request contains one subject attribute (*Student*), one action attribute (*Borrow*) and one resource attribute (*Book*). If this request is evaluated considering the XACML policy of Listing 1, a *Permit* decision is returned allowing the access. Specifically, this request will be first evaluated against the target of the policy set (line 6–15): it will be applicable to this target since it matches the resource *Book*; then it will be evaluated against the target of the policy (line 20–29): it will be applicable to this target since it matches the subject *Student*; finally it will be evaluated against the rules *Rule1* (line 33–67) and *Rule2* (line 68–96): it will be applicable to *Rule1* since it matches the subject *Student*, the resource *Book* and one of the two actions specified in that rule (the *Borrow* action), whereas it will be not applicable to the rule *Rule2* since it does not match the subject and action of this rule. Because the defined algorithm in the policy is *first-applicable*, the effect of the first rule, i.e., *Permit* will be returned.

Several common approaches for generating XACML requests are based on combinatorial strategies, as surveyed in Section 7. In this paper, among the tools available for test cases generation we refer to X-CREATE [13,5,14].[2] In particular, we use the *Simple Combinatorial* test strategy implemented in this tool for deriving the test suites used to empirically validate the effectiveness of the proposed XACML prioritization approach.

The *Simple Combinatorial* strategy applies a combinatorial approach to the policy values. Specifically, four data sets called *SubjectSet*, *ResourceSet*, *ActionSet* and *EnvironmentSet* are defined. These sets are filled with the values and the attributes of the policy elements ⟨`Subjects`⟩, ⟨`Resources`⟩, ⟨`Actions`⟩ and ⟨`Environments`⟩, respectively. The elements and attributes values in each set are then combined in order to obtain the entities. Specifically, a *subject entity* is defined as a combination of the values of elements and attributes of the *SubjectSet* set. Similarly the *resource entity*, the *action entity* and the *environment entity* represent combinations of the values of the elements and attributes of the *ResourceSet*, *ActionSet*, and *EnvironmentSet* respectively.

Then, an ordered set of combinations of *subject entities*, *resource entities*, *action entities* and *environment entities* is generated in the following way:

- First, pair-wise combinations are generated to obtain the *PW* set.
- Then, three-wise combinations are generated to obtain the *TW* set.
- Finally, four-wise combinations are generated to obtain the *FW* set.

These sets have the following inclusion propriety $PW \subseteq TW \subseteq FW$. Thus, the maximum number of requests derived by this strategy is equal to the cardinality of the *FW* set. The X-CREATE framework provides an ordered set of requests guaranteeing a coverage first of all pairs, then of all triples and finally of all quadruples of values entities derived by the policy. Since the *Simple Combinatorial* strategy relies only on the values entities specified in the policy, the derived test suite can be used for testing either the policy or the PDP. More details about this strategy are in [5].

3. Motivation

It is a shared understanding in testing environments that automated support tools for test cases generation and execution can drastically reduce the huge time and effort usually required for these activities. However, the activity of checking the testing outcomes remains largely a manual task and can become the bottleneck of the overall testing process. In fact, deciding whether each test result is correct or not can be a budget-consuming activity, especially when a (possibly large) number of tests is automatically executed.

During the TAS3 [1] project, we performed an experiment that aimed at evaluating the impact of test activities inside the development of a commercial access control system. We found that automatic test requests generation and execution required only 0.02% of the overall testing time, and that the (manual) analysis of test results took the remaining 99.98% [14]. Thus an emerging challenge is to provide applicable and efficient proposals to reduce the effort needed during the manual check of the test outputs.

[2] A release of the X-CREATE tool is available from http://labse.isti.cnr.it/tools/xcreate.

In software testing, solutions to reduce the cost of verdict analysis include either the development and adoption of automatic mechanisms (usually called the test oracles), or the application of proper strategies for test cases selection [15,16] or prioritization [12].

To the best of our knowledge, in the context of access control systems the only available proposal to automatically check whether the test outputs are correct, is provided in [17]. This work proposes to simultaneously observe the responses from different PDPs on the same test inputs, so that different responses can highlight possible issues. Although effective, the proposal is quite demanding, because it requires using different PDP implementations. The cost and effort necessary for the approach may prevent its applicability in a commercial settings.

In test case selection, the aim is to reduce the cardinality of the test suites while keeping the same effectiveness in terms of coverage or fault detection rate; in test case prioritization, the aim is to order the test cases so that those having the highest priority can be executed first. In this paper, we take the latter direction by performing XACML requests prioritization.

In our previously mentioned experiment [14], we learned several lessons for improving the test suite effectiveness and reducing the cost of verdict analysis. First, it is evident that only those requests that are applicable to a policy (namely those that contain values matching the target of the policy set, the target of the policy and the target of the rule) will trigger the rule decision and hence facilitate the identification of possible access problems related to the policy. This evidence has been used for the ad hoc selection of the test cases. Then, the execution of such selected test cases and the analysis of the obtained results highlighted that: (i) the effectiveness of the reduced test suites in terms of verdict coverage was preserved; (ii) the cardinality of the reduced test suites for some policies was drastically decreased; (iii) the analysis time of the verdicts of all reduced test suites was reduced of the 95%.

On the basis of the above experience, we reached the conclusion that only specific test case selection criteria taking into account the policy values and the request applicability to the XACML policy, represent feasible and effective solutions for testing access control systems.

In this paper we employ this lesson, learned in the context of XACML test case selection, for prioritization and present an approach, implemented into the SIMTAC tool, taking into account the applicability of the request to the policy. More specifically we propose an XACML test case prioritization approach based on similarity, using two different metrics: (i) a standard similarity metric applied to XACML test suites (we call it *simple similarity*), and (ii) a more specific similarity metric for the prioritization of test cases within an XACML test suite (we call it *XACML similarity*). In particular the latter implements the previously mentioned recommendation, by prioritizing those requests triggering the rule decision of an XACML policy.

4. Similarity metrics

Similarity is a heuristic that is used here to order access control requests, i.e., the test cases. Previous work on model-based testing, such as [18], has shown that dissimilar test cases bestow a higher fault detection power than similar ones. Analogously, the experiment results presented in this paper (see Section 5) show that two dissimilar access control requests are likely to find more access control faults than two similar ones.

In the following, we consider a test suite of r access control requests $\{R_1, \ldots, R_r\}$. A similarity-based prioritization approach consists of two steps. The first step involves the definition of a distance metric d between any two access control requests R_i and R_j, where $1 \leqslant i,j \leqslant r$. This metric is used to evaluate the degree of similarity between two given requests: the highest the resulting distance, the most different the two requests. The second step is the ordering of these r requests. To this end, we first compute the distance between each pair of requests. Then, a prioritization algorithm uses the computed distances to select the most dissimilar requests, resulting in a list where the first selected requests are the most dissimilar ones. In Sections 4.1–4.3 we introduce the similarity distances proposed in this paper (step 1), whereas in Section 4.4 we show the prioritization algorithm (step 2).

4.1. Distance metrics between access control requests

We present two methods for calculating a distance metric d between any two access control requests. The former, called the *simple similarity*, is based on the lexical distance of the requests parameters (subject, resource, action, environment). In this case the distance d_{ss} can be generally defined as follows:

$$d_{ss} : \begin{array}{ccc} R \times R & \rightarrow & \{0,1,2,3,4\} \\ (R_i, R_j) & \mapsto & d_{ss}(R_i, R_j) \end{array}.$$

The latter, called the *XACML similarity*, takes into account the requests attributes values (as the simple similarity) and the XACML policy. The idea is to go through all levels of a policy, from the policy set target to the rules targets, and compare the request attributes values with the targets values at each level.

The comparison between an XACML request and an XACML policy is performed following a relation called here *Applicability*. Specifically, if the request matches a target at some level (policy set, policy or rule), then it is considered to be applicable.

For the XACML similarity, the distance d_{xs} between requests is policy-dependent and can be generally defined as follows:

$$d_{xs} : \begin{array}{ccc} R \times R \times XP & \rightarrow & \mathbb{R}_+ \\ (R_i, R_j, XP_k) & \mapsto & d_{xs}(R_i, R_j, XP_k) \end{array}.$$

For both *simple similarity* and *XACML similarity*, we adopt the convention that the higher is the resulting distance value, the more dissimilar are the two requests, with a distance value equal to 0 meaning that two requests are identical.

Fig. 2 outlines the main steps for computing the two distance metrics given two requests R_i and R_j belonging to a test suite of r access control requests $\{R_1, \ldots, R_r\}$.

As shown in Fig. 2 (light gray part), the *simple similarity* distance $d_{ss}(R_i, R_j)$ is derived by parsing each pair of requests (R_i, R_j), where $1 \leqslant i,j \leqslant r$, so as to extract their attributes values $\{subject, resource, action$ and $environment\}$.

These values are represented in Fig. 2 by the vectors called (V_{R_i}, V_{R_j}), where $1 \leqslant i,j \leqslant r$. The similarity distance $d_{ss}(R_i, R_j)$ is computed by comparing the vectors (V_{R_i}, V_{R_j}) and counting the number of attributes having different values in the two vectors.

As an example we consider the attribute values of a set of six requests $R_1, R_2, \ldots, R_6$ obtained by the application of the Simple Combinatorial Strategy described in Section 2.2 to the policy of Listing 1.

- $R_1 = \{Student, Book, Buy, null\}$,
- $R_2 = \{Professor, Book, Borrow, null\}$,
- $R_3 = \{Student, Book, Return, null\}$,
- $R_4 = \{Professor, Book, Return, null\}$,
- $R_5 = \{Professor, Book, Buy, null\}$,
- $R_6 = \{Student, Book, Borrow, null\}$.

For instance, $d_{ss}(R_1, R_3) = 1$ and $d_{ss}(R_3, R_6) = 1$ since only the *action* attribute is different in the two requests, while $d_{ss}(R_1, R_2) = 2$ since both the *subject* and the *action* differ in R_1

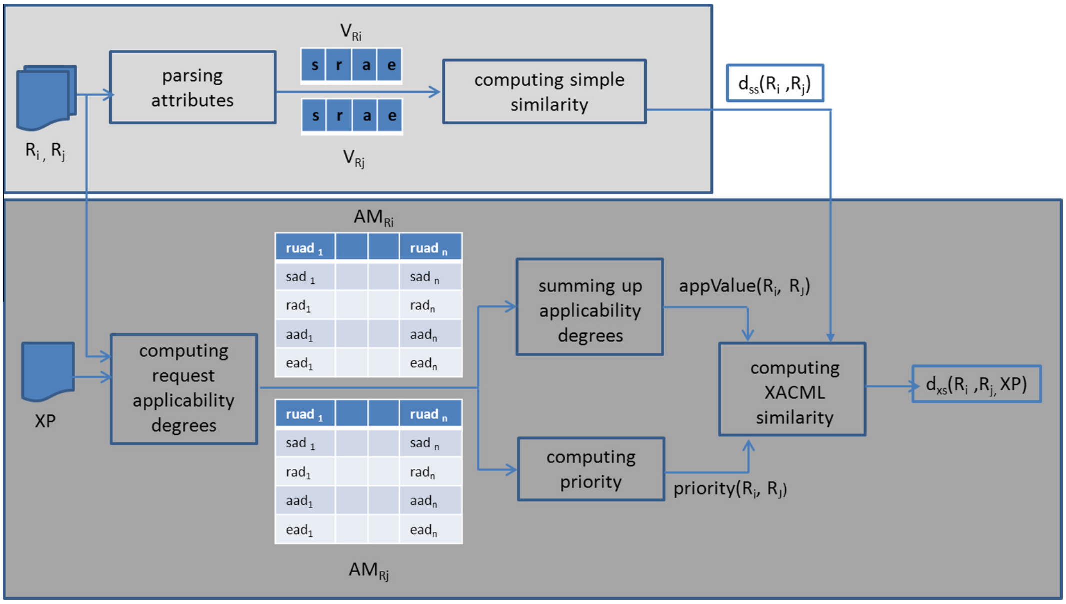


Fig. 2. Main steps for computing similarity metrics.

and R_2. More details about the *simple similarity* distance are provided in Section 4.2.

The XACML similarity distance $d_{xs}(R_i, R_j, XP)$, of each couple of requests $\{R_i, R_j\}$, where $1 \leqslant i, j \leqslant r$, is a policy-dependent measure that uses three different values: (i) the *simple similarity* distance $d_{ss}(R_i, R_j)$; (ii) the *Applicability* value of the couple (R_i, R_j) to the policy XP; and (iii) the value of a *priority* relation of R_i and R_j.

The computation of the *Applicability* value of a couple (R_i, R_j) to the policy XP includes two steps:

1. the derivation of the applicability degrees of each request to the XACML policy;
2. the summing up of the applicability degrees of the couple (R_i, R_j).

Precisely, for each request R_i, where $1 \leqslant i \leqslant r$, five degrees of applicability are considered: *rule applicability* (ruad): for each rule of the policy, it represents the degree to which the request can satisfy first the targets of the policy sets and the policy which the rule belongs to, and then the target of the rule itself; *subject applicability* (sad) [*resource applicability* (rad), *action applicability* (aad), *environment applicability* (ead) respectively]: it represents the degree to which the subject [resource, action, environment] of the request can match first the subjects [resources, actions, environments] of the targets of the policy sets and the target of the policy which the rule belongs to, and then the target of the rule itself.

For instance, considering the XACML request R_3 and the rule *Rule*1 of Listing 1 (line 33–67), the five applicability degrees of R_3 to the rule *Rule*1 are as follows:

$$(ruad, sad, rad, aad, ead) = \left(1, 1, 1, \frac{1}{2}, 0\right)$$

Specifically, the ruad value is 1 because the request R_3 exactly matches the target of the policy set, that of the policy and that of the rule *Rule*1; sad is 1 because the subject of R_3, i.e., *Student* exactly matches the subject of the target of the policy set, that of the policy and that of the rule *Rule*1; similarly rad is 1 because the resource of R_3, i.e., *Book* exactly matches the resource of the target of the policy set, that of the policy and that of the rule *Rule*1; aad is $\frac{1}{2}$ since the action of R_3, i.e., *Return* matches only one of the two actions of the target of the rule *Rule*1, i.e., *Borrow* and *Return* and the target of the policy set and that of the policy do not contain action values. Finally, *ead* is 0 since R_3 does not specify any environment value.

As shown in Fig. 2 (dark gray part), for each request R_i, the first step of the computation of the applicability values provides a $5 \times n$ matrix (where n is the cardinality of the XACML policy rules), Applicability Matrix, of the request R_i (AM_{R_i}). Each column h of the matrix AM_{R_i} contains the ruad, sad, rad, aad, ead values that refer to the h-th rule of the policy P.

For instance the applicability matrix of the request R_3 to the rules of the policy of Listing 1 is:

$$AM_{R_3} = \begin{bmatrix} 1.0 & \frac{2}{3} \\ 1.0 & 1.0 \\ 1.0 & 1.0 \\ \frac{1}{2} & 0.0 \\ 0.0 & 0.0 \end{bmatrix}$$

where the first column is the applicability vector of R_3 to rule *Rule*1 described above, whereas the second one is the applicability vector of R_3 to rule *Rule*2.

Similarly, the applicability matrix of the request R_6 to the rules of the same policy is:

$$AM_{R_6} = \begin{bmatrix} 1.0 & \frac{2}{3} \\ 1.0 & 1.0 \\ 1.0 & 1.0 \\ \frac{1}{2} & 0.0 \\ 0.0 & 0.0 \end{bmatrix}$$

The second step in the computation of the *Applicability* value consists in summing up the applicability degrees of each couple of matrices (AM_{R_i}, AM_{R_j}), where $1 \leqslant i, j \leqslant r$. In Fig. 2 the result of this sum is called *appValue*(R_i, R_j).

For instance the applicability value of (AM_{R_3}, AM_{R_6}) is 6.17 + 6.17 = 12.34 where 6.17 and 6.17 are the sum of all the values of AM_{R_3} and AM_{R_6}, respectively.

The third value used in the computation of the XACML similarity distance $d_{xs}(R_i, R_j, XP)$ is represented by the priority of each pair of access control requests. The priority value for the couple (R_i, R_j)

is computed according to the applicability of the requests to the XACML policy and can be equal to 3, 2, 1, 0. This priority value will be equal to: 3 when both requests are applicable to at least a rule, 0 when both requests are not applicable to any rule. The aim is to give higher priority to those couples of requests able to trigger the Effect (Permit or Deny) of the rules.

As in Fig. 2, the computation of the priority value (called $priority(R_i, R_j)$) is performed by analyzing each couple of matrices (AM_{R_i}, AM_{R_j}). For instance, considering the above presented matrices AM_{R_3} and AM_{R_6}, the priority value associated to the pair of requests (R_3, R_6) is 3 since both requests are applicable to the rule *Rule1*.

The XACML similarity distance $d_{xs}(R_i, R_j, XP)$ is finally computed by summing the three obtained values $(d_{ss}(R_i, R_j), appValue (R_i, R_j), priority(R_i, R_j))$.

For instance the XACML similarity distance between the requests R_3 and R_6 is $1 + 12.34 + 3\ \ = 16.34$ where 1 is the simple similarity distance between R_3 and R_6, 12.34 is the applicability value and 3 is the priority value computed as before.

In the following sections details about the computation of the two distance measures are provided.

4.2. Simple similarity

Given two requests (R_i, R_j), the simple similarity $d_{ss}(R_i, R_j)$ is defined based on a comparison between the request attributes values. There are four attributes in each request: $\{subject, action, resource$ and $environment\}$. For each attribute, the simple similarity compares the values in the two requests (R_i, R_j). The distance increases each time a given attribute has different values in the two requests. Since the evaluation is based on four attributes, the final distance varies between 0 and 4. Formally, the simple similarity is defined as follows:

$$d_{ss}(R_i, R_j) = \sum_{k=1}^{4} d^k_{attribute}(R_i, R_j)$$

where

$$d^k_{attribute}(R_i, R_j) = \begin{cases} 1 & R_i.attribute[k] \neq R_j.attribute[k] \\ 0 & \text{otherwise} \end{cases}$$

The similarity distance values relative to a set of requests $\{R_1, \ldots, R_r\}$ are represented by a $r \times r$ matrix, called the Simple Similarity Matrix (SSM)

$$SSM : (R \times R) \rightarrow \{0, 1, 2, 3, 4\}$$

defined as:

$$[SSM]_{i,j} = d_{ss}(R_i, R_j) \quad i,\ j = 1, 2, \ldots, r \text{ and } i < j.$$

Considering the six requests $R_1, R_2, \ldots, R_6$ presented in the previous section, the SSM matrix is

$$SSM = \begin{array}{c|cccccc|} & R_1 & R_2 & R_3 & R_4 & R_5 & R_6 \\ \hline R_1 & 0 & 2 & 1 & 2 & 1 & 1 \\ R_2 & 0 & 0 & 2 & 1 & 1 & 1 \\ R_3 & 0 & 0 & 0 & 1 & 2 & 1 \\ R_4 & 0 & 0 & 0 & 0 & 1 & 2 \\ R_5 & 0 & 0 & 0 & 0 & 0 & 2 \\ R_6 & 0 & 0 & 0 & 0 & 0 & 0 \end{array}$$

4.3. XACML similarity

In this section, we first provide some definitions about the applicability of a couple of requests to an XACML policy (Section 4.3.1), then we present the priority relation (Section 4.3.2), and finally we formally define the XACML similarity distance (Section 4.3.3).

4.3.1. Applicability definitions

Applicability is a relation between an XACML request and an XACML policy. We introduce first the *Applicability Degree*, which represents the percentage of a rule that is satisfied by a request. For each request the *Applicability Degree* values associated to the whole set of rules are collected into a matrix called the *Applicability Matrix* (Definition 1). This matrix summarizes the applicability of the request to the XACML policy. In particular, the sum of the elements of the *Applicability Matrix* provides the *Request Applicability Value*, which is used for assessing the requests against each other in terms of overall applicability to the policy. Then, for each couple of requests, we compute their *Applicability Value*, calculated as the sum of their respective *Request Applicability Value*s, which represents the overall applicability degree of a couple of requests to the XACML policy.

As already introduced in Section 4.1, for each request R_i, where $1 \leqslant i \leqslant r$, five degrees of applicability are considered: *rule applicability* (ruad); *subject applicability* (sad); *resource applicability* (rad); *action applicability* (aad), *environment applicability* (ead). These five values are represented into a Column Vector of length 5 called Applicability Degree, AD_{RU}, defined as follows:

$$\mathbf{AD_{RU}} = \begin{bmatrix} ruad_{RU} \\ sad_{RU} \\ rad_{RU} \\ aad_{RU} \\ ead_{RU} \end{bmatrix}$$

For instance, considering the XACML request R_3 and the rule *Rule*1 of Listing 1 (line 33–67), the Applicability Degree of R_3 to the rule *Rule*1, as already explained in Section 4.1, can be shown by the following Column Vector:

$$\mathbf{AD_{RU_{Rule1}}} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ \frac{1}{2} \\ 0 \end{bmatrix}$$

To each request we associate a $5 \times n$ matrix, where n is cardinality of the XACML policy rules, called Applicability Matrix of the request $R_i(AM_{R_i})$. Each column k of the matrix AM_{R_i} contains the $ruad_k$, sad_k, rad_k, aad_k, ead_k values that refer to the k-th rule of the XACML policy XP.

Definition 1 (*Applicability Matrix*). Given a request R and a set of n element Rules $\{RU_i\}$, where $i = 1, \ldots, n$, the Applicability Matrix of R, called AM_R, is a $5 \times n$ matrix defined as:

$$\mathbf{AM_R} = [\,\mathbf{AD_{RU_1}} \quad \mathbf{AD_{RU_2}} \quad \ldots \quad \mathbf{AD_{RU_n}}\,].$$

Considering the set of XACML requests introduced in Section 4.1, $\{R_1, R_2, \ldots, R_6\}$, and according to Definition 1, we have the following Applicability Matrices:

$$AM_{R_1} = \begin{bmatrix} \frac{2}{3} & \frac{2}{3} \\ 1.0 & 1.0 \\ 1.0 & 1.0 \\ 0.0 & 0.0 \\ 0.0 & 0.0 \end{bmatrix} \quad AM_{R_2} = \begin{bmatrix} \frac{1}{3} & \frac{1}{3} \\ 0.0 & 0.0 \\ 1.0 & 1.0 \\ 0.0 & 0.0 \\ 0.0 & 0.0 \end{bmatrix} \quad AM_{R_3} = \begin{bmatrix} 1.0 & \frac{2}{3} \\ 1.0 & 1.0 \\ 1.0 & 1.0 \\ \frac{1}{2} & 0.0 \\ 0.0 & 0.0 \end{bmatrix}$$

$$AM_{R_4} = \begin{bmatrix} \frac{1}{3} & \frac{1}{3} \\ 0.0 & 0.0 \\ 1.0 & 1.0 \\ 0.0 & 0.0 \\ 0.0 & 0.0 \end{bmatrix} \quad AM_{R_5} = \begin{bmatrix} \frac{1}{3} & \frac{1}{3} \\ 0.0 & 0.0 \\ 1.0 & 1.0 \\ 0.0 & 0.0 \\ 0.0 & 0.0 \end{bmatrix} \quad AM_{R_6} = \begin{bmatrix} 1.0 & \frac{2}{3} \\ 1.0 & 1.0 \\ 1.0 & 1.0 \\ \frac{1}{2} & 0.0 \\ 0.0 & 0.0 \end{bmatrix}.$$

Definition 2 (*Request Applicability Value*). Given an XACML request R, and its $5 \times n$ Applicability Matrix AM_R, the Request Applicability Value associated to R, called RA_R is defined as:

$$RA_R = \sum_{h=1}^{5}\sum_{k=1}^{n}[AM_R]_{h,k}.$$

Definition 3 (*Applicability Value*). Given a couple of requests (R_i, R_j) and their Request Applicability Values RA_{R_i}, RA_{R_j}, the Applicability Value associated to the couple of requests, called AppValue (R_i, R_j) is defined as:

$$AppValue(R_i, R_j) = RA_{R_i} + RA_{R_j}.$$

Considering the XACML requests, R_1 and R_3, their Applicability Matrices AM_{R_1}, AM_{R_3}, and according to Definition 2, the *Request Applicability Value* associated to R_1 is

$$RA_{R_1} = \sum_{h=1}^{5}\sum_{k=1}^{2}[AM_{R_1}]_{h,k} = 5.33$$

while the Request Applicability Value associated to R_3 is

$$RA_{R_3} = \sum_{h=1}^{5}\sum_{k=1}^{2}[AM_{R_3}]_{h,k} = 6.17$$

Hence, according to Definition 3, the *Applicability Value* associated to (R_1, R_3) is:

$$AppValue(R_1, R_3) = RA_{R_1} + RA_{R_3} = 11.5.$$

4.3.2. Priority Definition

The *Priority Value* (Definition 4) establishes the priority degree of a couple of requests. This *Priority Value* is computed considering the combined *Applicability Degree* of both requests to the policy. Depending on how the respective applicabilities combine, it can take four values (generically represented by α, β, γ, δ). Specifically, it yields the highest value when both requests trigger the effect of at least a rule. This value is decreased when only one request can trigger the effect of at least a rule, and further if none of the two requests can trigger the effect of a rule. From an empirical experimentation on a set of six policies described in Section 5.1, we observed that the best values for α, β, γ, δ are 3, 2, 1, 0 respectively. We cannot exclude though that different values of α, β, γ, δ could perform better for a different set of policies.

Definition 4 (*Priority Value*). Given a set of XACML requests, $\{R_1, R_2, \ldots, R_r\}$, and the set of their $5 \times n$ Applicability Matrices $\{AM_{R_1}, AM_{R_2}, \ldots, AM_{R_r}\}$, the Priority Value associated to a pair of requests (R_i, R_j), where $1 \leqslant i, j \leqslant r$ and $i \neq j$, called PriorityValue (R_i, R_j)

$$PriorityValue(R_i, R_j) : (R \times R) \;\rightarrow\; \{\alpha,\ \beta,\ \gamma,\ \delta\}$$

is defined as:

$$PriorityValue(R_i, R_j) = \begin{cases} \alpha & \text{if } [AM_{R_i}]_{1,h} = [AM_{R_j}]_{1,k} = 1 & \exists\, h,\ k\ s.t.\ 0 \leqslant h, k < n \\ \beta & \text{if } [AM_{R_i}]_{1,h} = 1 \ \wedge\ 0 \leqslant [AM_{R_j}]_{1,k} < 1 & \exists\, h\ s.t.\ 0 \leqslant h < n, \\ & & \forall k\ s.t.\ 0 \leqslant k < n \\ & OR & \\ & 0 \leqslant [AM_{R_i}]_{1,h} < 1 \ \wedge\ [AM_{R_j}]_{1,k} = 1 & \forall h\ s.t.\ 0 \leqslant h < n, \\ & & \exists\, k\ s.t.\ 0 \leqslant k < n \\ \gamma & \text{if } 0 < [AM_{R_i}]_{1,h},\ [AM_{R_j}]_{1,k} < 1 & \forall h, k\ s.t.\ 0 \leqslant h, k < n \\ \delta & \text{otherwise} & \end{cases}$$

According with Definition 4, the requests R_1 and R_3, and their Applicability Matrices, AM_{R_1}, AM_{R_3}, the Priority Value associated to the pair of requests (R_1, R_3) is *PriorityValue* $(R_1, R_3) = 2$ since $[AM_{R_1}]_{1,1} = [AM_{R_1}]_{1,2} = \frac{2}{3} \leqslant 1$ and $[AM_{R_3}]_{1,1} = 1$.

Otherwise considering also the request R_6 we have: *PriorityValue* $(R_3, R_6) = 3$ since $[AM_{R_3}]_{1,1} = [AM_{R_6}]_{1,1} = 1$.

4.3.3. XACML similarity definition

In this section, we formally specify the XACML similarity (Definition 5) representing the distance between a pair of requests. Specifically, given two requests (R_i, R_j), the XACML similarity distance $d_{xs}(R_i, R_j)$ is defined as the sum of the *simple similarity* distance, the *Applicability Value* and the *Priority Value* associated to the pair of requests (R_i, R_j).

In particular, if the two requests are identical, namely their *simple similarity* is equal to zero, then the XACML similarity distance is also set to zero.

Definition 5 (*XACML Similarity Distance*). Given a set of XACML requests, $\{R_1, R_2, \ldots, R_r\}$ and an XACML Policy XP, the XACML Similarity Distance between a pair of requests (R_i, R_j), where $1 \leqslant i,\ j \leqslant r$ and $i \neq j$, called $d_{xs}(R_i, R_j, XP)$, is defined as:

$$d_{xs}(R_i, R_j, XP) = \begin{cases} 0 & \text{if } d_{ss}(R_i, R_j) = 0 \\ d_{ss}(R_i, R_j) + & \\ AppValue(R_i, R_j) + & \\ PriorityValue(R_i, R_j) & \text{otherwise} \end{cases}$$

Using Definition 5 we define the XACML Similarity Matrix (XSM), in which the entry in the i-th row and j-th column with $i < j$ represents the XACML Similarity Distance between the pair of XACML Requests R_i, R_j. Formally, given an XACML Policy XP and given a set of XACML requests $\{R_1, R_2, \ldots, R_r\}$, the corresponding XACML Similarity Matrix is defined as:

$$XSM = \begin{array}{c} \\ R_1 \\ R_2 \\ \vdots \\ R_{r-1} \\ R_r \end{array} \begin{array}{c} \begin{array}{ccccc} R_1 & R_2 & \cdots & R_{r-1} & R_r \end{array} \\ \begin{bmatrix} 0 & d_{xs}(R_1, R_2, XP) & \cdots & d_{xs}(R_1, R_{r-1}, XP) & d_{xs}(R_1, R_r, XP) \\ & 0 & & & d_{xs}(R_2, R_r, XP) \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ & & & 0 & d_{xs}(R_{r-1}, R_r, XP) \\ 0 & 0 & \cdots & 0 & 0 \end{bmatrix} \end{array}$$

For instance, considering the XACML policy of Listing 1 and the set of XACML requests introduced in Section 4.1, $\{R_1, R_2, \ldots, R_6\}$, the corresponding XACML Similarity Matrix is the following:

$$XSM = \begin{array}{c|cccccc} & R_1 & R_2 & R_3 & R_4 & R_5 & R_6 \\ \hline R_1 & 0.0 & 11.0 & 14.5 & 11.0 & 10.0 & 14.5 \\ R_2 & 0.0 & 0.0 & 12.84 & 7.35 & 7.35 & 11.84 \\ R_3 & 0.0 & 0.0 & 0.0 & 11.84 & 12.84 & 16.34 \\ R_4 & 0.0 & 0.0 & 0.0 & 0.0 & 7.35 & 12.84 \\ R_5 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 12.84 \\ R_6 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \end{array}$$

where for instance, according to Definition 5, the XACML Similarity Distance between the pair of XACML Requests (R_1, R_3) is represented by the following element of XSM:

$$\begin{aligned}[XSM]_{R_1,R_3} &= d_{xs}(R_1, R_3, XP) \\ &= d_{ss}(R_1, R_3) + AppValue(R_1, R_3) + PriorityValue(R_1, R_3) \\ &= 1 + 11.5 + 2 = 14.5.\end{aligned}$$

4.4. Ordering the access control requests

This section presents the algorithm used for the prioritization of the requests, which can be applied to both distance metrics defined in the previous sections. The idea is to order the requests so that the first executed are those most dissimilar, i.e., the requests sharing the highest distance. To prioritize the XACML requests we adapt the technique proposed in [19]. The procedure steps are outlined in Algorithm 1 below.

Algorithm 1. Prioritization.

```
1: input: $S = \{R_1, \ldots, R_n\}$, distMatrix
2: output: $L$    ▹ Prioritized list of $n$ XACML requests
3: $L \leftarrow []$
4: Select $R_i, R_j$ where $max(distMatrix(R_i, R_j)), 1 \leqslant i, j \leqslant n$
5:                ▹ Take the first ones in case of equality
6: $L.add(R_i)$
7: $L.add(R_j)$
8: $S \leftarrow S \setminus \{R_i, R_j\}$
9: while $\#S > 0$ do
10:    $s \leftarrow size(L)$
11:    Select $R_i \in S$ where
       $max\left(\sum_{j=1}^{s} distMatrix(R_i, L.get(j))\right), 1 \leqslant i \leqslant n$
12:               ▹ Take the first one in case of equality
13:    $L.add(R_i)$
14:    $S \leftarrow S \setminus \{R_i\}$   ▹ Remove $R_i$ from $S$
15: end while
16: return $L$
```

Informally, the algorithm selects the request that is the most distant from all the requests already selected during the previous steps of the approach. It takes as input the set of XACML request $S = \{R_1, \ldots, R_n\}$ and a distance matrix (`distMatrix`), which can be either the SSM matrix or the XSM matrix defined in the previous sections. Using the distances between the requests collected into the matrix, it first selects the two XACML requests having the highest distance (Algorithm 1, line 4). In case of equality the first pair of requests is selected. Then these two requests are removed from the set of XACML requests to be prioritized, i.e., the set S (Algorithm 1, line 8). In the next step, the algorithm considers among the remaining XACML requests the one yielding the maximum sum of the distances from all the already selected requests (Algorithm 1, line 11). In case of equality, the first request is selected. Then, the selected request is removed from the XACML requests to be prioritized (Algorithm 1, line 14). The process is repeated until all requests are selected.

Considering for instance the SSM matrix at the end of Section 4.2 according to Algorithm 1 (line 4), the pair (R_1, R_2) is selected because this is the first pair having the maximum distance equal to 2 $(SSM(1,2) = 2)$. For the remaining set of requests $\{R_3, R_4, R_5, R_6\}$ the sum of the distances between each of them and the requests R_1 and R_2 is computed. Then the request having the maximum sum is selected (Algorithm 1, line 9–15). Specifically:

- for $R_3 : SSM(1,3) + SSM(2,3) = 1 + 2 = 3$,
- for $R_4 : SSM(1,4) + SSM(2,4) = 2 + 1 = 3$,
- for $R_5 : SSM(1,5) + SSM(2,5) = 1 + 1 = 2$,
- for $R_6 : SSM(1,6) + SSM(2,6) = 1 + 1 = 2$.

Thus the request R_3 is selected because it is the first one having the maximum sum (equal to 3). According to Algorithm 1 (line 9–15), the above described steps are repeated for the set of remaining requests. Finally, the obtained ordered set of requests is $\{R_1, R_2, R_3, R_4, R_5, R_6\}$.

Applying Algorithm 1 to the XSM matrix presented at the end of Section 4.3.3, instead, the obtained final ordered set of requests is $\{R_3, R_6, R_1, R_2, R_4, R_5\}$.

It is worth noting that this prioritization algorithm belongs to the category of prioritizations that do not rely on a feedback to adjust the selection of test cases as it goes forth (they are also called "total" prioritizations as opposed to the "additional" prioritizations that rely on a feedback).

5. Experiments

This section presents the experimental results obtained by applying the proposed similarity-based prioritization metrics. Specifically, we used the tool SIMTAC to evaluate the effectiveness of the simple similarity and XACML similarity metrics when applied to the test suites related to a set of real-world XACML policies. We aim at evaluating the effectiveness in terms of fault detection rate of the two similarity-based prioritization metrics, by answering to the following research questions:

> *RQ1: Similarity Effectiveness:* can the similarity-based prioritization techniques outperform other prioritization methods in terms of fault detection rate? In particular, we will assess whether the similarity-based prioritization techniques are more effective than prioritization based on a mutation-based heuristic, or on a random selection, or on n-wise combinatorial approaches.
> *RQ2: Similarity Variability:* is the effectiveness of a test suite prioritized using similarity-based approaches influenced by its size? In other words, we will assess whether the effectiveness in terms of fault detection of a test suite prioritized using similarity-based approaches depends on the size of the test suite.

By answering RQ1, we want to assess the effectiveness of similarity-based prioritization techniques against: (i) a mutation-based heuristic, which is able to optimize the fault detection rate and therefore represents the upper bound for the comparison; (ii) random selection, which is commonly used as baseline approach, and (iii) n-wise combinatorial approaches, which represent a widely adopted methodology for test cases derivation.

By answering RQ2, we want to show that the effectiveness of the proposed prioritization approaches does not depend on the size of the initial test suite of the X-CREATE tool. This experiment has been performed by using ten test suites of various sizes randomly selected from the initial X-CREATE test set.

To answer the first research question, we used the simple similarity and the XACML similarity metrics for ordering the test suites related to six XACML policies, and compared the effectiveness of the prioritized test suites in terms of fault detection rate. For measuring the latter, a mutation approach specifically conceived for XACML language has been used for introducing faults in the six XACML policies and the prioritized test suites have been run to

Table 1
Mutation operators [20].

ID	Description
PSTT	Policy Set Target True
PSTF	Policy Set Target False
PTT	Policy Target True
PTF	Policy Target False
RTT	Rule Target True
RTF	Rule Target False
RCT	Rule Condition True
RCF	Rule Condition False
CPC	Change Policy Combining Algorithm
CRC	Change Rule Combining Algorithm
CRE	Change Rule Effect
RPT (RTT)	Rule Type is replaced with another one
ANR	Add a New Rule
RER	Remove an Existing Rule
RUF	RemoveUniquenessFunction
AUF	AddUniquenessFunction
CNOF	Change-N-OF-Function
CLF	ChangeLogicalFunction
ANF	AddNotFunction
RNF	RemoveNotFunction
CCF	ChangeComparisonFunction
FPR	First the Rules having a *Permit* effect
FDR	First the Rules having a *Deny* effect

Table 2
Description of the six policies.

Policy name	Rules	Subjects	Resources	Actions	Environments
ASMS	117	8	5	11	3
Itrust	64	7	46	9	0
VMS	106	7	3	15	4
Continue-a	298	16	29	4	0
LMS	42	8	3	10	3
Pluto	21	4	90	1	0

Table 3
Structure of the six policies.

Policy name	♯ Policy sets	♯ Policies
ASMS	1	1
Itrust	1	1
VMS	1	1
Continue-a	111	266
LMS	1	1
Pluto	1	1

asses their capability to detect the introduced faults. In this paper for deriving a set of XACML mutants we have adopted the XACMUT tool.[3] Specifically, Table 1 lists the XACMUT mutation operators.

As said, the fault detection rate of the prioritized test suites has been compared with the ones obtained by: a greedy-optimal selection of test cases, computed based on the mutation coverage, which was able to maximize the fault detection rate (called mutation-based heuristic); a random selection of test cases (called random order); and the default X-CREATE requests order (see Section 2.2), which represents *per se* a possible prioritization technique based on an n-wise combinatorial approach (called X-CREATE order).

To address the second research question, we repeated the previously mentioned experiment considering several test suites of various sizes. In this experiment the test suites have been derived by a random selection of a different subset from the available requests set. The comparison between the different prioritized subsets has been provided again in terms of fault detection rate.

In the rest of this section we first provide details about the six XACML policies and the mutation approach used for introducing faults in them (Section 5.1). Then we describe the experiments performed to reply to RQ1 (Section 5.2) and to RQ2 (Section 5.3).

5.1. Policies and setup

Table 2 presents the sizes of the six XACML policies used in our experiments in terms of the number of subjects, resources, actions and environments in addition to the number of rules. Table 3 shows the structure of these policies in terms of policy sets and policies. Some policies contain several policy sets and the same rules appear in more than one policy.

With reference to this table, LMS is a Library Management System, VMS is a Virtual Meeting System and ASMS is an Auction Sales Management System. LMS, VMS, and ASMS are policies from three Java-based systems, which were used previously in several research papers (for instance in [21]). continue-a [22] is a policy that is used by the Continue application, a web-based conference management tool. pluto policy is used by the ARCHON system, a digital library management tool [23]. Finally, itrust policy is part of the itrust system, a health-care management system [24].

As explained previously, we have compared the effectiveness of the proposed similarity-based prioritization metrics (namely XACML similarity and simple similarity) with those of three other prioritization approaches: the random order, the mutation-based heuristic and the X-CREATE order. In particular the random order is presented in Algorithm 2. For each randomly selected request, we evaluated the number of killed mutants. To avoid experimental bias, we performed the random Algorithm 10 times and computed the average number of killed mutants on the 10 runs. We executed the requests with the original policy first and collected for each request the obtained response. Then, we run these requests with all mutated policies and collected the responses for each request. A given request kills a given mutant when the obtained response from the mutant is different from the original policy response.

Algorithm 2. Random Prioritization.

```
input: $S = \{R_1, \ldots, R_n\}$        ▷ Unordered set of $n$ XACML requests
output: $L$                             ▷ Prioritized list of $n$ XACML requests
$L \leftarrow []$
while $\#S > 0$ do
    $i \leftarrow random(1, \#S)$       ▷ Choose a random integer between 0 and $\#S$
    $L.add(R_i)$
    $S \leftarrow S \setminus \{R_i\}$  ▷ Remove $R_i$ from $S$
end while
return $L$
```

The mutation-based heuristic is a nearly optimal algorithm since it orders the requests according to the cumulative number of different mutants killed by the requests. The algorithm used in this case is similar to Prioritization Algorithm 1 (Section 4.4). Instead of using the distance to select the test cases, it relies on the numbers of mutants killed by each request (the mutation results) to order the requests. Therefore, this approach requires performing the mutation analysis by running all requests on mutated policies to get the mutation results. Then requests are ordered according to the numbers of killed mutants per requests. For the sake of consistency, we decided to always follow the alphabetical order when handling the XACML requests. The way files are

[3] A release of the XACMUT tool is available at http://labse.isti.cnr.it/tools/xacmut.

ordered by the Java virtual machine might change depending on the underlying platform (windows, Linux, etc.). In order to avoid any issue that could occur when running our Java tool on Linux based systems, Windows or MAC OS, we order files alphabetically before handling them.

Finally, the last prioritization approach that we consider is the default X-CREATE order. We consider the order in which the requests are generated by the X-CREATE tool, considered as a prioritization approach. This allows for evaluating the effectiveness of our approach compared to the default order of generation of requests.

5.2. Similarity effectiveness evaluation

In this section we discuss the results of the experiment performed to reply to RQ1 (see Figs. 3–8)

The results are depicted in a separate plot for each policy in the next six figures. The plots illustrate the cumulative number of mutants killed by each prioritized request set. They show the effectiveness of each approach, especially how effective are the first requests in improving the overall number of mutants killed. For instance, we can consider the first 200 requests and compare the number of mutants killed by each prioritized request set. After running the first 200 requests (out of the 1400 requests that are generated) for the continue-a policy, we can clearly see that the requests obtained from the XACML similarity prioritization are killing almost 1200 mutants (out of the 1800 mutants), which represents 66% of mutation score, while the other three approaches (X-CREATE, simple similarity and random prioritization) enable killing 800, which represents about 44% of mutation score.

The results that we obtained for the six policies allow us to evaluate how effective are the first tests. Specifically, the results presented in Table 4 show that: 10% of the XACML similarity-based prioritized test suite guarantees at least 50% of mutation score for 5 of the six policies (for pluto, it reaches 21%); 20% of the XACML similarity-based prioritized test suite guarantees at least 60% of the mutation score for 5 of the six policies (for pluto, it reaches 41%); 30% of the XACML similarity-based prioritized test suite guarantees at least 85% of the mutation score for 4 of the six policies (it reaches around 60% for pluto and itrust). It is out of the scope of the paper to provide a general criterion to select the best subset of the overall prioritized test suite. However, from the analysis of the above results, 20% of the test suite seems to be a good cutoff point. More detailed results are included in the additional material document, which show the numbers of mutants killed by first 10%, 20%, 30%, 40%, 50%, 60%, 70%, 80%, 90% of the prioritized test suites for all the presented prioritization criteria and policies.

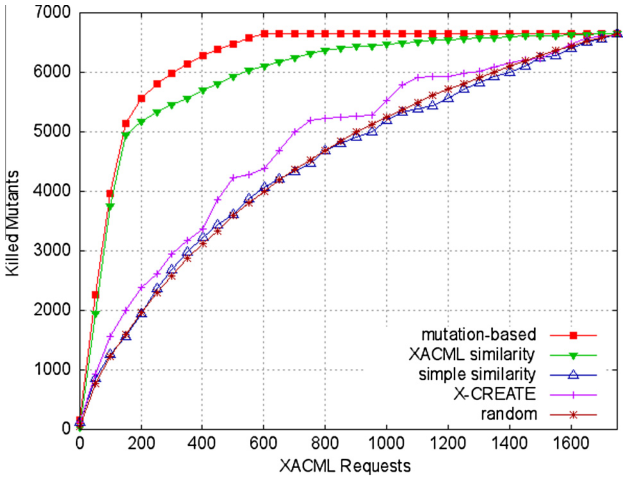


Fig. 3. ASMS policy.

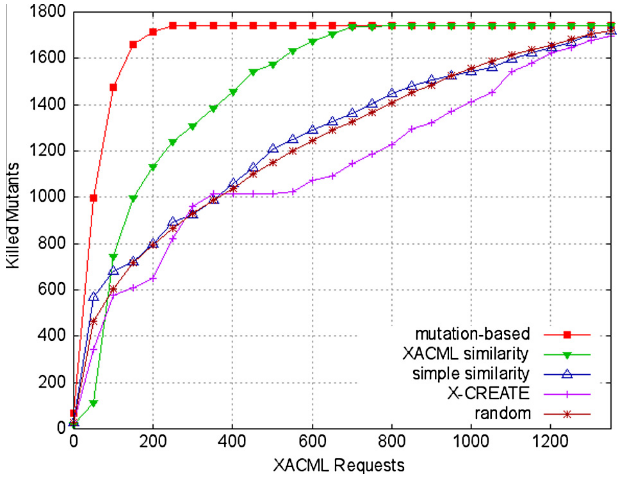


Fig. 4. Continue-a policy.

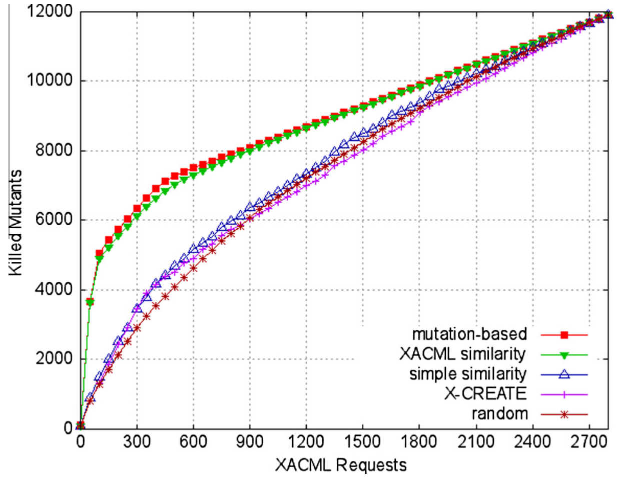


Fig. 5. Itrust policy.

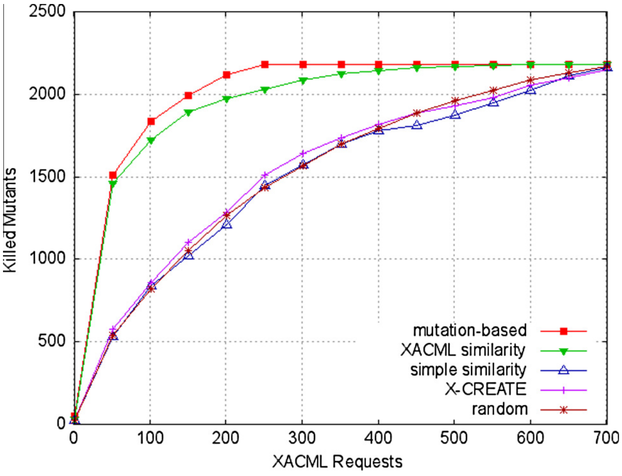


Fig. 6. LMS policy.

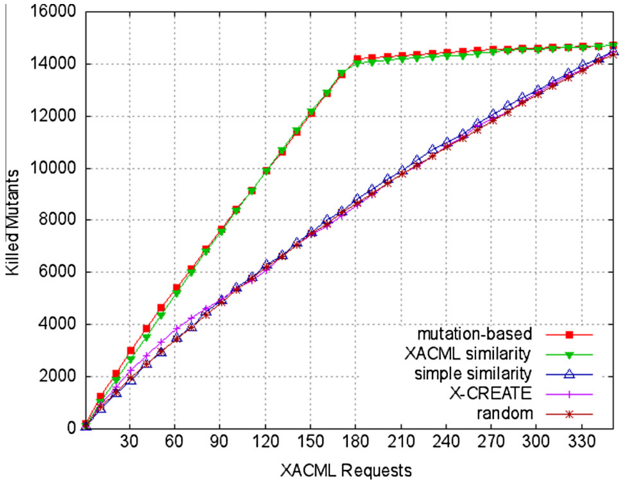


Fig. 7. Pluto policy.

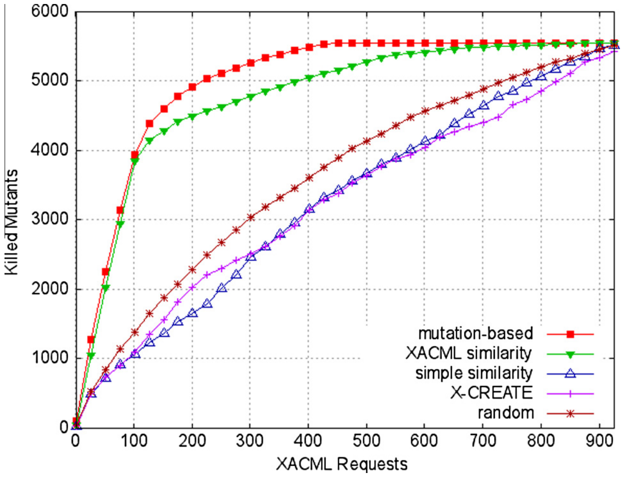


Fig. 8. VMS policy.

To evaluate the performance of the presented prioritization approaches we also computed the APFD metric that is commonly used in prioritization research. It is defined as follows [25]:

$$APFD = \frac{\sum_{i=1}^{n-1} F_i}{n \times l} + \frac{1}{2n}$$

where n is the number of test cases in the test suite T, l is the number of faults, and F_i is the number of faults detected by at least one test case among the first i test cases in T. The results in Table 5 show the APFD value of each proposed criterion for each of the six policies and confirm that the XACML Similarity approach outperforms the other approaches (except the mutation-based one).

To sum up, we can notice from the results that for all the policies the best results are, as expected, obtained from the mutation-based prioritization. Among the remaining prioritization approaches, there are three approaches leading to almost similar results; the random, the X-CREATE and the simple similarity. The plots also show that the XACML similarity outperforms these three techniques. This result is obtained for all the six policies.

From these obtained results, we can draw the following conclusions:

- Effectiveness of the XACML similarity approach: We can notice clearly that for all policies the XACML similarity provides always better results and is close to the nearly optimal solution (the mutation-based prioritization results). This indicates that taking into account the policy is very useful when it comes to test prioritization.
- Lack of effectiveness of the simple similarity: The obtained results show clearly that the simple similarity is providing poor prioritization results. Ignoring the policy and relying only on the requests content to perform similarity prioritization leads to poor results that are comparable to the random prioritization results.
- X-CREATE results are similar to random prioritization results: Interestingly, the six policies results demonstrate that the default order in which the X-CREATE tool creates the requests is providing a mutation-killing capability similar to the random one. This result is important because it shows clearly that we need to apply other prioritization approaches (like similarity) because the default order in which requests are created leads to poor results.

As a summary, the experiments that we conducted clearly recommend the use of the XACML similarity approach. It showed to be very effective for all the six policies that we used and outperformed the other prioritization approaches. In addition, the experiments confirm the need to use prioritization because the default order (the X-CREATE prioritization) is providing poor fault detection rate.

5.3. Influence of test-suite size

In this section we discuss the results of the experiment performed to reply to RQ2. We assess whether the effectiveness, measured in terms of fault detection rate of the prioritized test suites, depends on the size of the test suite. In the previous section, we showed that all the studied prioritization approaches, except of the XACML similarity, did not provide good mutation results when compared to the mutation-based prioritization. Therefore, here we only consider the XACML similarity.

For each of the six XACML policies of Section 5.1, different test suites of various sizes were selected at random from the set of requests (called here initial test population) generated by the X-CREATE tool. In particular, ten sets of size ratios 10%, 20%, 30%, 40%, 50%, 60%, 70% and 80% of the initial test population were selected per each considered policy. For each selected set, the proposed XACML similarity approach and the random order were applied. We record the APFD values for each policy and repetition of the experiment. Thus, a total of 60 values (6 policies ∗ 10 independent repetitions) per examined size were collected.

Fig. 9 shows the obtained results as box plots per selected size. Generally, the box plot representation graphically represents the distribution of the collected values. The area within the boxes represents the data that have values higher than the 25% and lower than the 75% of all the population data values. The horizontal line inside the box represents the median value. In Fig. 9 the two boxes of each graph represent the results of the XACML similarity. Each one of the graphs corresponds to the examined test suite sizes. Thus, the results evidence that XACML similarity kills a higher number of mutants than the random order. So, in reply to RQ2, we can conclude that the XACML similarity is not affected by the size of the prioritized test sets.

To investigate further RQ2 we formally compared the XACML similarity and the random approach, using the Mann–Whitney U test. This is a non-parametric statistical hypothesis test that allows for comparing two samples without making assumptions about the distribution of the underlying population. We test the hypothesis

Table 4
Mutant-kill ratios achieved by ordered sub-sets of X-CREATE requests.

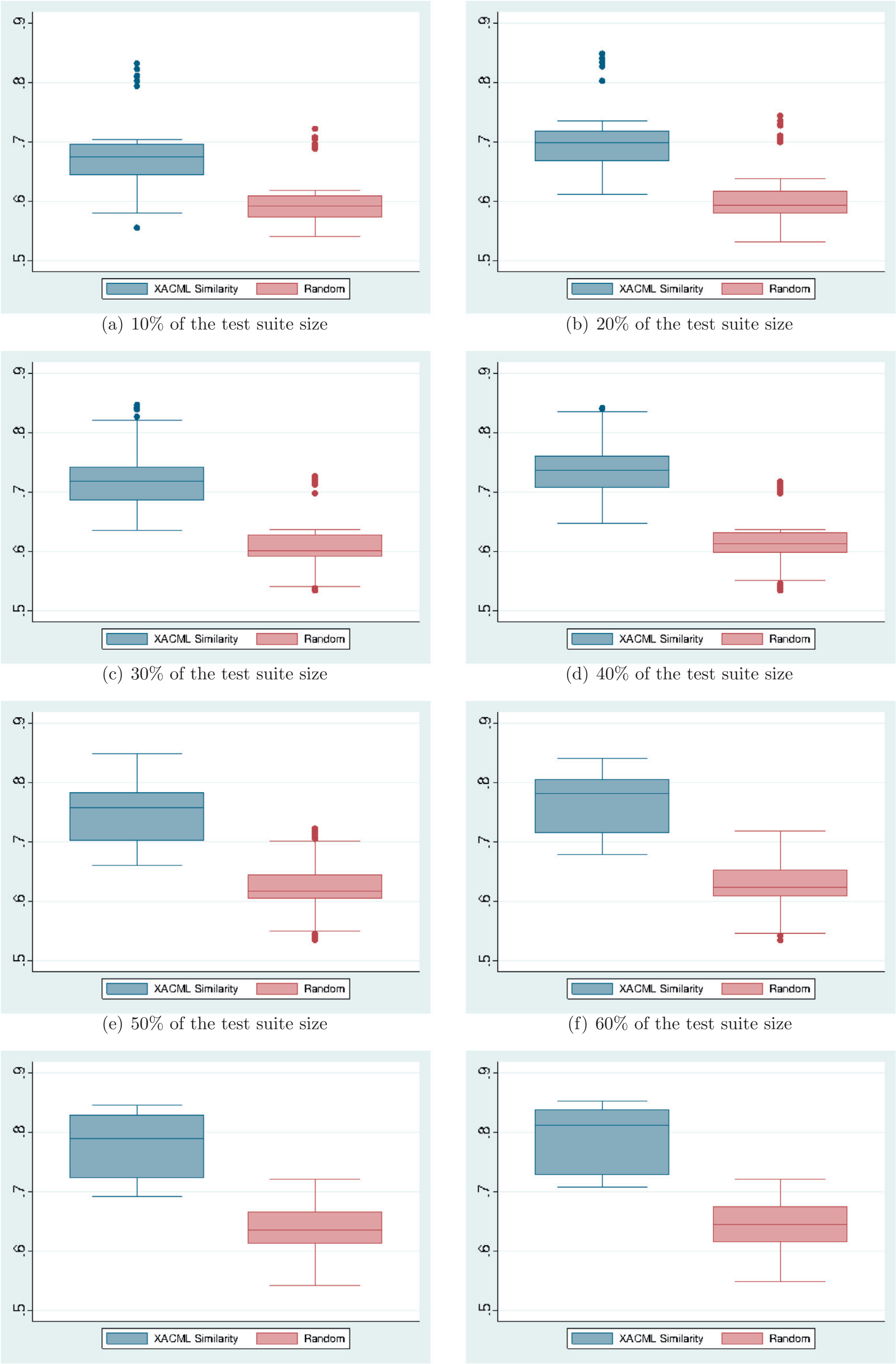
Row	Policy	%R	%M	%R	%M	%R	%M	%R	%M	%R	%M	%R	%M	%R	%M	%R	%M	%R	%M
1		ASMS																	
2		XACML requests									Killed mutants								
3		1760									6649								
4	Mutation-based	10	80	20	92	30	98	40	100	50	100	60	100	70	100	80	100	90	100
5	XACML similarity	10	76	20	83	30	89	40	94	50	96	60	97	70	98	80	99	90	99
6	Simple similarity	10	26	20	44	30	56	40	65	50	72	60	80	70	84	80	90	90	95
7	X-CREATE	10	33	20	47	30	63	40	76	50	79	60	87	70	89	80	92	90	96
8	Random	10	26	20	43	30	55	40	65	50	74	60	80	70	86	80	92	90	93
9	Policy	Continue-a																	
10		XACML requests									Killed mutants								
11		1392									1741								
12	Mutation-based	10	94	20	100	30	100	40	100	50	100	60	100	70	100	80	100	90	100
13	XACML similarity	10	53	20	73	30	85	40	93	50	99	60	99	70	99	80	99	90	100
14	Simple similarity	10	41	20	52	30	63	40	71	50	78	60	84	70	88	80	92	90	95
15	X-CREATE	10	34	20	51	30	58	40	59	50	65	60	72	70	79	80	88	90	94
16	Random	10	40	20	51	30	60	40	69	50	75	60	82	70	88	80	93	90	96
17	Policy	Itrust																	
18		XACML requests									Killed mutants								
19		2835									11949								
20	Mutation-based	10	52	20	62	30	66	40	71	50	76	60	81	70	85	80	90	90	95
21	XACML similarity	10	50	20	60	30	65	40	71	50	75	60	80	70	85	80	90	90	95
22	Simple similarity	10	26	20	41	30	51	40	59	50	68	60	76	70	83	80	88	90	94
23	X-CREATE	10	27	20	40	30	49	40	56	50	64	60	72	70	80	80	86	90	93
24	Random	10	23	20	37	30	48	40	58	50	66	60	74	70	81	80	88	90	94
25	Policy	LMS																	
26		XACML requests									Killed mutants								
27		720									2183								
28	Mutation-based	10	77	20	90	30	98	40	100	50	100	60	100	70	100	80	100	90	100
29	XACML similarity	10	73	20	86	30	91	40	94	50	97	60	98	70	99	80	99	90	99
30	Simple similarity	10	28	20	46	30	57	40	70	50	79	60	82	70	85	80	91	90	96
31	X-CREATE	10	33	20	48	30	60	40	74	50	80	60	86	70	88	80	92	90	96
32	Random	10	30	20	47	30	60	40	70	50	78	60	84	70	89	80	94	90	97
33	Policy	Pluto																	
34		XACML requests									Killed mutants								
35		360									14721								
36	Mutation-based	10	23	20	42	30	61	40	79	50	96	60	97	70	98	80	99	90	99
37	XACML similarity	10	21	20	41	30	61	40	79	50	95	60	96	70	97	80	98	90	99
38	Simple similarity	10	14	20	27	30	39	40	49	50	59	60	68	70	77	80	85	90	93
39	X-CREATE	10	17	20	29	30	38	40	48	50	58	60	67	70	76	80	84	90	92
40	Random	10	15	20	27	30	38	40	48	50	58	60	67	70	76	80	84	90	92
41	Policy	VMS																	
42		XACML requests									Killed mutants								
43		945									5550								
44	Mutation-based	10	67	20	87	30	93	40	98	50	100	60	100	70	100	80	100	90	100
45	XACML similarity	10	65	20	80	30	85	40	89	50	93	60	97	70	98	80	99	90	99
46	Simple similarity	10	18	20	28	30	41	40	53	50	63	60	71	70	79	80	87	90	95
47	X-CREATE	10	17	20	29	30	38	40	48	50	58	60	67	70	76	80	84	90	92
48	Random	10	19	20	34	30	44	40	52	50	63	60	70	70	77	80	83	90	91

Table 5
APFD values for the six policies.

Policy name	Mutation-based	XACML similarity	Simple similarity	X-CREATE	Random
ASMS	0.933	0.895	0.673	0.718	0.676
Itrust	0.748	0.742	0.645	0.626	0.627
VMS	0.904	0.864	0.594	0.591	0.657
Continue-a	0.962	0.856	0.728	0.659	0.715
LMS	0.923	0.902	0.695	0.716	0.708
pluto	0.747	0.741	0.565	0.563	0.558

that the mutation score achieved by the test cases of the XACML similarity (MS_{Sim}) is higher than the mutation score achieved by the test cases of the random order (MS_{Rand}). Thus, we test the following hypothesis ($\text{MS}_{Sim} > \text{MS}_{Rand}$) with the confidence level 95%.

Given a set of test cases of size *n*, the hypothesis test involves n comparisons of the mutation scores of the two methods (prioritization and random order). Since the random order involved ten different orderings, we compare the prioritization technique against



(a) 10% of the test suite size

(b) 20% of the test suite size

(c) 30% of the test suite size

(d) 40% of the test suite size

(e) 50% of the test suite size

(f) 60% of the test suite size

(g) 70% of the test suite size

(h) 80% of the test suite size

Fig. 9. APFD values for the different test set sizes.

the average values of these ten orders. Overall, based on the statistical analysis we can identify the following four cases:

True with significance (TS): the prioritization method kills statistically significantly more mutants than the random ordering.

True without significance (TNS): the prioritization method kills more mutants than the random ordering but without statistical significance.

False without significance (FNS): the prioritization method kills less mutants than the random ordering but without statistical significance.

False with significance (FS): the prioritization method kills statistically significantly less mutants than the random ordering.

Following these lines, we conduct 60 statistical tests (ten repetitions per subject policy for the six considered policies) per considered set size (10%, 20%, 30%, 40%, 50%, 60%, 70% and 80% of the whole test suite). The respective results are recorded on Table 6. The results show that the XACML similarity approach performs significantly better than the random orderings in all the cases. This fact signifies the ability of the proposed approach to effectively prioritize the test sets even of a small size.

6. Threats to validity

This section discusses threats to the internal, external and construct validity of the experiments presented in this paper. Concerning the internal validity, i.e., the amount of confidence on the reported results, different aspects can be considered: the used mutation operators, the employed test set, the correctness of the implementation and the tools used.

Since the effectiveness of the approach is evaluated in term of fault detection rate, 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 present study employs a combination of three different mutant sets: first, the set of mutants used in [26] (this set was adapted to XACML policies); second, the operators from Martin and Xie [27]; third, some new operators based on our most recent work [20]. However, it would be very interesting to investigate other mutants and even real faults to provide confidence in the proposed approach and reduce the threat related to the use of mutation analysis.

Another threat to our proposal is due to the employed test sets. We used those derived by X-CREATE, but it is likely that other test sets may produce different results. However, X-CREATE represents the current state of the art in XACML test generation tools. It employs combinatorial interaction testing, which is a well-established test technique in various domains.

Other threats may be attributed to the implementations of the SIMTAC tool, the XACMUT mutation tool and the X-CREATE test generation tool. These tools may have flaws, the presence of which may influence the reported results. To reduce these threats, several manual tests were performed. Additionally, at least two authors independently tested all the implemented parts.

External validity of the experiment concerns potential issues that may prevent the generalization of the results. While this is an issue concerning all empirical studies, including ours, to the authors' knowledge, the present study forms one of the largest studies conducted on XACML testing. Additionally, the six policies have quite different structures. Some have few rules whereas other ones have a large number of rules. In some cases, the number of resources is much bigger than the number of subjects and actions (this is the case for itrust and pluto policies) while in other cases it is the opposite (for VMS policy). Since similar results (our approach performs much better than random) are observed on all the cases, some confidence that our approach will behave similarly on other subjects is provided.

With respect to construct validity, i.e., threats regarding the extent of the utilized measures to the intended properties, some potential issues can also be identified. One such issue is the use of mutants as a means of effectiveness evaluation. While this is a potential problem of the conducted experiment, in practice evaluating one criterion in terms of another one is a usual practice, e.g., [28]. Since the similarity approach is independent from the employed mutants, this threat should be balanced. Moreover, using mutants for effectiveness evaluation forms a common practice in this kind of experiments, e.g., [29,30].

Table 6
Hypothesis tests for the similarity prioritization and the random orderings.

$MS_{Sim} > MS_{Rand}$ (%)	TS	TNS	FNS	FS
10	43	17	0	0
20	51	9	0	0
30	60	0	0	0
40	60	0	0	0
50	60	0	0	0
60	60	0	0	0
70	60	0	0	0
80	60	0	0	0

7. Related work

This work spans over several research directions, including: test case prioritization, access control testing and similarity approaches.

Test case prioritization. Test case prioritization relies on test cases re-ordering techniques to improve the fault detection rate at a given test execution time [22]. In [9], the authors have assessed the fault detection rate of JUnit and TSL test suites on open-source Java systems through mutation faults. This rate is impacted by mutation faults number and by test suites effectiveness to detect faults.

In [31], the authors have conducted a series of controlled experiments to evaluate test case prioritization techniques based on time constraints and fault detection rate. Their results favor the application of heuristics when the software contains considerable faults number and when the testing process has no time constraints. In [8,32], the authors have conducted experimental studies to show the effectiveness of prioritization techniques to improve fault detection rate in the context of regression testing. Our approach does not address regression testing, although it could be adapted to be applicable in a regression testing context when the access control policy evolves [33].

While most of the prioritization techniques in the literature rely on code coverage [34–36], some recent approaches have adopted different metrics. In [37], the authors use system models and system behavior to prioritize test cases. They have compared their approach with other prioritization techniques and have shown its effectiveness in early fault detection. The authors in [38] have used expert knowledge to achieve pair-wise comparison of test cases and have proposed similarity metrics, like we have done in the current work, between test cases clusters. Finally, the work in [12] improves the similarity-based test case prioritization using the ordered sequence of program elements measured by execution counts. The authors show that the proposed technique increases the rate of fault detection with respect to other coverage-based approaches.

Testing Access Control Systems. Testing Access Control Systems is a critical issue and the complexity of the XACML language specification prevents the manual specification of a set of test cases capable of covering all the possible interesting critical situations or faults. This implies the need of automated test cases generation for testing on the one side the XACML policy specification and on the other that the PDP behavior conforms to the policy specification.

Among the available proposals, the Targen tool [4] generates test inputs using combinatorial coverage of the truth values of independent clauses of XACML policy values. This approach has been proven to be more effective than random generation strategy in terms of structural coverage of the policy and fault detection rate [4].

A more recent tool is X-CREATE [13,5,14] that provides different strategies based on combinatorial approaches of the subject, resource, action and environment values taken from the XACML policy for deriving the access requests. Experimental results presented in [13] show that the fault detection rate of X-CREATE test suites is similar or higher than that of Targen test suites. Specifically, three main generation strategies are defined into X-CREATE: (i) the *Simple Combinatorial* testing strategy [5] that derives an XACML request for each of the possible combinations of the subject, resource, action and environment values taken from the policy; (ii) the *XPT-based* testing strategy [13,5] that generates requests using the structures obtained applying the *XPT* strategy [39] to the XACML Context Schema [3]; (iii) the *Multiple Combinatorial* strategy that relies on combinations of more than one subject, resource, action and environment values for generating XACML requests. This last strategy automatically establishes the number of subjects, resources, actions and environments of each request according to the complexity of the policy structure and targets the policy rules in which the effect is simultaneously dependent on more than one constraint [14]. A detailed comparison of X-CREATE test generation strategies in terms of fault detection is presented in [5,14]. Among the X-CREATE generation strategies we selected in this paper *Simple Combinatorial* for deriving test suites used to empirically validate the effectiveness of the proposed approach. This strategy is simple and easy-to-apply while at the same time it can reach the coverage of the policy values combinations. More detail about this strategy are presented in Section 2.2.

The work in [21] addresses model-based testing and provides a methodology for the generation of test cases based on combinatorial approaches of the elements of the model (role names, permission names, context names). Such approach automatically derives abstract test cases that have to be then refined into concrete XACML requests for being executed on a PDP.

Concerning the testing of the XACML PDP, the approach proposed in [17] focuses on running different XACML implementations for the same test inputs and can detect not correctly implemented XACML functionalities when different outputs are observed.

In software testing, mutation analysis [40] is commonly used to assess the effectiveness of a test suite. It consists of introducing single faults in a given program and running tests to assess their capability to detect these faults. Mutation analysis has been applied on access control policies [27,26,20] to qualify security tests. By means of mutation operators, the policy under test is modified to derive a set of faulty policies (mutants) each containing a fault. A mutant policy is killed if the response of an XACML request executed on the mutant policy differs from the response of the same request executed on the original policy. In [27] the authors define a fault model for access control policies and a set of mutation operators manipulating the predicates and logical constructs of target and condition elements of an XACML policy. They have used mutation analysis applied on access control policies to assess coverage criteria for test generation and test selection in terms of fault detection rate. In [26] the authors try to extend the mutation operators of [27], focusing on the use of a metamodel that allows to simulate the faults in the security models independently from the used role-based formalism (R-BAC or OrBAC). Finally, the work in [20] includes and enhances the mutation operators of [27,26] addressing specific faults of the XACML 2.0 language and providing a tool, called XACMUT, for the derivation of XACML mutation operators and their application to XACML policies. In this paper, the XACMUT tool has been adopted for deriving from an XACML policy a set of mutants used to assess the effectiveness of the proposed similarity-based prioritization approaches.

Similarity approaches. Similarity has been used in previous work for XACML-based policies comparison. In [41,42], the authors have defined similarity distances to enable comparing access control policies in order to locate providers that have similar policies in large scale environments like cloud systems. The number of policies that have to be evaluated at a given time can be reduced under the assumption that similar policies might provide the same decisions. Therefore they focus on comparing policies and do not compare requests as done in this current work. In fact, to the best of our knowledge, similarity has not yet been applied in the context of XACML policies testing. This heuristic has mainly been applied in the context of model-based testing. For instance, Cartaxo et al. [43] presented a strategy for automatic test case selection based on the use of a similarity function. Labeled transition systems are the model from which test cases are obtained. The similarity function *f* used is calculated by observing the number of identical transitions and the average between paths length. They, then use a greedy approach to select the test cases. On the same direction, Hemmati et al. [18,15] investigated and compared possible similarity functions that can be used for test cases selection in the context of state machine testing. The selection strategy used is based on genetic algorithms. Test cases are encoded using UML state machines with states, transitions and triggers/guards.

Similarity has also been used to cluster test cases. Sapna and Mohanty [44] used the Levenshtein distance to compare test cases and agglomerate hierarchical clustering in order to select dissimilar test scenarios with maximum coverage and fault detection rate. UML activity diagrams are the model used from which test cases are obtained. In this work, we use similarity also for improving the fault detection rate of the selected subset of requests.

Finally, in [19,45], similarity is used to generate and prioritize test suites in the context of Software Product Lines. In these works, similarity was evaluated in terms of (a) covering *t*-wise interactions [19] and (b) killing mutants related to the product line representation, i.e., feature models [45]. In our work, similarity is used to distinguish redundant test cases and prioritize them. In line with our work, Henard et al. [45] uses mutation to evaluate the effectiveness of the selected test suite. Additionally, in [19] the authors introduce a prioritization technique, called Global Maximum Distance [19], which is also used in this paper (the mutation-based strategy).

In our work, we are targeting a completely different and new context, which is XACML policies testing and we rely on similarity to prioritize XACML requests. All these previous work are however using the same heuristic and applying it to other contexts.

8. Conclusions and future work

In this paper, we presented a new approach based on similarity, and implemented into the SIMTAC tool, aiming at prioritizing tests in the context of XACML access control systems. We proposed two similarity-based prioritization metrics: the first strategy is the simple similarity, which is policy-independent and involves comparing the content of requests; the second approach is called XACML similarity and considers the applicability of the requests to the XACML policy. We performed an empirical study to evaluate the effectiveness of the simple and the XACML similarity metrics when applied to the test suites related to a set of six real-world XACML policies. The results showed that the second approach is effective and provides a mutation coverage that is significantly better than random prioritization and close to a greedy-optimal heuristic cognizant of the requests effectiveness.

In future work, we plan to investigate several other issues related to the proposed approach. In particular, we want to refine the proposed applicability relation taking into account further elements of the XACML policy such as the condition or the combining algorithm. Indeed this last plays an important role in case of policies with conflicting rules. Moreover, we plan to extend the similarity-based prioritization metrics in order to consider other test case generation strategies, also based on the combination of more than one subject, resource, action, environment.

Future work will also include further experimentation considering more XACML policies and the application of the SIMTAC tool to evaluate the effectiveness of the similarity-based prioritization metrics applied to different test suites.

Acknowledgment

This work has been partially funded by the Network of Excellence on Engineering Secure Future Internet Software Services and Systems (NESSoS) FP7 Project 256980.

Appendix A. Supplementary material

Supplementary data associated with this article can be found, in the online version, at http://dx.doi.org/10.1016/j.infsof.2014.07.003.

References

[1] TAS3 Project, Trusted Architecture for Securely Shared Services. <http://www.tas3.eu/>.

[2] NESSoS Project, Network of Excellence on Engineering Secure Future Internet Software Services and Systems. <http://www.nessos-project.eu/>.

[3] OASIS, extensible access control markup language (XACML) version 2.0, 1 February 2005.

[4] E. Martin, T. Xie, Automated Test Generation for Access Control Policies, in: Supplemental Proc. of 17th International Symposium on Software Reliability Engineering (ISSRE), 2006.

[5] A. Bertolino, S. Daoudagh, F. Lonetti, E. Marchetti, Automatic XACML requests generation for policy testing, in: Proc. of The Third International Workshop on Security Testing (SECTEST), 2012, pp. 842–849.

[6] S. Elbaum, A.G. Malishevsky, G. Rothermel, Prioritizing test cases for regression testing, SIGSOFT Softw. Eng. Notes 25 (5) (2000) 102–112.

[7] Z. Li, M. Harman, R.M. Hierons, Search algorithms for regression test case prioritization, IEEE Trans. Softw. Eng. 33 (4) (2007) 225–237.

[8] G. Rothermel, R.H. Untch, C. Chu, M.J. Harrold, Prioritizing test cases for regression testing, IEEE Trans. Softw. Eng. 27 (10) (2001) 929–948.

[9] H. Do, G. Rothermel, On the use of mutation faults in empirical assessments of test case prioritization techniques, IEEE Trans. Softw. Eng. 32 (9) (2006) 733–752.

[10] S. Elbaum, G. Rothermel, S. Kanduri, A.G. Malishevsky, Selecting a cost-effective test case prioritization technique, Software Qual. J. 12 (3) (2004) 185–210.

[11] L. Zhang, S.-S. Hou, C. Guo, T. Xie, H. Mei, Time-aware test-case prioritization using integer linear programming, in: Proc. of the Eighteenth International Symposium on Software Testing and Analysis, ACM, 2009, pp. 213–224.

[12] K. Wu, C. Fang, Z. Chen, Z. Zhao, Test case prioritization incorporating ordered sequence of program elements, in: Proc. of 7th International Workshop on Automation of Software Test (AST), 2012, pp. 124–130.

[13] A. Bertolino, F. Lonetti, E. Marchetti, Systematic XACML request generation for testing purposes, in: Proc. of 36th EUROMICRO Conference on Software Engineering and Advanced Applications (SEAA), 2010, pp. 3–11.

[14] A. Bertolino, S. Daoudagh, F. Lonetti, E. Marchetti, L. Schilders, Automated testing of extensible access control markup language-based access control systems, IET Software 7 (4) (2013) 203–212.

[15] H. Hemmati, L. Briand, An industrial investigation of similarity measures for model-based test case selection, in: Proc. of the 21st International Symposium on Software Reliability Engineering (ISSRE), 2010, pp. 141–150.

[16] C. Zhang, Z. Chen, Z. Zhao, S. Yan, J. Zhang, B. Xu, An improved regression test selection technique by clustering execution profiles, in: Proc. of 10th International Conference on Quality Software (QSIC), 2010, pp. 171–179.

[17] N. Li, J. Hwang, T. Xie, Multiple-implementation testing for xacml implementations, in: Proc. of the Testing, Analysis, and Verification of Web Services and Applications (TAV-WEB), 2008, pp. 27–33.

[18] H. Hemmati, A. Arcuri, L. Briand, Achieving scalable model-based testing through test case diversity, ACM Trans. Softw. Eng. Methodol. 22 (1) (2013) 1–42.

[19] C. Henard, M. Papadakis, G. Perrouin, J. Klein, P. Heymans, Y. Le Traon, Bypassing the combinatorial explosion: Using similarity to generate and prioritize t-wise test configurations for software product lines, IEEE Trans. Software Eng. 40 (7) (2014) 650–670. http://doi.ieeecomputersociety.org/10.1109/TSE.2014.2327020, bibsource: http://dblp.uni-trier.de.

[20] A. Bertolino, S. Daoudagh, F. Lonetti, E. Marchetti, XACMUT: XACML 2.0 mutants generator, in: Proc. of 8th International Workshop on Mutation Analysis (associated with ICST 2013), 2013, pp. 28–33.

[21] A. Pretschner, T. Mouelhi, Y.L. Traon, Model-based tests for access control policies, in: Proc. of First International Conference on Software Testing, Verification (ICST), 2008, pp. 338–347.

[22] G. Rothermel, R.H. Untch, C. Chu, M.J. Harrold, Test case prioritization: an empirical study, in: Proc. of IEEE International Conference on Software Maintenance (ICSM), IEEE, 1999, pp. 179–188.

[23] K. Maly, M. Zubair, M. Nelson, X. Liu, H. Anan, J. Gao, J. Tang, Y. Zhao, Archon – a digital library that federates physics collections.

[24] Realsearch group at NCSU, iTrust: Role-Based Healthcare. <http://agile.csc.ncsu.edu/iTrust/wiki/doku.php>.

[25] L. Zhang, D. Hao, L. Zhang, G. Rothermel, H. Mei, Bridging the gap between the total and additional test-case prioritization strategies, in: Proc. of the International Conference on Software Engineering (ICSE), 2013, pp. 192–201.

[26] T. Mouelhi, F. Fleurey, B. Baudry, A generic metamodel for security policies mutation, in: Proc. of Software Testing Verification and Validation Workshop (ICSTW), 2008, pp. 278–286.

[27] E. Martin, T. Xie, A fault model and mutation testing of access control policies, in: Proc. of 16th International Conference on World Wide Web (WWW), pp. 667–676.

[28] D.F. Yates, N. Malevris, An objective comparison of the cost effectiveness of three testing methods, Inform. Software Technol. 49 (9) (2007) 1045–1060.

[29] J.H. Andrews, L.C. Briand, Y. Labiche, A.S. Namin, Using mutation analysis for assessing and comparing testing coverage criteria, IEEE Trans. Softw. Eng. 32 (8) (2006) 608–624.

[30] H. Do, G. Rothermel, On the use of mutation faults in empirical assessments of test case prioritization techniques, IEEE Trans. Softw. Eng. 32 (9) (2006) 733–752.

[31] H. Do, S. Mirarab, L. Tahvildari, G. Rothermel, The effects of time constraints on test case prioritization: a series of controlled experiments, IEEE Trans. Softw. Eng. 36 (5) (2010) 593–617.

[32] S. Elbaum, A.G. Malishevsky, G. Rothermel, Test case prioritization: a family of empirical studies, IEEE Trans. Softw. Eng. 28 (2) (2002) 159–182.

[33] J. Hwang, T. Xie, D. El Kateb, T. Mouelhi, Y. Le Traon, Selection of regression system tests for security policy evolution, in: Proc. of the 27th International Conference on Automated Software Engineering (ASE), 2012, pp. 266–269.

[34] A. Kaur, S. Goyal, A genetic algorithm for regression test case prioritization using code coverage, Int. J. Comput. Sci. Eng. 3 (5) (2011) 1839–1847.

[35] D. Leon, A. Podgurski, A comparison of coverage-based and distribution-based techniques for filtering and prioritizing test cases, in: Proc. of 14th International Symposium on Software Reliability Engineering (ISSRE), 2003, pp. 442–453.

[36] K.R. Walcott, M.L. Soffa, G.M. Kapfhammer, R.S. Roos, Timeaware test suite prioritization, in: Proc. of the 2006 International Symposium on Software Testing and Analysis, 2006, pp. 1–12.

[37] L. Tahat, B. Korel, M. Harman, H. Ural, Regression test suite prioritization using system models, Softw. Test. Verif. Reliab. 22 (7) (2012) 481–506.

[38] S. Yoo, M. Harman, P. Tonella, A. Susi, Clustering test cases to achieve effective and scalable prioritisation incorporating expert knowledge, in: Proc. of the 18th International Symposium on Software Testing and Analysis, 2009, pp. 201–212.

[39] A. Bertolino, J. Gao, E. Marchetti, A. Polini, Automatic test data generation for XML schema-based partition testing, in: Proc. of Second International Workshop on Automation of Software Test (AST), 2007, pp. 4–10.

[40] Y. Jia, M. Harman, An analysis and survey of the development of mutation testing, IEEE Trans. Softw. Eng. 37 (5) (2011) 649–678.

[41] D. Lin, P. Rao, E. Bertino, J. Lobo, An approach to evaluate policy similarity, in: Proc. of the 12th ACM Symposium on Access Control Models and Technologies (SACMAT), 2007, pp. 1–10.

[42] D. Lin, P. Rao, R. Ferrini, E. Bertino, J. Lobo, A similarity measure for comparing XACML policies, IEEE Trans. Knowl. Data Eng. 25 (9) (2013) 1946–1959.

[43] E.G. Cartaxo, P.D.L. Machado, F.G.O. Neto, On the use of a similarity function for test case selection in the context of model-based testing, Softw. Test. Verif. Reliab. 21 (2011) 75–100.

[44] P.G. Sapna, H. Mohanty, Clustering test cases to achieve effective test selection, in: Proc. of the 1st Amrita ACM-W Celebration on Women in Computing in India (A2CWiC), 2010, pp. 15:1–15:8.

[45] C. Henard, M. Papadakis, G. Perrouin, J. Klein, Y.L. Traon, Assessing software product line testing via model-based mutation: an application to similarity testing, in: ICST Workshops, 2013, pp. 188–197.