CHAPTER FIVE

Inroads in Testing Access Control

Tejeddine Mouelhi*, Donia El Kateb†, Yves Le Traon†
*itrust consulting, Berbourg, Luxembourg
†Interdisciplinary Research Centre, SnT, University of Luxembourg, Kirchberg, Luxembourg

Contents

Abstract

In the last few years, a plethora of research has addressed security testing issues. Several commercial tools have emerged to provide security testing services. Software security testing goes beyond functional testing to reveal flaws and vulnerabilities in software design and behavior. Access control is a major pillar in computer security. This chapter pursues the goal of describing the landscape in the research area of access control testing. We provide an outline of the different existing research over the literature according to the taxonomy reflecting the different phases of common software testing processes

Advances in Computers, Volume 99
ISSN 0065-2458
http://dx.doi.org/10.1016/bs.adcom.2015.04.003

(generation, selection, prioritization, quality assessment, regression). We also provide an outline of some existing initiatives that support usage control besides access control by testing obligation policies. Finally, we point out future research directions that emerge from the current research study. Through this work, we aim at providing useful guidelines for software testers to improve the current trends in access control testing.

1. INTRODUCTION

Software security is one major concern that is required to build trustworthy software systems. In the last decades, we have witnessed an increasing interest in the security testing research area. Several researchers have explored this topic by providing new solutions in terms of security modeling, security features development, and the specification and implementation of the security mechanisms that have to be embedded in software systems. In parallel to the emergence of security concerns, security testing has also gained a considerable interest as it has to be developed conjointly to software security hardening. As a matter of fact, it is crucial to guarantee that the security mechanisms that are in place are correctly implemented. Testing these security mechanisms is very important in order to avoid ending up with security flaws inside the system or the application.

Access control is one of the major and the most critical security mechanisms. It ensures that only eligible users are able to access protected resources in a given system. This book chapter explores the landscape of access control testing and shows advances in access control testing approaches.

We start by providing recent advances in access control testing by surveying recent contributions in this research domain. We present the research contributions according to how they fit in a given research process. In a nutshell, the process of testing access control implemented in a given system or application follows the different steps highlighted in Fig. 1. The first and the most important step aims at *generating* a set of test cases that have to be exercised on the system under test.

Based on real-world applications, a large number of test cases are generated. Due to budget, time, and resources constraints, testers have to choose the tests that have to be run among all the generated tests. The subset of test cases to be run is defined based on business-related criteria according to available budget, computing resources, and the time allocated to testing. Commonly, there are two options, either *selecting* a fixed number of tests

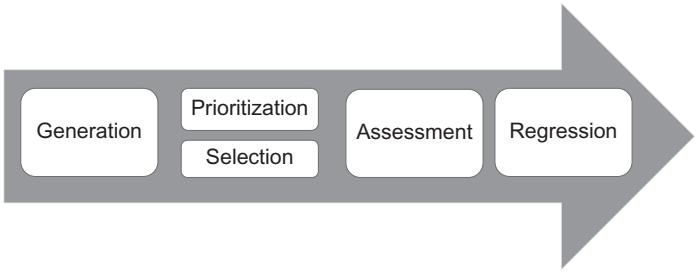

Figure 1 Testing process in a nutshell.

or ordering (*prioritizing*) tests. When *prioritizing* tests, the tests that have highest priority are executed first until the resources that are available for testing such as time or budget are consumed. Finally, once tests are executed and their verdict is checked, we need to assess the quality of these tests to provide guarantee that the test suite is of high quality. Tests assessment enables also to evaluate the fault-detection capability of tests cases. This book chapter goes first through the overall testing process by providing a detailed description of existing research contributions that aim at *generating*, *selecting*, *prioritizing*, and *assessing* test cases. Second, we provide an overall view of international projects which tackled security testing and the emerging commercial products for security testing.

Third, we describe ongoing research that extends the work on access control testing to encompass usage control testing.

We conclude this chapter by discussing the main security testing challenges that are worth exploring in the near future. The remainder of this chapter is organized as follows. In Section 2, we give an overview about access control concepts and mechanisms by focusing on the XACML policy model. In Section 3, we go through the different approaches for access control testing according to a classification according to test targets. Section 4 outlines the research proposals in each step of common testing processes. Section 5 gives an overview about usage control testing. Section 6 discusses future research challenges and finally Section 7 concludes this work.

2. ACCESS CONTROL

In the last few years, XACML (eXtensible Access Control Markup Language) has gained momentum as a standard to develop security solutions. In this section, we introduce key concepts related to access control, XACML architecture, and policy language.

2.1 Access Control as a Security Concept

Unauthorized access to sensitive data is one of the main challenging issues in IT security. Access control is a security mechanism that regulates the actions that users can/cannot perform on system resources. Access control policies are used to manage authorizations in the objective to protect sensitive data. Access control research spans mainly over access control policies models [1], access control policies enforcement mechanisms [2, 3], and access control policies languages definition [4]. Access control policies are defined based on several access control models such as Role-Based Access Control (RBAC) [5], Mandatory Access Control (MAC) [6], Discretionary Access Control (DAC) [7], and Organization-Based Access Control (OrBAC) [8]. Access control policies are specified in various policy specification languages such as the XACML and Enterprise Privacy Authorization Language [9]. An access control policy is composed of authorization rules that regulate the access to data and services. At the decision making time, a request to access a service/resource is evaluated against the rules in the policy. A response is then sent to the user, which authorizes/prohibits her/him to/from access/accessing the requested resource.

2.2 Security Policies

In the last few years, XACML has gained momentum as a standard to develop security solutions. Several commercial and open source solutions have been developed to help build access control systems.

2.2.1 XACML Model

XACML proposes a conceptual model of an access control architecture and defines the interactions between the components in this conceptual model. It also defines an access control policy language and a protocol for access control requests and responses. XACML policy-based systems rely on the separation of concerns, by implementing independently a software system and its associated security policy. Such separation eases policy management and increases the degree of policy interoperability with heterogeneous platforms. It also limits potential risks arising from incorrect policy implementation or maintenance when the policy is hard-coded inside the business logic. In the context of policy-based systems, a Policy Enforcement Point (PEP) is located inside an application's code (i.e., business logic of the system). Business logic describes functional algorithms to govern information exchange between access control decision logic and a user interface

(i.e., presentation). To determine whether a user can access which resources, a request is formulated from a PEP located in an application code. Given a request, a Policy Decision Point (PDP) evaluates the request against an access control policy and returns its access decision (i.e., permit or deny) to the PEP.

XACML architecture is based on the following components:

- Policy Administration Point (PAP): It is the policy repository which sends policies to the PDP.
- PDP: The PDP is responsible for making decisions based on the collected information from other actors.
- PEP: It receives an access request whenever a service which is regulated by a security policy is called, sends a request in an XACML format to the PDP and enforces the decision sent by the PDP.
- Policy Information Point (PIP): The PIP retrieves necessary attributes from external resources (i.e., LDAP).
- Context Handler: It transforms requests/responses in an XACML format.

Figure 2 presents the interactions between the different components to handle an access control request: (1) Policies are written and managed in the

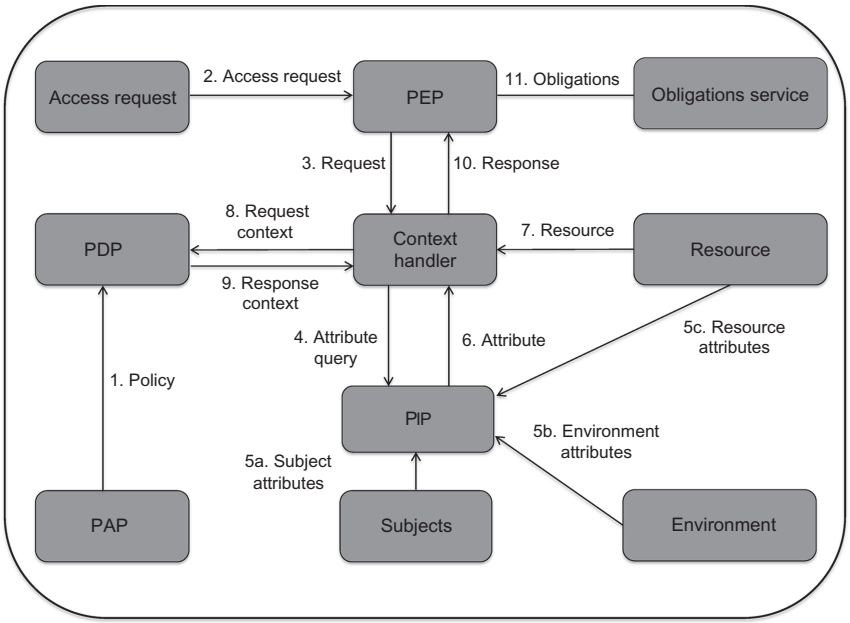

Figure 2 XACML data flow.

PAP. The PDP will fetch the policies in the PAP in each access control request evaluation. (2) The PEP is triggered whenever a service which is regulated by a security policy is called. (3) The PEP sends the request to a context handler that transforms the native format of the request into an XACML format. (4) The context handler sends a request to the PIP to collect attributes. (5) The PIP obtains the requested attributes from subject, resource, and environment. (6) The PIP sends the attributes to the context handler. (7) The context handler may include the resource in the context. (8) The context handler sends an XACML request to the PDP for evaluation. (9) The request is evaluated against the policies in the PAP. (10) The context handler returns the response to the PEP in a format that can be interpreted by the PEP. (11) If some obligations are returned with the decision, the PEP has to discharge those obligations.

2.2.2 XACML Policies

XACML policy has a hierarchical structure as shown in Fig. 3. On the top level of the this structure, a *policy set* can contain 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*

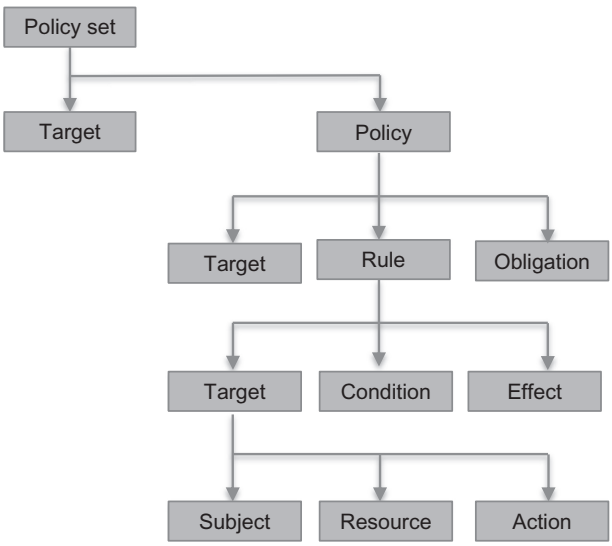

Figure 3 XACML policy structure.

(*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, and 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 enables to resolve conflicts when there is more than one rule that can be applicable to a given request. For instance, if the *permit-overrides* algorithm is used:

- If there is one single rule that is evaluated to permit, the permit decision takes precedence regardless of the result of the evaluation of other rules.
- If one rule is evaluated to Deny and all other rules are evaluated to NotApplicable, the final decision is Deny.
- 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. 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. Listing 1 illustrates an XACML policy with one rule. The rule (lines 26–58) states that a student can borrow and return books from the library. Listing 2 illustrates an XACML request in which a student requests to borrow a book.

2.3 Beyond Access Control: Usage Control

Usage control extends the notion of access control to consider what can happen to the data in the future [10]. A security policy reflects usage control concepts [11] when it includes some actions that have to be carried out before the access (i.e., the user has to authenticate before accessing a Web site), during the access (i.e., the user has to keep an open window while he is accessing a Web site), or after access (i.e., the user has to submit a form after his access).

In security policies paradigm, those actions are usually referred to as (pre, ongoing, post) obligations [12, 13]. They accurately allow to extend the

```
<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>
 </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.

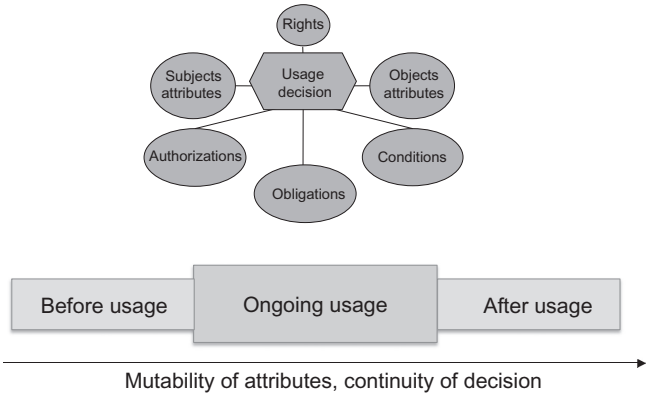

Figure 4 UCON model.

notion of access rights with related duties, called obligations. Obligations, which have been introduced in Ref. [14], are considered as an important research direction in the domain of usage control [15].

A complete security policy should encompass both rights and duties, both access authorizations and obligations. Usage control model (UCON) [16–18] is a popular model that is built based on the following concepts that we illustrate in Fig. 4:

- *Continuity of decision:* Access control is verified before and during the access. Access to resources can be revoked after it has been granted due to a change of some object or subject attributes.
- *Mutability of attributes:* Subject's or object's attributes can be mutable (i.e., subject name) or immutable (i.e., resource cost). Immutable attributes can be updated before, during, or after the access.

The model is based on a mapping of subjects and objects to access rights. Rights are evaluated based on (1) authorizations which are predicates on subject and object attributes, (2) conditions which represent predicates on environmental attributes, and (3) obligations which represent actions that have to be performed before or during access.

3. TEST TARGETS WHEN TESTING ACCESS CONTROL

XACML policy specification language defines access control policies in an XML format and defines a standardized way to exchange requests/responses. It relies on an abstract architecture consisting of abstract

components interacting with each other to handle a decision making process. XACML relies on a standardized encoding since it enables to encode a policy independently from the underlying platform, to make it thus interoperable with heterogeneous platforms. In XACML architecture, the policy is externalized from the application code and from the decision engine. This eases the maintenance of software systems since the update of the policy, usually a frequent task, can be done without changing the system's implementation.

The main components in the access control architecture are the policy, the evaluation engine that evaluates the policy against access control requests, and the underlying implementation. A comprehensive access control framework has to cover these three main components. The massive body of research in access control testing can be classified according to the aforementioned main components. In what follows, we survey the research contributions that have targeted each element in an access control architecture.

3.1 Testing the Policy

The domain of testing security enables to verify that the policy has to behave as expected. Testing XACML access control policies is complex due to the verbosity of the language, the recursive structure of XACML policies and the combining algorithms.

In Ref. [19], Martin has proposed an approach for policy testing based on the analysis of access control responses. Access control requests are triggered and a policy specification error is detected in the policy specification if the access control response related to the triggered request is different from the expected response.

3.2 Testing the Implementation

In most cases, PEPs are implemented manually, which can introduce errors in policy enforcement and lead to security vulnerabilities. To systematically test and validate the correct enforcement of access control policies in policy-based system, Mouelhi *et al.* [20] have used mutation analysis to verify the correct policy enforcement.

3.3 Testing the Evaluation Engine

Assuming that the policy specification is correct, Daoudagh *et al.* [21] evaluate a given policy evaluation engine like Sun's XACML implementation[1]

[1] http://sunxacml.sourceforge.net/.

by comparing expected responses to the real access control responses that result from the evaluation of requests against the policy. The approach has not identified any error in the evaluation engine in their conducted evaluation; however, the approach can be used to test any policy evaluation engine.

4. ACCESS CONTROL TESTING

Access control testing [22] is based on the evaluation of actual access control responses against expected responses. Tests inputs are access control requests that are evaluated by the PDP against the access control policy. The test outputs are the authorization responses that testers compare against what they expect in terms of authorization response. In what follows, we revisit recent advances in the main building blocks of access control testing by starting with test qualification, then test generation, selection, prioritization, and finally regression testing.

4.1 Access Control Test Qualification

Mutation is widely known and used technique in software testing in general. The aim is to evaluate the test quality in terms of fault detection. A good test suite should be able to detect all faults in a given program. As shown in Fig. 5, to evaluate the effectiveness of the tests, faults are seeded in the program. A unique fault is seeded each time, leading to create a faulty version of the program that we call mutant. Then, tests are executed against these mutants to detect the seeded faults. A good test suite should detect all seeded

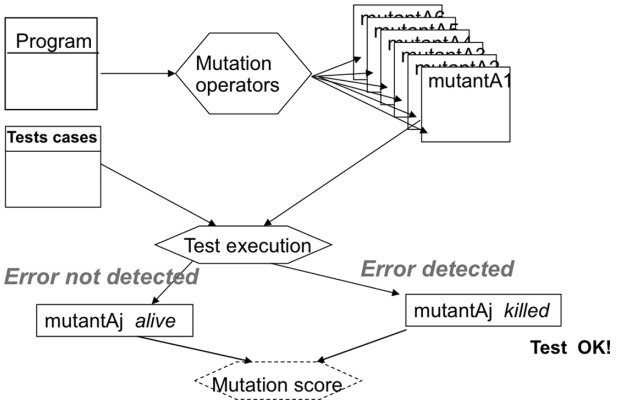

Figure 5 Mutation analysis process.

faults. In some particular cases, the mutated program is semantically equivalent to the original program. In this case, it behaves like the original program and tests cannot differentiate it from the original. These so-called equivalent mutants should be detected and removed. Finally, a mutation score is computed to evaluate the effectiveness of the test suite. Usually, faults are seeded using what we call mutation operators, which seed a very specific fault.

Mutation analysis was adapted and applied in the context of access control testing [23–25]. In this chapter, we present how it was applied to XACML policies. Martin and Xie [23] proposed a first approach to mutating XACML policies. This work was later extended by Bertolino *et al.* [25] with additional mutation operators by including the operators proposed in Ref. [24]. They also performed a better assessment of the proposed mutation operators.

The mutation process for access control testing is described in Fig. 6. Instead of mutating directly the program, the strategy in this case is to mutate the access control policy. Then, since the policy is included inside the code and used to evaluate the requests, the objective of the security test cases will be to detect seeded faults in the policy. A good test suite should be able to detect all faults seeded in the access control policy rules. It is important to highlight the fact that mutation should be applied only to the security part,

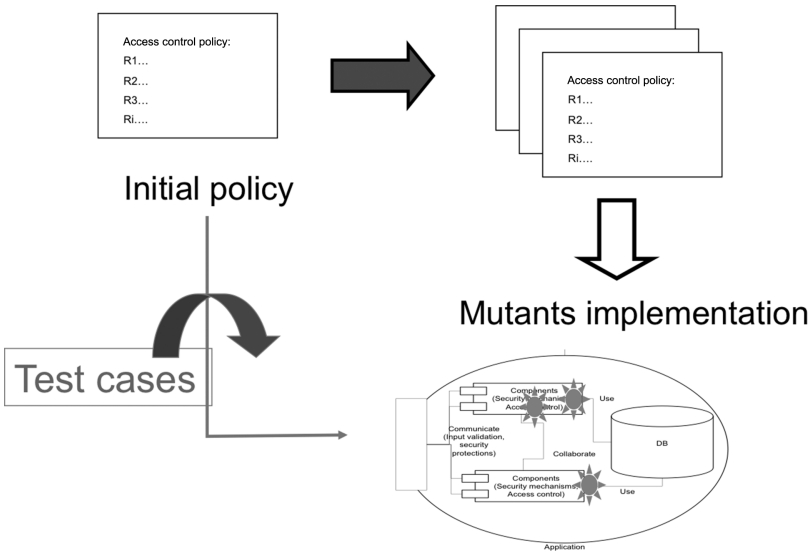

Figure 6 Mutation analysis for AC process.

in this case the XACML policy by mutating the rules. In fact, mutating the application code will not provide useful faults from the access control point of view because we are not testing the system' functionality but rather the implementation of the policy.

The mutation operators are specific to the XACML language and target the modification of the rules and the way response is computed. The list of these operators and their description is shown in Table 1. Most of these operators change the rules, for instance, by impacting the rule effect (inverting deny into permit and vice versa) or changing policy or rule combining algorithm. An interesting operator is definitely the Add New Rule (ANR) operator, which changes the policy by adding a new rule. This operator is an interesting one because it is difficult to detect. Indeed, in order to detect that the policy contains an additional rule, we need to create a test case that checks this added rule. Therefore, the test request should contain the values included in that added rule.

Table 1 XACML Mutation Operators [25]

ID	Description
PSTT, PSTF	Policy Set Target True/False
PTT, PTF	Policy Target True/False
RTT, RTF	Rule Target True/False
RCT, RCF	Rule Condition True/False
CPC, CRC	Change Policy/Rule Combining Algorithm
CRE	Change Rule Effect
RPT	Rule Type is replaced with another one
ANR	Add a New Rule
RER	Remove an Existing Rule
AUF, RUF	Add/Remove Uniqueness Function
CNOF	Change N-OF Function
CLF	Change Logical Function
ANF, RNF	Add/Remove Not Function
CCF	Change Comparison Function
FPR, FDR	First the Rules having a *Permit*/*Deny* effect

4.2 Access Control Test Generation

In the last few years, Model-Driven Engineering (MDE) [26] has appeared as a paradigm that promotes models as the central key element of all the phases of software development. MDE enables to build an abstraction layer to handle complex software systems and to automate various tasks. Model-Driven Architecture (MDA) illustrates a normalization of MDE that has been proposed in 2000 by OMG (Object Management Group) [27]. MDA offers several techniques that allow denying families of languages and supporting tools. MDE has introduced a new vision to the testing activities which are considered as a core activity in the software development cycle. This vision is illustrated through model-based testing [28] which refers to the automatic generation of test cases based on requirements models. Model-based testing requires to invest effort on building models that can be used to derive test cases. However, this effort is repaid through automation and easiness in capturing the changes in the model.

Model-based testing has been used in the context of security testing. In Ref. [29], Pretschner *et al.* have used the policy model to generate tests. Abstract test cases are derived from all the rules in the policy by a combination of roles, permissions, and contexts. Concrete test cases are derived from abstract test cases. The whole process for security testing is presented in Ref. [30]. The authors use requirements elicitation to build a security model from the application. This model is transformed into platform-specific security code that is woven using aspect-oriented programming [31] into running code. The security tests are derived from the requirements to test the code. A model-based approach based on Petri nets has been proposed by Xu *et al.* in Ref. [32]. The authors have introduced contracts and access control rules into Petri nets and generated access control test cases from Petri nets models. A MIM (Model-Implementation Mapping) description is used to map the model elements into executable tests.

To facilitate the task of test generation to test firewall policies, Hwang *et al.* [33] have defined four techniques for packets generation. The first technique is based on packet generation and aims at generating entities within a given specific domain. The second and the third techniques are based on local and global constraint solving. The local constraint technique generates packets to maximize the satisfaction of constraints defined at the rules level, while the global constraint technique aims at maximizing the satisfaction of the constraints defined at the policy level. The fourth technique automatically generates test cases based on boundary values.

In Refs. [34, 35], Martin *et al.* have proposed a framework to automate tests generation for access control policies. The framework takes as input different policy versions and uses a change impact analysis tool to derive policy portions that are syntactically different. For two different policy versions, a request is generated that provides different responses when evaluated with two policies versions. Requests are finally reduced by identifying the minimal set of test cases that covers all the requirements that have to be satisfied to reach a defined policy coverage criteria.

In Ref. [36], the authors have proposed a model-based testing approach to test PolPA authorizations systems. PolPA authorizations systems are based on a process-algebraic language to specify policies according to UCON [17]. The policy is transformed into a tree structure and the tests are derived from a depth-first exploration of the tree.

In the context of XACML policies testing, Bertolino *et al.* [37] have proposed the X-CREATE tool which proposes two strategies for XACML request generation. The first strategy is called XPT and is based on XACML context schema analysis. Requests are generated with the respect to border values of the different elements in the XACML context schema. The combinatorial strategy is based on pairwise, three-wise, and four-wise combinations of subject, resource, action, and environment elements in the XACML policy.

In Refs. [38, 39], Mallouli *et al.* have proposed a framework for security tests generation that they have applied on a Weblog system. The generated tests check the conformance of the policy specification with the business logic. The business model is expressed through an extended finite-state machine (EFSM). Prohibitions, permissions, and obligations are added to the EFSM as transition or restriction predicates. The authors have developed the SIRIUS test generation tool to derive tests automatically from the system model.

In Ref. [40], Masood *et al.* have proposed approaches to reduce tests that are generated from finite-state models. The approaches are based on heuristics and random selection of paths in the policy model. First-order mutation has been used to evaluate the fault detection capability of the selected test cases.

In Ref. [41], Brucker *et al.* have used the HOL-TESTGEN tool to transform high-level requirements into a formal specification that is encoded in higher-order logic (HOL) and convertible into sequence of test cases.

4.3 Access Control Test Selection

The selection of relevant tests among a test suite is a very challenging task. A selection process usually is based on effectiveness criteria such as performance, execution cost, fault detection, or coverage. Bertolino *et al.* [42] have proposed an approach for XACML test selection based on coverage criteria. The XACML rule coverage criterion is based on selecting tests that match the Rule Target Sets. The Rule Target Set is the union of the target of the rule, and all enclosing policy and policy sets targets.

ALGORITHM 1 Coverage-Based Selection of Test Cases [42]

```
1:  input: S = {Req_1,...,Req_n}          ▷Unordered set of n XACML
    requests
2:  input: P                                          ▷The XACML policy
3:  output: Result          ▷List of m selected XACML requests with m ¡ n
4:  Result ← {}
5:  TargetsConds ← computeAllRulesTargetsConds(P)
6:  i ← 0
7:  while i < TargetsConds.size() do
8:    ContainsReq ← False
9:    j ← 0
10:   while !ContainsReq do
11:     ReqTarget_j ← extractReqTarget(Req_j)
12:     if containsReq(TargetCond_i, ReqTarget_j) then
13:       Result ← Result ∪ {Req_j}
14:       ContainsReq ← True
15:     end if
16:     j ← j + 1
17:     if then j == n
18:       Break loop
19:     end if
20:   end while
21:   i ← i + 1
22: end while
23: return Result
```

According to the XACML specification, in order to match the rule target, requests must first match the enclosing policy and policy sets targets (note that there could be several enclosing policy sets). Here are the different configurations in terms of rule target matching.

- If a rule R_1 contains no condition and if it has a target containing the elements $\{\text{Subject}_1,\text{Action}_1,\text{Resource}_1\}$ and the policy and policy set

targets related to the rule are both empty, then in order to match R_1, a request should contain the same values of {Subject_1,Action_1, Resource_1}.

- If a rule R_1 contains no condition and if the rule target has several subjects, resources, actions, and environments and the enclosing policy and policy set targets are empty, the request should include a subject contained in target subjects set, a resource contained in the target resources set, an action contained in the target actions set, and an environment contained in the target environments set in order to cover the rule target.
- Finally, if the Rule Target Set of a rule R_1 is empty and its condition is evaluated to True or False, all requests are covering the rule R_1.

Algorithm 1 is used for test cases selection. The algorithm takes as input the Rule Target Sets and a set of access requests. Then, it iterates through the requests and selects the ones that match one Rule Target Set according to the aforementioned configurations. However, the coverage criteria that have been defined in Ref. [42] do not take into consideration the combining algorithms, which play an important role when it comes to selecting which rule applies in case of conflicts. The impact of combining algorithms on the quality of the selected test cases in terms of fault detection effectiveness has to be explored. This impact can be investigated by using policies that contain conflicting rules.

In the context of access control tests selection, Mouelhi *et al.* [20] select security tests among the set of functional tests by tracking the test cases that are impacted by security rules. Basically, the authors mutate each rule in the policy for instance by inverting the prohibition rule to a permission rule or vice versa and they identify the tests that kill the mutants. Those tests are considered as security tests.

4.4 Access Control Test Prioritization

Test prioritization is a very important task in the testing process. Indeed, usually projects have limited and strict budget and time resources and since exhaustive testing is practically impossible, it is important to order the tests cases and run the first tests until the resources are exhausted. This is exactly what prioritizing tests is about, ordering test cases in terms of a given testing criteria, for instance fault detection. If for a given policy, there are let us assume 10K test cases and only the first 100 tests can be executed and checked manually due to time constraint, and then it is important to choose the first 100 tests, which achieve the highest fault detection capability. To

measure the fault detection capability, it is possible to rely on mutation analysis and then we are able to compare a given prioritization approach to the random prioritization, considered as baseline.

This section presents in details the work done recently by Bertolino *et al.* [43]. It shows the approach in a nutshell and then presents the formal description of the algorithms that were used to order the test cases. Finally, it shows the main limitation and potential improvements.

4.4.1 The Similarity-Based Prioritization Strategy

For XACML policies, there is a recent work from Bertolino *et al.* [43], which proposes a novel approach for prioritizing test cases based on similarity. The main idea of this work boils down to ordering the tests by considering the test cases, which are different from each other, putting first the most different test cases, since a request in XACML contains usually a subject, an action, a resource, and an environment. The objective is to choose the combinations containing different values. Concretely, a distance is computed between each given test case and the others; then, test cases having the largest distance with the other test cases are selected and put first in the list. Then, the process continues for the remaining tests by calculating again the distances and selecting the most interesting test cases in terms of distance.

It is important to highlight the fact that the way the distance is computed has an important impact on the resulting order. Indeed, the distance computation algorithm could include the notion of coverage in order to give priority to test cases, which increase the policy coverage. This way, the tests will trigger the rules defined inside the policy instead of triggering the default rule effect or a rule that is not defined in the policy.

Bertolino *et al.* proposed two criteria to perform similarity-based prioritization in the context of XACML testing, namely the simple similarity and the XACML similarity.

The simple similarity

The simple similarity is straightforward. Given two requests, the distance is equal to 0, when the four values of the two requests (the subject, the action, the resource, and the environment) match. This is the case, when the two requests are identical. If all request values do not match, then the distance value is 4. In the other cases, when corresponding values match partly the distance can vary between 1 and 3. Therefore, the distance value is computed simply by comparing one to one each of the two requests attributes.

XACML similarity

For each request, an applicability matrix is computed. The size of this matrix is $5 \times n$ (n is the number of rules). The five rows contain values related to applicability of the request to subject, action, resource, environment, and the full rule. A request is applicable when it matches the given element (subject, action, etc.). In addition to this matrix, a priority value is computed for all pairs of requests and contains applicability value of the two rules combined. Finally, the distance matrix is computed by adding all values in applicability matrix to the priority value and simple similarity value. If the simple similarity equals to 0, then the value is set to 0; otherwise, it is computed as stated before.

4.4.2 Assessment of the Approach

Assessment of this proposed approach was performed on real work case studies. A set of real world access control policies written in XACML were used in addition to mutation analysis in order to evaluate the effectiveness of the proposed criteria and compare them to the random prioritization, which involves ordering the test cases randomly. The random ordering is considered as a baseline. In addition, the prioritization is compared to the near-optimal heuristic, computed *a posteriori* based on mutation results. In their paper [43], the authors applied the experiment on six policies. In this book chapter, we show two interesting results for the policy with the relatively smallest number of rules and the one with the largest number of rules.

Figure 7 shows the obtained results for continue-a policy, which includes 398 rules, while Fig. 8 presents the results for pluto which contains only 21 rules. Clearly, for both policies the XACML similarity is outperforming the random prioritization and is very close to the near-optimal heuristic.

4.4.3 Limitations and Potential Improvements

The main limitation of this prioritization approach is related to the fact that combining algorithms are ignored and not considered during the prioritization process. Indeed, combining algorithms might play an important role in case the policy contains many conflicting rules. In that case, the proposed strategies to order test cases might not be effective. This claim is yet to be checked by performing more experiments involving policies with several conflicting rules. This future work was mentioned by the authors in their paper [43] and is yet to be done.

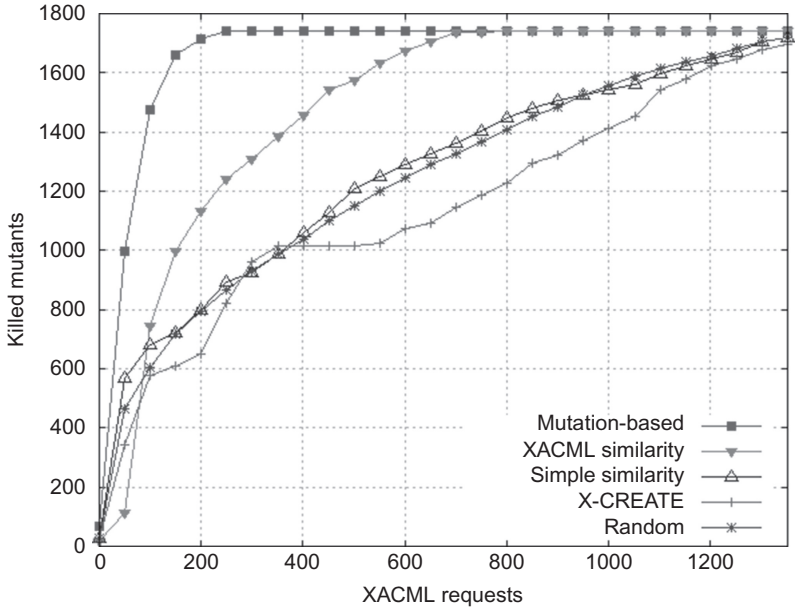

Figure 7 Prioritization experiment results for continue-a policy [43].

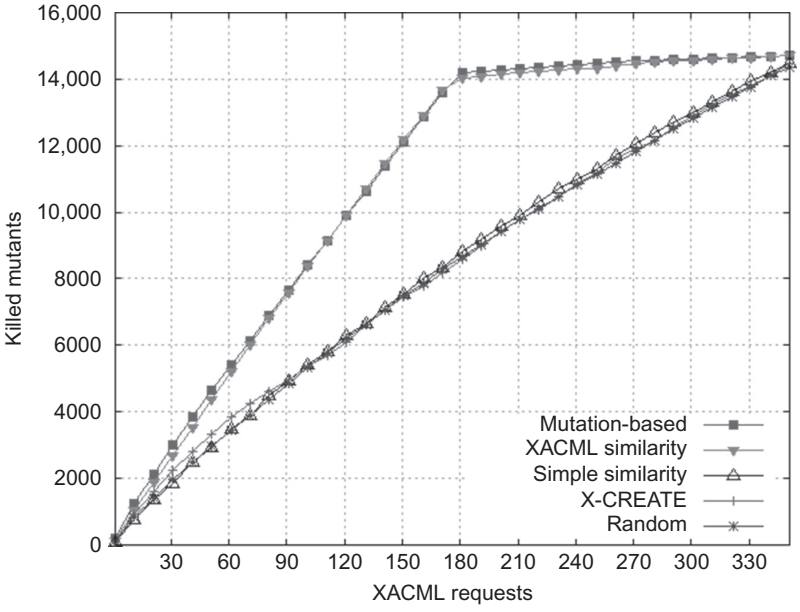

Figure 8 Prioritization experiment results for pluto policy [43].

4.5 Regression Testing

Regression testing [44] aims at verifying that the modified parts of the software are correct. Regression test selection aims at reducing the costs inherent from rerunning all the test cases by selecting the test cases that maximize the fault detection capabilities of the modified version of the software. In Ref. [45], a regression test selection approach has been applied in the context of the policy evolution. A regression test selection process in the context of security policy evolution is illustrated in Fig. 9. The authors have proposed three techniques for regression test selection that reveal implementation faults when the policy evolves. The three techniques are summarized as follows:

- *Mutation-based technique:* This technique establishes a mapping between test cases and rules by exercising tests for a normal policy and a mutated one. The mutated policy has a flip in one of its rules. When a test case presents a different result with a normal policy and a mutated one, a test case is thus mapped to the rule. The second step considers an initial policy and a policy that has been evolved, identifies the different rules in the two policy versions, and uses the correlation step to identify the test regression test selection set.
- *Coverage-based technique:* This technique uses two steps similar to the first technique; however, the mapping step uses coverage analysis instead of mutation analysis. The coverage step aims at tracking the impacted rules when a test executes.

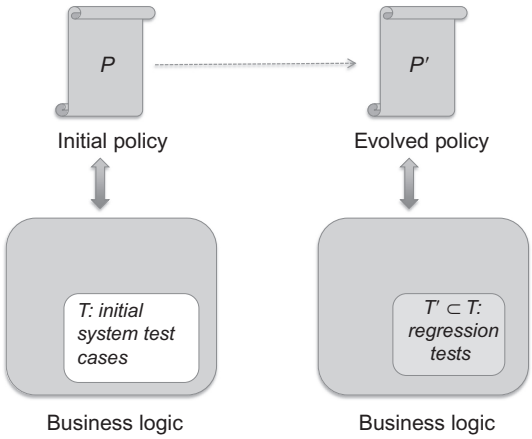

Figure 9 Regression testing in the context of policy evolution.

- *Request evaluation-based technique:* This technique is more efficient than the two previous techniques in terms of execution time since it does not require a mapping step. This technique analyzes requests results at runtime and selects test cases that have different evaluations with an initial policy and its evolved version.

5. USAGE CONTROL TESTING

Even though the research in access control testing has been quite active in the last few years. There is still a major effort to improve usage control testing. Rubab *et al.* [46] have initiated this effort by developing a model-based approach for usage control testing. Their approach is based on an obligations profile that extends the concepts of UML Class and State Machine Diagrams. They have used the Object Constraint Language to define constraints on the obligations profile. The profile completeness has been validated through the modeling of 47 different obligations for four different systems. They have developed the tool to generate executable test cases from UML class diagrams and UML state machines. The quality of the generated test cases has been assessed using mutation analysis. Rubab *et al.* have used the mutation operators that have been defined in the work of Elrakaiby *et al.* [47]. The mutation analysis results showed that on average the generated test cases have been able to kill 75% of the mutants.

6. DISCUSSION

Even though the domain of access testing has been widely explored in the last few years, there is still big room for research to tackle the current limitations and challenges in this research area. These limitations and challenges are the following:

Lack of benchmarking policies and access control implementations

XACML is based on an Attribute-Based Access Control Model (ABAC) that captures several access control scenarios. The lack of benchmarks in XACML access control policies and underlying implementations makes it difficult to define access control testing frameworks that are able to capture all XACML features (i.e., delegation, obligations, combining algorithms). A benchmark for distributed environment such as the cloud environment

that illustrates for instance how resources can be regulated for different tenants [48] would provide a room for more challenging testing issues.

Lack of interaction between academia and research

There has always been a lack of communication and collaboration between researchers and industry in general in the software testing and even in software engineering. For access control testing, this lack of collaboration is flagrant and questions the usefulness of the academic research in this area. The vast majority of papers cited in this book chapter were done without any collaboration with industry partners. Even more, some research done in this area is probably not applicable in an industrial setting simply because the case studies that were considered are toy systems or small policies. Therefore, the community should put more effort in bringing the research ideas, approaches, and prototypes to industry by setting more collaborations and more partnerships with companies, specially those involved in building tools for specifying, deploying, and verifying the XACML language like IBM, Oracle, and Axiomatics. Furthermore, the research community should be actively involved in the standardization process. Indeed, as shown on the OASIS XACML technical committee Web page,[2] all members contributing to the XACML standard are from industry. Researchers should be involved and added to the list of contributing members.

Automation challenges

Building automated frameworks for test generation, selection, prioritization, and assessment is still an area of active research. With the growing complexity of policy-based systems, there is a need to develop automated tools that are able to test properties satisfaction and to scale. Moreover, one major challenge that automation has to face is the interoperability issues. In most of the cases, the testing tool is developed based on a specific application and a specific access control model. Subsequently, its usage with a different target application or with a different access control model may require to use specific adapters.

7. CONCLUSION

This book chapter presented a detailed summary of existing approaches tackling access control testing. It has focused on testing the

[2] https://www.oasis-open.org/committees/membership.php?wg_abbrev=xacml.

XACML policies and showed the different components of XACML policies testing, namely, the automated test generation approaches, test selection and prioritization approaches, and finally test qualification and assessment based on mutation analysis. This area of research has been very active during the last decade and researchers made significant progress and proposed very effective contributions, in terms of testing strategies and also in terms of tools implementing the proposed ideas.

On the other hand, the area of usage control testing is not as mature and substantial research work in this area is yet to be done. We believe that the work in this area is still in its early phases and there is much to be done for instance, to target testing XACML policies, which include usage control rules.

ACKNOWLEDGMENTS

We would like to thank itrust consulting staff for their help in proofreading this book chapter and their help in providing feedback and comments specially in the discussion part.

In addition, we would like to thank the members of the SerVal research group, from the University of Luxembourg, for their help, comments, and interesting inputs that helped us in improving the contents of this book chapter.

REFERENCES

[1] P. Samarati, S. de Capitani di Vimercati, Access control: policies, models, and mechanisms, in: Foundations of Security Analysis and Design: Tutorial Lectures, 2001, pp. 137–196.
[2] F. Siewe, A. Cau, Z. Hussein, A compositional framework for access control policies enforcement, in: Proceedings of the 2003 ACM Workshop on Formal Methods in Security Engineering (FMSE '03), ACM, New York, NY, USA, 2003, pp. 32–42.
[3] J. Ligatti, L. Bauer, D. Walker, Edit automata: enforcement mechanisms for run-time security policies, Int. J. Inf. Secur. 4 (1–2) (2005) 2–16.
[4] N. Damianou, N. Dulay, E. Lupu, M. Sloman, The ponder policy specification language, in: Policies for Distributed Systems and Networks, Springer, Berlin Heidelberg, 2001, pp. 18–38.
[5] D.F. Ferraiolo, R.S. Sandhu, S.I. Gavrila, D.R. Kuhn, R. Chandramouli, Proposed NIST standard for role-based access control, ACM Trans. Inf. Syst. Secur. 4 (3) (2001) 224–274.
[6] D.E. Bell, L.J. La Padula, Secure Computer System: Unified Exposition and Multics Interpretation (No. MTR-2997-REV-1), MITRE Corp, Bedford MA, 1976.
[7] B. Lampson, Protection, in: Proceedings of the 5th Princeton Conference on Information Sciences and Systems, 1971.
[8] A.A.E. Kalam, S. Benferhat, A. Miège, R.E. Baida, F. Cuppens, C. Saurel, P. Balbiani, Y. Deswarte, G. Trouessin, Organization based access control, in: Proceedings of 10th IEEE International Conference on Policies for Distributed Systems and Networks, 2003, pp. 120–131.
[9] P. Ashley, S. Hada, G. Karjoth, C. Powers, M. Schunter, 2003, Enterprise privacy authorization language (EPAL 1.2), Submission to W3C, p. 1.
[10] A. Pretschner, M. Hilty, D.A. Basin, Distributed usage control, Commun. ACM 49 (2006) 39–44.

[11] J. Park, R. Sandhu, Towards usage control models: beyond traditional access control, in: Proceedings of the Seventh ACM Symposium on Access Control Models and Technologies, SACMAT'02, 2002, pp. 57–64.
[12] M. Hilty, D.A. Basin, A. Pretschner, On obligations, in: ESORICS, 2005, pp. 98–117.
[13] X. Zhang, Formal model and analysis of usage control, Ph.D. thesis, 2006.
[14] N.H. Minsky, A.D. Lockman, Ensuring integrity by adding obligations to privileges, in: Proceedings of the 8th International Conference on Software Engineering, ICSE'85, 1985, pp. 92–102.
[15] K. Irwin, T. Yu, W.H. Winsborough, On the modeling and analysis of obligations, in: Proceedings of the 13th ACM Conference on Computer and Communications Security, CCS'06, 2006, pp. 134–143.
[16] J. Park, R. Sandhu, Towards usage control models: beyond traditional access control, in: Proceedings of the Seventh ACM Symposium on Access Control Models and Technologies, ACM, 2002, pp. 57–64.
[17] J. Park, R. Sandhu, The UCON ABC usage control model, ACM Trans. Inf. Syst. Secur. 7 (1) (2004) 128–174.
[18] C. Danwei, H. Xiuli, R. Xunyi, Access control of cloud service based on ucon, in: Cloud Computing, Springer, Berlin Heidelberg, 2009, pp. 559–564.
[19] E. Martin, Testing and analysis of access control policies, in: Companion to the Proceedings of the 29th International Conference on Software Engineering, IEEE Computer Society, 2007, pp. 75–76.
[20] T. Mouelhi, Y. Le Traon, B. Baudry, Transforming and selecting functional test cases for security policy testing, in: Software Testing Verification and Validation, 2009. ICST'09. International Conference on, IEEE, 2009, pp. 171–180.
[21] S. Daoudagh, D. El Kateb, F. Lonetti, E. Marchetti, T. Mouelhi, A toolchain for model-based design and testing of access control systems, in: MODELSWARD, 2015.
[22] J.H. Hwang, E. Martin, T. Xie, et al., Testing access control policies, in: Encyclopedia of Software Engineering, 2010, pp. 673–683.
[23] E. Martin, T. Xie, A fault model and mutation testing of access control policies, in: Proceedings of the 16th International Conference on World Wide Web, ACM, 2007, pp. 667–676.
[24] T. Mouelhi, Y. Le Traon, B. Baudry, Mutation analysis for security tests qualification, in: Testing: Academic and Industrial Conference Practice and Research Techniques—MUTATION, 2007, TAICPART-MUTATION 2007, IEEE, 2007, pp. 233–242.
[25] A. Bertolino, S. Daoudagh, F. Lonetti, E. Marchetti, Xacmut: Xacml 2.0 mutants generator, in: Software Testing, Verification and Validation Workshops (ICSTW), 2013 IEEE Sixth International Conference on, IEEE, 2013, pp. 28–33.
[26] A. Bertolino, S. Daoudagh, F. Lonetti, E. Marchetti, The X-CREATE framework—a comparison of XACML policy testing strategies, in: WEBIST, 2012, pp. 155–160.
[27] R. Soley, Model driven architecture, OMG White Paper 308 (2000) 308.
[28] L. Apfelbaum, J. Doyle, Model based testing, in: Software Quality Week Conference, 1997, pp. 296–300.
[29] A. Pretschner, T. Mouelhi, Y. Le Traon, Model-based tests for access control policies, in: Software Testing, Verification, and Validation, 2008 1st International Conference on, IEEE, 2008, pp. 338–347.
[30] T. Mouelhi, F. Fleurey, B. Baudry, Y. Le Traon, A model-based framework for security policy specification, deployment and testing, in: Model Driven Engineering Languages and Systems, Springer, Berlin Heidelberg, 2008, pp. 537–552.
[31] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. Lopes, J.M. Loingtier, J. Irwin, Aspect-oriented programming, Springer, Berlin Heidelberg, 1997, pp. 220–242.

[32] D. Xu, L. Thomas, M. Kent, T. Mouelhi, Y. Le Traon, A model-based approach to automated testing of access control policies, in: Proceedings of the 17th ACM Symposium on Access Control Models and Technologies, ACM, 2012, pp. 209–218.
[33] J. Hwang, T. Xie, F. Chen, A.X. Liu, Systematic structural testing of firewall policies, in: Reliable Distributed Systems, 2008, SRDS'08. IEEE Symposium on, IEEE, 2008, pp. 105–114.
[34] E. Martin, Automated test generation for access control policies, in: Companion to the 21st ACM SIGPLAN Symposium on Object-Oriented Programming Systems, Languages, and Applications, ACM, 2006, pp. 752–753.
[35] E. Martin, T. Xie, Automated test generation for access control policies via change-impact analysis, in: Proceedings of the Third International Workshop on Software Engineering for Secure Systems, IEEE Computer Society, 2007, p. 5.
[36] A. Bertolino, S. Daoudagh, F. Lonetti, E. Marchetti, F. Martinelli, P. Mori, Testing of PolPA authorization systems, in: Proceedings of the 7th International Workshop on Automation of Software Test, IEEE Press, Zurich, Switzerland, 2012, June, pp. 8–14.
[37] A. Bertolino, S. Daoudagh, F. Lonetti, E. Marchetti, The X-CREATE framework—a comparison of XACML policy testing strategies, in: WEBIST, 2012, pp. 155–160.
[38] W. Mallouli, A. Cavalli, Testing security rules with decomposable activities, in: High Assurance Systems Engineering Symposium, 2007, HASE'07, 10th IEEE, IEEE, 2007, pp. 149–155.
[39] W. Mallouli, J.-M. Orset, A. Cavalli, N. Cuppens, F. Cuppens, A formal approach for testing security rules, in: Proceedings of the 12th ACM Symposium on Access Control Models and Technologies, ACM, 2007, pp. 127–132.
[40] A. Masood, R. Bhatti, A. Ghafoor, A.P. Mathur, Scalable and effective test generation for role-based access control systems, IEEE Trans. Softw. Eng. 35 (5) (2009) 654–668.
[41] A.D. Brucker, L. Brügger, P. Kearney, B. Wolff, An approach to modular and testable security models of real-world health-care applications, in: Proceedings of the 16th ACM Symposium on Access Control Models and Technologies, ACM, 2011, pp. 133–142.
[42] A. Bertolino, Y.L. Traon, F. Lonetti, E. Marchetti, T. Mouelhi, Coverage-based test cases selection for XACML policies, in: Software Testing, Verification and Validation Workshops (ICSTW), 2014 IEEE Seventh International Conference on, IEEE, 2014, pp. 12–21.
[43] A. Bertolino, S. Daoudagh, D. El Kateb, C. Henard, Y. Le Traon, F. Lonetti, E. Marchetti, T. Mouelhi, M. Papadakis, Similarity testing for access control, Inf. Softw. Technol. 58 (2015) 355–372.
[44] G. Rothermel, M.J. Harrold, Analyzing regression test selection techniques, IEEE Trans. Softw. Eng. 22 (8) (1996) 529–551.
[45] J. Hwang, T. Xie, D. El Kateb, T. Mouelhi, Y. Le Traon, Selection of regression system tests for security policy evolution, in: Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering, ACM, 2012, pp. 266–269.
[46] I. Rubab, S. Ali, L. Briand, Y.L. Traon, Model-based testing of obligations, in: Quality Software (QSIC), 2014 14th International Conference on, IEEE, 2014, pp. 1–10.
[47] Y. Elrakaiby, T. Mouelhi, Y. Le Traon, Testing obligation policy enforcement using mutation analysis, in: Software Testing, Verification and Validation (ICST), 2012 IEEE Fifth International Conference on, IEEE, 2012, pp. 673–680.
[48] J.M.A. Calero, N. Edwards, J. Kirschnick, L. Wilcock, M. Wray, Toward a multi-tenancy authorization system for cloud services, IEEE Secur. Priv. 8 (6) (2010) 48–55.

ABOUT THE AUTHORS

Dr. Tejeddine Mouelhi is a currently a senior security researcher at itrust consulting. Prior to that, he was a research associate at the University of Luxembourg for 4 years. He was involved in the Dynosoar project, focusing on access control testing and modeling. He holds a PhD degree in Computer Science. His PhD subject was about "Testing and Modeling Security Mechanisms in Web Applications."

Prof. Dr. Yves Le Traon is a professor at University of Luxembourg, in the Faculty of Science, Technology and Communication (FSTC). His domains of expertise are related software engineering and software security, with a focus on software testing and model-driven engineering. He received his engineering degree and his PhD in Computer Science at the "Institut National Polytechnique" in Grenoble, France, in 1997. From 1998 to 2004, he was an associate professor at the University of Rennes, in Brittany, France. During this period, Professor Le Traon studied design for testability techniques, validation, and diagnosis of object-oriented programs and component-based systems. From 2004 to 2006, he was an expert in Model-Driven Architecture and Validation in the EXA team (Requirements Engineering and Applications) at "France Télécom R&D" company. In 2006, he became professor at Telecom Bretagne (Ecole Nationale des Télécommunications de Bretagne) where he pioneered the application of testing for security assessment of Web applications, P2P systems, and the promotion of intrusion detection systems using contract-based techniques.

He is currently the head of the Computer Science Research Unit at University of Luxembourg. He is a member of the Interdisciplinary Centre for Security, Reliability and Trust (SnT), where he leads the research group SERVAL (SEcurity Reasoning and VALidation). His research interests include software testing, model-driven engineering, model-based testing,

evolutionary algorithms, software security, security policies, and Android security. The current key topics he explores are related to Internet of Things (IoT) and Cyber-Physical Systems (CPS), Big Data (stress testing, multi-objective optimization, analytics, models@run.time), and mobile security and reliability. He is author of more than 140 publications in international peer-reviewed conferences and journals.

Dr. Donia El Kateb is a research associate in the Interdisciplinary Centre for Security, Reliability and Trust (SnT). She obtained her Computer Science Engineering degree at ENSI (National School of Computer Sciences) in Tunisia in 2005. She worked 4 years at the National Digital Certification Agency in Tunisia as a security engineer. She received her PhD in Computer Science at the University of Luxembourg in 2015. Her research interests span over multiobjective optimization, software engineering, and security.