Access Control Policy Coverage Assessment Through Monitoring

Antonello Calabrò, Francesca Lonetti(✉), and Eda Marchetti

ISTI-CNR, 56124 Pisa, Italy
{antonello.calabro,francesca.lonetti,eda.marchetti}@isti.cnr.it

Abstract. Testing access control policies relies on their execution on a security engine and the evaluation of the correct responses. Coverage measures can be adopted to know which parts of the policy are most exercised. This paper proposes an access control infrastructure for enabling the coverage criterion selection, the monitoring of the policy execution and the analysis of the policy coverage assessment. The framework is independent from the policy specification language and does not require the instrumentation of the evaluation engine. We show an instantiation of the proposed infrastructure for assessing the XACML policy testing.

1 Introduction

Nowadays, the criticality and the importance of ruling the access and the usage of the different distributed resources is becoming a stringent need. Episodes in which the cloud private profiles have been violated and personal data distributed[1], are unfortunately a more frequent part of our daily-news. Security problems motivate the research and industry to find solutions for data protection that involve the improvement of the security mechanisms and the policy specification. Security testing and assessment has also gained a lot of attention in order to avoid security flaws and violations inside the systems or applications. Indeed, as detailed more in the rest of this paper, policy-based testing is the process to ensure the correctness of policy specifications and implementations. By observing the execution of a policy implementation with a test input (i.e., an access request), the testers may identify faults in policy specifications or implementations, and validate whether the corresponding output (i.e., an access decision) is as intended. However, most of the test cases generation approaches available in literature for access control policies are based on combinatorial methodologies [3,11,16], thus the generated number of test cases can rapidly grow to cope with the policy complexity. Considering the strict constraints on testing budget, it is extremely important to focus the testing activity in the generation or selection of the test cases that cover the most important features and/or policy constructs. The purpose is to reduce as much as possible the number of tests to be executed

[1] http://edition.cnn.com/2014/10/02/showbiz/celebrity-news-gossip/nude-celeb-photos-google-hack/.

S. Tonetta et al. (Eds.): SAFECOMP 2017 Workshops, LNCS 10489, pp. 373–383, 2017.
DOI: 10.1007/978-3-319-66284-8_31

trying, from one side, to maximize the fault detection effectiveness, and from the other, to cover the most important elements/aspects defined into the policy itself. In this paper, we focus on the testing of the access control policies and in particular on the coverage assessment of the derived test suites. In literature, the available coverage facilities are divided into two groups: those that are embedded in the execution engine such as [8,14] and those that can be integrated into the execution framework as an additional component such for instance [1,7]. Both the solutions have specific advantages. For sure, an embedded solution reduces the performance delay of the execution framework. The main disadvantage of these last approaches is the lack of flexibility both in the data collection, coverage measures definition and language adopted. Moreover, any change requires to redesign or improve the execution engine itself, preventing in such manner the possibility of dynamic modification.

In this paper, we would like to overcome the above mentioned issues by proposing a solution through which the implementation of the access policy can be made more transparent for coverage purposes, while maintaining the flexibility and access control language independency. The proposed access control policy infrastructure is based on an external monitoring facility and enables language independent coverage measurements. The basic behavior of the proposed infrastructure consists into the derivation of the relevant coverage information from the policy specification, the collection of events during the policy execution by means of a monitoring facility, and the analysis of them so to assess the coverage level reached by a test strategy. Additionally, some corrective actions could be triggered in case of violations or problems without modifying the structure of the policy execution engine, enabling to dynamically update or modify the policy when necessary. The type of data to be collected during the execution is independently specified by the execution engine and is not linked to the specific notation used for the policy specification.

The contribution of this paper can be summarized into: (i) the integration for the first time of a monitoring framework into an access control system architecture; (ii) the definition of the architecture of the Policy Assessment Infrastructure enabling the coverage criterion selection, the policy analysis, the monitoring of the policy execution, and the policy coverage assessment; (iii) an instantiation of the proposed architecture on the XACML access control language.

The remainder of this paper is structured as follows: Sect. 2 introduces the basic concepts of access control systems and coverage testing; Sect. 3 presents the architecture of the Policy Assessment Infrastructure; Sect. 4 presents an instantiation of the proposed infrastructure on the XACML context; finally, Sect. 5 concludes the paper also hinting at future work.

2 Basic Concepts

Access control is one of the most adopted security mechanisms for the protection of resources and data against unauthorized, malicious or improper usage or modification. It is based on the implementation of access control policies expressed by a specific standard such for instance the wildly adopted eXtensible Access Control Markup Language (XACML) [15]. The recent approaches for testing access control systems can be classified into [9]: (i) Policy testing which includes fault models and mutation based proposals [4], testing criteria based on structural coverage [2,6] and proposals exploiting the access control policies structure [3,11]; (ii) testing the policy enforcement to discover its possible security vulnerability [9,14]; (iii) testing the evaluation engine by means of model-based approach [5,7] or combination of access control policies values [3].

In this paper, we focus on the testing of the access control policies and in particular on the coverage assessment of the derived test suite. Here below some basic concepts about the XACML-based access control system and the coverage testing used for the specific instantiation of the proposed Policy Assessment Infrastructure presented in Sect. 4.

XACML [15] is a platform-independent XML-based language for the specification of access control policies. Briefly, an XACML policy has a tree structure whose main elements are: PolicySet, Policy, Rule, Target and Condition. The PolicySet includes one or more policies. A Policy contains a Target and one or more rules. The Target specifies a set of constraints on attributes of a given request. The Rule specifies a Target and a Condition containing one or more boolean functions. If the Condition evaluates to true, then the Rule's Effect (a value of Permit or Deny) is returned, otherwise a NotApplicable decision is formulated (Indeterminate is returned in case of errors). The PolicyCombiningAlgorithm and the RuleCombiningAlgorithm define how to combine the results from multiple policies and rules respectively in order to derive a single access result.

The main components of an XACML based access control system are shown in Fig. 1: the Policy Administration Point (PAP) is the system entity in charge of managing the policies; the Policy Enforcement Point (PEP), usually embedded into an application system, receives the access request in its native format, constructs an XACML request and sends it to the Policy Decision Point (PDP); the Policy Information Point (PIP) provides the PDP with the values of subject, resource, action and environment attributes; the PDP evaluates the policy against the request and returns the response, including the authorization decision to the PEP.

Measurement of test quality is one of the key issues in software testing and coverage measures represent an effective mean for evaluating the different testing approaches. Adequacy criteria evaluate the testing strategy through the percentage of exercised set of elements in the program or in the specification. Usually, test coverage can be used for different purposes: (i) improve the test suite so to exercise elements that have not been tested; (ii) test suite augmentation and test suite minimization in case of regression testing; (iii) test cases selection, prioritization and test suite effectiveness evaluation. A systematic review of coverage based testing is presented in [17]. Many proposals for test coverage measurement and analysis, embedded directly in the evaluation engine, have been proposed depending on the adopted policy specification language. Considering

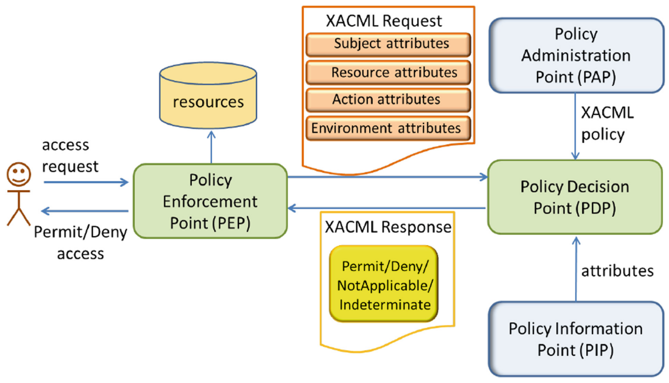

Fig. 1. Access control system architecture

in particular the XACML context, in [6,12] the authors propose PDP embedded solutions for coverage analysis and selection, while [10] focuses on regression test selection techniques.

3 Coverage Testing Framework

In this section, we propose a possible architecture of Policy Assessment Infrastructure based on an on-line monitor. The proposal has been conceived to be independent from the language adopted for the policy specification and flexible enough to be adapted to the different testing purposes. In particular, Fig. 2 (part A) shows the main components of the proposed Policy Assessment Infrastructure (top left component), referring to a generic structure of an access control system (top right component):

– *Test case generator* is in charge of test cases generation starting from the policy specification. In literature, depending on the access policy language there are several proposals such for instance XCREATE [3] and Targen [11] focused on XACML-based combinatorial approaches;
– *Test case executor* takes in input the test suite derived by the *Test case generator*, and sends one by one the test cases to the *Policy executor engine*. Moreover, it extracts the required information by each test case and transforms them into events readable by the *Monitor infrastructure*.
– *Trace generator* is in charge of implementing the different policy coverage criteria. It takes in input the policy from the *Policy Administration Point* and, according to the selected coverage criterion, derives all the possible policy traces. Usually, the traces extraction is realized by an optimized unfolding algorithm that exploits the policy language structure. Intuitively, the main goal is to derive an acyclic graph, defining a partial order on policy elements. Several proposals are available such as [6,12] for XACML policy specification. Once extracted, the traces are provided to the *Monitor infrastructure*.

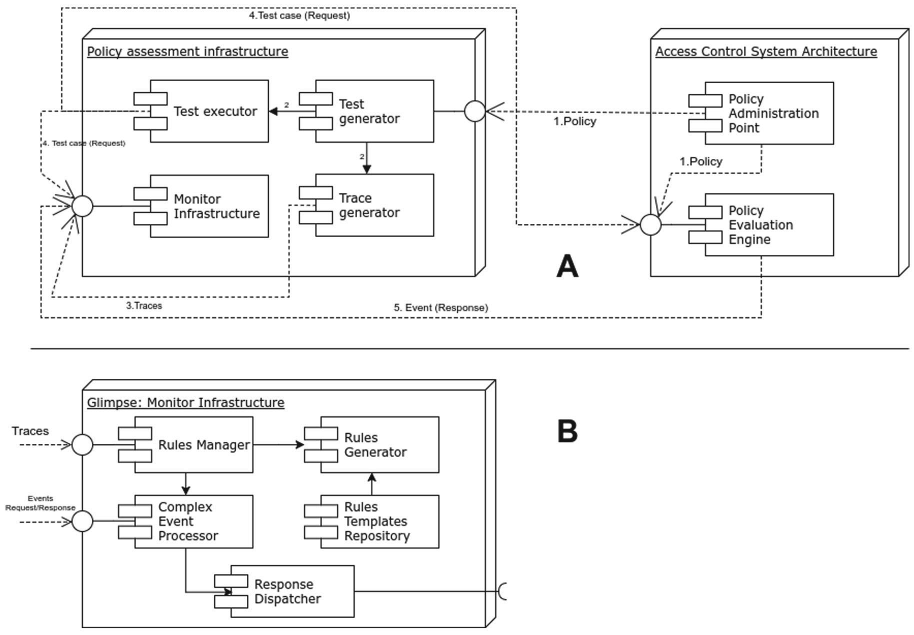

Fig. 2. Policy Assessment Infrastructure

- *Policy evaluation engine* is in charge of the execution of the policy and the derivation of the associated response. It communicates with the *Monitoring infrastructure* though a dedicated interface such as a REST one.
- *Monitoring infrastructure* is in charge of collecting data of interest during the run-time policy execution. There can be different solutions for monitoring activity. In this paper, we rely on Glimpse [1] infrastructure which has the peculiarity of decoupling the events specification from their collection and processing. As detailed in Fig. 2 (part B), the main components of Glimpse are: (i) *Complex Events Processor (CEP)* which analyzes the events and correlates them to infer more complex events; (ii) *Response Dispatcher* keeps track of the monitoring requests and sends back the final coverage evaluation; (iii)*Rules Generator* generates the rules using the templates stored into the *Rules Template Repository* starting from the derived policy traces to be monitored. A generic rule consists of two main parts: the events to be matched and the constraints to be verified, and the events/actions to be notified after the rule evaluation; (iv) *Rules Template Repository* stores predetermined rules templates that will be instantiated by the *Rules Generator* when needed; (v) *Rules Manager* instructes the CEP by loading and unloading the set of rules. We refer to [1] for a more detailed description of the Glimpse architecture.

In a typical workflow of the proposed framework, the *Policy Administration Point* sends the policy both to the *Test case generator* and the *Policy evaluation engine* (step 1 of Fig. 2). The *Test case generator* derives from the policy specification a set of test cases and sends them to both the *Test case executor* and the *Trace generator* (step 2 of Fig. 2). The *Trace generator* derives from the policy all the possible policy traces and sends them to the *Monitoring infrastructure* (step 3 *Monitoring infrastructure*). The *Test case executor* sends the test cases to the *Policy evaluation engine*, moreover it extracts from these test cases the events that are forwarded to the *Monitoring infrastructure* (step 4 of Fig. 2). Finally, the responses associated to the execution of the test cases are forwarded by the *Policy evaluation engine* to the *Monitoring infrastructure* (step 5 of Fig. 2).

4 Application Example

In this section, we present an instantiation of the proposed Policy Assessment Infrastructure to XACML based access control systems and its application to the XACML policy showed in Listing 1.1. Specifically, the policy defines the accesses to a library. It includes a policy set target (line 3) that is empty; a policy target (lines 5–12) allowing the access only to the *books* resource; a first rule (*ruleA*) (lines 13–30) with a target (lines 14–29) specifying that this rule applies only to the access requests of a *read* action of *books* resource with any environment; a second rule (*ruleB*)(lines 31–46), which effect is *Deny* when the subject is "Julius", the action is "write", the resource and environment are any resource and any environment respectively; a third rule (*ruleC*) (lines 47–69) that allows subject "Julius" the action "write", if he is also "professor" or "administrator"; finally, the default rule (line 70) denies the access in the other cases.

The *Test cases generator* has been implemented by X-CREATE tool [3] using the available *Simple-Combinatorial* test generation strategy. Specifically, it derives an XACML request for each of the possible combinations of the subject, resource, action and environment values taken from the policy and some additional requests containing random values. Listing 1.2 shows an example of a request generated by *Simple-Combinatorial* strategy: the subject *Julius* wants to *write* the *books* resource. Each generated XACML request is then transformed into an event by the *Test case executor* and sent to the *Monitor infrastructure.* The *Trace generator* has been implemented using the *XACML smart coverage* approach presented in [6] which focuses on the policy rules coverage. Briefly, the criterion computes the *Rule Target Set*, i.e., the union of the target of the rule, and all enclosing policy and policy sets targets. The main idea is that in order to match the rule target, the requests must first match the enclosing policy and policy sets targets. For instance, the *Rule Target Sets* of Listing 1.1 are presented in Table 1. We refer to [6] for the definition of the *XACML smart coverage* criterion. Each *Rule Target Set* is sent to the *Monitoring Infrastructure* as a event. The *Policy evaluation engine* is instantiated with the XACML Sun' Policy Decision Point (PDP) [13] which executes the XACML requests against the XACML policy and sends the corresponding XACML responses to the *Monitor infrastructure.* The *Monitor infrastructure* observes the on-line execution of the XACML policy on the PDP, and, according to the values of the requests, the responses and the set of traces generated from the XACML policy, assesses the coverage of the XACML requests on the traces.

Table 1. *Rule Target Sets* of Listing 1.1

$T_1 = \{(\emptyset, \emptyset, \emptyset, \emptyset), (\emptyset, \{books\}, \emptyset, \emptyset), (\emptyset, \{books\}, \{read\}, \emptyset), Permit\}$
$T_2 = \{(\emptyset, \emptyset, \emptyset, \emptyset), (\emptyset, \{books\}, \emptyset, \emptyset), (\{Julius\}, \{books\}, \{write\}, \emptyset), Deny\}$
$T_3 = \{(\emptyset, \emptyset, \emptyset, \emptyset), (\emptyset, \{books\}, \emptyset, \emptyset), (\{Julius, professor\}, \{books\}, \{write\}, \emptyset), Permit\}$
$T_4 = \{(\emptyset, \emptyset, \emptyset, \emptyset), (\emptyset, \{books\}, \emptyset, \emptyset), (\{Julius, administrator\}, \{books\}, \{write\}, \emptyset), Permit\}$
$T_5 = \{(\emptyset, \emptyset, \emptyset, \emptyset), (\emptyset, \{books\}, \emptyset, \emptyset), (\emptyset, \emptyset, \emptyset, \emptyset), Deny\}$

As an example, in Listings 1.3 there is a rule definition for trace T_3 of Table 1. The monitor infrastructure extracts from the payload of the event *Glimpse-BaseEventRequest* the field data that contains the values for the (*Subjects*, *Resources*, *Actions*, *Environments*) and checks if they are included in the sets of *policySet*, *policy* and *rules* target values of the trace T_3 (lines [21–37]). If this is verified the monitoring infrastructure extracts from the same trace the *Response* value and checks whether it is equal to the corresponding PDP response (line 43). If this is true the trace is considered covered (line 49).

In this application example, we executed the XACML policy of Listing 1.1 with all the requests generated by the *Simple-Combinatorial* strategy and we reached a coverage of 60% (only T_1, T_2, T_5 were covered). Specifically, the request of Listing 1.2 is able to cover T_2 trace. Similarly, other requests having *read* as action, *books* as resource and any value for subject are able to cover T_1 trace; whereas requests having any subject, action, resource, and environment are able to cover T_5 trace. From the analysis of the coverage assessment results, it was evident that, by construction, the test suite derived from *Simple-Combinatorial* strategy can only cover traces including only one subject, resource, action and environment value. The coverage of traces T_3 and T_4 requires XACML requests having more than one subject, resource, action and environment values because the effect of the corresponding XACML policy rule (*Rule C*) is simultaneously dependent on more than one constraint.

This simple experiment evidences that the *Simple-Combinatorial* strategy is not effective enough to reach 100% coverage of the traces and should be enriched. By the identification of not covered traces, the Policy Assessment infrastructure provides important hints to testers and can guide them in the generation of ad hoc test cases or selection of more effective test strategies.

```
<PolicySet PolicySetId=‘‘policySetExample’’
 PolicyCombiningAlgId=‘‘first-applicable’’>
 <Target/>
 <Policy PolicyId=‘‘policyExample’’ RuleCombiningAlgId=‘‘first-applicable’’>
  <Target>
   <Resource>
    <ResourceMatch MatchId=‘‘anyURI-equal’’>
     <AttributeValue DataType=‘‘anyURI’’>books</AttributeValue>
     <ResourceAttributeDesignator AttributeId=‘‘resource-id’’ DataType=‘‘anyURI’’/>
    </ResourceMatch>
   </Resource>
  </Target>
  <Rule RuleId=‘‘ruleA’’ Effect=‘‘Permit’’>
   <Target>
    <Resources>
     <Resource>
      <ResourceMatch MatchId=‘‘anyURI-equal’’>
       <AttributeValue DataType=‘‘anyURI’’>books</AttributeValue>
       <ResourceAttributeDesignator AttributeId=‘‘resource-id’’ DataType=‘‘anyURI’’/>
      </ResourceMatch>
     </Resource>
   </Resources>
    <Actions><Action>
      <ActionMatch MatchId=‘‘string-equal’’>
       <AttributeValue DataType=‘‘string’’>read</AttributeValue>
       <ActionAttributeDesignator AttributeId=‘‘action-id’’ DataType=‘‘string’’/>
      </ActionMatch>
     </Action></Actions>
   </Target>
  </Rule>
  <Rule RuleId=‘‘ruleB’’ Effect=‘‘Deny’’>
   <Target>
    <Subjects><Subject>
      <SubjectMatch MatchId=‘‘string-equal’’>
       <AttributeValue DataType=‘‘string’’>Julius</AttributeValue>
       <SubjectAttributeDesignator AttributeId=‘‘subject-id’’ DataType=‘‘string’’/>
      </SubjectMatch>
     </Subject></Subjects>
    <Actions><Action>
      <ActionMatch MatchId=‘‘string-equal’’>
       <AttributeValue DataType=‘‘string’’>write</AttributeValue>
       <ActionAttributeDesignator AttributeId=‘‘action-id’’ DataType=‘‘string’’/>
      </ActionMatch>
     </Action></Actions>
   </Target>
  </Rule>
  <Rule RuleId=‘‘ruleC’’ Effect=‘‘Permit’’>
   <Target>
    <Subjects><Subject>
     <SubjectMatch MatchId=‘‘string-equal’’>
      <AttributeValue DataType=‘‘string’’>Julius</AttributeValue>
      <SubjectAttributeDesignator AttributeId=‘‘subject-id’’ DataType=‘‘string’’/>
     </SubjectMatch>
    </Subject></Subjects>
    <Actions><Action>
     <ActionMatch MatchId=‘‘string-equal’’>
      <AttributeValue DataType=‘‘string’’>write</AttributeValue>
      <ActionAttributeDesignator AttributeId=‘‘action-id’’ DataType=‘‘string’’/>
     </ActionMatch>
    </Action></Actions>
   </Target>
   <Condition FunctionId=‘‘string-at-least-one-member-of’’>
    <SubjectAttributeDesignator SubjectCategory=‘‘access-subject’’ AttributeId=‘‘Role’’ DataType=‘‘string’’/>
    <Apply FunctionId=‘‘string-bag’’>
     <AttributeValue DataType=‘‘string’’>professor</AttributeValue>
     <AttributeValue DataType=‘‘string’’>administrator</AttributeValue>
    </Apply>
   </Condition>
  </Rule>
  <Rule RuleId=‘‘ruleD’’ Effect=‘‘Deny’’/>
 </Policy>
</PolicySet>
```

Listing 1.1. An XACML policy

```
<Request xmlns=''urn:oasis:names:tc:xacml:2.0:context:schema:os''>
 <Subject>
  <Attribute AttributeId=''subject-id1'' DataType=''string''>
   <AttributeValue>Julius</AttributeValue>
  </Attribute>
 </Subject>
 <Resource>
  <Attribute AttributeId=''resource-id'' DataType=''string''>
   <AttributeValue>books</AttributeValue>
  </Attribute>
 </Resource>
 <Action>
  <Attribute AttributeId=''action-id'' DataType=''string''>
   <AttributeValue>write</AttributeValue>
  </Attribute>
 </Action>
 <Environment/>
</Request>
```

Listing 1.2. An XACML request

```
import it.cnr.isti.labsedc.glimpse.event.GlimpseBaseEventAbstract;
import it.cnr.isti.labsedc.glimpse.event.GlimpseBaseEventRequest;
import it.cnr.isti.labsedc.glimpse.event.GlimpseBaseEventPdpResponse;
import it.cnr.isti.labsedc.glimpse.engine.xacml.TraceEngine;
import it.cnr.isti.labsedc.glimpse.utils.Notifier;

declare GlimpseBaseEventAbstract
    @role( event )
    @timestamp( timeStamp )
end

rule ''policySetExampleRule''
    no-loop true
    salience 20
    dialect ''java''
when
    $aEvent : GlimpseBaseEventRequest(
        this.isConsumed == false,
        this.isException == false,

    //policySetCheck
        this.data.getSubjectsSection().areValidForPolicySetOfTrace(''T3''),
        this.data.getResourcesSection().areValidForPolicySetOfTrace(''T3''),
        this.data.getActionSection().areValidForPolicySetOfTrace(''T3''),
        this.data.getEnvironmentSection().areValidForPolicySetOfTrace(''T3''),

    //policyCheck
        this.data.getSubjectsSection().areValidForPolicyOfTrace(''T3''),
        this.data.getResourcesSection().areValidForPolicyOfTrace(''T3''),
        this.data.getActionSection().areValidForPolicyOfTrace(''T3''),
        this.data.getEnvironmentSection().areValidForPolicyOfTrace(''T3''),

    //rulesCheck
        this.data.getSubjectsSection().areValidForRulesOfTrace(''T3''),
        this.data.getResourcesSection().areValidForRulesOfTrace(''T3''),
        this.data.getActionSection().areValidForRulesOfTrace(''T3'')),
        this.data.getEnvironmentSection().areValidForRulesOfTrace(''T3''));

    $bEvent : GlimpseBaseEventPdpResponse(
        this.isConsumed == false,
        this.isException == false,
        this.data.getId().compareTo($aEvent.getId().toString()) == 0,
        this.data.getResponse.compareTo(TraceEngine.getTraceResponse(''T3'')) == 0,
        this after $aEvent);

then
```

```
        $aEvent.setConsumed(true);
        $bEvent.setConsumed(true);
        Notifier.setPolicyMatch($aEvent.data.getId(), ''T3'');
        update($aEvent);
        retract($aEvent);
        update($bEvent);
        retract($bEvent);
    end
```

Listing 1.3. Monitoring rule

5 Discussion and Conclusions

In this paper, we presented an access control infrastructure for enabling the coverage criterion selection, the monitoring of the policy execution and the analysis of the policy coverage assessment. We provided an instantiation inside the XACML-based access control systems. The preliminary obtained results showed the effectiveness of the proposed infrastructure in evaluating the coverage of an XACML policy. Moreover, traces analysis highlighted weaknesses in the test suite and provided hints for the generation of ad-hoc test cases.

The application of the proposed access control infrastructure is not limited to the coverage policy assessment as shown in this paper, but can be used to detect on-line criticalities of the policy execution. Indeed, the monitoring infrastructure is able to detect some inconsistencies between the responses belonging to the not covered traces and the corresponding PDP 'responses. These inconsistencies could evidence potential flaws either in the policy specification or in its implementation. Moreover, the proposed access control infrastructure could be used in real word environments for profiling the resource usage and the user behaviors. This could be a very important starting point for identifying the most critical policy traces and improving their security enforcement.

Concerning threats to validity of the presented experiment, three aspects can be considered: the test case generation, the traces generation and the policy evaluation. Indeed, the tools adopted and the algorithms implemented may have influenced the reported results. It could be that different choices might have provided different effectiveness results.

We are currently working to include in the access control infrastructure more coverage criteria. We plan also to enhance the monitor infrastructure with facilities for proactively detecting, by the off-line trace analysis, possible security inconsistencies of the tested access control policy. Other future work deals with the instantiation of the proposed infrastructure by considering different access and usage control policy specification languages.

Acknowledgements. This work has been partially supported by the GAUSS national research project (MIUR, PRIN 2015, Contract 2015KWREMX).

References

1. Bertolino, A., Calabrò, A., Lonetti, F., Di Marco, A., Sabetta, A.: Towards a model-driven infrastructure for runtime monitoring. In: Troubitsyna, E.A. (ed.) SERENE 2011. LNCS, vol. 6968, pp. 130–144. Springer, Heidelberg (2011). doi:10.1007/978-3-642-24124-6_13
2. Bertolino, A., Daoudagh, S., El Kateb, D., Henard, C., Le Traon, Y., Lonetti, F., Marchetti, E., Mouelhi, T., Papadakis, M.: Similarity testing for access control. Inf. Softw. Technol. **58**, 355–372 (2015)
3. Bertolino, A., Daoudagh, S., Lonetti, F., Marchetti, E.: Automatic XACML requests generation for policy testing. In: Proceedings of ICST, pp. 842–849. IEEE (2012)
4. Bertolino, A., Daoudagh, S., Lonetti, F., Marchetti, E.: Xacmut: Xacml 2.0 mutants generator. In: Proceedings of ICST Workshops, pp. 28–33 (2013)
5. Bertolino, A., Daoudagh, S., Lonetti, F., Marchetti, E., Martinelli, F., Mori, P.: Testing of polpa-based usage control systems. Software Qual. J. **22**(2), 241–271 (2014)
6. Bertolino, A., Le Traon, Y., Lonetti, F., Marchetti, E., Mouelhi, T.: Coverage-based test cases selection for xacml policies. In: Proceedings of ICST Workshops, pp. 12–21 (2014)
7. Carvallo, P., Cavalli, A.R., Mallouli, W., Rios, E.: Multi-cloud applications security monitoring. In: Au, M.H.A., Castiglione, A., Choo, K.-K.R., Palmieri, F., Li, K.-C. (eds.) GPC 2017. LNCS, vol. 10232, pp. 748–758. Springer, Cham (2017). doi:10.1007/978-3-319-57186-7_54
8. Daoudagh, S., Lonetti, F., Marchetti, E.: Assessment of access control systems using mutation testing. In: Proceedings of TELERISE, pp. 8–13 (2015)
9. Felderer, M., Büchler, M., Johns, M., Brucker, A.D., Breu, R., Pretschner, A.: Chapter one-security testing: a survey. Adv. Comput. **101**, 1–51 (2016)
10. Hwang, J., Xie, T., El Kateb, D., Mouelhi, T., Le Traon, Y.: Selection of regression system tests for security policy evolution. In: Proceedings of ASE, pp. 266–269 (2012)
11. Martin, E.: Automated test generation for access control policies. In: Proceedings of OOPSLA, pp. 752–753 (2006)
12. Martin, E., Xie, T., Yu, T.: Defining and measuring policy coverage in testing access control policies. In: Ning, P., Qing, S., Li, N. (eds.) ICICS 2006. LNCS, vol. 4307, pp. 139–158. Springer, Heidelberg (2006). doi:10.1007/11935308_11
13. Microsystems, S.: Sun's XACML implementation (2006)
14. Mouelhi, T., El Kateb, D., Le Traon, Y.: Chapter five-inroads in testing access control. Adv. Comput. **99**, 195–222 (2015)
15. OASIS: extensible access control markup language (XACML) version 2.0 (2005)
16. Pretschner, A., Mouelhi, T., Le Traon, Y.: Model-based tests for access control policies. In: Proceedings of ICST, pp. 338–347 (2008)
17. Shahid, M., Ibrahim, S., Mahrin, M.N.: A study on test coverage in software testing. Advanced Informatics School (2011)