2014

Approaches for Testing and Evaluation of XACML Policies

Izzat M. Alsmadi
Texas A&M University-San Antonio, ialsmadi@tamusa.edu

International Journal of Security and Its Applications
Vol.8, No.4 (2014), pp.101-112
http://dx.doi.org/10.14257/ijsia.2014.8.4.10

Approaches for Testing and Evaluation of XACML Policies

Izzat M Alsmadi

Yarmouk University
ialsmadi@yu.edu.jo

Abstract

Security services are provided through: The applications, operating systems, databases, and the network. There are many proposals to use policies to define, implement and evaluate security services. We discussed a full test automation framework to test XACML based policies. Using policies as input the developed tool can generate test cases based on the policy and the general XACML model.

We evaluated a large dataset of policy implementations. The collection includes more than 200 test cases that represent instances of policies. Policies are executed and verified, using requests and responses generated for each instance of policies. WSO2 platform is used to perform different testing activities on evaluated policies.

Keywords: *Software Defined Networks or Networking (SDN), Policy management, Change impact*

1. Introduction

Policies related to security and some business processes are implemented across all enterprise applications. They are created, continuously monitored and applied. On the other hand, there is a need to have policies that are agile and flexible. In addition to the need to have them easy to change and update, there is a need to be able to detect who can be impacted by the policy change and who will not.

We used WSO2 enterprise architecture (WSO2.com). Particularly, the WSO2 security component that can define and interact with security is called identity server Figure 1. The application is built to fit the cloud architecture and to enable adding and configuring components very flexibly.

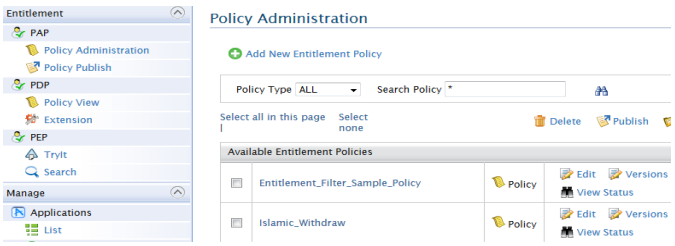

Figure 1. WSO2 Identity Server Example

In this context, XACML (the Extensible Access Control Markup Language) from OASIS (http://www.oasis-open.org) can be used for policy management and change analysis. This is an authorization markup language based on the popular widely used XML, the defacto Internet data and messaging communication language. It is also considered a security policy creation and management application. XACML includes components to define a security

ISSN: 1738-9976 IJSIA

policy to access computer resources (*e.g.*, a data base, an application, and a web service), *etc.*, It also includes rules to specify users and their permissions or privileges. Figure 2 shows XACML authorization elements including: Policy component, policy set, policy, policy administration point, rule, target, action, resource, subject and environment. We will describe those components later on with a context example related to the paper subject.

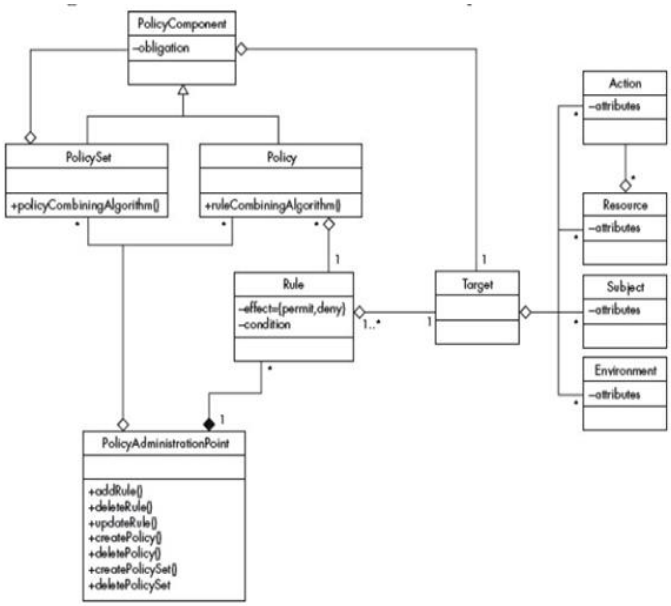

Figure 2. XACML Policy Authorization Elements (Conceptual Diagram) [1]

2. Related Work

In this section, a description of some paper utilizes XACML policies in the area of testing will be introduced. Software change impact analysis is a research field with many research publications.

Fisler, *et al.*, 2005 [2, 3] paper is a popular paper in terms of citations related to policy impact analysis. The paper discussed policy architecture based on XACML. The paper discussed Margrave software tool for role-based access control policies management. The tool includes verification methods for policies against properties. Different rules in a policy are modeled in multi-terminal_binary_decision_diagram (MTBDD) where output can be permit or deny. Authors listed some deficiencies in the approach related to data values reasoning and incomplete processing of XACML policies. The tool itself conducts change impact analysis through comparing new and old tool statically without connecting those policies with the actual system and measuring impact analysis on the actual system.

Martin and Xie paper 2007 discussed automatic testing for XACML policies [4]. A framework and a tool called Cirg (for change impact request generation) are developed for this purpose. The proposed system evaluates changes between different policies and making a comparison of requests between them. Policies contain rules and rules contain target elements that should be satisfied to fulfill a rule. Change impact between different versions of policies is conducted using counter examples or mutants to evaluate differences between those policies or versions of policies. Several metrics related to testing and coverage was also used to evaluate effectiveness of test case generation methods.

Biskup and J. Lopez 2007 paper discussed change impact analysis for firewall policies [5]. As firewall policies may need to change very often, testing them and their impact is necessary and should be conducted smoothly and transparently. The input to the proposed system is a policy and required changes. The output is then the impact of such proposed changes. Impact is classified based on the nature of policy change (*e.g.*, policy deletion, insertion, or update).

3. Goals and Approaches

The policy testing framework allows policies to be tested before enforcing them in live systems. To allow this, the configuration or impact analysis component should be able to evaluate system objects and those that are impacted by a particular policy.

Using some libraries, we developed a test automation framework to: read XACML files and serialize them into their attributes, rules, targets, etc. The system then generates test cases based on that information. Test execution and verification is then conducted to judge test cases' results based on predefined outputs. Test cases can be also used for regression testing when policies are changed to evaluate test cases that are affected by policy change.

The developed system can be used offline where its input is XACML policies. It can be also used part of a system to evaluate its current or applied policies.

Figure three below shows a context diagram for XACML showing its major architectural components. The figure shows that XCAML develop, regulate, implement and test rules through four components: PAP, PDP, PEP, PIP.

1. Policy Administration Point (PAP). This includes the management component that also includes policies' repository. Different rules can be written in one or more policies that are stored and managed by PAP.

2. Policy Enforcement Point (PEP). This is the interface of the whole XACML to the system or the users. It received access requests and evaluates them with the help of other components (especially PDP). Decision to permit or deny access to the resource is then taken communicated to the user by PEP.

3. Policy Decision Point. This is the decision engine for access request. Data is collected by PDP from other components. The component includes an analysis system or component to make inference decisions.

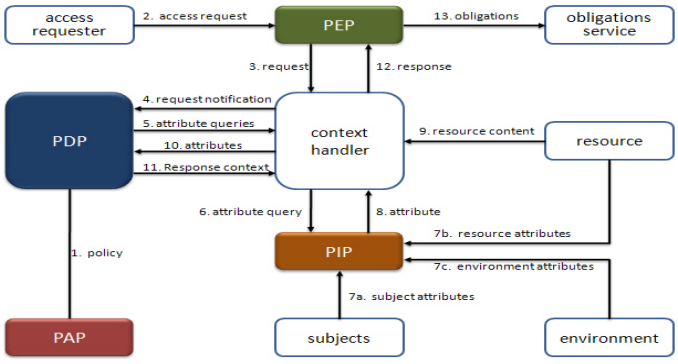

Figure 3. XACML Context and Data Flow Diagram

4. Policy Information Point (PIP). This represents the memory or the kitchen where all necessary information from other components, resources, or environment are collected.

For testing XACML authorization systems, test cases can simulate PEP role in actual systems. Test cases are represented by requests sent to policies where judgment is made based on policy and the request as a test case instance.

Figure 4 below summarizes tasks for XACML test automation framework major activities.

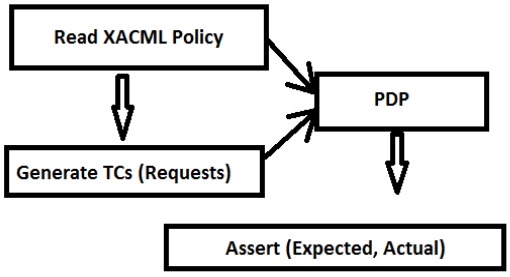

Figure 4. XACML based Test Framework

The test automation framework starts from (Read XACML Policy). In test automation framework, model based approaches are used formally describe framework inputs (in this case XACML policies). The output is a formal model that parses XACML different components. From this model test cases or requests can be automatically generated. In this case, they represent instances from the earlier developed model. The input to the PDP is then both policy and requests generated based on that particular policy. Test case evaluation or assertion is conducted based on comparing actual input with expected one where the request or the test case passes if expected and actual outputs are the same. Parser and builder components are required whenever we want to convert from XAML to an abstract representation necessary for testing or vice versa. WSO2 and SOAPUI can do most of test framework tasks. However, they are semi-automated and need user administration in each step.

The above approach assumes single policy-single request case. However, in many cases PDP or policy decision may need to combine more than one policy or more than one request for a particular permit or deny decision. Policies r requests may have contradictory rules and conflict resolution maybe expected. Verification process can have some other challenges specially where in some cases; it is difficult to describe expected correct output. In typical XACML architecture, response is sent as a XACML message with a binary decision of whether to permit or deny the request.

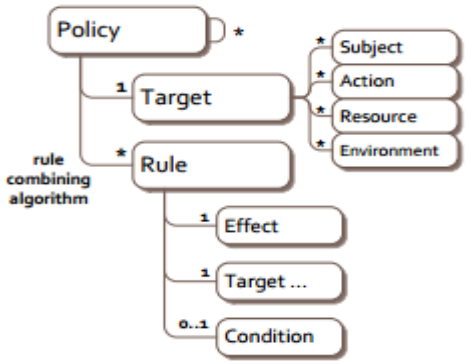

Figure 5. A XACML Simplified Policy Model [19]

4. Experiments and Analysis

In order to test the conformance of policies according to standards test cases that represent instances of policies are created. A policy request represents a test case that we can evaluate a policy through. Results are then comparison with expected output (aka policy response). Policy Decision Point (PDP) is tools used to perform test execution and verification. In real scenarios, PEP (Policy Enforcement Point) sends policy requests to PDP. Along with policy repository, PDP used incoming request from PEP and relevant policy to generate policy response and sends it back to PEP to communicate decisions with users or client. Request includes attributes of: Subject, Action, Resource and Environment while response includes obligation attributes (*e.g.,* Deny, accept, not applicable, or indeterminate).

In this case study we used WSO2. WSO2 IS XACML implementation is based on sunxacml. Its administration section, allows users to add or import policies and test them through policy requests. The actual output from executing policy requests over policy represents actual policy response that can be compared with expected one in conformance testing.

In this case study, we selected: Policies, policy requests and responses defined in the dataset (xacml2.0-ct-v.0.4.zip). Details on this dataset can be found at: https://www.oasis-open.org/committees/document.php?document_id=14846 [20]. Author indicates two versions of the dataset. XACML 2.0 Conformance Tests V3 published in Sep. 21st 2005 and V4 published in Oct. 10 2005. The contribution here is that author took original conformance tests published in OASIS for XACML version 1 and evaluates them against XACML version 2. History of dataset versions and details are available in: https://www.oasis-open.org/committees/document.php?-document_id=14846. Policy numbers in the dataset are labeled according to the sections in the document: II: Mandatory-to-Implement Functionality Tests, B: Target Matching, C: Function evaluation, D: combining algorithms, E: schema components, F: XACML 2.0 new features, G: Optional, but Normative Functionality Tests, GA: DefaultsType, Hierarchical Resources, D: <ResourceContent> Element, E: Multiple Decisions, F:Attribute Selectors, G: Non-mandatory Functions.

This dataset is used as a baseline to test PEP/PDP implementation engines against conformance with XACML guidelines and standards. One example of an engine that was evaluated against the dataset is XACMLight (http://www.immagic.com/eLibrary/TECH/OASIS/O110306G.pdf). Figure 6 below shows summary of XAMLight conformance test.

- **The total number of tests was: 333, out of which 328 were successful, 5 were not applicable.**

IIA004, IID029.1-2 (not compliant with XSD), IID030.1-2, IIE001, IIE002 are the policies that produce (not applicable) response.

Figure 6. XACMLight Conformance Test

XACML standards moved from standard 1 to 2 and then to 3. Policies and their related requests and responses can then be related to one of those three standards. Hence some tests may fail due to standards inconsistency between what the policy standard is and what the testing framework is based upon. We used WSO2 framework for testing and evaluation.

WSO2 framework is currently using XACML 3.0 standard. Figure 7 below summarizes change history of the current dataset. Numbers like IIC086 represents policy number or name.

IIC086-IIC091: change some attribute types to string.
*** IIA004, IIA005 - these files contained intentional syntax errors, which were accidentally "fixed" when converting to xacml 2.0. Syntax errors are reintroduced.**
*** IID029, IID030, IIE001, IIE002 - these files were not converted properly to to xacml 2.0: Condition had FunctionId attribute like in pre xacml 2.0 schemas.**

Figure 7. XACML 2 Tests V(4) Change Summary

Comparing Figures 6 and 7, we can see that all policies listed in Figure 8 are shown to have issues according to initial dataset. Some of those policies include intentional errors such as: IIA004, IIA005. Others have issues related to conformance with XACML standards. IIA005 is mentioned to have an intentional error but is now shown in XACMLight conformance test.

In another example, tester evaluates XACML policies dataset against the tool (http://xmlsoft.org/xmllint.html) for policies evaluation. However, no elaborate details are shown to indicate detail results of conformance testing results (http://comments.gmane.org/gmane.comp.java.castor.user/9164). General results showed that 388 policies were tested, II004Policy.xml has a deliberate error, and 387 are valid according to xmllint. Results were posted in 2010. This means that authors were evaluating against XACML version 2 standards. 72 policies passed conformance testing. The rest 315 indicates conformance or validation issues including: Missing abstract element in XML instances. According to author this was largely since Castor parser deals with "Expression" element as mandatory while standard does not. The second reported problem is related to miss conception between policies and policy sets where policy sets are objects to represent containers. A policy set can have policy sets or policies.

Some of the main features that are shown to be different between XACML V1 and 2 include [20]:

- Empty Target element is allowed in XACML 2.0.
- "AnySubject" and other Any* elements in the Target are not allowed in XACML 2.0
- Environment element is required in Target for Request in XACML 2.0.
- "FunctionId" is not allowed in Condition element in XACML 2.0.
- "IssueInstant" attribute is not allowed in Attribute in XACML 2.0

We expect to see more issues in our conformance test as WSO2 is evaluating based on XACML 2 while the dataset is prepared based on version 1 and then version 2 standards.

1. Mandatory-to-Implement Functionality Tests

This section includes 21 policies with their names start with (IIA). This part includes totally 63 files where each policy file has two files representing request and response. Two policies (2 and 4) include one extra file called (special). Those special files include extra instructions to run policies 2 and 4. Originally, there are three possible outputs representing the status of request evaluation. Those are: Permit, deny or indeterminate. In addition, a

fourth status: Not applicable can be generated where there is a general mismatch between policy and request. Table 1 below shows a summary of results.

Table 1. Conformance Testing for Policies of Section A

	Response	
Policy	Expected	Actual
IIA001Policy	Permit	Indeterminate
IIA002Policy	Permit	NotApplicable
IIA003Policy	NotApplicable	NotApplicable
IIA004Policy	Invalid schema	
IIA005Policy	Indeterminate	Indeterminate
IIA006Policy	Permit	Indeterminate
IIA007Policy	Indeterminate	Indeterminate
IIA008Policy	Permit	Permit
IIA009Policy	Indeterminate	Indeterminate
IIA0010Policy	Permit	Permit
IIA0011Policy	Indeterminate	Indeterminate
IIA0012Policy	Permit	Permit
IIA0013Policy	Indeterminate	Indeterminate
IIA0014Policy	Permit	Indeterminate
IIA0015Policy	Permit	Permit
IIA0016Policy	Permit	Permit
IIA0017Policy	Permit	Permit
IIA0018Policy	Permit	Permit
IIA0019Policy	Permit	Permit
IIA0020Policy	Permit	Permit
IIA0021Policy	Permit	Permit

Here are some comments on policies in this section:

- Some policies such as IIA004 were not loaded due to improper schema. The policy has an intentional error according to its internal documentation (This policy contains INTENTIONAL syntax error in Subject Attribute Designator, Attribute It attribute is omitted).
- Many policies have incorrect internal ID (policy ID, rule ID or both) and hence it should be modified before processing. The main identifier of policies by engines is not their name but those IDs. Those should be all checked in policies' dataset.
- Each policy includes two locations to define its ID: Policy ID and Rule ID. Investigations showed that some conformance error issues are related to inconsistency between those two IDs that should be the same (Based on tests intentions).
- In summary, all tests' were according to expectations except for policies one and two.

2. B:Target Matching

"A Target is basically a set of simplified conditions for the Subject, Resource and Action that must be met for a Policy Set, Policy or Rule to apply to a given request" (OASIS). Request is then compared with the target section of the policy to make final judgment. For size limitations, we will show only significant issues in conformance testing in Tables. Table 2 summarizes significant results.

Table 2. Conformance Testing for Policies of Section B

Total Number of Policies		53
Total Expected Permit		26
Total Expected Deny		0
Total Expected NotApplicable		27
Total Expected Indeterminate		0
Total mismatch		4
Policy	Expected	Actual
IIB0016Policy	Permit	Indeterminate
IIB0017Policy	NotApplicable	Indeterminate
IIB0028Policy	Permit	Indeterminate
IIB0029Policy	NotApplicable	Indeterminate

Policies that show mismatch between actual and expected were: IIB0016, IIB0017 (Subject with specific KeyInfo value), IIB0028 and IIB0029 (multiple Subjects).

3. C: Function Evaluation

This section includes tests to test mandatory policy functions.

Table 3. Conformance Testing for Policies of Section C

Total Number of Policies		226
Total Expected Permit		180
Total Expected Deny		0
Total Expected Not-Applicable		46
Total Expected Indeterminate		0
Total mismatch		0
Policy	Expected	Actual
IIC003Policy	Parsing error	
IIC012Policy	Parsing error	
IIC014Policy	Parsing error	

With the exception of the three policies that were not parsed correctly (03: Apply with single-element bag where function expects primitive type, 12: ERROR: Condition Evaluation - non-boolean data type and 14: ERROR: function: integer-add - non-integer data type) all other polices passed conformance tests.

4. D: Combining Algorithms

Table 4 shows summary of policy testing for this section.

Table 4. Conformance Testing for Policies of Section D

Total Number of Policies		31
Total Expected Permit		8
Total Expected Deny		9
Total Expected Not-Applicable		8
Total Expected Indeterminate		6
Total mismatch		2
Policy	Expected	Actual
IID029-1	NotApplicable	Permit
IID030	Deny	Indeterminate

Two test cases failed. However, special notes are included with policies (29: Permit, Multiple initial policies, but only one applies and 30: Indeterminate: Multiple initial policies, more than one applies) where it seems that they are expected to fail.

5. E: Schema Components

As the name implies tests in this section evaluate schema conformance. Table 5 shows results of this section tests.

Table 5. Conformance Testing for Policies of Section E

Total Number of Policies		8 (3 basic ones)
Total Expected Permit		3
Total Expected Deny		2
Total Expected Not-Applicable		2
Total Expected Indeterminate		1
Total mismatch		5
Policy	Expected	Actual
IE001(policy)	NotApplicable	Permit
IE001(policy set)	Deny	Permit
IE002(policy set)	Deny	Permit
IE002(policy)	NotApplicable	Permit
IE003(policy set)	Indeterminate	Permit

Most of tests in this section fail. This is expected since WSO2 tests based on XACML2 while collected dataset was prepared based on XACML2.

6. F: XACML 2.0 New Features

This set is supposed to test new features in XACML2. No test case was included in this section. Further, such tests may not be relevant for XACML3 based conformance testing.

7. G: Optional, but Normative Functionality Tests

This section of test cases test optional policy sections.

8. GA: Defaults Type

The first section is related to optional Obligations. This section includes 28 policies. Table 6 includes details of testing section G.A.

Table 6. Conformance Testing for Policies of Section G.A.

Total Number of Policies		28
Total Expected Permit		8
Total Expected Deny		8
Total Expected Not-Applicable		7
Total Expected Indeterminate		5
Total mismatch		2
Policy	Expected	Actual
IID029-1	NotApplicable	Permit
IID030	Deny	Indeterminate

9. Hierarchical Resources

Table 7 shows results of executing test cases to test this policy section.

Table 7. Conformance Testing for Policies of Section G.A.

Total Number of Policies		3
Total Expected Permit		2
Total Expected Deny		1
Total Expected Not-Applicable		0
Total Expected Indeterminate		0
Total mismatch		1
Policy	Expected	Actual
IIIC003	Deny	Permit

All policies in the left sections: D: <ResourceContent> Element, E: Multiple Decisions, F: Attribute Selectors and G: Non-mandatory Functions failed to load through WSO2 architecture with parsing problems.

5. Conclusion

Policies should be continuously tested and evaluated as their proper functionalities are very critical to systems especially in the cloud and web environments. For testing to be effective its activities should be conducted with little or no human intervention. In addition to typical testing activities, the testing system should be able to monitor and evaluate changes in policies and possible system objects that may be affected by such changes.

This paper describes a test automation framework dedicated to test XACML based security policies. This can be implemented as a standalone testing framework or part of a web, enterprise, or cloud infrastructure.

Test automation can improve quality without the need for extensive resources. We proposed a test automation framework to generate, execute and evaluate test cases on XACML policies.

In the case study section, we evaluated the dataset of test cases available in OASIS website for testing XACML2. We used WSO2 architecture that contain the framework to export and test policies. The framework is based on XACML3 standard. We showed test cases that fail based on either initial specifications or based on conformance issues between XACML 2 and 3 standards.

References

[1] E. Fernandez-Buglioni, "Security Patterns in Practice", Designing Secure Architectures Using Software Patterns, Wiley Software Patterns Series, Wiley, 1 edition, **(2013)** May 28**.**

[2] K. Fisler, S. Krishnamurthi, L. A. Meyerovich and M. C. Tschantz, "Verification and change-impact analysis of access-control policies", ICSE '05: Proceedings of the 27th international conference on Software engineering, New York, NY, USA, ACM Press, **(2005),** pp. 196—205.

[3] K. Fisler, S. Krishnamurthi, L. A. Meyerovich and M. C. Tschantz, "Verification and change-impact analysis of access-control policies", ICSE, ACM, **(2005)**, pp. 196-205.

[4] E. Martin and T. Xie, "Automated test generation for access control policies via change-impact analysis", In Proc. of SESS, **(2007),** pp. 5-12.

[5] J. Biskup and J. Lopez, "Change-Impact Analysis of Firewall Policies, (Eds.): ESORICS 2007, LNCS, vol. 4734, **(2007),** pp. 155–170.

Authors

Izzat Alsmadi, he is an associate professor in software engineering affiliated with Yarmouk University in Jordan. He is currently in a leave at Prince Sultan University, KSA. He has his PhD in software engineering from NDSU, USA 2008. His main research interests are in software engineering and security.