Towards a JSON-based Fast Policy Evaluation Framework
(short paper)

Hao Jiang and Ahmed Bouabdallah

IMT Atlantique, SRCD Dept., Site of Rennes, 35510 Cesson-Sévigné, France
`{hao.jiang, ahmed.bouabdallah}@imt-atlantique.fr`

Abstract. In this paper we evaluate experimentally the performance of JACPoL, a previously introduced JSON-based access control policy language. The results show that JACPoL requires much less processing time and memory space than XACML by testing generic families of policies expressed in both languages.

Keywords: low-latency access control, efficient policy evaluation, JSON

1 Introduction

Policies represent sets of properties of information processing systems [1]. Their implementation mainly rests on the IETF architecture [2] initially introduced to manage QoS policies in networks. It consists in two main entities namely the PDP (Policy Decision Point) and the PEP (Policy Enforcement Point). The first one which is the smart part of the architecture acts as a controller the goal of which consists in handling and interpreting policy events, and deciding in accordance with the policy currently applicable, what action should be taken. The decision is transmitted to the PEP which has to concretely carry it out.

Access control policies are a specific kind of security policies aiming to control the actions that principals can perform on resources by permitting their access only to the authorized ones. Typically, the access requests are intercepted and analyzed by the PEP, which then transfers the request details to the PDP for evaluation and authorization decision [2]. In most implementations, the stateless nature of PEP enables its ease of scale. However, the PDP has to consult the right policy set and apply the rules therein to reach a decision for each request and thus is often the performance bottleneck of policy-based access control systems. Therefore, a policy language determining how policies are expressed and evaluated is important and has a direct influence on the performance of the PDP.

Especially, in nowadays, protecting private resources in real-time has evolved into a rigid demand in domains such as home automation, smart cities, health care services and intelligent transportation systems, etc., where the environments are characterized by heterogeneous, distributed computing systems exchanging enormous volumes of time-critical data with varying levels of access control in a dynamic network. An access control policy language for these environments needs to be very well-structured, expressive but lightweight and easily extensible [3].

In the past decades, a lot of policy languages have been proposed for the specification of access control policies using XML, such as EPAL [4], X-GTRBAC [5] and the standardized XACML [6]. Nevertheless, it is generally acknowledged that XACML suffers from providing poorly defined and counterintuitive semantics [7], which makes it not good in simplicity and flexibility. On the other hand, XML performs well in expressiveness and adaptability but sacrifices its efficiency and scalability, compared to which JSON is considered to be more well-proportioned with respect to these requirements, and even simpler, easier, more efficient and thus favored by more and more nowadays' policy designers [8][9][10][11]. To address the aforementioned inefficiency issues of the XML format, the XACML Technical Committee recently designed the JSON profile [12] to be used in the exchange of XACML request and response messages between the PEP and PDP. However, the profile does not define the specification of XACML policies, which means, after the PDP parses the JSON-formatted XACML requests, it still needs to evaluate the parsed attributes with respect to the policies expressed in XML.

Leigh Griffin and his colleagues [10] have proposed JSONPL, a policy vocabulary encoded in JSON that semantically was identical to the original XML policy but stripped away the redundant meta data and cleaned up the array translation process. Their performance experiments showed that JSON could provide very similar expressiveness as XML but with much less verbosity. On the other hand, as much as we understand, JSONPL is merely aimed at implementing XACML policies in JSON and thus lacks its own formal schema and full specification as a policy specification language [13]. Major service providers such as Amazon Web Services (AWS) [14] have a tendency to implement their own security languages in JSON, but such kind of approaches are normally for proprietary usage thus provide only self-sufficient features and support limited use cases, which are not suitable to be a common policy language.

To the best of our knowledge, there are very few proposals that combine a rich set of language features with well-defined syntax and semantics, and such kind of access control policy language based on JSON has not even been attempted before. According to these observations, we proposed in a previous paper [15] a simple but expressive access control policy language (JACPoL) based on JSON. JACPoL by design provides a flexible and fine-grained ABAC (Attribute-based Access Control), and meanwhile it can be easily tailored to express a broad range of other access control models. We carefully positioned JACPoL in comparison with the traditional policy language XACML and we showed that JACPoL is more lightweight, scalable and flexible [15].

This paper presents a quantitative evaluation of JACPoL. We focus experimentally on the PDP performances. Using a common Python implementation, we assess the speed of writing, loading and processing generic families of policies expressed in JACPoL and XACML together with their relative memory consumption. The comparisons show that JACPoL systematically requires much less time and memory space.

The rest of the paper provides a quick overview of JACPoL in Section 2, a detailed performance evaluation in Section 3 and a conclusion in Section 4.

2 Fundamentals of JACPoL

This section recall the foundations of JACPoL, and then introduce its structures with an overview of how an access request is evaluated with respect to JACPoL policies. A detailed introduction of JACPoL can be found in [15].

JACPoL is JSON-formatted [16]. It is also attribute-based by design but meanwhile supports RBAC [17]. When integrating RBAC, user roles are considered as an attribute (ARBAC) [18], or attributes are used to constrain user permissions (RABAC) [19], which obtains the advantages of RBAC while maintaining ABAC's flexibility and expressiveness. JACPoL adopts hierarchically nested structures similar to XACML. The layered architecture as shown in Fig. 1 not only enables scalable and fine-grained access control, but also eases the work of policy definition and management for policy designers. JACPoL supports *Implicit Logic Operators* which make use of JSON built-in data structures (*Object* and *Array*) to implicitly denote logic operations. This allows a policy designer to express complex operations without explicitly using logical operators, and makes JACPoL policies greatly reduced in size and easier to read and write by humans. Finally JACPoL supports *Obligations* to offer a rich set of security and network management features.

JACPoL uses hierarchical structures very similar to the XACML standard [20]. As shown in Fig. 1, JACPoL policies are structured as *Policy Sets* that consist of one or more child policy sets or policies, and a *Policy* is composed of a set of *Rules*.

Because not all *Rules*, *Policies*, or *Policy Sets* are relevant to a given request, JACPoL includes the notion of a *Target*. A *Target* determines whether a *Rule*/*Policy*/*Policy Set* is applicable to a request by setting constraints on attributes using simple Boolean expressions. A *Policy Set* is said to be *Applicable* if the access request satisfies the *Target*, and if so, then its child *Policies* are evaluated and the results returned by those child policies are combined using the policy-combining algorithm; otherwise, the *Policy Set* is skipped without further examining its child policies and returns a *Not Applicable* decision. Likewise, the *Target* of a *Policy* or a *Rule* has similar semantics.

The *Rule* is the fundamental unit that is evaluated eventually and can generate a conclusive decision (*Permit* or *Deny* specified in its *Effect* field). The *Condition* field in a rule is a simple or complex Boolean expression that refines the applicability of the rule beyond the predicates specified by its target, and is optional. If a request satisfies both the *Target* and *Condition* of a rule, then the rule is applicable to the request and its *Effect* is returned as its decision; otherwise, *Not Applicable* is returned.

For each *Rule*, *Policy*, or *Policy Set*, an *id* field is used to be uniquely identified, and an *Obligation* field is used to specify the operations which should be performed (typically by a PEP) before or after granting or denying an access request, while a *Priority* is specified for conflict resolution between different *Rules*, *Policies*, or *Policy Sets*.

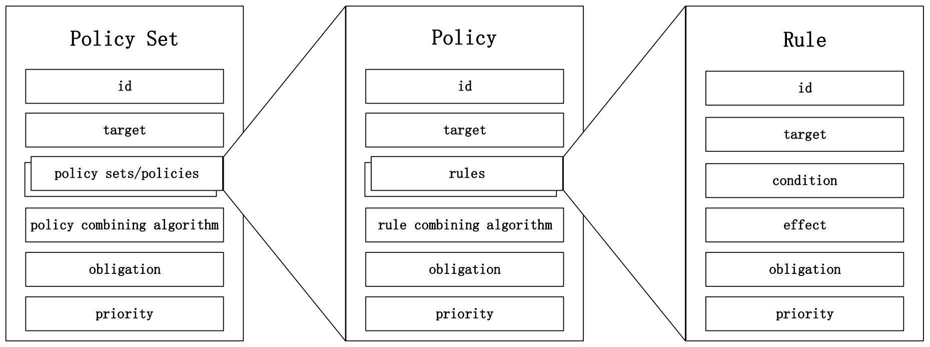


Fig. 1: JACPoL's hierarchical nested structure

3 Performance Evaluation

This section presents the results of performance tests on JACPoL and XACML. We assessed respectively how both languages' evaluation time and memory usage are affected with regards to the increase of nesting policies (policy depth) and the increase of sibling ones (policy scale). The values are ranged from 0 to 10000 with a step of 1000, as shown in Fig. 2.

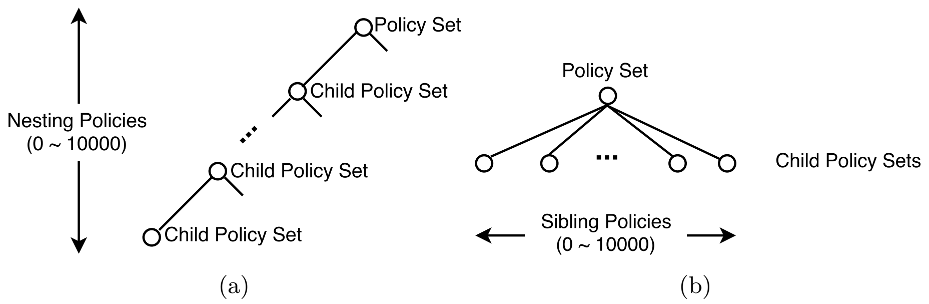


Fig. 2: Examples of nesting policies (a) and sibling policies (b)

We used the two policy languages to express the same policy task and compared their performances with different depth (number of nesting policies) and scale (number of sibling policies). Fig. 3 provides XACML and JACPoL policy examples used for assessment. The two ellipses in each example indicate nesting (the upper and inner ellipsis) and sibling (the lower and outer ellipsis) child policies respectively. When processing a given request, the Boolean expression in the *Target* field of each policy is evaluated to verify the applicability of its child policies until the last child policy is evaluated. Note that the JSON profile of XACML defines XACML requests and responses but not policies, therefore we are unable to compare JACPoL with JSON-formatted XACML.

```
<PolicySet Id="0" PolicyCombiningAlgorithm="firstApplicable" Update="2017-03-14 17:18:31" Version="1">
   <Target>
      <Subjects>
         <Subject>
            <SubjectMatch MatchId="…#string-equal">
               <AttributeValue DataType="…#string">Sam</AttributeValue>
               <SubjectAttributeDesignator AttributeId="…:subject:subject-id" DataType="…#string" />
            </SubjectMatch>
         </Subject>
      </Subjects>
   </Target>
   <PolicySet Id="1" PolicyCombiningAlgorithm="firstApplicable" Update="2017-03-14 17:18:31" Version="1">
      …
   </PolicySet>
   …
</PolicySet>
```

(a)

```
{
   "Target": {"Subject": {"equals": "Sam"}},
   "Update": "2017-03-14 17:18:31",
   "PolicyCombiningAlgorithm": "firstApplicable",
   "Version": 1,
   "Id": 0,
   "Policies": [
      {
         "Target": {"Subject": {"equals": "Sam"}},
         "Update": "2017-03-14 17:18:31",
         "PolicyCombiningAlgorithm": "firstApplicable",
         "Version": 1,
         "Id": 1,
         "Policies": […]
      },
      …
   ]
}
```

(b)

Fig. 3: Examples of XACML policies (a) and JACPoL policies (b)

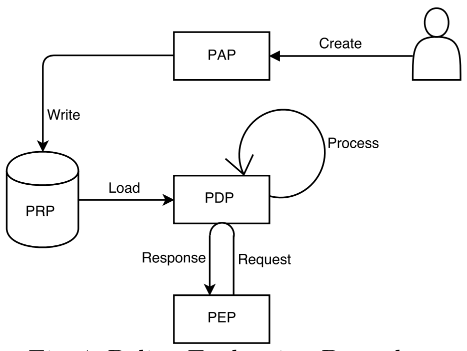
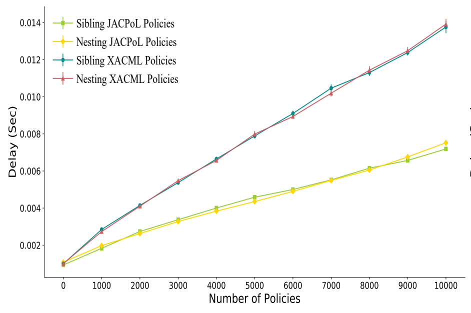


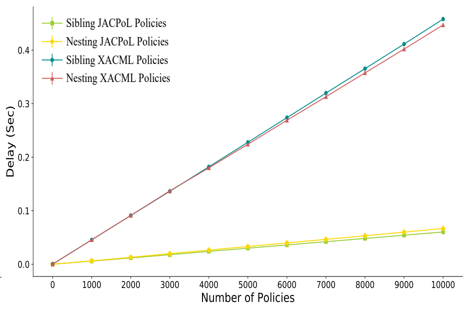
Fig. 4: Policy Evaluation Procedure

As shown in Fig. 4, the policy evaluation performance of the PDP is very related to the writing, loading and processing performance of a policy language. We agree that the schemes and technologies used in requesting/responding operations would also influence the performance, which is however beyond the scope of current JACPoL's specification.

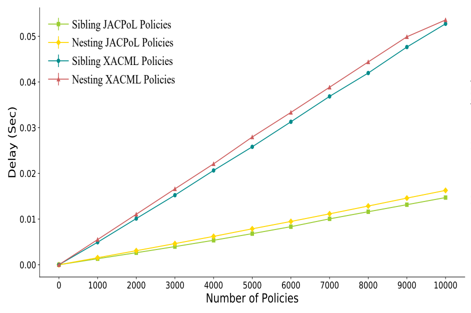
Therefore, we have conducted 4 sets of tests in order to evaluate the writing, loading and processing speed and the memory consumption of the two policy languages. Respectively, the writing test measures the average time (in seconds) to write policies from the memory into a single file; the loading test measures the average time (in seconds) to load policies from a single file into the memory; the processing test measures the average time (in seconds) to process a request against policies that are already loaded in the memory; and the last test measures the space consumption (in Megabytes) of policies in the memory. Each test evaluates the two languages with different policy nesting depth and sibling scale, and was repeated 10000 times conducted using Python on a Windows 10 ASUS N552VW laptop with 16G memory and a 2.6GHz Intel core i7-6700HQ processor.



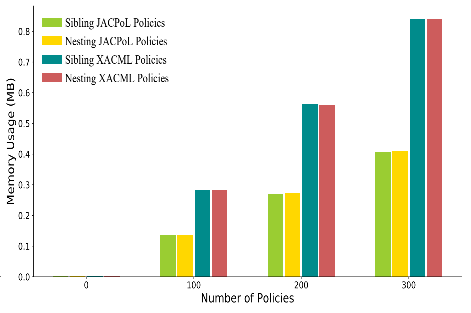
(a) writing time consumption



(b) loading time consumption



(c) processing time consumption



(d) memory consumption

Fig. 5: Comparative performance evaluation between the JACPoL and XACML

Fig. 5a shows the writing time versus the number of policies. We observe that there are near linear correlations between the average writing time and number policies (both in depth and in scale) for both languages. Compared to XACML, JACPoL consumes only approximately half of the time taken by the former.

Fig. 5b shows the average loading time versus the number of policies in depth and in scale. Similar to Fig. 5a, there is an almost linear correlation between the loading time and the number of policies. We can see that the loading time of JACPoL becomes much less than XACML as the number of policies increases.

Fig. 5c shows the processing time versus the number of policies. Similar as the writing time and loading time, there is also an almost linear correlation between the processing time and the number of policies. On the other hand, unlike Fig. 5a and Fig. 5b, we can see that the policy structure would influence the processing time given the same policy size. We also observe that JACPoL is processed faster than XACML policies.

Fig. 5d shows the memory consumption versus the number of policies. Similar as all figures above, the memory consumption increases also almost linearly with the growth of the number of policies. Given the same policy size, the memory space used by JACPoL is nearly half of that used by XACML.

These figures demonstrate that JACPoL is highly scalable and efficient in comparison with XACML, with much faster writing, loading, processing speed and lower memory consumption.

4 Conclusion

Real-time requirements rarely intervene in the design of access control systems. Applications are emerging, however, that require policies to be evaluated with a very low latency and high throughput. We proposed in a previous paper [15] JACPoL, as a new JSON-based, attribute-centric and light-weight access control policy language which was showed through a comprehensive qualitative analysis, as expressive as XACML but more simple, scalable and flexible. We introduce in this paper a first level of quantitative evaluation focusing on the PDP performances by assessing the speed of writing, loading and processing generic families of policies expressed in JACPoL and XACML together with their relative memory consumption. The comparisons show that JACPoL systematically requires much less time and memory space.

5 Acknowledgement

We acknowledge the reviewers of C&TC'17 for their constructive comments.

References

1. Clarkson M.R., Schneider F.B.: "Hyperproperties." Journal of Computer security 18, 2010, pp. 1157-1210.

2. Yavatkar R, Pendarakis D, Guerin R : "A Framework for Policy-based Admission Control." IETF, RFC 2753, January 2000.
3. Borders K, Zhao X, Prakash A: "CPOL: High-performance policy evaluation." the 12th ACM conference on Computer and communications security. ACM (2005).
4. Ashley P, Hada S, Karjoth G, Powers C, Schunter M: "Enterprise privacy authorization language (EPAL)." IBM Research. (2003 Mar.)
5. Bhatti R, Ghafoor A, Bertino E, Joshi JB: "X-GTRBAC: an XML-based policy specification framework and architecture for enterprise-wide access control." ACM Transactions on Information and System Security (TISSEC) 8.2 (2005): 187-227.
6. OASIS XACML Technical Committee: "eXtensible access control markup language (XACML) Version 3.0." Oasis Standard, OASIS (2013). URL: http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-os-en.html, last accessed 2017/05/17.
7. Crampton J, Morisset C: "PTaCL: A language for attribute-based access control in open systems." In International Conference on Principles of Security and Trust, pp. 390-409. Springer Berlin Heidelberg (2012).
8. Crockford D: "JSON — The fat-free alternative to XML (Vol. 2006)." URL: http://www.json.org/fatfree.html. last accessed 2017/05/17.
9. El-Aziz AA, Kannan A: "JSON encryption." Computer Communication and Informatics (ICCCI), 2014 International Conference on. IEEE (2014).
10. Griffin L, Butler B, de Leastar E, Jennings B, Botvich D: "On the performance of access control policy evaluation." In Policies for Distributed Systems and Networks (POLICY), 2012 IEEE International Symposium on, pp. 25-32. IEEE (2012).
11. W3schools: "JSON vs XML." URL: www.w3schools.com/js/js_json_xml.asp, last accessed 2017/05/24.
12. David Brossard: "JSON Profile of XACML 3.0 Version 1.0." XACML Committee Specification 01, 11 December 2014. URL: http://docs.oasis-open.org/xacml/xacml-json-http/v1.0/cs01/xacml-json-http-v1.0-cs01.pdf, last accessed 2017-05-26.
13. Steven D., Bernard B., Leigh G.: "JSON-encoded ABAC (XACML) policies." FAME project of Waterford Institute of Technology. Presentation to OASIS XACML TC concerning JSON-encoded XACML policies, 2013-05-30.
14. Amazon Web Services: "AWS Identity and Access Management(IAM) User Guide." URL: http://docs.aws.amazon.com/IAM/latest/UserGuide/introduction.html, last accessed 2017-05-27.
15. Jiang H, Bouabdallah A: "JACPoL: A Simple but Expressive JSON-based Access Control Policy Language." The 11th WISTP International Conference on Information Security Theory and Practice (WISTP'2017). September 28-29, 2017, Heraklion, Crete, Greece. To appear in the LNCS series - Ed. Springer.
16. ECMA International: "ECMA-404 The JSON Data Interchange Standard." URL: http://www.json.org/, last accessed 2017-05-27.
17. Ferraiolo DF, Kuhn DR. "Role-based Access Controls." arXiv preprint arXiv: 0903.2171. (2009 Mar 12).
18. Obrsta L., McCandlessb D., Ferrella D.: "Fast semantic Attribute-Role-Based Access Control (ARBAC) in a collaborative environment" 2012 8th International Conference on Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom), Pittsburgh, PA, USA, 14-17 Oct. 2012.
19. Jin X, Sandhu R, Krishnan R: "RABAC: role-centric attribute-based access control." International Conference on Mathematical Methods, Models, and Architectures for Computer Network Security. Springer Berlin Heidelberg (2012).
20. Ferraiolo David, et al. "Extensible Access Control Markup Language (XACML) and Next Generation Access Control (NGAC)." Proceedings of the 2016 ACM International Workshop on Attribute Based Access Control. ACM (2016).