Idea: Efficient Evaluation of Access Control Constraints

Achim D. Brucker and Helmut Petritsch

SAP Research, Vincenz-Priessnitz-Str. 1, 76131 Karlsruhe, Germany
{achim.brucker,helmut.petritsch}@sap.com

Abstract. Business requirements for modern enterprise systems usually comprise a variety of dynamic constraints, i. e., constraints that require a complex set of context information only available at runtime. Thus, the efficient evaluation of dynamic constraints, e. g., expressing separation of duties requirements, becomes an important factor for the overall performance of the access control enforcement.

In distributed systems, e. g., based on the service-oriented architecture (SOA), the time for evaluating access control constraints depends significantly on the protocol between the central Policy Decision Point (PDP) and the distributed Policy Enforcement Points (PEPs).

In this paper, we present a policy-driven approach for generating customized protocol for the communication between the PDP and the PEPs. We provide a detailed comparison of several approaches for querying context information during the evaluation of access control constraints.

Keywords: distributed policy enforcement, XACML, access control.

1 Introduction

Business regulations, e. g., Basel II [3] or the Sarbanes-Oxley Act [16], often require the enforcement of dynamic or context-aware access control policies [9, 17], for example, (dynamic) separation-of-duties. As many widely used policy languages, e. g., [6, 15], are not supporting such context requirements as first class citizens, they are usually encoded as access control constraints [7]. For example, XACML [15], PERMIS [6], or SecureUML [4] are supporting access control constraints. By definition, these constraints depend on context information such as attributes of a resource (e. g., its owner, last modification date, shipping destination) and, as such, are not amenable for caching access control decisions [12]. Thus, an efficient attribute retrieval becomes even more important; astonishingly this is left unspecified in many policy frameworks, e. g., XACML [1] or EPAL [2].

In distributed systems following the service-oriented architecture (SOA), business services are provided by orchestrating a set of loosely coupled technical services. Still, authentication and authorization rely on centralized components: single sign on implementations authenticate users within highly distributed systems using a centralized authority. Moreover, while access control policies are enforced by distributed Policy Enforcement Points (PEPs), the central Policy

Decision Point (PDP) evaluates the underlying access control requests. Such a centrally managed and administered PDP stores access control policies for all secured services. For evaluation of dynamic constraints, attributes need to be resolved (i. e., accessible) at runtime within the PDP. Albeit, often the resolution of attributes can only be done within the service (i. e., its PEP) requesting the policy evaluation. Thus, resolving attributes often requires substantial network communication between the centralized PDP and the various distributed PEPs. All PDP implementations we are aware of are based on an iterative approach. In contrast, we propose to optimize the required communication overhead by using a static policy analysis that allows for sending all attributes required for a specific access control request at once.

Our contributions are three-fold: first, we present an approach for pre-computing the required attributes for evaluating access control constraints, second, we analyze and present the performance tests for several attribute resolution strategies for distributed enterprise systems, and, third, we give guidance on choosing an optimal resolution strategy based on multiple criteria.

2 Efficient Attribute Resolution

2.1 Context Attributes

Context attributes can be classified with respect to the runtime environment in which they can be resolved efficiently. For example, while information about the accessed object is usually only available in the context of the service issuing the access control request, information about a user (e. g., his roles) are often only available within the PDP. Thus, attributes can be categorized into:

PDP attributes are only available within the centralized security infrastructure, i. e., the PDP. The information about role hierarchy membership is, usually, an example for this kind of information.

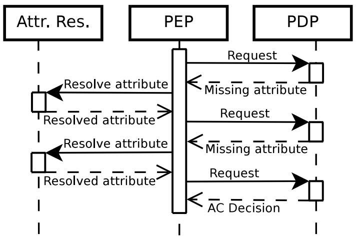
Service attributes are only available within the client application or service, e. g., owner of a resource, specific attributes of a resource (e. g., balance of a bank account) or number of threads running on the application server.

Global attributes are equally resolvable from either the PDP or the services. For example, this is the case for attributes that need to be resolved by an additional service, e. g., the single sign-on or identity provider.

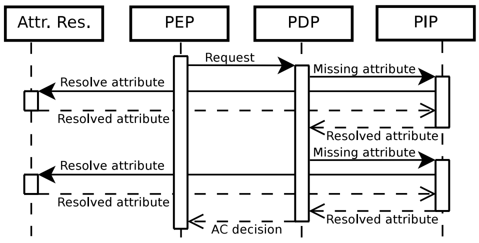
While in common implementations such a classification is left implicit, we propose to use such a classification explicitly.

2.2 Attribute Resolution Strategies

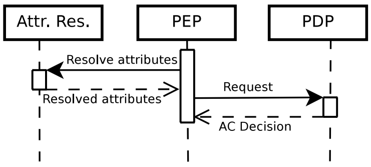
There exist several approaches for the resolution of context attributes during the evaluation of access control requests. Fig. 1 illustrates the approaches we will discuss in the following. These sequence diagrams only illustrate the resolution of service attributes, i. e., within the PEP. The resolution of global or PDP attributes using a Policy Information Point (PIP) or context provider is left out.



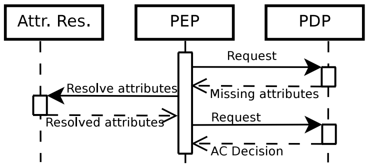
(a) The "trial and error" strategy.



(b) Resolving attributes using a PIP.



(c) PEP-prefetch.



(d) PDP-prefetch.

Fig. 1. Different strategies for resolving context attributes

Trial and error: The PDP requests the required attributes from the PEP using an iterative approach (see Fig. 1a). If the evaluation of an access control request requires a service attribute, the PDP returns a missing attribute message back to the client. As the PDP is stateless, the initial request has to be re-send by the PEP until all required attributes are supplied.

PIP: The PIP is responsible for resolving all types of attributes (see Fig. 1b). For example, the PIP queries the service or PEP over an additional Web service interface, which requires this interface be reachable from the PIP.

For increasing the performance of access control requiring the resolution of context attributes, we propose to pre-compute the set of required attributes using a static analysis of the policy together with one of the following two strategies:

PEP Prefetch: Using a pre-computed look-up-table mapping access control request to the (potentially over-approximated) set of required service attributes, the PEP can easily determine which attributes *could* be required during the evaluation of a concrete access control request. Thus, the PEP resolves as much services attributes that are potentially required and sends them, proactively, together with the initial request to the PDP (see Fig. 1c).

PDP Prefetch: By deploying the pre-computed look-up-table only in the central PDP implementation, we avoid the need of customizing the various PEPs. As the PDP can easily check which service attributes might be necessary for a specific access control request, the PDP is able to request (in its first answer) all potentially required attributes at once. Therefore the number of evaluation attempts required is strictly bound to two and does not depend on the number of attributes required (see Fig. 1d).

2.3 Pre-computing Attribute Sets

In the following, we briefly discuss the steps required for pre-computing the mapping from access control requests to set of service attributes required by the two approaches based on pre-fetching. We assume that service attributes are not used within the core policy language, e. g., in case of XACML within the target match.

On an abstract level, a security policy P is a mapping of rules to access control decisions (e. g., deny or allow). For example, in case of RBAC with access control constraints, a rule is a four-tuple (g, r, a, c) where g is the required role, r is the resource, a is the action on that resource, and c is a constraint (i. e., a Boolean expression over context attributes). An access control request is a triple (u', r', a') where u' is the requesting user, r' is the resource the user is requesting access for executing action a'. Moreover, we say a rule (g, r, a, c) matches a request (u', r', a') if and only if, the user u' is a member of the role g, both $r = r'$ and $a = a'$ hold, and the access control constraint c' evaluates to true.

For PDP Prefetch, we need to compute a mapping from triple (g, r, a) to the set of potentially required service attributes. Given a policy P,

1. for each rule (g, r, a, c) in the policy, we compute a four-tuple $(g, r, a, A_s(c))$ where $A_s(c)$ is the set of service attributes that are syntactically referenced in the access control constraint c.
2. we group the set of four-tuples $(g, r, a, A_s(c))$ with respect to their lexicographic order of group, resource, and action (i. e., the triple (g, r, a)).
3. in each group from the previous step, we build die union of all required sets of service attributes, e. g., given $\big\{(g, r, a, A_s(c_1)), \dots, (g, r, a, A_s(c_n))\big\}$, we compute the triple $(g, r, a, \bigcup_{1 \leq i \leq n} A_s(c_i))$.
4. we use the consolidated set of triples as input for initializing the hash-table used in the PDP implementation.

For the PEP Prefetch strategy, we compute analogously a mapping from resource-action tuples to the set of required service attributes.

While for policy written in an high-level language (e. g., SecureUML [4]) the set of attributes can be computed exactly, for complicated technical policies (e. g., XACML using various policy sets together with different policy combining algorithms) an over-approximation seems to be a good compromise avoiding the need for the semantical analysis of the given policy. For policy languages supported by a model-driven-security toolchain, e. g., [5], the required configurations (or even the complete PEP implementation) can be generated automatically.

3 Empirical Results

3.1 Scenario

We evaluate how the different strategies for resoling context attributes behave in three different scenarios, whereas these three scenarios differ in the size of the policies and the requirement to resolve service attributes:

Scenario I: This scenario uses a relatively small policy consisting out of 200 rules for ten resources, 50 users that are mapped onto fifteen roles, about 0.5 service attributes are resolved per access control request.

Scenario II: Compared to scenario I, this scenario has an increased likelihood that service attributes need to be resolved (about 1.8 per request), the policy has 500 rules for ten resources, 200 users are mapped onto 40 roles.

Scenario III: This scenario uses a large policy with 5000 rules for 100 resources, restricting the access of 800 users mapped to 100 roles. The policies extensively use service attributes, causing that an average access control request requires the resolution of over six service attributes.

For every scenario, we generated the policy, the user-group assignments, the required configurations, and a set of 2000 test requests. Attributes are resolved to a configurable random value. Generating the access control list explicitly allows for deterministic repetition of our benchmarks ensuring the comparability of the results for the different resolution strategies.

3.2 Experimental Results

We compare the different resolution strategies using a distributed prototype based on the Sun's XACML implementation (`http://sunxacml.sourceforge.net`) running on standard hardware. The PDP offers a Web service interface for the client PEPs and we use the `AttributeFinderModule` of XACML for implementing the PIPs used by the PDP.

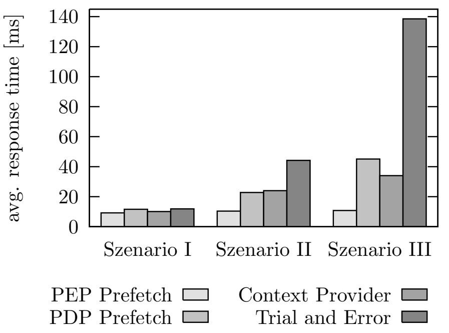
First, for each attribute resolution strategy we compare the average response times for each scenario (see Fig. 2a). In general, this time reflects a significant portion of the time a system needs for reacting on a users actions. Second, we analyze the average network load, i. e., the amount of data transmitted during an access control request (see Fig. 2b).

For a small scenario (i. e., scenario I), we do not observe a significant difference in both the average response time and the average network traffic (see Fig. 2). For larger policies (i. e., scenario II), PDP Prefetch and PIP require twice as long as PEP Prefetch and the trial and error strategy takes more than four times longer. A similar behavior can be observed in scenario III.

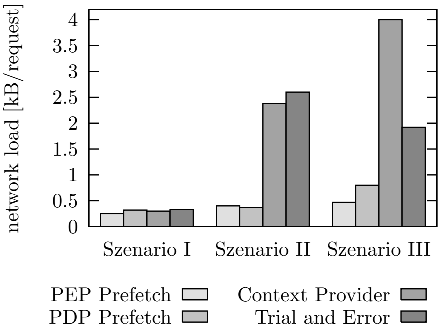
Overall, PEP Prefetch is the fastest approach in all scenarios. Interestingly, the PIP is in scenario three faster than PDP Prefetch, although the PIP has to execute on average six Web service calls back to the client, compared to one additional XACML request done by PDP Prefetch.

The average network traffic shows a similar behavior: PEP Prefetch results in the minimal amount of network traffic per access control request. Both the trail and error strategy and the PIP are resulting in an increased network traffic that does grow larger than linear with respect to the increase of the number service attribute that need to be resolved. Thus, the overhead caused by Web service calls seems to exceed the overhead of sending additional attributes.

Further, we compare the average response time depending on the number of (required) service attributes. Fig. 3 shows that the overhead caused by PEP

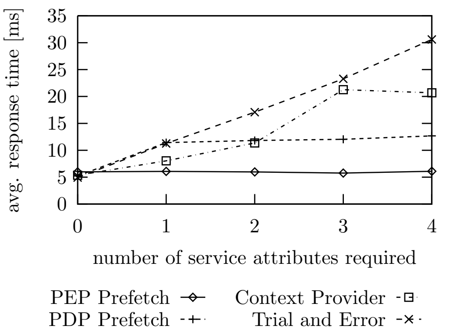


(a) Comparing the average response time per request in milliseconds.

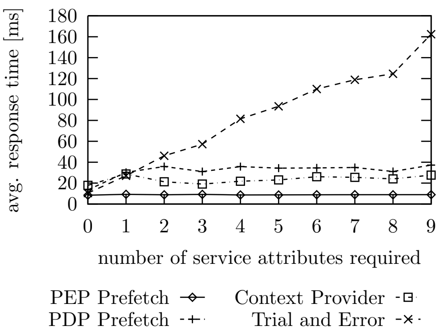


(b) Comparing the average network load (i. e., traffic in kilobytes).

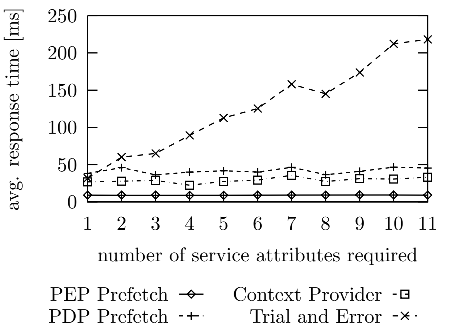
Fig. 2. Comparing the average response times and the overall network traffic for the different usage scenarios and resolution strategies



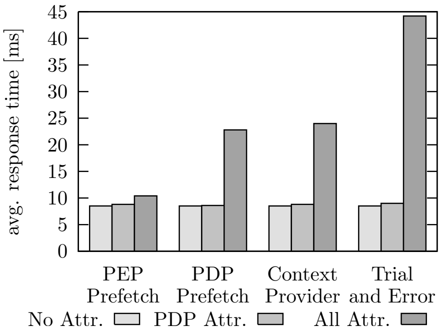
(a) Scenario I: average response time.



(b) Scenario II: average response time.



(c) Scenario III: average response time.



(d) Comparing different attribute typs.

Fig. 3. Comparing response times based on (effective) required service attributes, and comparing scenarios without attributes

Prefetch causes only minor delay in contrast to the three other approaches. In scenario I (Fig. 3a), PDP Prefetch results in the worst response time for one resolved service attribute. Albeit, it remains on this level for two and more service attributes. In contrast, the trial and error strategy has a similar response time as the PDP Prefetch approach for one service attribute, but the response time increases nearly linear with additionally required attributes.

For larger policies, i. e., scenario II and III, the size of the policy seems not to have a significant effect on how the different approaches behave relatively to each other (see Fig. 3b and Fig. 3c). Here, the PEP Prefetch strategy enjoys the fastest response time, which, moreover, also does not depend on the number of attributes required during the request evaluation.

Marshaling and unmarshaling the XACML (i. e., XML) seems to be a time consuming task: first, compared to Web service requests the effort for handling XACML seems to be very high. While for both the PDP Prefetch and the trail and error approach the same number of Web service requests are executed, the trial and error approach has a significantly slower response time.

Fig. 3d compares variants of scenario II using policies where all attributes are resolved on the PDP (i. e., causing no network overhead and delay), and without any attributes (except group resolution). As expected, using no service attributes is significantly faster. The four approaches differ only in service resolution significantly. The resolution of evaluation of local attributes (i. e., within the PDP) results in a minor overhead.

4 Discussion

Overall, our proposed resolution strategies (i. e., PEP Prefetch and PDP Prefetch) do not eliminate or replace the concept of PIPs. For resolving global or PDP attributes, a context provider directly attached to the PDP should be used.

The PEP Prefetch strategy for resolving context attributes from service side, produces only little overhead. Thus, even in scenarios rarely using service attributes, our approach is the fastest implementation—leading to the best user experience, i. e., fast response time of the system. The main drawback of the PEP Prefetch strategy is the fact that it requires an update of the PEP configuration whenever the security policy (i. e., the set of rules) requires additional service attributes for evaluation. The PDP Prefetch strategy overcomes this drawback while being only resulting in slightly higher response times and network traffic. As our implementation allows to use the PEP Prefetch strategy with a fallback strategy (PDP Prefetch or trial and error, e. g., for the time the updates need to be distributed to all PEPs after a policy update), we propose to use the PEP Prefetch strategy with a fallback strategy as a general solution.

Overall, the overhead for marshaling and unmarshaling XACML requests encoded in XML seems to be very high. Unfortunately, our experiments cannot give hints if this is an implementation specific or a general problem. Thus, a more efficient implementation may increase the overall performance in general and the performance of the PDP Prefetch approach in particular.

We assume that resolving attributes is possible with low costs. We think that this is a reasonable assumption (i. e., many attributes required will be assigned to or available within the current context of the access control request, e. g., owner of the accessed resource as attribute or property of the resource itself). Nonetheless, this cannot be guaranteed for all attributes. Especially for the context provider approach, this assumption is problematic: as the call back to the service is asynchronous, the resolution is not executed within the context of the access control request. Thus, the performance and costs of the resolution depends mainly on the underlying application: an intelligent session handling may allow a fast access of the request context and, therefore, allow the resolution with nearly the same costs as from within the context itself.

Classifying the attribute types depending in which parts of the overall system landscape they can be resolved efficiently seems to be a valuable add-on to existing policy frameworks. Overall, such a classification avoids, first, the need to transport any kind of status from the client over PDP and PIP to the attribute resolving component of the client. Second, in the context of the access control request they are for usual accessible without or with low costs.

5 Conclusion and Future Work

Work on improving the performance of PDPs in distributed service-oriented environments mainly focuses on three aspects: first, providing highly efficient PDP implementations for static policies (e. g., [13]), second, optimizing the evaluation performance by optimizing, statically, the policy (e. g., [14]). Third, while there is a large body of literature, e. g., [8, 10, 11], on improving the performance of security frameworks using caching strategies, only proactive caching [11, 12] addresses the caching of dynamic access control properties. Still, proactive caching is only able to cache dynamic access control constraints if they are first-class citizens of the underlying access control language.

As pre-computing the set of required attributes significantly helps in reducing both the delays caused by resolving context attributes and the overall network traffic, combining access control caching strategies with the attribute resolution approaches presented in this paper seems to be an attractive option for improving the overall performance of today's enterprise systems.

Furthermore, the performance overhead caused by encoding requests in XACML needs to be explored: either by switching to a different XACML policy decision engine, e. g., [13] or by looking at different policy decision languages and evaluation techniques.

Acknowledgments. This work has been supported by the German "Federal Ministry of Education and Research" in the context of the projects "SoKNOS" and "Polytos." The authors are responsible for the content of this publication.

References

[1] Anderson, A.H.: A comparison of two privacy policy languages: EPAL and XACML. In: ACM workshop on Secure Web services (SWS), pp. 53–60. ACM Press, New York (2006)

[2] Ashley, P., Hada, S., Karjoth, G., Powers, C., Schunter, M.: Enterprise privacy authorization language (EPAL 1.2). Tech. rep., IBM (2003), `http://www.zurich.ibm.com/security/enterprise-privacy/epal`

[3] Basel Committee on Banking Supervision: Basel II: International convergence of capital measurement and capital standards. Tech. rep., Bank for International Settlements, Basel, Switzerland (2004), `http://www.bis.org/publ/bcbsca.htm`

[4] Basin, D.A., Doser, J., Lodderstedt, T.: Model driven security: From UML models to access control infrastructures. ACM Transactions on Software Engineering and Methodology 15(1), 39–91 (2006)

[5] Brucker, A.D., Doser, J., Wolff, B.: An MDA framework supporting OCL. Electronic Communications of the EASST 5 (2006)

[6] Chadwick, D., Zhao, G., Otenko, S., Laborde, R., Su, L., Nguyen, T.A.: PERMIS: a modular authorization infrastructure. Concurrency and Computation: Practice & Experience 20(11), 1341–1357 (2008)

[7] Chen, H., Li, N.: Constraint generation for separation of duty. In: ACM symposium on access control models and technologies (SACMAT), pp. 130–138. ACM Press, New York (2006)

[8] Crampton, J., Leung, W., Beznosov, K.: The secondary and approximate authorization model and its application to Bell-LaPadula policies. In: ACM symposium on access control models and technologies (SACMAT), pp. 111–120. ACM Press, New York (2006)

[9] Kapsalis, V., Hadellis, L., Karelis, D., Koubias, S.: A dynamic context-aware access control architecture for e-services. Computers & Security 25(7), 507–521 (2006)

[10] Karjoth, G.: Access control with IBM Tivoli access manager. ACM Transactions on Information and System Security 6(2), 232–257 (2003)

[11] Kohler, M., Brucker, A.D., Schaad, A.: ProActive Caching: Generating caching heuristics for business process environments. In: Conference on Computational Science and Engineering (CSE), vol. 3, pp. 207–304. IEEE Computer Society, Los Alamitos (2009)

[12] Kohler, M., Schaad, A.: Pro active access control for business process-driven environments. In: Annual Computer Security Applications Conference (ACSAC) (2008)

[13] Liu, A.X., Chen, F., Hwang, J., Xie, T.: XEngine: A fast and scalable XACML policy evaluation engine. In: Conference on Measurement and Modeling of Computer Systems, Sigmetrics (2008)

[14] Miseldine, P.L.: Automated XACML policy reconfiguration for evaluation optimisation. In: Software engineering for secure systems (SESS), pp. 1–8. ACM Press, New York (2008)

[15] OASIS: eXtensible Access Control Markup Language (XACML) 2.0 (2005), `http://docs.oasis-open.org/xacml/2.0/XACML-2.0-OS-NORMATIVE.zip`

[16] Sarbanes, P., Oxley, G., et al.: Sarbanes-Oxley Act of 2002. 107th Congress Report, House of Representatives, pp. 107–610 (2002)

[17] Schaad, A., Spadone, P., Weichsel, H.: A case study of separation of duty properties in the context of the Austrian "eLaw" process. In: ACM symposium on applied computing (SAC), pp. 1328–1332. ACM Press, New York (2005)