Policy Management in Cloud Computing Environments

Hassan Takabi
School of Information Sciences
University of Pittsburgh
Pittsburgh, PA, USA
hatakabi@sis.pitt.edu

James B. D. Joshi
School of Information Sciences
University of Pittsburgh
Pittsburgh, PA, USA
jjoshi@sis.pitt.edu

ABSTRACT

In this paper, we focus on access control systems in Cloud environments. Currently, users must use diverse access control solutions available for each cloud service provider to secure data. Heterogeneity and distribution of these policies pose problems in managing access policy rules for cloud environment. Here, we introduce a semantic based policy management framework that is designed to give users a unified control point for managing policies that control access to their data no matter where the data is stored. We also present a proof of concept implementation and results of performance evaluation.

1. INTRODUCTION

The cloud computing environment does not allow use of a single authorization mechanism, single policy language or single management tool. Currently, users must use diverse access control solutions available for each cloud service provider to secure their data and control its dissemination. Access control policies may be composed in incompatible policy languages and maintained separately at every cloud service provider.

An ideal access control scheme must be able to work with all types of content regardless of where they are stored. Users should be able to manage policies to govern access to their information and resources from a central location. Heterogeneity and distribution of access policies pose significant problems in managing them in cloud computing environments. Furthermore, traditional non-semantic-based access control approaches are inadequate for supporting interoperability for cloud computing environments.

Using OWL for specifying access policies provides some important advantages that are critical in cloud computing environments involving interoperation across multiple cloud providers. It also enables involved entities to better understand and share the security policies. In order to specify a policy, one should be able to precisely specify classes of subjects, objects, actions, etc. Having a centralized policy management helps the users to use same access policies in multiple CSPs; each provider must understand and enforce the policy rules even though they have their own schemas or data models. There needs to be a platform that provides access control to the resources and services of all the different CSPs. Using OWL gives us a natural and efficient way of specifying these classes and access policies. The second advantage is that OWL is based on description logic and we can translate access policies expressed in OWL to other policy languages and formalisms.

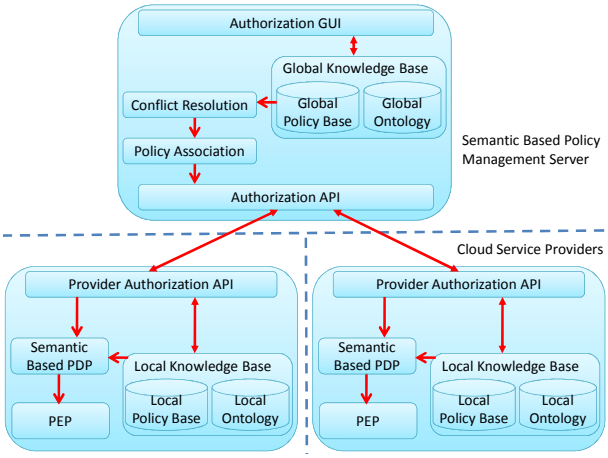

Figure 1: The Proposed Semantic Based Policy Management Framework

2. THE SEMANTIC BASED POLICY MANAGEMENT FRAMEWORK

Our proposed semantic based policy management framework is built on the concept of centrally expressing a user's security requirements that are applied to a user's resources scattered across the cloud. The customers do not manage or control the underlying cloud infrastructure, network, servers, operating systems, storage, or even individual application capabilities. Figure 1 illustrates the framework and its components which are briefly described in the followings. At a high level, the two components of the proposed architecture are the *cloud service provider* and the *semantic based policy management service*.

Each CSP includes a *semantic based policy decision point (PDP)*, a *policy enforcement point (PEP)* and its own *local knowledge base*. The local knowledge base stores all the on-

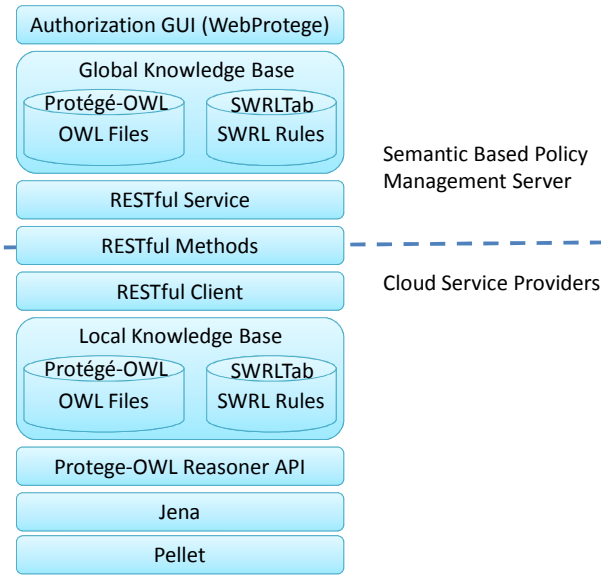

Figure 2: The Implementation Architecture of the Proposed Framework

tologies and the policy rules of the CSP. The semantic based PDP component is in charge of making authorization decisions while the PEP module is in charge of enforcing those decisions.

The SBPMS provides authorization services and interfaces for cloud users to manage access policies in a single centralized location. Once the policies are specified by cloud users, it detects and resolves possible conflicts among access policies. Then, it exports policies into the CSPs. In order to do this, it first separates the policies related to each CSP based on the resource-provider association and then exports them into the associated CSP via the *provider authorization API*. The SBPMS has several components. The *global knowledge base* is a central repository that stores all the ontologies and the policy rules gathered from different CSPs' local knowledge bases. This is done using the *provider authorization APIs*.

The *authorization GUI* provides users with information that is required for specifying their access policies that are retrieved from the global knowledge base. The *conflict resolution* module then detects and resolves possible conflicts among the specified access policies while the *policy association* module is responsible for associating the policies with their target CSPs and services. Finally, an *authorization API* provides all the services related to the management of authorization services and it is accessed via the provider authorization API.

3. THE IMPLEMENTATION ARCHITECTURE

Our implementation contains two different Java based components: the CSP and the SBPMS. The SBPMS contains authorization API, knowledge management and authorization GUI and the CSP contains the provider authorization API, knowledge management and semantic based PDP.

As shown in Figure 2, the authorization API and the provider authorization API are published by means of RESTful Web service technology, which enables the remote invocation of the methods for the different parties involved. A RESTful Web service is implemented in the SBPMS side and the CSPs are implemented as RESTful clients. The authorization API provides methods to *insert*, *remove*, *update*, *access* and *search* information in the knowledge base.

Our proposed framework uses *Protege* to provide an optimized solution to efficiently manage a large knowledge base with authorization information. We use *Protege-OWL* that is an editor to enable users to build ontologies for the Semantic Web, in particular in the OWL. For SWRL rules, our implementation makes use of SWRLTab which is a development environment for working with SWRL rules in Protege-OWL. It supports the editing and execution of SWRL rules.

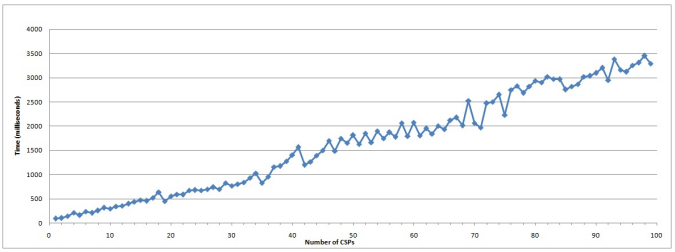

Figure 3: Global Ontology Construction Process

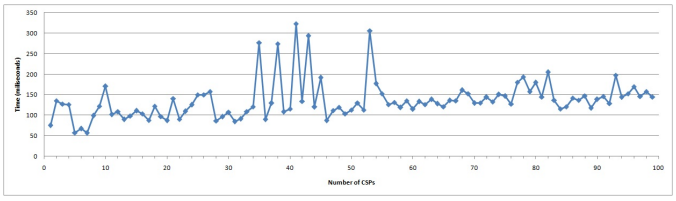

Figure 4: OWL Ontology Update

4. PERFORMANCE EVALUATION

In order to enable users to specify their access policy rules, the SBPMS first needs to provide them with information about all their resources. In this process, that is done only once at the beginning of the deployment of the framework, the SBPMS connects to knowledge bases of all the CSPs and retrieves information about all the resources the user has stored at the CSPs. This is essentially a construction of the global ontology in which the SBPMS gather information from OWL files of all the CSPs and construct the global ontology to be used for policy specification by the user. Figure 3 shows the performance of this process. As we can see, for the first CSP it takes less than 100 milliseconds to fetch required information to the server. For 10 CSPs, it takes

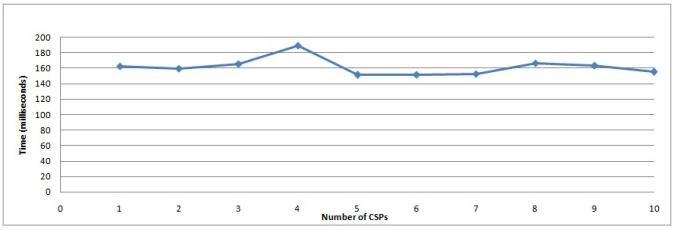

Figure 5: SWRL Rules Update

about 300 milliseconds and for 40 CSPs the time it takes to fetch the ontologies is 1 second. For 100 CSPs, it takes less than 3.5 seconds to fetch all the ontologies stored at all the CSPs to the server; This may seem hight but considering the large number of CSPs and size of the OWL files, it is an acceptable and reasonable performance.

Next, we looked at performance of the authorization API and the provider authorization API. Whenever there is a change in one of the CSPs, we need to update the OWL file in the SBPMS side to reflect these changes. The results of this experiment are shown in Figure 4. As can be seen, when there is only one CSP it takes only 75 milliseconds to update the ontology base of the SBPMS server. If we increase the number of CSPs to 40, it takes 150 milliseconds to update the server. With 100 CSPs, the time it takes to update the server when there is a change in the CSPs' side is always less than 300 milliseconds.

Similar to OWL files, when there is a change in SWRL rules at the SBPMS side, these changes need to be reflected at the CSPs side too. The SBPMS should update SWRL rules files of the CSPs that are related to the change. The results of this experiment are shown in Figure 5. As it can be seen, for 10 CSPs it takes less than 200 milliseconds to update the policy base of the CSPs when there is a change in the policy base of the SBPMS server and the number of the CSPs does not have much effect on performance of this process.

Note that for all the experiments, we ran each of them five rounds and the results reported here are average of those five rounds of experiments. Overall, the results show that our proposed framework performs well and the time required for policy management is reasonable.