Design of Authorization-Pull Service for Community Authorization Service

SANDEEP KUMAR SINGH and ANNAPPA
Department of Computer Engineering
National Institute of Technology Karnataka
Surathkal, Karnataka-575025
INDIA

Abstract: - This paper proposes an Authorization-Pull support for Community Authorization Services (CAS), an authorization-push model for the Grid authorization by the Globus Alliance, to evaluate it in the role of a pull model. The proposed system tries to evaluate the advantages and use of an authorization-pull model in the grid scenario making use of CAS and compares the same with the push-model originally proposed. The paper gives a design view of the proposed authorization system, and also discusses the security considerations and performance of the system proposed.

Key-Words: - authorization; Community Authorization Service; authorization-pull model; grid computing

1 Introduction

Community Authorization Service [1] [2] [3] was proposed as a solution to the grid authorization problem. CAS utilized the fact that the grids are composed of distributed communities of resource providers and resource consumers to use a community-based approach to the authorization of these resources and services. This approach allows resource providers to delegate some of the authority for maintaining fine-grained access control policies to communities, while still maintaining ultimate control over their resources.

The community based approach of CAS provides scalability [1] by making the cost of administering a resource to be proportional to the number of Virtual Organizations (VOs), and not their size or dynamics. The community-based approach also provides flexibility and experssiblity [1], and simplifies the administering of the authorization policies in a VO. The CAS provides a policy hierarchy [1], by allowing various institutions within a VO to define their own institutional policies. Thus, the community-based approach allows resource owners to grant access to blocks of resources to a community as a whole, and lets the community itself manage fine-grained access control within that framework.

This makes CAS an interesting and viable authorization system for grids, and hence our choice. CAS was designed to be an authorization-push model [4] to make it better suit for the grid application scenario. Therefore, it differs from a pull-model [4] in that the examination of policy and granting of rights is done before the gatekeeper is contacted. This means the user must ask for all the rights she will need in advance of referencing the resource. Also, the user must determine in advance all the resources and services she will utilize for her application. The solution for this in CAS was to acquire the complete set of rights the user has in advance and push them to the services and resources the application needs to use.

The proposed Authorization-Pull system for CAS is used to gather and check the policy after the call to the gatekeeper to perform a certain action. This alleviates the need to collect all the rights for all the resources and services a user has, in advance. This is done by the services/resources, the user intends to use, as and when the user application contacts the same for performing certain action(s). The service can check the rights of the user for the intended actions with the CAS server of the user's VO. This model requires less knowledge and interaction at the user end, and hence the security can become more transparent, alleviating the need for the user to acquire her rights from the CAS server of her VO on her own. The proposed system also gives the owners and other stake-holders of the resources more control over their resources and services.

2 Authorization-Pull Support for CAS

The Authorization-Pull support for CAS has been designed to look into the possibility of using CAS

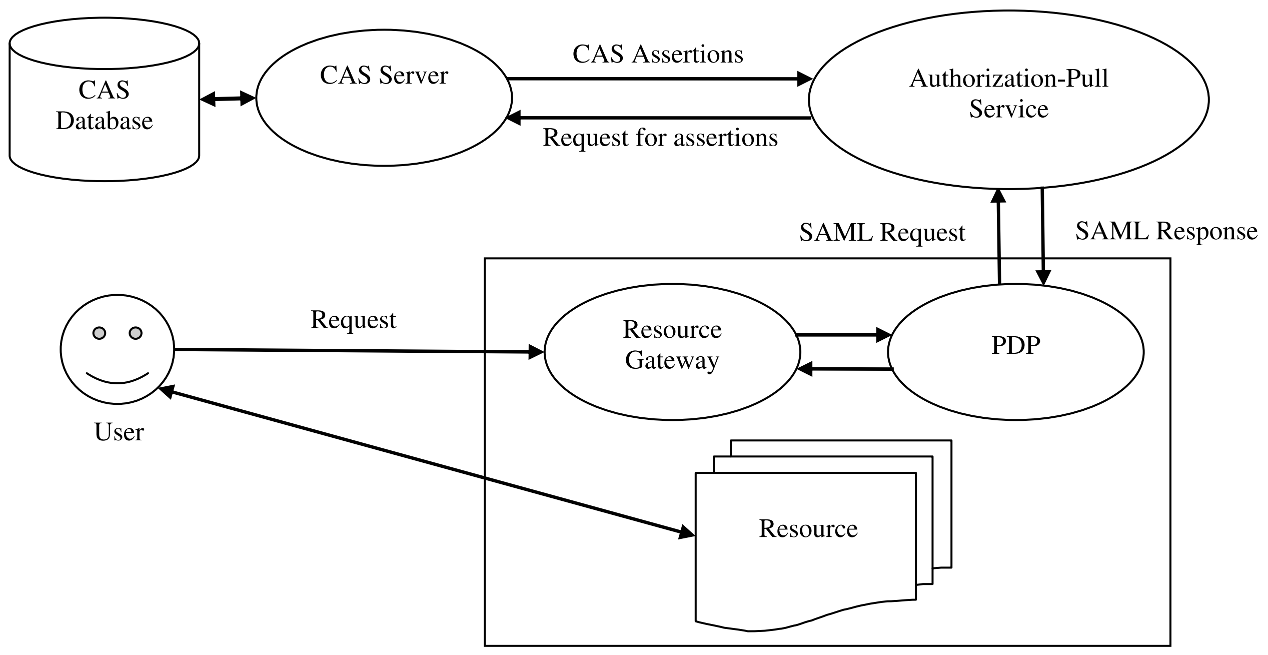

Fig. 1: A Grid authorization scenario with the proposed Authorization-Pull Service.

through a pull model to provide it with some of the added advantages of the authorization-pull models [4] (like Akenti [5], PERMIS [6], etc) and evaluate its performance and benefits over the push-model [4].

This Authorization-Pull support has been designed as a Grid Service [7], which will provide a resource or service with the authorization service using the CAS assertions. A typical grid authorization scenario with the proposed Authorization-Pull Service in place is shown in Fig. 1. The resource or service needs to contact this service to get the CAS assertions of a user, which has called the gatekeeper of the resource or service to perform certain action(s) and has been authenticated by the gatekeeper. The Authorization-Pull service caches the rights that the user was granted, to deal with the common case of several calls in rapid succession for resources in the same realm.

The proposed system is designed as a separate service instead of integrating it with the CAS service itself, in order to provide the calling resource or service with the user’s assertions from one or more CAS servers, if the need be. Also, this service, with little modifications, is being envisioned to provide the assertions from various other authorization services, which can be the likes of CAS, Akenti [5], PERMIS [6], or any other authorization system. This may be needed for resources or services, which has several stake-holders each with a different set of policies defined through different authorization mechanisms. In such cases, the resource or service needs to permit a user to perform the desired action(s) only after ascertaining that the user is permitted to do so as per the policies of all the stake-holders of the resource or service. The proposed system can then be used to acquire the assertions of the user from all such authorities given their URIs. This can be implemented by standardizing the authorization request and response messages, making use of Grid Services [7] and XML-based standards, like SAML [8], XACML [9], etc.

This service will alleviate the need for user to collect the complete set of the user's permissions in advance, as the required user assertions and rights are now being pulled by the Authorization-Pull service. Therefore, the user is allayed of botheration of identifying in advance the services and/or resources her application will or may use, in order for her to get her rights on them. The service, thus, makes the grid authorization system transparent to the user. Also, the proposed service will give the resource owners and other stake-holders greater control over their resources and services.

3 Working of Authorization-Pull Service

When a user submits a request to the gatekeeper of the service/resource to perform a certain action on a resource, the gatekeeper authenticates the user through the credentials of the user submitted along with the request. If the user is authenticated, the gatekeeper (also, the PEP [4] of the resource), then, contacts the PDP [4] of the resource to authorize the requested user action, which in turn submits a SAML Request [8] for the requested action to the Authorization-Pull service.

The Authorization-Pull service then extracts the CAS server URI(s) from the Evidence element(s) [10] of the SAML Request, and requests this (these) CAS server(s) for the required assertions. The Authorization-Pull service caches the assertions returned from the CAS server, to deal with the common case of several calls in rapid succession for resources in the same realm. It then constructs a SAML Response [8] using the assertions returned by the CAS, and returns this response to the requesting PDP. The PDP makes a decision based on the SAML Response returned to it, and sends the same to the gatekeeper, which when allows or denies the requested action as per the decision returned from PDP.

4 Security Considerations

The proposed system uses CAS assertions to authorize a user action; hence it take care of various security considerations, such as the Restricted Proxy Certificates, Compromised CAS server, Revocation Mechanism and Compromised Resource Server as discussed in [1]. The Authorization-Pull service is GT4 GSI [11] [12] compliant and any service which needs its services needs to have the same. The Authorization-Pull service needs the required permissions to acquire the CAS assertions of a user, which must be delegated to it by the VOs of the user as well as the service/resource.

5 Implementation and Future Work

The proposed Authorization-Pull service is been implemented as a Grid Service [7] to provide with the PDPs of the resources and/or services requesting for the CAS assertions of a user from the CAS service URL provided by the PDP. This is being implemented in Java on Grid test-bed consisting of three Linux (Fedora Core 2) workstations. The Grid test-bed is created using the Globus Toolkit 3.9.4 (development release). So far we have been able to retrieve the desired CAS assertions of a user requesting to perform a certain action. We are yet to complete the implementation and compare the same with the push-model [4] of CAS for performance, security and other related issues.

A system based on the CAS server has a performance bottle-neck as well as a single-point of failure. We will look into the ways to alleviate this problem by replicating the CAS server or developing a caching server.

We are also planning, as explained in the earlier sections, to extend the same for other authorization systems, like Akenti [5], PERMIS [6], etc.

6 Related Work

Akenti [5] and PERMIS [6], while having differences in implementation and features, are architecturally similar in that they provide a resource with an authorization decision in regards to a request. Both follow the basic authorization pull model [4].

The Akenti [5] system comprises the compliance checker, called the Akenti server, which can be called either via a function call in the gateway or as a standalone server via TCP/IP. The Akenti system identifies a set of stakeholders with a resource, where each stakeholder is allowed to place restrictions on who and how the resource can be used. These restrictions are specified in terms of what attributes a user must possess in order to perform specific requests. If all stakeholders approve a request, then the request may be performed.

PERMIS [6] operates in multi-step decision making mode. Firstly, the user's assertions are obtained and validated, and the roles that conform to the policy are passed back to the calling application for caching. Then the requested action and target are passed, along with the user's validated roles, and a simple Boolean decision is returned, either granting or denying access. The last step can be repeated as often and as many times as required for different targets and different actions, as the user attempts to perform different tasks.

7 Conclusion

The proposed authorization service is expected to add the advantages of an authorization-pull model to the existing CAS service. This will, thus, alleviate the need for user to collect the complete set of the user's permissions in advance. Therefore, the user is allayed of botheration of identifying in advance the services and/or resources her application will or may use. The service, thus, makes the authorization system transparent to the user. Also, the proposed service will give the resource owners and other stake-holders more control over their resources and services.

References:

[1] L. Pearlman, V. Welch, I. Foster, C. Kesselman, and S. Tuecke. "A Community Authorization Service for Group Collaboration" *IEEE 3rd International Workshop on Policies for Distributed Systems and Networks*, 2002.

[2] Pearlman, L., Welch, V., Foster, I., Kesselman, C. and Tuecke, S., "The Community Authorization Service: Status and Future". *CHEP03*, 2003.

[3] V. Welch, F. Siebenlist, I. Foster, J. Bresnahan, K. Czajkowski, J. Gawor, C. Kesselman, and S. Meder. "Security for Grid Services." Twelfth International Symposium on High Performance Distributed Computing (HPDC-12*), IEEE Press*, June 2003.

[4] RFC2904, Vollbrecht J., Calhoun P., Farrell S., Gommans L., Gross G., de Bruijn B., de Laat C., Holdrege M., Spence D., "AAA Authorization Framework", August 2000.

[5] M. Thompson, W. Johnston, S. Mudumbai, G. Hoo, K. Jackson, and A. Essiari. "Certificate-based Access Control for Widely Distributed Resources," *In Proceedings of the 8th USENIX Security Symposium*, Washington, D.C., August 1999.

[6] D. W. Chadwick and A. Otenko. "The PERMIS X.509 Role Based Privilege Management Infrastructure". *7th ACM Symposium on Access Control Models and Technologies*, 2002.

[7] T. Sandholm and J. Gawor. "*Globus Toolkit 3 Core — A Grid Service Container Framework*". Technical report, 2003. http://www-unix.globus.org/toolkit/3.0/ogsa/docs/

[8] "*Security Assertion Markup Language (SAML) 1.1 Specification*", OASIS, May 2004. http://www.oasisopen.org/committees/security/

[9] "*eXtensible Access Control Markup Language (XACML) 1.0 Specification*", OASIS, February 2003. http://www.oasisopen.org/committees/xacml/

[10] Von Welch et. al. "*Use of SAML for OGSA Authorization*" December 14, 2004. https://forge.gridforum.org/projects/ogsa-authz

[11] I. Foster, C. Kesselman, G. Tsudik, and S. Tuecke. "A Security Architecture for Computational Grids". *ACM Conference on Computers and Security*, 1998, pp. 83-91.

[12] Von Welch et. al. "Globus Toolkit Version 4 Grid Security Infrastructure: A Standards Perspective" Version 2, December 2004 http://www-unix.globus.org/toolkit/3.0/ogsa/docs/