Building a Modular Authorization Infrastructure

David Chadwick, Gansen Zhao, Sassa Otenko, Romain Laborde, Linying Su, Tuan Anh Nguyen

University of Kent

Abstract

Authorization infrastructures manage privileges and render access control decisions, allowing applications to adjust their behavior according to the privileges allocated to users. This paper describes the PERMIS role based authorization infrastructure along with its conceptual authorisation, access control, and trust models. PERMIS has the novel concept of a credential validation service, which verifies a user's credentials prior to access control decision making and enables the distributed management of credentials. Details of the design and the implementation of PERMIS are presented along with details of its integration with Globus Toolkit, Shibboleth and GridShib. A comparison of PERMIS with other authorization and access control implementations is given, along with our plans for the future.

1. Introduction

Authorization infrastructures provide facilities to manage privileges, render access control decisions, and process the related information. Normally, an authorization infrastructure will follow a certain set of authorization policies to make decisions, such as Credential Issuing Policies, Access Control Policies, Delegation Policies, and Credential Validation Policies. These policies contain the rules and criteria that specify how privileges (or credentials) are managed and access control decisions are made. Following these policies, an authorization infrastructure issues credentials to users, who might belong to one domain, whilst the credentials are then presented and validated by the authorization infrastructure of the resource which might belong to a different domain. Once the credentials are validated, the authorization infrastructure then renders an access control decision, and returns this to the application for enforcement. When enforcing the access control decisions, the application should only grant those requests that are authorized by the authorization infrastructure, and should forbid all others.

The authorization infrastructure that we have built is called PERMIS [1]. This paper describes the various components of the PERMIS authorization infrastructure, the conceptual models that are behind them, and the standards that we have used. We also describe some of our plans for future enhancements and work that still needs to be done. We also compare our work to that of others. The rest of this paper is structured as follows. Section 2 provides the conceptual models of our authorization infrastructure. Section 3 describes the design and implementation of PERMIS. Section 4 presents PERMIS's integration with Globus Toolkit, Shibboleth and GridShib. Section 5 compares PERMIS to other related research. Section 6 concludes and indicates our plans for the future.

2. Conceptual Models

2.1 The Authorisation Model

The authorization model paradigm adopted is the "Subject – Action – Target" paradigm and the Role Based Access Control model [18], where roles are presented as credentials issued to the subjects. In our model, a role is not restricted to an organizational role, but can be any attribute of the subject, such as a professional qualification or their current level of authentication [23]. Each subject represents a real world principal, which is the action performer. Action is the operation that is requested to be performed on the target. It can be either a simple operation, or a bundle of complex operations that is provided as an integrated set. Target is the object of the action, over which the action is to be performed. A target represents one or more critical resources that need to be protected from unauthorized access.

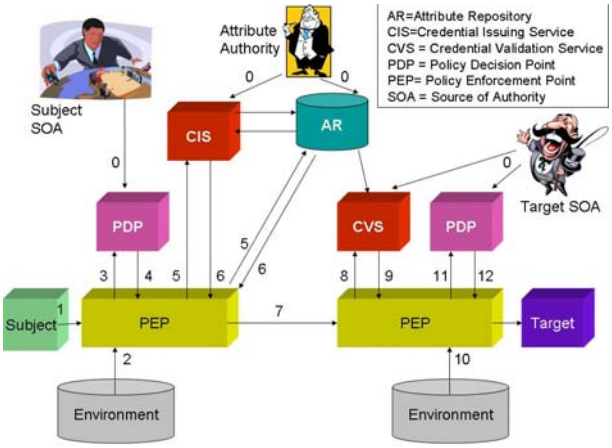

Figure 1: High Level Conceptual Model of an Authorisation Infrastructure

Figure 1 shows our high level conceptual model for an authorization infrastructure. Step 0 is the initialization step for the infrastructure, when the policies are created and stored in the various components. Each subject may possess a set of credentials from many different Attribute Authorities (AAs), that may be pre-issued, long lived and stored in a repository or short lived and issued on demand, according to their Credential Issuing Policies. The Subject Source of Authority (SOA) dictates which of these credentials can leave the subject domain for each target domain. When a subject issues an application request (step 1), the policy decision point (PDP) informs the application's policy enforcement point (PEP) which credentials to include with the user's request (steps 3-4). These are then collected from the Credential Issuing Service (CIS) or Attribute Repository by the PEP (steps 5-6). The user's request is transferred to the target site (step 7) where the target SOA has already initialized the Credential Validation Policy that says which credentials from which issuing AAs are trusted by the target site, and the Access Control policy that says which privileges are given to which attributes. The user's credentials are first validated (steps 8-9) and then the validated attributes, along with any environmental information, such as current date and time (step 10), are passed to the PDP for an access control decision (steps 11-12). If the decision is granted the user's request is allowed by the PEP, otherwise it is rejected. In more sophisticated systems there may be a chain of PDPs called by the PEP, in which case each PDP may return granted, denied or don't know; the latter response allowing the PEP to call the next PDP in the chain.

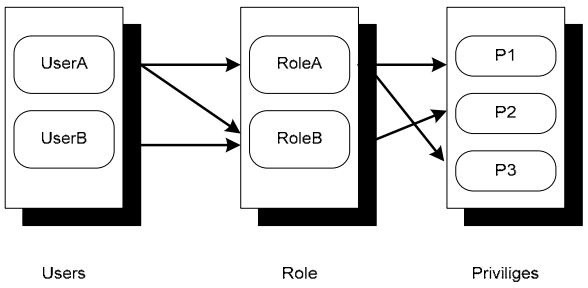

Figure 2 : Example User-Role-Privilege Assignments

PERMIS uses the Role Based Access Control Model [18]. Roles are used to model organization roles, user groups, or any attribute of the user. Subjects are assigned attributes, or role memberships. A subject can be the member of zero, one or multiple roles at the same time. Conversely, a role can have zero, one or more subject occupants at the same time.

Privileges are directly allocated to roles or assigned attributes. Thus each role (or assigned attribute) is associated with a set of privileges, representing the authorised rights the role/attribute has been given by the system administrator. These are rights to perform operations on target objects. Thus a subject is authorised to perform the operations corresponding to his role memberships (or attribute assignments). Changing the privileges allocated to a role/attribute will affect all subjects who are members of the role or who have the assigned attribute.

Figure 2 shows that UserA is a member of both RoleA and RoleB, and UserB is a member of RoleB. RoleA has been granted privileges P1 and P3, whilst RoleB has been granted privilege P2. With Role Based Access Controls this means that UserA is granted privileges P1, P2 and P3 whilst UserB only has privilege P2.

PERMIS supports hierarchical RBAC in which roles (or attributes) are organized in a hierarchy, with some being superior to others. A superior role inherits all the privileges allocated to its subordinate roles. For example, if the role Staff is subordinate to Manager, the Manager role will inherit the privileges allocated to the Staff role. A member of the Manager role can perform operations explicitly authorized to Managers as well as operations authorised to Staff. The inheritance of privileges from subordinate roles is recursive, thus a role r_o will inherit privileges from all its direct subordinate roles r_s, and indirect subordinate roles which are direct or indirect subordinate roles of r_s.

2.2 The Trust Model

Credentials are the format used to securely transfer a subject's attributes/roles from the Attribute Authority to the recipient. They are also known as attribute assertions [20]. PERMIS only trusts valid credentials. A valid credential is one that has been issued by a trusted AA or his delegate in accordance with the current authorization policies (Issuing, Validation and Delegation policies).

It is important to recognize the difference between an authentic credential and a valid credential. An authentic credential is one that has been received exactly as it was originally issued by the AA. It has not been tampered with or modified. Its digital signature, if present, is intact and validates as trustworthy by the underlying PKI, meaning that the AA's signing key has not been compromised, i.e. his public key (certificate) is still valid. A valid credential on the other hand is an authentic credential that has been issued according to the prevailing authorization policies. In order to clarify the difference, an example is the paper money issued by the makers of the game Monopoly. This money is authentic, since it has been issued by the makers of Monopoly. The money is also valid for buying houses on Mayfair in the game of Monopoly. However, the money is not valid if taken to the local supermarket because their policy does not recognize the makers of Monopoly as a trusted AA for issuing money.

Recognition of trusted AAs is part of PERMIS's Credential Validation Policy. PERMIS checks that the AA is mentioned in this policy directly, or that the credential issuer has been delegated a privilege by a trusted issuer, recursively (i.e. a recursive chain of trusted issuers is established controlled by the Delegation Policies of the Target SOA and the intermediate AAs in the chain). The PERMIS Credential Validation Policy contains rules that govern which attributes different AAs are trusted to issue, along with a Delegation Policy for each AA. These rules

separate AAs into different groups and assign them different rights to issue different attributes to different sets of subjects. Further each AA will have its own Credential Issuing Policy and Delegation Policy. PERMIS assumes that if a credential has been issued and signed by an AA, then it must be conformant to the AA's Issuing Policy, so this need not be checked any further. However, if the credential was subsequently delegated this may or may not have conformed to the original AA's Delegation Policy. Therefore when PERMIS validates a credential it checks that it conforms to the AA's delegation policy as well as the Target SOA's delegation policy. PERMIS also makes sure that all credentials conform to the delegation paradigm that an issuer cannot delegate more privileges than he has himself, to ensure constrained propagation of privileges from issuers to subjects.

The net result of this trust model is that PERMIS can support multiple AAs issuing different sets of attributes to different groups of users, in which each AA can have different delegation policies, yet the target SOA can specify an overall Credential Validation Policy that constrains which of these (delegated) credentials are trusted to be used to access the resources under his control.

3. PERMIS: A Modular Authorization Infrastructure

The PERMIS authorization infrastructure is shown in Figure 3. The PERMIS authorisation infrastructure provides facilities for policy management, credential management, credential validation and access control decision making.

3.1 Policy Management

PERMIS Policies are rules and criteria that the decision making process uses to render decisions. It mainly contains two categories of rules, trust related rules (Credential Validation Policy) and privilege related rules (Access Control Policy). Trust related rules specify the system's trust in the distributed Attribute Authorities. Only credentials issued by trusted AAs within their authority will be accepted. Privilege related rules specify the domains of targets, the role hierarchies, the privileges assigned to each role and the conditions under which these privileges may be used, for example, the times of day or the maximum amount of a resource that may be requested.

PERMIS provides a policy composing tool, the Policy Editor [13], which users can use to compose and edit PERMIS policies. The GUI interface of the Policy Editor comprises: the subject (user) policy window, the trusted AA policy window, the role assignment policy window, the role hierarchy policy window, the target resource policy window, the action policy window and the role-privilege policy window. These windows provide forms for users to fill in, then the tool generates the corresponding PERMIS policy. Policies can be saved in pure XML format, or the XML can be embedded in an X.509 Attribute Certificate (AC) [3] and signed with the policy author's private key.

The Policy Editor is capable of retrieving information from LDAP directories, such as subject names, and writing policy ACs back to the author's entry in the LDAP directory. Authors can use the Policy Editor to browse the directories and select existing policies to update them.

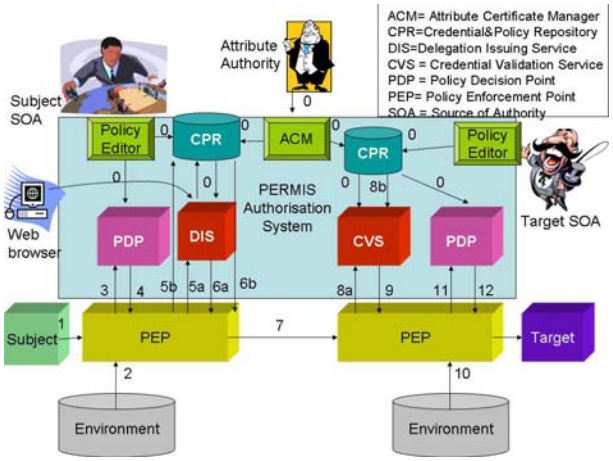

Figure 3: Architecture of the PERMIS Authorization Infrastructure

3.2 Credential Management

The Credential Management system is responsible for issuing and revoking subject credentials. The Attribute Certificate Manager (ACM) tool is used by administrators to allocate attributes to users in the form of X.509 ACs. These bind the issued attributes with the subject's and issuer's identities in a tamper-proof manner. The ACM has a GUI interface that guides the manager through the process of AC creation, modification and revocation. The manager can search for a user in an attached LDAP directory, or enter the DN of the user directly. There is then a picking list of attribute types (e.g. role, affiliation etc.), to which the manager can add his own value (e.g. project manager, University of Kent). There is a pop up calendar allowing the manager to select the dates between which the AC is valid, plus the option of adding appropriate times of day to these. Finally the manager can add a few standard selected extensions to the AC, to say whether the holder is allowed to further delegate or not, and if so, how long the delegation chain can be ("basic attribute constraints" extension [3]), or if the holder may assert the attributes or only delegate them to others ("no assertion" extension [4]). Finally, the manager must add his digital signature to the AC, so the GUI prompts him for the PKCS#12 file holding his private key and his password to unlock it. Once the AC is signed, the manager has the option of storing it in an LDAP directory or local filestore. Besides creating ACs, the ACM allows the manager to edit existing ACs and to revoke existing ACs by deleting them from their storage location. Note that at present revocation lists have not been implemented, because short validity times or deletion from storage have been sufficient to satisfy our current user requirements.

The Delegation Issuing Service is a web service that dynamically issues X.509 ACs on demand. It may be

called directly by an application's PEP after a user has invoked the application, to issue short lived ACs for the duration of the user's task. Alternatively there is a http interface that lets users invoke it via their web browers. This enables users to dynamically delegate their existing longer lived credentials to other users, to enable them to act on their behalf. This is especially powerful, as it empowers users to delegate (a subset of) their privileges to others without any administrative involvement. Because the DIS is controlled by its own PERMIS policy, written by the Subject SOA, an organization can tightly control who is allowed to delegate what to whom, and then leave its subjects to delegate as they see fit. More details of the DIS can be found in [2].

3.3 Authorization Decision Engine

The PERMIS Authorization Decision Engine is responsible for credential validation and access control decision making. Credential validation is the process that enforces the trust model of PERMIS described in Section 2.2. Access control decision making is the process that implements the Role Based Access Control Model described in Section 2.1. The CVS extracts the subset of valid attributes from the set of available credentials, according to the Target SOA's Credential Validation Policy. The PDP makes access control decisions based on the Target SOA's access control policy and the valid attributes passed from the CVS. The PERMIS authorization decision engine is superior to conventional PDPs since it has the ability to validate credentials and delegation chains, which is not a common capability of conventional PDPs e.g. the XACML PDP [15].

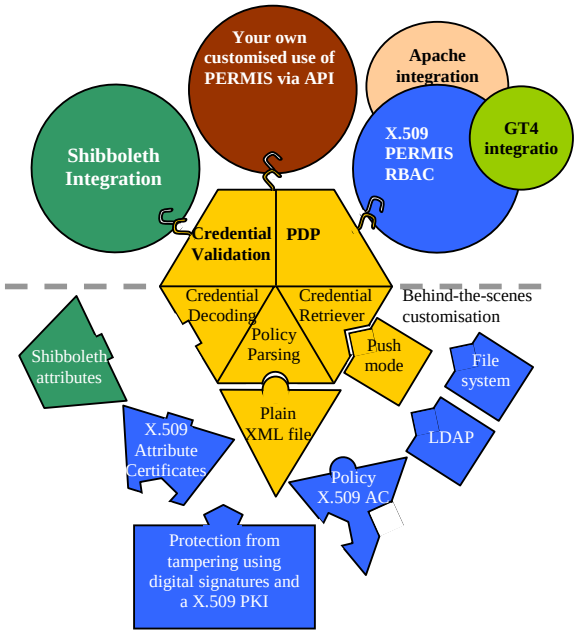

Figure 4: The PERMIS Authorization Decision Engine

As an authorisation infrastructure, PERMIS is not responsible for the actual enforcement of the authorisation decision. This responsibility lies with the application dependent PEP.

Figure 4 depicts the overall architecture of the PERMIS Authorization Decision Engine. It comprises five main components: the PDP, the CVS, the Credential Retriever, the Credential Decoder, and the Policy Parser.

3.4 The PDP

The PDP component is responsible for making access control decisions based on the valid attributes of the user and the Target SOA's access control policy. As stated before, this is based on the Role Based Access Control (RBAC) Model, with support for attribute/role hierarchies.

At initialization time the Target SOA's access control policy is read in and parsed by the Policy Parser so that the PDP is ready to make decisions. Both plain XML policies and digitally signed and protected policies can be read in. The former are stored as files in a local directory whilst the latter are stored as X.509 policy ACs in the LDAP entry of the Target SOA. The latter are tamper resistant and integrity protected, whereas the former have to be protected by the operating system.

Each time the user makes a request to the application to perform a task, the PEP passes this request to the PERMIS PDP along with user's valid attributes and any required environmental attributes such as the time of day. The PEP needs to know which environmental attributes are needed by the access control policy, and since the PEP is software, then it is more likely that the access control policies will be restricted to constraints based on the environmental attributes that the PEP is capable of passing to the PDP.

3.5 The CVS

As described in Section 2.2, all credentials allocated to subjects will be validated by the PERMIS CVS according to the Target SOA's credential validation policy. Figure 5 illustrates the detailed architecture of the CVS, along with the internal data flows.

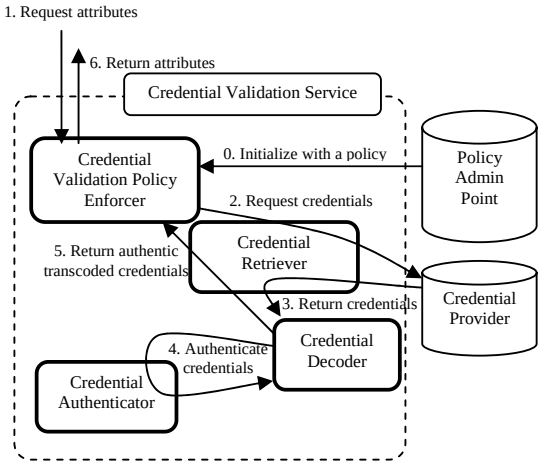

Figure 2 : Data Flow Diagram for Credential Validation Service Architecture

In Figure 5 we can see the general flow of information and sequence of events. First of all the service is initialised by giving it the credential validation policy. The policy parsing module is responsible for this (see Section 3.4). When the user activates the application, the target PEP requests the valid attributes of the subject (step 1). Between the request for attributes and returning them (in step 6) the following events may occur a number of times, as necessary i.e. the CVS is capable of recursively calling itself as it determines the path in a delegation tree from a given credential to a trusted AA specified in the policy.

The Credential Validation Policy Enforcer requests credentials from the Credential Retriever (step 2). PERMIS can operate in either credential pull mode or credential push mode. In credential push mode the application passes the user's credentials along with his request to the target PEP (Step 7 in Figure 3) and the PEP passes them to the CVS (Step 8a in Figure 3). In credential pull mode, the credentials are dynamically pulled from one or more remote credential providers (these could be AA servers, LDAP repositories etc.) by the CVS (step 8b in Figure 3). The actual attribute request protocol (e.g. SAML or LDAP) is handled by the Credential Retriever module. When operating in credential push mode, the PEP stores the already obtained credentials in a local Credential Provider repository and pushes the repository to the CVS, so that the CVS can operate in logically the same way for both push and pull modes. After credential retrieval, the Credential Retriever module passes the credentials to the Credential Decoding module (step 3). From here they undergo the first stage of validation – credential authentication (step 4). Because only the Credential Decoder is aware of the actual format of the credentials, it has to be responsible for authenticating the credentials using an appropriate Credential Authenticator module. Consequently, both the Credential Decoder and Credential Authenticator modules are encoding specific modules. For example, if the credentials are digitally signed X.509 ACs, the Credential Authenticator uses the configured X.509 PKI to validate the signatures. If the credentials are XML signed SAML attribute assertions, then the Credential Authenticator uses the public key in the SAML assertion to validate the signature. The Credential Decoder subsequently discards all unauthentic credentials – these are ones whose digital signatures are invalid. Authentic credentials are decoded and transformed into an implementation specific local format that the Policy Enforcer is able to handle (step 5).

The task of the Policy Enforcer is to decide if each authentic credential is valid (i.e. trusted) or not. It does this by referring to its Credential Validation Policy to see if the credential has been issued by a trusted AA or not. If it has, it is valid. If it has not, the Policy Enforcer has to work its way up the delegation tree from the current credential to its issuer and from there to its issuer, recursively, until a trusted AA is located, or no further issuers can be found (in which case the credential is not trusted and is discarded). Consequently steps 2-5 are recursively repeated until closure is reached (which, in the case of a loop in the credential chain, will be if the same credential is encountered again). Remember that in the general case there are multiple trusted credential issuers, who each may have their own Delegation Policies, which must be enforced by the Policy Enforcer in the same way that it enforces the Target SOA's Delegation Policy.

The CVS can be customized by PERMIS implementers, by either enabling or disabling the credential services built-in with the PERMIS Authorisation Decision Engine, or by implementing their own credential decoding services and plugging them into PERMIS. The latter enables implementers to adopt credential formats that are not implemented by PERMIS, such as local proprietary formats. PERMIS can theoretically be customized to support most application specific credential validation requirements.

4. Integrating PERMIS

4.1 Integration with GT4

Globus Toolkit (GT) is an implementation of Grid software, which has a number of tools that make development and deployment of Grid Services easier [9]. One of the key features of this toolkit is secure communications. However, Globus Toolkit has limited authorisation capabilities based on simple access control lists. To improve its authorization capabilities a Security Assertions Markup Language (SAML) authorization callout has been added. SAML [20] is a standard designed by the Organization for the Advancement of Structured Information Standards (OASIS) to provide a universal mechanism for conveying security related information between the various parts of an access control system. The Global Grid Forum has produced a profile of SAML for use in Grid authorisation [19]. The important consequence of this is that it is now possible to deploy an authorisation service that GT will contact to make authorisation decisions about what methods can be executed by a given subject. A PERMIS Authorisation Service has been developed to provide authorisation decisions for the Globus Toolkit through the SAML callout [8]

4.2 Integration with Shibboleth

Shibboleth [21] is a cross-institutional authentication and authorisation architecture for single sign on and access control of web resources. Shibboleth defines a protocol for carrying authentication information and user attributes from the user's home site to the resource site. The resource site can then use the user attributes to make access control decision about the user's request. A user only needs to be authenticated once by the home site in order to visit other Shibboleth protected resource sites in the federation, as the resulting authentication token is recognized by any member of the federation. In addition to this, protection of the user's privacy can be achieved, since the user is able to restrict what attributes will be released to the resource providers from his/her home site. However Shibboleth's built in access control decision making based on the user's attributes, is simplistic in its functionality, and the

management of the access controls is performed together with web server administration at the resource site. Furthermore, distributed management of credentials and dynamic delegation of authority are not supported. To rectify these deficiencies, a Shibboleth-Apache Authorisation Module (SAAM) has been developed which integrates PERMIS with Shibboleth. SAAM plugs into Apache and replaces the Shibboleth authorization functionality with calls to the PERMIS authorization decision engine. A full description is provided in [5]

PERMIS extends the access control model used in Shibboleth by introducing hierarchies of roles, distributed management of attributes, and policy controlled decisions based on dynamically evaluated conditions. PERMIS supports the existing semantics of Shibboleth attributes, but also allows X.509 ACs to be used instead, where more secure credentials are needed.

4.3 Integration with GridShib

GridShib [10] provides interoperability between Globus Toolkit [9] and Shibboleth [21]. The GridShib Policy Information Point (PIP) retrieves a user's attributes from the Shibboleth Identity Provider (IdP). These attributes are parsed and passed to the GT4 PEP. The GT4 PEP then feeds these attributes to the corresponding PDP to request an authorisation decision. GridShib integrates Shibboleth's attribute management functionality with GT4's authorisation decision making for Grid jobs. However, like GT4, GridShib provides only limited PDP functionality, which is based on access control lists and is not capable of coping with dynamically changing conditions, which a policy based engine is.

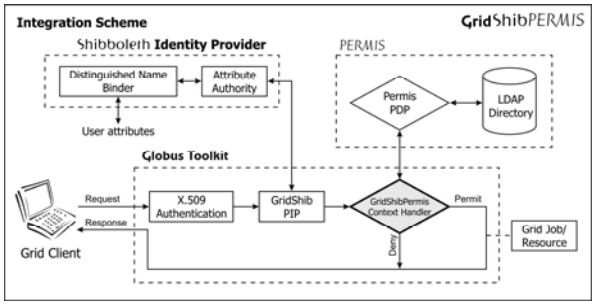

Figure 6: GridShibPERMIS Integration Scheme

Figure 6 shows the GridShibPERMIS integration scheme. GridShibPERMIS provides a GridShibPERMIS Context Handler that can be integrated with GT4 as a callable PDP. The Context Handler is invoked by GT4 when an authorisation decision is to be made. The Context Handler is fed with the user's attributes that have been retrieved from the Shibboleth IdP. They are parsed and stored in a local Credential Provider Repository, ready to be accessed by the PERMIS CVS as described in Section 3.5. The Context Handler calls the PERMIS CVS followed by the PDP, which renders an access control decision based on the Target SOA's policy, and returns it to GT4. In the case of multiple PDPs being configured into GT4, the final authorisation decision is made based on combining all the decisions returned by all the different PDPs. The combining algorithm currently adopted by GT4 is "Deny-Override", which means that the user's request is authorised if and only if no PDP denies the request and at least one PDP grants it.

5. Related Work

Manandhar et at. [12] present an application infrastructure in which a data portal allows users to discover and access data over Grid systems. They propose an authorization framework that allows the data portal to act as a proxy and exercise the user's privileges. When a user authenticates to the data portal, a credential is generated stating that the data portal is authorized to exercise the user's privileges for a specific period. The credential is then used by the data portal to retrieve the user's authorization tokens from various providers. When requesting a service from a service provider, the data portal presents both the credential and the authorization tokens. The authorization decision is then made by the service provider. The proposed infrastructure mainly focuses on the interaction between different systems in the Grid environment, with no in depth discussion about the access control model or the trust model. Credential verification is also missing from the discussion.

XACML [14] defines a standard for expressing access control policies, authorization requests, and authorization responses in XML format. The policy language allows users to define application specific data types, functions, and combining logic algorithms, for the purpose of constructing complex policies. Sun's open source XACML implementation [15] is a java implementation of the XACML 2.0 standard and provides most of the feature in the standard. The XACML policy language is richer than that of PERMIS's PDP policy, but XACML has not yet addressed the issue of credential validation and is only now working on dynamic delegation of authority [22].

The Community Authorisation Service (CAS) [11] was developed by the Globus team to improve upon the manageability of user authorisation. CAS allows a resource owner to grant access to a portion of his/her resource to a VO (or community – hence the name CAS), and then let the community determine who can use this allocation. The resource owner thus partially delegates the allocation of authorisation rights to the community. This is achieved by having a CAS server, which acts as a trusted intermediary between VO users and resources. Users first contact the CAS asking for permission to use a Grid resource. The CAS consults its policy (which specifies who has permission to do what on which resources) and if granted, returns a digitally self-signed capability to the user optionally containing policy details about what the user is allowed to do (as an opaque string). The user then contacts the resource and presents this capability. The resource checks that the capability is signed by a known and trusted CAS and if so maps the CAS's distinguished name into a local user account name via the Gridmap file. Consequently

the Gridmap file now only needs to contain the name of the trusted CAS servers and not all the VO users. This substantially reduces the work of the resource administrator. Further, determining who should be granted capabilities by the CAS server is the task of other managers in the VO community, so this again relieves the burden of resource managers. For finer grained access control, the resource can additionally call a further routine, passing to it the opaque policy string from the capability, and using the returned value to refine the access rights of the user. Unfortunately this part of the CAS implementation (policy definition and evaluation routine) were never fully explored and developed by the Globus team. This is precisely the functionality that PERMIS has addressed.

The main purpose of SPKI [16] is to provide public key infrastructures based on digital certificates without depending upon global naming authorities. SPKI binds local names and authorizations to public keys (or the hash values of public keys). Names are allocated locally by certificate issuers, and are only of meaning to them. SPKI allows authorizations to be bound directly to public keys, removing the process of mapping from authorization to names and then to public keys. SPKI supports dynamic delegation of authorizations between key holders, and allocation of authorizations to groups. Though SPKI can convey authorization information, it does not cover authorization decision making or access control policy issues. One can thus regards SPKI as an alternative format to X.509 ACs or SAML attribute assertions for carrying credentials, and PERMIS could easily be enhanced to support this format of credential if it were required.

The EU DataGrid and DataTAG projects have developed the Virtual Organisation Membership Service (VOMS) [6] as a way of delegating the authorisation of users to managers in the VO. VOMS is a credential push system in which the VOMS server digitally signs a short lived X.509 role AC for the VO user to present to the resource. The AC contains role and group membership details, and the Local Centre Authorisation Service (LCAS) [7] makes its authorisation decision based upon the user's AC and the job specification, which is written in job description language (JDL) format. This design is similar in concept to the CAS, but differs in message format and syntax. However what neither VOMS nor CAS nor LCAS provide is the ability for the resource administrator to set the policy for access to his/her resource and then let the authorisation infrastructure enforce this policy on his/her behalf. This is what systems such as PERMIS and Keynote [17] provide. It would therefore be relatively easy to replace LCAS with the PERMIS decision engine, so that VOMS allocates role ACs and pushes them to the resource site, whilst PERMIS makes the policy controlled authorization decisions.

KeyNote [17] is a trust management system that provides a general-purpose mechanism for defining security policies and credentials, and rendering authorization decisions based on security policies and credentials. KeyNote provides a language for defining both policies and assertions, where policies state the rules for security control, and assertions contain predicates that specify the granted privileges of users. KeyNote has been implemented and released as an open source toolkit. But KeyNote is not without its limitations. Keynotes policies and credentials are in their own proprietary format. KeyNote credentials have no time limit, and Keynote has no concept of revocation of credentials. Further, policies define the roots of trust, but the policies themselves are not signed and therefore have to be stored securely and are only locally trusted.

6. Conclusions

This paper presents our work on building a modular authorization infrastructure, PERMIS. We have explained the conceptual models of PERMIS by describing the authorization model, the access control model, and the trust model of PERMIS. The design and the implementation of PERMIS have also been presented, with details of the architecture and an explanation of the facilities PERMIS provides to support policy management, attribute management, and decision making. Details of the decision making process and the credential validation service are also given, showing how PERMIS implements hierarchical RBAC decision making based on user credentials and various authorization policies.

Finally, we have presented a comparison of related work, pointing out their relative advantages and disadvantages as compared to PERMIS.

6.1 Future Work

In an application, sometimes coordination is needed between access control decisions. For example, in order to support mutually exclusive tasks (Separation of Duties), the PDP needs to know if the same user is trying to perform a second task in a set of mutually exclusive ones. Alternatively, if multiple resources are available but their use is to be restricted, for example a maximum of 30GB of storage throughout a grid, then enforcing this becomes more difficult. The use of a stateful PDP allows coordination between successive access control decisions, whilst passing coordination data between PDPs allows coordination over the use of restricted multiple resources. In order to achieve the latter, the access control policies should state what coordination between decision making is needed, which coordination data is used to facilitate this, and how this coordination data is updated afterwards. We have already implemented Separation of Duties in a prototype stateful PDP and we are currently working on more sophisticated distributed coordination between PDPs.

Obligations are actions that are required to be fulfilled on the enforcement of access control decisions. Existing authorization infrastructures are mainly concerned with access control decision making, but this is not sufficient in scenarios where obligations are needed. XACML policies already support obligations and we are currently incorporating these into PERMIS. We are building an obligation engine that will evaluate the obligations that are embedded in a policy e.g. Add the amount requested to the

amount already consumed, and will return the obligated action to the PEP for enforcement, since it is the PEP that ultimately has to interpret and fulfill the obligations.

Acknowlegements

We would like to thank the UK JISC for funding part of this work under the DyCOM, DyVOSE, SIPS and GridAPI projects, and the EC for funding part of this work under the TrustCoM project (FP6 project number 001945).

References

[1] D.W.Chadwick, A. Otenko "The PERMIS X.509 Role Based Privilege Management Infrastructure". Future Generation Computer Systems, 936 (2002) 1–13, December 2002. Elsevier Science BV.

[2] D.W.Chadwick. "Delegation Issuing Service". NIST 4th Annual PKI Workshop, Gaithersberg, USA, April 19-21 2005

[3] ISO 9594-8/ITU-T Rec. X.509 (2001) "The Directory: Public-key and attribute certificate frameworks"

[4] ISO 9594-8/ITU-T Rec. X.509 (2005) "The Directory: Public-key and attribute certificate frameworks"

[5] Wensheng Xu, David Chadwick, Sassa Otenko. "Development of a Flexible PERMIS Authorisation Module for Shibboleth and Apache Server". Proceedings of 2nd EuroPKI Workshop, University of Kent, July 2005

[6] R. Alfieri et al. "VOMS: an Authorization System for Virtual Organizations", 1st European Across Grids Conference, Santiago de Compostela, February 13-14, 2003

[7] Martijn Steenbakkers "Guide to LCAS v.1.1.16", Sept 2003. Available from http://www.dutchgrid.nl/DataGrid/wp4/lcas/edg-lcas-1.1

[8] David Chadwick, Sassa Otenko, and Von Welch. "Using SAML to Link the GLOBUS Toolkit to the PERMIS Authorisation Infrastructure". In Proceedings of Eighth Annual IFIP TC-6 TC-11 Conference on Communications and Multimedia Security, Windermere, UK, September 2004.

[9] I. Foster. "Globus Toolkit Version 4: Software for Service-Oriented Systems". IFIP International Conference on Network and Parallel Computing, Springer-Verlag LNCS 3779, pp 2-13, 2005.

[10] Barton, T., Basney, J., Freeman, T., Scavo, T., Siebenlist, F., Welch, V., Ananthakrishnan, R., Baker, B., and Keahey, K. "Identity Federation and Attribute-based Authorization through the Globus Toolkit, Shibboleth, Gridshib, and MyProxy", 5th Annual PKI R&D Workshop. April 2006.

[11] Ian Foster, Carl Kesselman, Laura Pearlman, Steven Tuecke, and Von Welch. "The Community Authorization Service: Status and Future". In Proceedings of Computing in High Energy Physics 03 (CHEP '03), 2003.

[12] Ananta Manandhar, Glen Drinkwater, Richard Tyer, Kerstin Kleese. "GRID Authorization Framework for CCLRC Data Portal", Second Earth Science Portal Workshop: Web Portal Framework Design/Implementation, September 2003.

[13] Sacha Brostoff, M. Angela Sasse, David Chadwick, James Cunningham, Uche Mbanaso, Sassa Otenko. "*"R-What?"* Development of a Role-Based Access Control (RBAC) Policy-Writing Tool for e-Scientists" Software: Practice and Experience
Volume 35, Issue 9, Date: 25 July 2005, Pages: 835-856

[14] OASIS. "XACML 2.0 Core: eXtensible Access Control Markup Language (XACML) Version 2.0", Oct, 2005.

[15] Sun's XACML Implementation available on http://sunxacml.sourceforge.net/.

[16] C. Ellison, B. Frantz, B. Lampson, R. Rivest, B. Thomsa, and T. Ylonen. "SPKI Certificate Theory". RFC 2693, September 1999.

[17] M. Blaze, J. Feigenbaum, J. Ioannidis, and A. Keromytis. "The KeyNote Trust Management System Version 2". RFC 2704, Sept. 1999.

[18] David F. Ferraiolo and Ravi Sandhu and Serban Gavrila and D. Richard Kuhn and Ramaswamy Chandramouli. "Proposed NIST standard for role-based access control". ACM Transactions on Information and System Security Volume 4, Issue 3. August 2001.

[19] Von Welch, Rachana Ananthakrishnan, Frank Siebenlist, David Chadwick, Sam Meder, Laura Pearlman. "Use of SAML for OGSI Authorization", Aug 2005, Available from https://forge.gridforum.org/projects/ogsa-authz

[20] OASIS. "Security Assertion Markup Language (SAML) 2.0 Specification", November 2004.

[21] S. Cantor. "Shibboleth Architecture, Protocols and Profiles", Working Draft 02. 22 September 2004, see http://shibboleth.internet2.edu/

[22] XACML v3.0 administration policy Working Draft 05 December 2005. http://www.oasis-open.org/committees/documents.php?wg abbrev=xacml.

[23] N. Zhang, L. Yao, A. Nenadic, J. Chin, C. Goble, A. Rector, D. Chadwick, S. Otenko and Q. Shi; "Achieving Fine-grained Access Control in Virtual Organisations", to appear in Concurrency and Computation: Practice and Experience, published by John Wiley and Sons publisher.