First Experiences Using XACML for Access Control in Distributed Systems

Markus Lorch
Virginia Tech
Dept. of Computer Science, m/c 106
Blacksburg, VA 24061
+1 206 337 0428
mlorch@vt.edu

Seth Proctor
Sun Microsystems Laboratories
1 Network Drive
Burlington, MA 01803
+1 781 442 2090
seth.proctor@sun.com

Rebekah Lepro
NASA Ames Research Center
Mail Stop 258/6
Moffett Field, CA 94035
+1 650 604 4359
rlepro@arc.nasa.gov

Dennis Kafura
Virginia Tech
Dept. of Computer Science, m/c 106
Blacksburg, VA 24061
+1 540 231 5568
kafura@cs.vt.edu

Sumit Shah
Virginia Tech
Dept. of Computer Science, m/c 106
Blacksburg, VA 24061
+1 540 951 4636
sshah@vt.edu

ABSTRACT

Authorization systems today are increasingly complex. They span domains of administration, rely on many different authentication sources, and manage permissions that can be as complex as the system itself. Worse still, while there are many standards that define authentication mechanisms, the standards that address authorization are less well defined and tend to work only within homogeneous systems. This paper presents XACML, a standard access control language, as one component of a distributed and inter-operable authorization framework. Several emerging systems which incorporate XACML are discussed. These discussions illustrate how authorization can be deployed in distributed, decentralized systems. Finally, some new and future topics are presented to show where this work is heading and how it will help connect the general components of an authorization system.

Categories and Subject Descriptors

C.2.0 [**Computer-Communication Networks**]: General – *Security and protection.* D.4.6 [**Operating Systems**]: Security and Protection – *Access controls.* I.7.2 [**Document and Text Processing**]: Document Preparation – *Markup languages.*

General Terms

Design, Security, Human Factors, Standardization, Languages.

ACM Workshop on XML Security, October 31, 2003, Fairfax, VA, USA

Keywords

Distributed system security, authorization, policy language, policy management, access control decision, access control enforcement.

1. INTRODUCTION

In modern systems, security is a critical feature. Beyond providing strong protection, security systems must also be flexible and promote inter-operability between domains of trust. However, flexibility can come at the price of simplicity and manageability, especially in the complex realm of authorization. Thus, the authorization components of a secure system must be able to work together across domains, but must be manageable to maintain their collaborative value.

Authorization determines whether or not a given action, for example reading a file or logging into a server, is allowed. This is typically, though not always, achieved by authenticating a user and then using their locally assigned attributes or rights to make access decisions according to locally defined policies. Unfortunately, most systems use either proprietary policy languages or formats that apply only to a specific application (like traditional file access modes), leading to interoperability problems. As systems evolve from a central to a distributed model, this limited ability to interoperate authorization components creates additional administrative requirements and hinders overall scalability. Further, heterogeneity restricts the development of standard management tools and toolkits that serve common policy needs, leaving developers and administrators without a common solution to use when creating policy-driven systems.

Authorization in a distributed system often requires cooperation among separate and autonomous administrative domains. Maintaining a consistent authorization strategy requires each system to maintain at least some knowledge of its potential collaborators throughout the entire system. Further, any authorization decision that spans two or more authorization domains requires each participant be capable of correctly

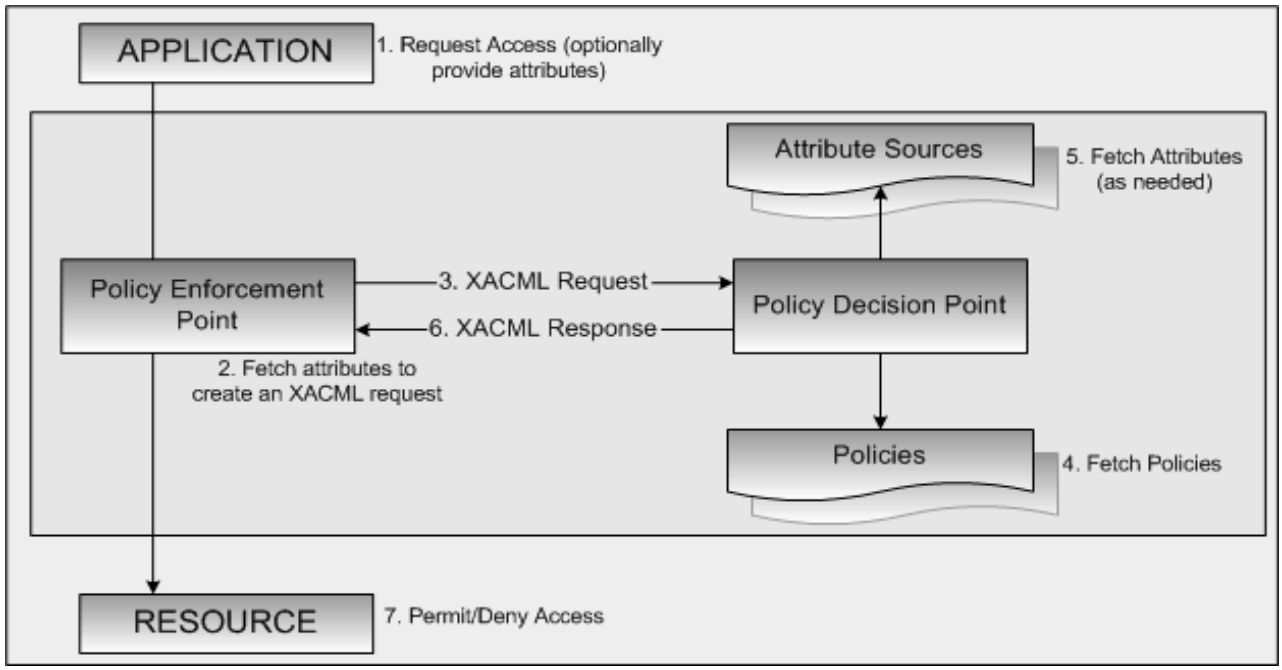

Figure 1. XACML Overview

producing, accepting and interpreting authorization information from a group of potentially heterogeneous peers. This capability requires agreement on protocol, syntax and semantics for each piece of shared authorization data. Additionally, existing enforcement mechanisms typically associate authorization data with identities that are unique to an individual authorization domain. This requires coordination of local identities between the domains, forcing administrative domains to cede partial control of local authorizations to a literal or figurative central authority.

In an attempt to help with these and other problems, OASIS ratified XACML [11] (eXtensible Access Control Markup Language), a standard, general purpose access control policy language defined using XML. XACML was designed to accommodate most system needs, so it may serve as a single interface to policies for multiple applications and environments. In addition to defining a policy language, XACML also specifies a request and response format for authorization decision requests, semantics for determining policy applicability, and a host of advanced features that make it well-suited for tying together large-scale authorization systems. Although XACML does not standardize a complete authorization solution, it provides a foundation upon which cooperative solutions can emerge.

What follows is a brief discussion of XACML. The full details of the language are discussed in [11]. Sufficient explanation of the new standard is presented to support the following sections, which discuss early experiences using XACML in current authorization systems and with existing and emerging protocols. Several systems are presented to illustrate different operating scenarios that can leverage XACML, different means to incorporate XACML into varying authorization approaches and show various ways to leverage XACML. Finally, we present some future directions for XACML and its use in distributed authorization systems.

2. The eXtensible Access Control Markup Language - XACML

XACML is a general purpose policy system, designed to support the needs of most authorization systems. At its core, XACML defines the syntax for a policy language and the semantics for processing those policies. There is also a request and response format to query the policy system, and semantics for determining applicability of policies to requests. The request and response formats represent a standard interface, between a Policy Decision Point (PDP) presents standard behavior when processing policy and a Policy Enforcement Point (PEP) that issues requests and handle responses. A PEP can be embedded in an application-specific system (see Figure 1). This is based on policy framework definitions used in the IETF [30].

XACML policies consist of an arbitrary tree of sub-policies. Each tree represents a target, while the leaves of the tree contain a collection of rules. The target is a simple set of criteria used to determine a policy's applicability to a request, while the rules contain more complex logic that makes XACML extremely expressive, and therefore able to handle myriad policy requirements. A request consists of attributes associated with the requesting subjects, the resource acted upon, the action being performed, and the environment. A response contains one of four decisions: permit, deny, not applicable (no applicable policies or rules could be found), or indeterminate (some error occurred during processing). In the case of an error, optional information is available to explain the error. Responses may also include obligations, which are directives from the applicable policies for the PEP to execute.

The logic within a policy uses an extensible system of datatypes and functions to promote interoperability. All attributes used in XACML are of a well-known type, and all functions have well-known signatures that use these same datatypes. XACML defines

a set of standard datatypes (like string, boolean, integer, time, email address, set, etc.), and a set of standard functions (like equality and comparisons, arithmetic, set comparison, etc.). While these standard datatypes and functions can express many access control policies, XACML also specifies a standard extension mechanism for defining additional datatypes and functions.

In addition to expressing access control logic within a single policy, policies can include references to other policies. In effect, a single policy can consist of any number of decentralized, distributed rules, each managed by different organizational groups. A key supporting language feature is XACML's use of combining algorithms, which define how to take results from multiple rules or policies and derive a single result. As with datatypes and functions, there are a number of standard combining algorithms defined (first applicable, deny overrides, etc.), as well as a standard extension mechanism used to define new algorithms.

Two mechanisms are used to resolve attribute values within policy logic: AttributeDesignators (which reference values by identifier, datatype, and other optional meta-data), and AttributeSelectors (which use XPath expressions to find values). If the needed values aren't found in a request during policy processing, the PDP is free to look elsewhere. This means that XACML can work with existing attribute systems either by including values in a request or by using some custom retrieval module during evaluation.

Policy referencing and retrieval, and attribute value resolution are both specified as arbitrary operations that the PDP is free to perform in any way it sees fit. All policies and attributes, however, are handled in a standard manner once within the PDP. This facilitates inter-operation with legacy systems, and cooperation between different modern attribute and policy management [12] components.

The systems discussed in this paper use the open source XACML implementation [26] originally developed at Sun Microsystems Laboratories. The implementation supports the complete XACML 1.1 specification, handles all the extension points discussed in this section, and includes several optional features of the specification as well. It is implemented in the Java™ Programming Language, and is available at http://sunxacml.sourceforge.net. The PRIMA system (discussed in section 5) also uses the free Jiffy binary distribution implementation of XACML available at http://www.jiffysoftware.com.

3. XACML and Shibboleth

A part of the middleware suite of tools being defined by the Internet2 group, Shibboleth [7] provides a web-based authentication and authorization system. The primary use case is securing interaction between higher education sites, though it is generally useful for any environments that must work across domains of trust. The system will work entirely within the scope of a web browser, so it's easy to setup a resource at one site (for instance, slides for some course), and then let a student at another site access the resource through the web. After the student's browser issues a request for some resource, a series of exchanges between the target site and the user's site verify the user's identity, gather attributes, and perform the access check.

For the authorization step, the target site must determine attributes associated with the subject. As a simple web request contains the initial message, no attribute values are available by default. For security, privacy, and management reasons, an attribute authority at the subject's site maintains all attributes associated with a subject. Thus, the resource site contacts the subject site to request attribute values needed by the policy system. Despite this flexible attribute management system, actual policy decisions are ultimately made using htaccess files in an Apache module. The limitations of htaccess syntax and the difficulties involved with sharing them or storing them in arbitrary locations severely reduces access control system flexibility. These drawbacks also restrict the opportunities to share access control policies among system components.

Current research at Sun Microsystems and Brown University focuses on XACML as a solution to these problems. Specifically, researchers are considering XACML to replace current access control functionality in Shibboleth, though the work applies to other systems as well. In particular, XACML addresses the problems with htaccess files and scales well to a distributed, decentralized environment. In addition to adding basic access control, they are also exploring XACML as a language for defining release policies. This setup provides input into usability and management issues for XACML in general as well as for each of these specific environments.

3.1 Online Access

A basic PEP library, built in C, and an online PDP, implemented using the open source XACML library, support the access control needs of Shibboleth. Incorporating this functionality into Shibboleth Apache modules supports more expressiveness than previously permitted in htaccess syntax. In fact, XACML's policy referencing mechanism allows scenarios such as incorporating policy from a subject's site into the host site policy. This change required no modifications to the majority of Shibboleth's features and there is no difference from a user perspective. Obviously this functionality is generally useful outside of Shibboleth as any application or web server plugin can use this library to talk to an online PDP. Further, the simplicity of the PEP library provides an easy way to add XACML support into older systems. The existing open source project provides a PEP interface in the Java Programming Language, and PEP interfaces in other languages are being developed.

XACML does not specify a protocol for communication between a PEP and a PDP. As is discussed in the next section, SAML [13] is a highly suitable candidate for this protocol. Indeed, the original XACML request and response format came from the SAML specification. Further, the current request and response format from XACML may be included directly in the next version of SAML together with ways to include XACML related data. One of the current projects this framework is being used for is to investigate different exchange protocols, like SAML over SOAP [4] or the Common Open Policy Service [6], to understand what will work best both for Shibboleth and authorization systems in general. Different authorization systems may have different performance or bandwidth requirements, so an online PDP may need to support multiple protocols.

3.2 Release Policy

Another issue that the Shibboleth design raises is the management of attributes and the circumstances under which an attribute authority should release a user's attributes to another site. Currently Shibboleth employs a proprietary system using XML configuration files in which a user defines some simple rules

about when and to whom attributes may be revealed. Other people have explored this same idea in Shibboleth [21] and in other systems [25][29]. Unfortunately, no standards address this problem, nor do good tools for managing these proprietary solutions exist.

To this end a profile of XACML called Web Services Policy Language [20] is being prototyped to provide this and other functionality. The name implies its original goal, which is to provide Web Services the ability to publish policy requirements for communication. Since it can also define release criteria (a similar application), and because it is using a standard language, it is a good choice for replacing proprietary release languages. Again, this addition doesn't typically affect the applications or the user experience. The attribute exchange step requires additional work only if the user wants to add extra levels of protection to their attributes. Early results suggest that XACML and WSPL can be used effectively to protect the privacy of both the user and the authorization system at the other end during the attribute exchange process

3.3 Management

Strong support for policy management is integral to the usefulness of Shibboleth's features. More expressive policies can be very useful, but if they're difficult to write and maintain, they may cancel the benefits of expressiveness. Worse still, while people with some technical knowledge define most access policies, average users will typically define attribute release policies. Thus, release policies must be easy to write and manage or no one will use them. To this end some initial investigation is being done into fundamental, low-level management techniques for XACML, especially in reasoning about policies to provide feedback at a level that most humans can understand.

3.4 Results

Initial investigations have shown that XACML is a good match for Shibboleth. With relative ease a new access control system has been plugged in, and the resulting infrastructure can also be used by other web plugins and stand-alone applications, which helps pull the authorization components together. Additionally, policies can now be shared between applications, regardless of whether they're using Shibboleth, which makes it easier to work across different kinds of authorization systems in the same network. Finally, XACML's ability to work with policies and attributes managed in arbitrary locations greatly supports the distributed nature of Shibboleth . The next steps for this project are to continue investigating protocols and their relative efficiencies, support other languages for the PEP (for instance so Perl modules can use the same features), and continue exploring the usability challenges.

4. Cardea – Combining XACML and SAML to support distributed authorization

Cardea is a distributed authorization system, developed as part of the NASA Information Power Grid [15], which dynamically evaluates authorization requests directly according to a set of relevant characteristics of the resource and requester rather than considering specific local identities that represent those characteristics. Potentially accessed resources are protected by local access control policies, specified with the XACML syntax, in terms of subject and resource characteristics. Further, potential users are modeled only by the characteristics that they can demonstrate. The exact values needed to complete an authorization decision are assessed and collected during the decision process itself. Once assembled, this information is presented to a PDP that returns a final authorization together with any relevant details.

Cardea is currently implemented in the Java Programming Language as a collection of web service portTypes. Much of the communication between components follows the XACML and SAML [13] request and response formats. Although XACML and SAML are transport independent, the initial implementation binds these protocols to SOAP and utilizes the Apache Axis [2] architecture as a SOAP engine. Custom handlers specified for the request and response flows within Axis provide common mechanisms to optionally sign and verify, using the XML Digital Signature [3] specification, the content of each generated or received SOAP message. Cardea interacts with each SOAP message directly via the JAXM or JAX-RPC API. Therefore, no functionality strictly depends on custom Axis functionality

Cardea addresses several specific unmet needs that emerge when authorization spans multiple administrative domains. The system reduces reliance on locally defined identities to define authorizations for each potential user. Therefore, it reduces the system state that must be replicated at each site. Further, it allows separate administrative domains to coordinate local authorization decisions while retaining control over access to its local resources.

The remainder of this section examines the way Cardea combines the power of XACML and SAML to address those needs and identifies distinct gaps that were handled. Then, the ways that XACML were applied within the system architecture are highlighted, and areas that could benefit from additional research and future directions are outlined.

4.1 Assumptions and pre-requisites

Although the system minimizes the amount of negotiation and configuration required to implement distributed authorization, there are several site-specific items that must be defined according to the standard semantics of XACML and SAML. First, local access control policies must be defined using the characteristics of pertinent user-resource combinations. Additionally, authorities must be populated with verified attribute values. Although there is no inherent restriction on how attributes are maintained or represented internally to its authority, each attribute value must be available to a qualified requester as a SAML Assertion.

4.2 The authorization decision process

Cardea evaluates each authorization decision according to a general procedure that requires minimal a priori knowledge of participants. This section illustrates several critical steps in the authorization process (see Figure 2). It specifically highlights communication between distinct system components, how XACML and SAML functionality is leveraged and how the components work together to complete the authorization process.

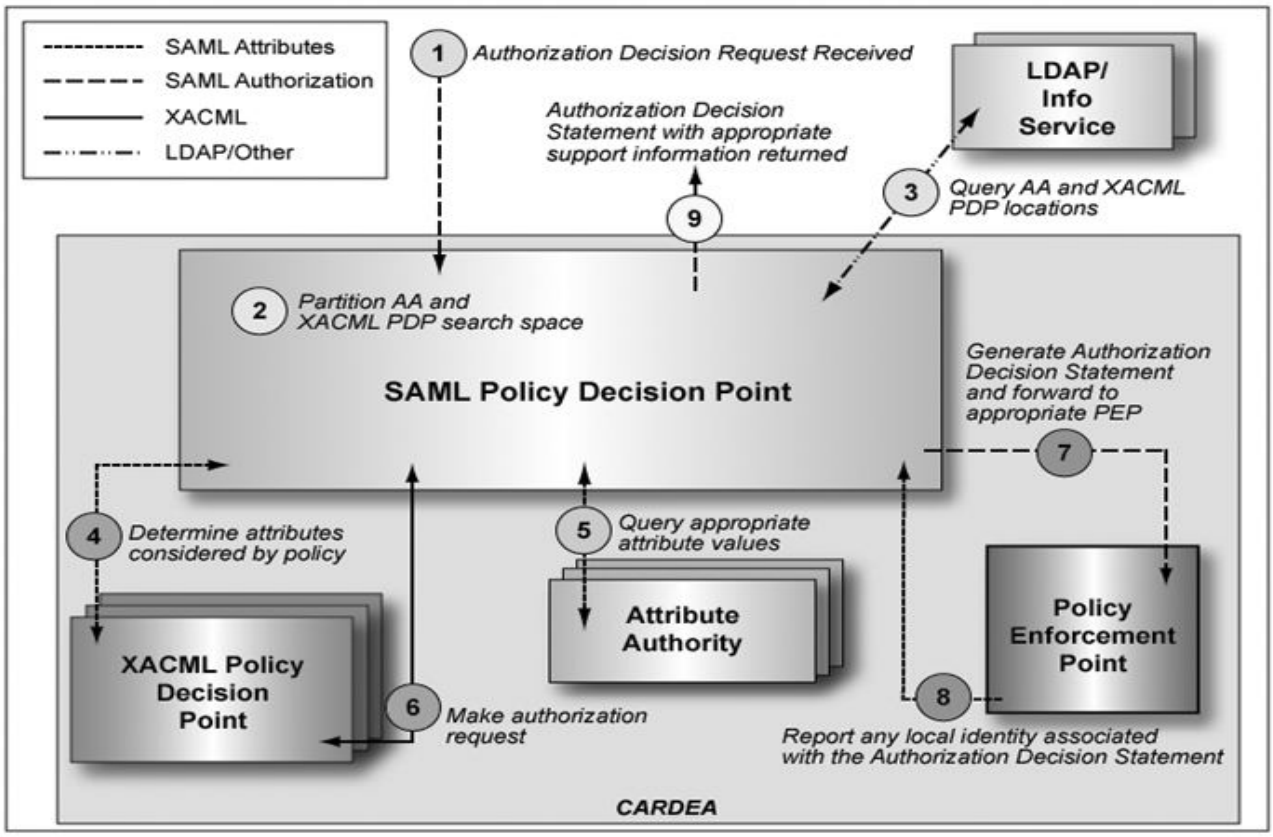

Figure 2. The Cardea Architecture

4.2.1 Authorization decision request received

Initially, the system receives a SAMLAuthorizationDecisionQuery. There are no mandatory restrictions on the origin of any accepted request other than what is required to enforce local access control policy. For example, an authorization domain may require that any request it processes be authenticated by a trusted source. Any request presenting from an untrusted source would be discarded, even if it could actually be completely processed by the system. Cardea processes all requests that are digitally signed by an identity guaranteed by a trusted authority.

4.2.2 Partition search space for locating attribute authorities

All access control requests present a set of identifying credentials to Cardea when requesting an authorization decision. Cardea extracts the credential authority identities from the authorization request to locate the desired attribute authority.

4.2.3 Query an information service to locate the authoritative AA and PDP locations

Cardea assumes that a directory service contains the necessary location and binding data for available attribute authorities. Cardea places no requirements on the security of interaction with the directory server. Each implementation must directly define and support the appropriate means to identify and interact with trusted information stores. Currently, Cardea assumes location data will be in URL format and needs no authority-specific binding data.

4.2.4 Determine attributes considered by controlling policy

Location information for an attribute authority is used to construct a SOAP endpoint representing an interface to that authority. To minimize the set of attribute assertions presented to the PDP for evaluation, a custom interface was built into the PDP to report the attribute identifiers expected within each request. XACML and SAML provide sufficient functionality in their current form to extract all attributes associated with the principals of a request. As an authority may store a large number of attributes for a single principal, this custom interface offers an optimization to reduce the number of attributes communicated between entities. Such an optimization must be balanced against the need for policy delegation and the complexity in evaluating policies for attribute designators. This custom interface assumes that the identification of attributes within an XACML policy corresponds to their identity within the attribute authority. The initial functionality maps resource identifiers to the set of subject attributes required by the policy governing that resource. XACML does not specify a format for reporting the set of attributes required by a PDP. Therefore, this custom function formats each required attribute set as SAML attribute statements, permitting a standard interpretation of each result.

4.2.5 Query appropriate attribute values

Cardea must insert actual attribute values into the final XACML request. XACML does not address how to collect the values contained within that set. Thus, a SAMLAttributeQuery is executed for each attribute. Depending on the initial authorization request, this may require interaction with several distinct attribute authorities. Regardless of the actual attribute authority contacted, the SAML protocol specifies the semantics of extracting the appropriate attribute values.

4.2.6 Execute XACML authorization request

Once the complete set of requester attributes is known, all returned values are formatted as XACML subject attributes. Resource and action attributes are handled in a similar fashion Cardea employs custom functionality to transform collected SAMLAttributeAssertions into a valid XACML attribute format. This functionality presumes a correspondence between the attribute identities used in the XACML and SAML representations of each logically equivalent attribute. After populating the request, it is enclosed in a SOAP message destined for the PDP that controls the desired resource. The payload of the response received contains the evaluation decision made by that PDP.

4.2.7 Generate an authorization decision statement for the enforcing PEP

Only an XACML context handler maintains information about the access request. However, enforcing an authorization decision often requires information from the request context. Thus, the original SAMLAttributeAssertion contains the identity of a group whereas the XACML authorization decision specifies membership validity. Therefore, the system bundles the XACML authorization decision together with all the attribute values from the request context to forward to the appropriate PEP. Although not currently incorporated into the final SAMLAuthorization-DecisionStatement, evidence used to evaluate the request and conditions attached to the decision may also be presented to the PEP.

4.2.8 Report any local identity associate with the authorization decision statement

Once the PEP receives a SAMLAuthorizationDecision-Statement, its verifies the identity of the PDP that generated the statement by authenticating the attached digital signature. The PEP must define rules that govern how authorization decision statements will be enforced. Several alternative technologies may be used to implement the rules. The only constraint placed on enforcement functionality by Cardea design requires a PEP to report any local identity bound to the authorization decision statement be returned to the initial PDP in the form of a SAMLAttributeAssertion. This constraint facilitates further distribution of the authorization process between distinct yet cooperating PDPs.

4.3 Results

Initial prototypes have shown XACML a key aspect of Cardea. XACML provides the means for resource stakeholders to uniformly express complex access control policies. It also allows standard evaluation of access control requests across heterogeneous resources and external subjects. Additionally, it integrates fluidly with SAML when used as a means to provide the attribute information presented in an attribute request, thus further facilitating interoperability across different kinds of authorization systems and domains. Finally, XACML's defines a framework which encourages the separation of authorization decisions from enforcement mechanisms, providing resource owners the ability to enforce policy decision in their locally preferred manner.

5. Privilege and Policy Management in the PRIMA System

In this section the use of XACML in an access control mechanism for grid computing systems is described. The access control mechanism is unique in that it allows users to act authoritatively for resources they control by directly creating, delegating, and combining access privileges among themselves without the intervention of resource administrators. An interesting issue is how XACML can be used to express privileges and how these XACML-expressed privileges relate to XACML-expressed policies. What is evolving from this research is the concept of a dynamic policy based on privileges that complements the more static access policy traditionally associated with XACML.

5.1 The PRIMA model

A grid computing system, like many other distributed systems, has multiple entities that are authoritative for a resource at different levels of granularity. For example, a site authority may be responsible for a site wide acceptable use policy. An authority for a specific hardware resource may define which individuals will have access to the resource itself and which services are hosted on that machine. An authority for a specific service may want to define the access rules for the service and associated data. In addition to these resource and service oriented authorities there are entities that want to exercise control over data they own and define who may have access to (parts of) their data. Individual users would like to be authoritative for resources they control and be empowered to delegate access to these resources to other users directly and efficiently. On top of this there are authorities for virtual organizations that describe collaborative groups which may incorporate resources from multiple physical organizations.

PRIMA [18][19], a system for distributed access control in grid computing environments, supports multiple authorities by allowing users as well as administrative personnel to delegate access to resources for which they are authoritative. The scope of such access can be as fine grained as access to individual data files or as encompassing as access to a whole set of compute resources. Subjects (users) can possess and delegate to other subjects fine-grained privileges to resources for which they are authoritative. Resource authorities can use the same mechanisms to grant privileges to users and to issue policy statements for their resources.

In addition to the definition of individual, delegated privileges, PRIMA allows for the definition of privilege management policies (PMPs) that are used to define the permissible actions with regard to the creation and delegation of individual privileges. Resource based access control policies (ACPs) are used to abridge or extend the set of actions allowed based on privileges held by subjects. This provides for additional flexibility in the definition of access control rules, allows for the combination of a variety of rules from different authorities and also enables the timely and uncomplicated revocation of delegated privileges.

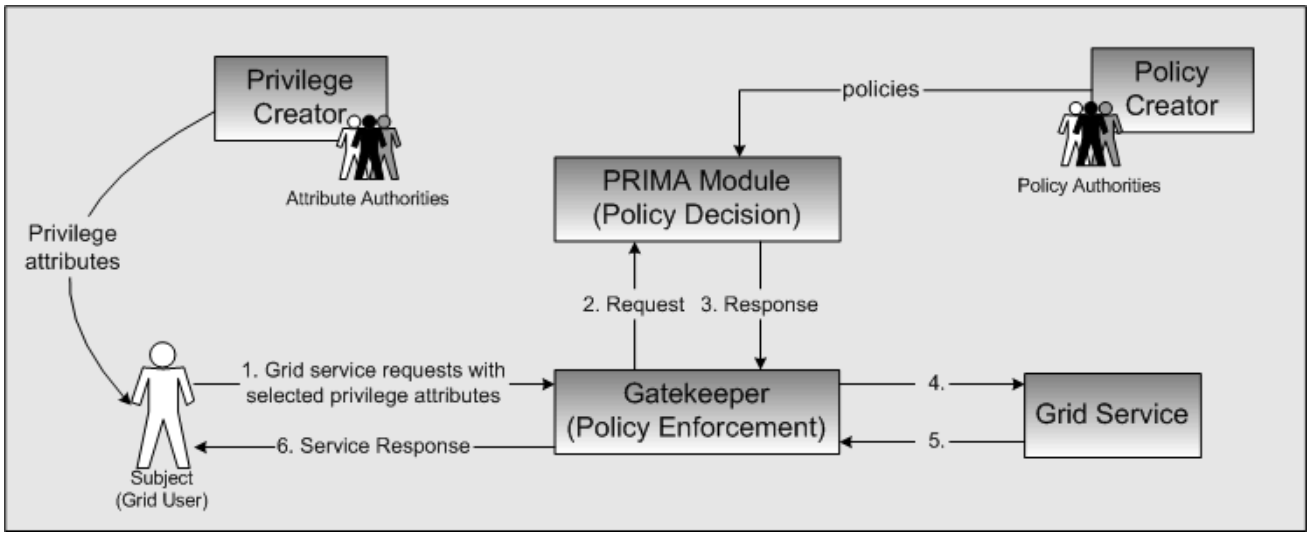

Figure 3. The PRIMA System Architecture

Recently, XACML was introduced into PRIMA to allow for a more flexible specification of access control rules in privileges and policies. The ability to reuse code for the creation and parsing of these constructs as well as possible interoperability benefits also played a key role in our decision to use XACML. PRIMA now leverages XACML to express three different types of access control information:

1. **Privilege Attributes**
   Privilege attributes are created by ordinary users, group leaders and managers and convey individual access rights to the recipient, they have a lifetime and may be relatively short lived. Privilege attributes supplied with a specific access request are complied into a dynamic policy document which is used as a unique context by the resource PDP in conjunction with the more static resource's access control policies to determine access.
2. **Privilege Management Policies**
   Privilege management policies (PMPs) define the authorities for a resource and the delegation and privilege management rules. PMPs are relatively static and typically created and maintained by system administrators.
3. **Access Control Policies**
   Traditional access control policies are used to complement the dynamic policies created from privilege statements. The combination of these two mechanisms not only provides for added flexibility in the specification of access control rules but also provides a mechanism to limit or revoke rights that were issued or delegated using privilege statements.

The definition and management of access control policies in a platform independent format such as XACML is a complex task requiring high level tools. In traditional systems, this task is often left to administrators. For ordinary users, group leaders and managers with little or no system administration background, advanced graphical user interfaces and appropriate abstractions are required to enable such users to exert their authority. Two such tools developed for PRIMA are described later in this section. One tool, the Privilege Creator, facilitates the creation of privilege attributes and their secure association with an issuer and holder. A second tool is being developed that allows for the creation and maintenance of access control as well as privilege management policies without requiring knowledge of the policy language syntax.

5.2 The PRIMA system components

Figure 3 shows an overview of the PRIMA system architecture. The three principal entities in an authorization system are subjects, which initiate requests, authorities, which provide access rules (e.g. via policies), and resources which provide services and enforce access rules. In Figure 3 two different types of authorities are shown, the attribute authorities that issue privilege statements to subjects, and the more traditional policy authorities that create access-control and privilege management policies and provision them to the resources. The resource is split into three logical components, a policy enforcement point (gatekeeper PEP), a policy decision point (the PRIMA PDP) and the service. The interaction between the components can be characterized as a traditional authorization pull model [28]. The inclusion of privilege attributes with a request (attribute push, see [8]) which state specific access permissions in the form of rule statements bound to the specific individual is a distinct feature of PRIMA. The decision on which of a subject's attributes will be provided with a specific request lies with the subject and thus provides the basis for a least privilege access scheme.

The PRIMA system has been implemented specifically to complement the security mechanisms present in the Globus Toolkit [9]. The PRIMA PDP is located on the Globus resource itself and communicates with the PEP through a direct, local communication channel. Originally PRIMA used proprietary formats to define privilege attributes, resource access control policies, and privilege management policies. A proprietary API was used for communication between the PEP and PDP. By replacing these proprietary formats with XACML, we leveraged its standardized open language and message formats, achieved greater flexibility and expressiveness and facilitated reuse of parser and evaluation libraries. The possible use of standard tools for the management of policy documents in XACML is another important advantage.

5.2.1 Flow of Access Control Information

Access control information (ACI) encompasses all the data provided to make access control decisions. In PRIMA, ACI consists primarily of the privilege statements (in the form of privilege attributes), the ACPs, and the PMPs. Information about the requested action and environmental data is also taken into consideration by the PDP. Privilege statements are provided by attribute authorities to subjects at admin time, i.e. de-coupled from the point in time where a request is made to a resource (access time). Access control policies and privilege management policies are provided to the PDPs by the respective administrative entities, also at admin time.

The sequence of actions at request time (as indicated in Figure 3) is as follows: (1) a subject contacts a resource (it's PEP), mutually authenticates and provides a resource request along with privileges of the subject's choosing. The PEP in turn compiles all provided privileges into a ***dynamic policy,*** which will provide the individual, least-privilege policy context for the specific access**.** During creation of the dynamic policy, the PEP checks that each included privilege is applicable and valid through queries to the PDP, which bases its answers on compliance with the PMP. Once the dynamic policy has been assembled it is provisioned to the PDP. (2) The PEP contacts the PDP to determine if the actual request is permissible with respect to all applicable access-control policies and with respect to the dynamic policy. (3) The PDP provides a response to the PEP. (4) If the authorization was successful the PEP will permit the subject's request to pass through to the service and the service response (5) will be provided to the subject (6).

5.2.2 The Privilege Creator

The Privilege Creator, ACGen, is a graphical user tool implemented in the Java Programming Language. It allows the user to create privilege statements that will be embedded in an X.509 Attribute Certificate (AC) [28] as the payload. Existing grid infrastructures are versed in certificate formats, thus minimizing the required infrastructure modifications. A single attribute certificate may contain a set of privileges and can also be bound to a set of entities. The “Issuer” and “Holder” entities of the AC are filled with the respective X.500 distinguished names (DN), and the AC structure is signed with the issuer's private key. The holder DN can either be acquired by searching an LDAP server or entered manually. The privilege statement itself is an individual XACML rule. The rule specifies the subject to which the attribute is bound (holder), the resource to which it applies, the permitted action, and, optional conditions. Appendix A1 shows such a rule that grants access to a specific file. Currently supported are privileges that define system access (the right to a local user account), file access and network access.

5.2.3 Policy Creator

The policy creator also is a GUI tool implemented in the Java Programming Language that aids the user in creating XACML ACPs. While the current implementation only supports the creation of very limited policies for grid resources, it allows an authoritative party to define basic, predefined access rules with relative ease. Policy creation and enactment may be done remotely, without the need to edit proprietary access control lists at the resource through shell access. The tool mainly aids with the syntactical complexity of XACML but eventually will also provide semantic support, possibly through policy templates. Embedding the access control policies in X.509 ACs and provisioning them to the PDPs using grid middleware file staging performs secure movement. A small utility at the PDP verifies received policies for issuer authority and integrity (leveraging the PDP to query the privilege management policies) and configures them into the PDP's policy store.

5.2.4 The PRIMA Policy Decision Point

The policy decision point accepts XACML requests for access control and privilege management decisions. It provides answers based on three different policies, the (set of) ACPs, the dynamic policies and the PMPs. ACPs are provided to the PDP by the respective policy authority via our policy creator. PMPs, due to their crucial role in defining the sources of authority and thus bootstrapping the PDPs operation, have to be manually made available to the PDP by a traditional system administrator.

The initial version of our PDP uses the C++ implementation of XACML by JiffySoftware [16], which is currently available as a binary alpha release. We plan to switch to use Sun Microsystem's open source XACML library for future releases, as it provides richer functionality.

5.3 Results

The change from simple proprietary formats for privilege statements and access policies to XACML allowed for the reuse of creation, parsing and evaluation code that already exists and has been tested. A drawback of this change was the significantly greater size of policies and privilege statements due to the XML encoding overhead and verbosity of the language.

The creation of dynamic policies augments the more static policy model XACML was originally developed for and shows that XACML policies and other language components can be applied in a variety of access control scenarios. The integration of XACML into the PRIMA infrastructure, which leverages X.509 attribute certificates for the transport of access control information, was without problems and did not require changes to the protocols used between grid nodes. The support for X.500 names in XACML enabled us to link policy rules directly to the entities identified by X.509 certificates.

6. Implementation considerations

The initial implementations presented in this paper needed to address several challenges common to distributed authorization systems that are not addressed directly within XACML. Several of the subjects fall outside the scope of the XACML, such as management and retrieval of authorization attributes, or the location of applicable policy decision points. Complimentary technologies are required to provide the needed functionality. Other issues arise when extending XACML functionality, either for expressiveness or manageability, such as management of actual policy files. The remainder of this section presents a number of such issues that require careful consideration when incorporating XACML into a distributed authorization system and some of the approaches adopted by these initial implementations.

6.1 Creation and management of access control policies

XACML provides a mechanism independent representation of access rules that vary in granularity via a standard yet flexible language. This flexibility permits the combination of multiple

policies (e.g. from different authoritative parties) into a single applicable policy set to use when making access control decisions for resources in a widely distributed system with overlapping competencies. Further, this mechanism-independent representation of access rules allows a single policy to be applied to heterogeneous resources throughout and across administrative domains. This common representation greatly reduces errors, discrepancies, and auditing complexity.

However, creation of actual XACML policies is not a simple task. Further, supporting XACML in heterogeneous environment calls for fully specified data type and function definitions that produce a highly verbose document even if the actual policy rules are trivial. Manual creation of such policies by ordinary users, as required in the PRIMA distributed authority model (see § 5.2), or by resource administrators, as required in the Cardea system (see §4.2), is not reasonable. Therefore, additional management tools, such as the introduced PRIMA policy creator, to support policy file management and administration are required. As previously noted, the standard open format of XACML encourages reuse of these tools and libraries across many diverse systems.

6.2 Encoding of Privilege Management Policies in XACML

The flexibility of the XACML language allows its application to emerging scenarios without modification to the existing vocabulary. XACML is not directly targeted at specifying sources of authority and privilege management rules. PRIMA's use of XACML demonstrates the flexibility of allowing such policy encodings without changes to the basic XACML vocabulary. A sample privilege management policy in XACML is shown in appendix A2. This policy states that "Markus Lorch" and "Sumit Shah" can grant access rights (action: "delegate GRAM access") for gram://zuni.cs.vt.edu (a Globus resource) to all the users belonging to the Virginia Tech domain (OU=Virginia Tech User). Current work in prototyping attribute release policies through the Web Services Policy Language (an XACML profile) underlines the versatility and flexibility of XACML with respect to new applications of the language.

6.3 Locating the correct PDP

Before an authorization decision can be obtained, an authoritative PDP must be located. This boot strapping problem is common to any distributed system and not specific to authorization systems based on XACML. Thus, XACML does not provide a standard mechanism to resolve this issue but relies on individual implementations to handle it appropriately to their environment. Initial system implementations either assume fixed PDP locations with policy file discovery dependant on the requested resource or discovery of a PDP via an information service query to a trusted source. For example, Cardea assumes that a directory service contains the necessary location and binding data for the appropriate PDP. Once a PDP is identified, XACML functionality provides for the location of applicable policy files, including policies to be retrieved from a remote location.

6.4 XACML request preparation and request context management

XACML considers the collection and encoding of attributes used in an authorization system to lie outside its core focus. Further, XACML views attributes as an external form of access control information that must be converted from their native form to be included in an XACML authorization decision request in the form of a request context by a context manager component. Therefore, XACML does not standardize interactions to retrieve this data for an authorization request. Two distinct approaches have been implemented within the introduced systems to share subject data used for authorization. The first provides a framework by which this information is shared via SAML. The second uses privilege attributes managed by subjects to directly influence the context creation.

The XACML model is based on the authorization pull sequence [28] and requires the context manager to maintain state information to associate requests that it created with received responses. If another authorization sequence such as the push or the agent sequence [28] are desired, the contextual information necessary for a PEP to enforce an access decision response from a PDP has to be supplied to the PEP through a supplementary mechanism. Current work on SAML 2.0 proposes to include the original authorization decision request context with an authorization decision response, which would address this issue.

6.4.1 Encoding of descriptive attributes in Cardea

Cardea employs SAMLAttributeAssertions to collect and encode attribute data for an authorization decision request. Custom functionality transforms the collected SAMLAttributeAssertions into a valid XACML attribute format. Although specific mappings need not be predefined, the functionality presumes a correspondence between the attribute identities used in the XACML and SAML representations of each logically equivalent attribute. By supporting such transformations, these attributes are available both within the decision and enforcement phases of authorization. Therefore, Cardea augments XACML functionality with SAML functionality to provide this data to all participants in an authorization decision.

6.4.2 Encoding of privilege attributes in PRIMA

In PRIMA, individual XACML rule statements are used to represent individual privilege attributes. A secure container provides issuer and validity information outside of the attribute definition. The attribute itself consists of a "rule" construct within which the holder of the privilege and the resource for which the right is targeted are specified in a "target" construct. The rule has a "permit" effect if matched and specifies request details in the "action" construct, while a "condition" construct may optionally be used to provide for more complex rules. The container (X.509 Attribute Certificate) provides the information on authority (issuer identifier, signature) and validity (time frame), which are not defined in a standard XACML rule. This separation of the validity and authority information from the actual access control rules is not a drawback but rather promotes the separation of concerns in the system. The validity and authority information is used when a request context and dynamic policy is built by the PEP, whereas the access control rules will provide the content for the dynamic policy that later will be used by the PDP.

A change to an XML based container for PRIMA privilege attributes (i.e. SAML attribute assertions) was not necessary as the integration of XACML structures into ASN.1 (as an IA5String) did not pose any problems, thus the existing

infrastructure, protocols and tools for the creation and exchange of the attribute containers could remain unchanged.

7. Related Work and Ongoing Work

This section provides general descriptions of systems that provide similar features to those found in XACML and the systems described in this paper. Following that is a brief introduction to other work being done to use XACML in future systems.

7.1 Related Work

There are several other projects that deal with distributed authorization. Although each of these systems takes a unique approach to the authorization problem, the features of XACML directly benefit or improve the existing functionality.

The Community Authorization Service (CAS) [23] reduces administrative overhead by separating resource administration from community specific administration. Resource administrators grant bulk rights to a specific community (e.g. a Virtual Organization (VO) [10]) while community administrators manage membership and privileges associated with members without resource administrator intervention. Group members authenticate to grid resources with a group credential (limited proxy credential) that limits the individual's rights to a subset of the rights the community has at the resource. To promote scalability, CAS requires only a shared group account per community rather than an individual account for each member. The CAS system is independent of the policy language used to define restrictions in proxy credentials. XACML is being evaluated as an alternative to the proprietary policy statement format currently used in restricted proxy credentials.

Akenti [27] addresses issues raised when multiple authorities (stakeholders) control access to resources. Akenti provides a policy language to define, as well as infrastructure components to enforce, flexible access control policies. Akenti leverages a collection of proprietary XML-based certificates to encapsulate policy, use-condition and attribute statements. For a flat set of resources there is only one policy certificate. For hierarchical resources there may be multiple policy certificates, one for each level of the hierarchy. Akenti allows the certificates to be stored in remote repositories and provides mechanisms to ensure that all applicable use-conditions (from possibly a group of stakeholders) are combined when making an access control decision. The Akenti team will investigate the use of XACML for the representation of distributed policies and the applicability and effectiveness of the policy combining mechanism.

Like Akenti, PERMIS [5] provides a Privilege Management Infrastructure (PMI). PERMIS uses X.509 Attribute Certificates [8] to specify subject attributes such as roles and permissions. Each permission represents the right to access a target in a particular mode. PERMIS defines a hierarchical role based access control (RBAC) policy language in terms of those roles and permissions. The RBAC policy (in XML format) is used to control access to all the targets within the policy domain and is composed of a number of sub-policies. The PERMIS project is currently investigating the use of XACML as a core language to replace parts of their proprietary policy language.

7.2 Ongoing Work

The systems discussed in this paper represent current and future work that leverage XACML for authorization. This paper also discusses several standards that will integrate with XACML to streamline authorization. However, there are many other systems that already integrate XACML or are starting to experiment with doing so now. This section provides some small insight into a few categories of such systems.

One such class of systems is peer-to-peer (P2P) which typically lacks any form of centralized administration and usually leaves users to manage their own data and policies. P2P projects like JXTA [17] provide a general framework for building applications. Thus, the underlying security systems must be flexible enough to handle any application but still be manageable. Work is currently underway within JXTA projects to explore XACML's role in its authorization framework and the tools needed for JXTA and P2P environments. Other P2P systems are exploring using XACML to address privacy concerns. Examples include a research project at Sun Microsystems looking at human interaction and personal privacy protection.

Another class of systems evaluating XACML is Role Based Access Control (RBAC), an increasingly important component in distributed systems, but one that is often hard to support in heterogeneous environments. NIST has a project [14] to define flexible RBAC systems, and it has strong authorization requirements. The XACML Technical Committee is working with NIST to define this relationship, and a draft [1] is available.

Finally, there are several projects that are evaluating XACML as a core policy language within its authentication engine, like the ebXML Registry [22]. ebXML Registry includes support for XACML in its latest specification draft and prototypes [24] using XACML are already working well. There are also proposals, for example, defining how to use XACML as a complementary language in systems like the Java Policy framework, though many of these are in an early discussion stage.

8. Summary

Early experiences using XACML in distributed systems have proven positive. The language is indeed useful for specifying arbitrarily complex policies in a wide variety of (distributed) applications and environments. While targeted at traditional access control systems, XACML also proves practical for expressing privilege management policies and defining privilege statements. The standard format works well in tying together heterogeneous systems, and already fosters development of common tools. Its open standard status, definition in XML, and availability of open source projects has already drawn support from diverse applications. XACML's ability to tie into other authorization systems makes it a natural inter-operability point, even for legacy systems. Its expressive semantics and extensible nature also make it useful as an intermediary language. The ability to work with decentralized policies, and the ease with which it integrates into the systems presented in this paper point to XACML as an excellent choice for distributed authorization systems.

XACML does have some limitations at present, however. The language's flexibility and expressiveness comes at the cost of complexity and verbosity. As such, it's hard to work directly with the language or policy files. Tools are underway, but until there is widespread availability, it will be hard for average users to work with any XACML-based system. Even with good tools in place, there is an inherent semantic complexity that's separate from the syntactic complications. This too will need to be addressed, and tools are needed that help people understand the meaning of

policies. Finally, there are remaining issues in how XACML presently works with other standards, some of which are fairly critical, such as online protocols and storage systems. Again, these issues are currently being addressed, but until they are resolved, it will remain difficult to leverage the full power of XACML.

In conclusion, XACML is an important and useful component for a distributed system's authorization needs. It supports varied authorization models and approaches within its basic definitions. Further, several potential areas to evaluate optimizations against flexibility are known. Finally, missing pieces have been identified and are being addressed. As has been illustrated in this paper, XACML will work well with real systems today, and it has the features required to help tie authorization systems together in the future.

9. Acknowledgments

This work was supported in part by the Virginia Commonwealth Information Security Center (CISC) and the Pennsylvania State University.

The authors would like to thank the following people for their efforts in the work described in this paper: Anne Anderson, Steve Carmody, Tom Doeppner, Yassir Elley, Tracy Hadden, Steve Hanna, Linda Null, Radia Perlman, Roberto Tamassia, Bill Thigpen, and Danfeng Yao.

10. References

[1] Anne Anderson, “XACML RBAC Profile”, OASIS TC Working Draft, June 5th, 2003

[2] http://xml.apache.org/axis, visited 2003-09-29

[3] Mark Bartel et al, “XML Signature Syntax and Processing”, World Wide Web Consortium Recommendation, February 2002

[4] Don Box et al “Simple Object Access Protocol (SOAP) 1.1” World Wide Web Consortium Note, May 2000

[5] D. Chadwick and A. Otenko, “The Permis X.509 Role Based Privilege Management Infrastructure”, SACMAT 2002 Conference Proceedings, ACM Press, NY, pp. 135 - 140

[6] D Durham et al, “The COPS (Common Open Policy Service) Protocol”, IETF Proposed Standard, RFC 2748, Jan. 2000

[7] Marlena Erdos and Scott Cantor, “Shibboleth Architecture v5”, Internet2/MACE, May 2002

[8] S. Farrell, R. Housley, “An Internet Attribute Certificate Profile for Authorization“, IETF RFC, April 2002

[9] I. Foster and C. Kesselman, “Globus: A Toolkit-Based Grid Architecture”, The Grid, Blueprint for a Future Computing Infrastructure, Morgan Kaufmann, San Francisco, 1999, pp. 259-278

[10] I. Foster, C. Kesselman, and S. Tuecke, "The Anatomy of the Grid: Enabling Scalable Virtual Organizations", International Journal of Supercomputer Applications, 2001.

[11] Simon Godik, Tim Moses, et al, “eXtensible Access Control Markup Language (XACML) Version 1.0”, OASIS Standard, February 18th, 2003

[12] Cheh Goh, “Policy Management Requirements”, Hewlett Packard Laboratories Technical Report, HPL-98-64, April 1998, http://www.hpl.hp.com/techreports/98/HPL-98-64.html

[13] Phillip Hallam-Baker, Eve Maler, et al, “Assertions and Protocol for the OASIS Security Assertion Markup Language (SAML), Oasis Standard, November 5th, 2002

[14] Information Technology Industry Council, “Role Based Access Control”, Proposed ANSI Standard, April 4th, 2003

[15] http://www.ipg.nasa.gov, visited 2003-09-29

[16] http://www.jiffysoftware.com/, visited 2003-09-29

[17] http://www.jxta.org, visited 2003-09-29

[18] Markus Lorch and Dennis Kafura, "Supporting Secure Ad-hoc User Collaboration in Grid Environments", Proc. 3rd Int. Workshop on Grid Computing - Grid 2002, Pages 181 - 193, Baltimore, USA, November 18th, 2002

[19] Markus Lorch, David Adams, Dennis Kafura, Madhu Koneni, Anand Rathi, Sumit Shah “The PRIMA System for Privilege Management, Authorization and Enforcement in Grid Environments”, communicated to the 4th Ind. Workshop on Grid Computing – Grid 2003

[20] Tim Moses, Anne Anderson, Seth Proctor, and Simon Godik, “XACML Profile for Web Services”, OASIS TC Working Draft, September 29th, 2003

[21] Sidharth Nazareth, “SPADE: SPKI/SDSI for Attribute Release Policies in a Distributed Environment”, Dept of Computer Science, Dartmouth College Technical Report TR2003-453, May 30, 2003

[22] OASIS Registry Technial Committee, “OASIS/ebXML Registry Services Specification v2.0”, April 2002

[23] L. Pearlman et al, “A Community Authorization Service for Group Collaboration”, 2002 IEEE Workshop on Policies for Distributed Systems and Networks

[24] Registry Reference Project, http://ebxmlrr.sourceforge.net/, visited 2003-09-29

[25] K E Seamons et al, “Protecting Privacy during On-line Trust Negotiation”, Second Workshop on Privacy Enhancing Technologies, April 2002

[26] http://sunxacml.sourceforge.net, visited 2003-09-29

[27] M. Thompson, A. Essiari, S. Mudumbai, “Certificate-based Authorization Policy in a PKI Environment,” ACM Transactions on Information and System Security, to appear, August 2003

[28] J. Vollbrecht et al, “AAA Framework“, IETF RFC 2904, August 2000

[29] William Winsborough and Ninghui Li “Protecting Sensitive Attributes in Automated Trust Negotiation”, WPES, November 2002

[30] R Yavatkar, D Pendarakis, and R Guerin, “A Framework for Policy-based Admission Control”, IETF Informational Standard, RFC 2753, January 2000

11. Appendix

11.1 A File Privilege Encoded as an XACML Rule Component

```
<Rule RuleId="File-Privilege-Rule" Effect="Permit">
 <Target>
  <Subjects>
   <Subject>
    <SubjectMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:x500Name-equal">
     <AttributeValue DataType="urn:oasis:names:tc:xacml:1.0:data-type:x500Name">
         CN=Sumit Shah (sshah),OU=Virginia Tech User,OU=Class 2,O=vt,C=US
     </AttributeValue>
     <SubjectAttributeDesignator AttributeId="urn:oasis:names:tc:xacml:1.0:subject:subject-id"
DataType="urn:oasis:names:tc:xacml:1.0:data-type:x500Name" />
    </SubjectMatch>
   </Subject>
  </Subjects>

  <Resources>
   <Resource>
    <ResourceMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:anyURI-equal">
     <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#anyURI">
         gridftp://zuni.cs.vt.edu/data/collaboration/results.dat
     </AttributeValue>
     <ResourceAttributeDesignator AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-id"
DataType="http://www.w3.org/2001/XMLSchema#anyURI" />
    </ResourceMatch>
   </Resource>
  </Resources>

  <Actions>
   <Action>
    <ActionMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
     <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
         Read
     </AttributeValue>
     <ActionAttributeDesignator AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"
DataType="http://www.w3.org/2001/XMLSchema#string" />
    </ActionMatch>
   </Action>
  </Actions>
 </Target>
</Rule>
```

11.2 A Simple Privilege Management Policy in XACML

```
<?xml version="1.0" encoding="UTF-8" ?>
<Policy xmlns="urn:oasis:names:tc:xacml:1.0:policy"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="urn:oasis:names:tc:xacml:1.0:policy cs-xacml-schema-policy-01.xsd"
        PolicyId="IssuerVerification"
        RuleCombiningAlgId="urn:oasis:names:tc:xacml:1.0:rule-combining-algorithm:permit-overrides">
<Description>
  This is a privilege management policy. The RULE defines two subjects as issuers which are
  authoritative to grant "Delegate GRAM access" privileges for resource "gram://zuni.cs.vt.edu/"
  to other entities. The Condition element adds a constraint by specifying that only entities
  that belong to the Virginia Tech domain can be holders of the privilege.
</Description>

<Target>
  <Subjects>
   <AnySubject />
  </Subjects>
  <Resources>
   <AnyResource />
  </Resources>
  <Actions>
   <AnyAction />
  </Actions>
</Target>
```

```
<Rule RuleId="IssuerVerificationRule" Effect="Permit">

<Description>
  This is the main rule of this policy, if a request is successfully matched against
  this rule an evaluating PDP will return Permit
</Description>

 <Target>
   <Subjects>
    <Subject>
     <SubjectMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:x500Name-equal">
        <AttributeValue DataType="urn:oasis:names:tc:xacml:1.0:data-type:x500Name">
           CN=Markus Lorch (mlorch),OU=Virginia Tech User,OU=Class 2,O=vt,C=US
        </AttributeValue>
              <SubjectAttributeDesignator
                AttributeId="urn:oasis:names:tc:xacml:1.0:subject:subject-id"
                          DataType="urn:oasis:names:tc:xacml:1.0:data-type:x500Name" />
     </SubjectMatch>
    </Subject>
    <Subject>
     <SubjectMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:x500Name-equal">
        <AttributeValue DataType="urn:oasis:names:tc:xacml:1.0:data-type:x500Name">
           CN=Sumit Shah (sshah),OU=Virginia Tech User,OU=Class 2,O=vt,C=US
        </AttributeValue>
        <SubjectAttributeDesignator
                AttributeId="urn:oasis:names:tc:xacml:1.0:subject:subject-id"
                           DataType="urn:oasis:names:tc:xacml:1.0:data-type:x500Name" />
     </SubjectMatch>
    </Subject>
  </Subjects>
  <Resources>
    <Resource>
     <ResourceMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:anyURI-equal">
      <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#anyURI">
               gram://zuni.cs.vt.edu/</AttributeValue>
      <ResourceAttributeDesignator
                AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-id"
                           DataType="http://www.w3.org/2001/XMLSchema#anyURI" />
     </ResourceMatch>
    </Resource>
  </Resources>

  <Actions>
    <Action>
     <ActionMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
       <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
                 delegate GRAM access  </AttributeValue>
       <ActionAttributeDesignator AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"
DataType="http://www.w3.org/2001/XMLSchema#string" />
     </ActionMatch>
    </Action>
   </Actions>
 </Target>

 <Condition FunctionId="urn:oasis:names:tc:xacml:1.0:function:x500Name-match">
   <AttributeValue DataType="urn:oasis:names:tc:xacml:1.0:data-type:x500Name">
      OU=Virginia Tech User,OU=Class 2,O=vt,C=US</AttributeValue>
   <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:x500Name-one-and-only">
     <SubjectAttributeDesignator DataType="urn:oasis:names:tc:xacml:1.0:data-type:x500Name"
AttributeId="urn:oasis:names:tc:xacml:1.0:attribute:holder" />
   </Apply>
 </Condition>
</Rule>

<Rule RuleId="FallThroughReturnDeny" Effect="Deny">
<Description>
  This second rule is a "fall-through" rule. It ensures that a PDP evaluating the
  policy will return DENY if the main rule cannot be fulfilled. By default if
  a condition in a rule evaluates to FALSE then the rule's effect is NotApplicable
  and consecutive rules will be processed.
</Description>
</Rule>

</Policy>
```