Authorization Policy in a PKI Environment

Mary R. Thompson, Srilekha Mudumbai, Abdelilah Essiari, Willie Chin

National Energy Research Scientific Computing Division
Ernest Orlando Lawrence Berkeley National Laboratory
Berkeley, CA, 94720
pkidev@george.lbl.gov

Abstract

The major emphasis of Public Key Infrastructure has been to provide a cryptographically secure means of authenticating identities. However, procedures for authorizing the holders of these identities to perform specific actions still needs additional research and development. While there are a number of proposed standards for authorization structures and protocols. [17, 5, 22, 10, 6] based on X.509 or other key-based identities, none have been widely adopted. As part of an effort to use X.509 identities to provide authorization in highly distributed environments, we have developed and deployed an authorization service based on X.509 identified users and access policy contained in certificates signed by X.509 identified stakeholders. The major goal of this system, called Akenti, is to produce a usable authorization system for an environment consisting of distributed resources used by geographically and administratively distributed users. Akenti assumes communication between users and resources over a secure protocol such as secure socket layer (TLS) which provides mutual authentication with X.509 certificates. This paper explains the authorization model and policy language used by Akenti, and how we have implemented an Apache authorization module to provide Akenti authorization.

Background

There is significant and growing set of distributed computing environments where the resources, resource stakeholders and users are geographically and organizationally distributed. The DOE sponsored Collaboratories [1] and various "Computational Grids" [13] are examples of these as well as the ubiquitous Web-controlled sets of documents and services. These systems effectively define a *Virtual Organization* whose members and resources span many different real organizations. These virtual organizations need a way to authenticate and then authorize their users.

One of the characteristics of a collaboratory or Grid is that both the stakeholders and users may come from many different administrative domains. Thus the virtual organization needs to identify its users in a domain neutral manner. The most common candidates for cross-domain identities are Kerberos and PKI. Kerberos is mostly used within a single administrative domain, but there are many examples of cross-authenticated Kerberos realms, where the Kerberos administrators have agreed to accept tokens from another realm. Negotiating cross-realm agreements is often a lengthy and complex process. Some examples of such domains are universities where there may be multiple Kerberos realms within the university, and the DOE's ASCI-DisCom^2 program [9] that connects Lawrence Livermore National Laboratory, Los Alamos National Laboratory and Sandia National Laboratories in a computational Grid.

Looser collaborations, such as Grids based on Globus [14] middleware, [24,27] Collaboratories [8,25] and portals [20] have chosen to use PKI identities to authenticate members. These organizations either run a Certificate Authority of their own and/or accept certificates from a set of trusted CAs. Establishing trusted CA relationships can also be a lengthy process, but since many current collaboratories and grids are experimental in nature, the trust relations have been established on an informal basis by the researchers, rather than the system security administrators. Once a collaboration has decided to use PKI identities to authenticate users, it needs to develop an authorization system using those identities plus some additional access policy information for each of its resources.

This work is supported by the U. S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research, Mathematical, Information and Computation Sciences office (http://www.er.doe.gov/production/octr/mics), under contract DE-AC03-76SF00098 with the University of California.See the disclaimer at http://www-library.lbl.gov/teid/tmRco/howto/RcoBerkeleyLabDisclaimer.htm.This document is report LBNL-49512.

Another characteristic of collaboratories and Grids is that their resources, such large scientific instruments, computing resources and data stores, may have more than one person (called a stakeholder) who needs to control access to the resource. For example, when remote control of an instrument is allowed the instrument administration may want assurance that any user who can control the instrument has passed a local training course, while the principal investigator may be mostly concerned that the person controlling the instrument during his allowed time is a member of his research group. An authorization system that allows access policy to be defined independently and remotely from the resource gateway is needed.

However, standard access control methods typically require that the stakeholder has privileged access to the machine on which the resource resides to set the access control. Also such systems, to the extent that they use the underlying operating system for actual access control, require that all users of a shared resource must have a local account on the system. The requirement for individual system accounts on the resource machine does not scale well.

We have developed the Akenti [32] authorization system to meet these two needs: to use a virtual organization-wide user identity (in our case an X.509 identity certificate); and to facilitate setting access policy by multiple independent stakeholders remote from the actual resource gateway.

This paper explains the authorization model and policy language that we use, and how we have implemented an Apache authorization module to provide the same authorization policy and mechanism for resources accessed via a Web browser as accessed by other remote methods such as Globus job submission [14] or CORBA object invocation.

Akenti

Akenti is built using X.509 identity certificates [18] and the SSL/TLS [7] connection protocols to securely identify a user that is requesting access to a resource. It represents the authorization policy for a resource as a set of (possibly) distributed digitally signed certificates. These policy certificates are independently created by authorized stakeholders. When an authorization decision needs to be made, the Akenti policy engine gathers up all the relevant certificates for the user and the resource, validates them, and determines the users rights with respect to the resource.

Authorization model

The Akenti model consists of *resources* that are being accessed via a *resource gateway* by *users*. These users connect to the resource gateway using the SSL handshake protocol to present authenticated X.509 identity certificates. The *stakeholders* for the resources express *access constraints* on the resources as a set of *signed certificates*, a few of which are self-signed and must be stored on a known secure host (probably the resource gateway machine), but most of which can be stored remotely. These certificates express what attributes a user must have in order to get specific rights to a resource, who is trusted to make such Use-condition statements and who can attest to a user's attributes. At the time of the resource access, the resource gatekeeper asks a trusted Akenti server, what access the user has to the resource. The Akenti server finds all the relevant certificates, verifies that each one is signed by an acceptable issuer, evaluates them, and returns the allowed access. See Figure 1.

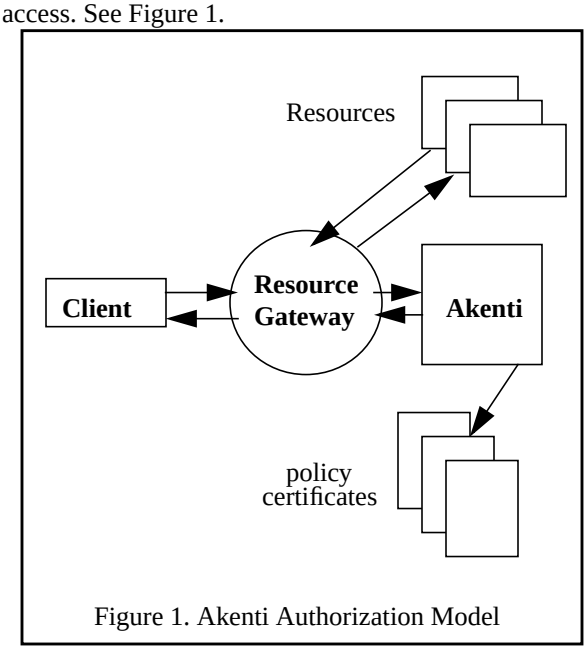

Figure 1. Akenti Authorization Model

There are several models for arriving at access control decisions. One is the classical access control list model, where the user just presents an identity to the gatekeeper who finds the policy information for the resource and evaluates the users access. Another is the capability model, where the user presents a capability which grants the holder specific rights to the resource, and the gatekeeper has to verify that the user has come by the capability legitimately and then interpret the rights that have been presented. There are also hybrids of the two models, where a user may present some identity information and possibly a restricted set of his full rights.

We have mostly concentrated on the first model in order to allow applications to use Akenti authorization over standard SSL connections which can transport and verify X.509 identity certificates. We have also experimented with s capability model where Akenti will return a signed capability certificate containing a subject's Distinguished Name (DN), public key, the Certificate Authority (CA) that signed for this name, the name of the resource and the subject's rights. If this is presented to a resource gatekeeper, along with an authenticated identity certificate, the gatekeeper need only verify the signature of the certificate by using its copy of the Akenti server's public key, and verify that the subject named in the capability is the same as that in the identity certificate. These capability certificates are short-lived in order to avoid the problems of revocation.

Akenti policy language

Akenti policy is expressed in XML and stored in three types of signed certificates: *Policy certificates*, *Use-condition certificates* and *Attribute certificates*. Policy certificates are self-signed, co-located with the resources to which they apply and contain only minimal information. Use-condition certificates contain the constraints that control access to a resource. Attribute certificates assign attributes to users that are needed to satisfy the use constraints. Akenti attribute certificates are simpler than the proposed IETF Attribute certificates. See the section on Related Work for a more detailed comparison. See Figure 2 for an example of a Use-condition certificate and Appendix A for the DTD definition of the complete Akenti Certificate schema.

Policy certificates specify who the resource stakeholders are, and thus who may sign Use-condition certificates. The Use-condition certificates specify who can attest to the required attributes and thus who may sign Attribute certificates. Whenever a certificate is used, the Akenti policy engine will check that it has been signed by an acceptable issuer, and that the signature verifies.

```
<?xml version="1.0" encoding="US-ASCII"?>
<!DOCTYPE AkentiCertificate SYSTEM "/home/g1/proj/akenti/release/common/AkentiCertificate.dtd">

<AkentiCertificate>
 <SignablePart>
 <Header Type="useCondCertificate" SignatureDigestAlg="RSA-MD5" CanonAlg="AkentiV1">
    <Version ver="V1"/>
    <ID id="griffy.lbl.gov#4e6ba338#Mon Mar 01 10:56:51 PST 1999"/>
    <Issuer>
       <UserDN> /C=US/O=Lawrence Berkeley National Laboratory/OU=ICSD/CN=Mary R. Thompson </UserDN>
       <CADN> /C=US/O=Lawrence Berkeley National Laboratory/OU=ICSD/CN=IDCG-CA </CADN>
    </Issuer>
    <ValidityPeriod start="981224003646Z" end="020123003646Z"/>
 </Header>
 <UseConditionCert scope="local" enable="false">
  <ResourceName> LBL </ResourceName>
  <Condition>
    <Constraint>( o=Lawrence Berkeley National Laboratory | ( group = distrib ) ) </Constraint>
    <AttributeInfo type="X509">
            <AttrName> o </AttrName>
            <AttrValue> Lawrence Berkeley National Laboratory </AttrValue>
            <CADN> /C=US/O=Lawrence Berkeley National Laboratory/OU=ICSD/CN=IDCG-CA </CADN>
    </AttributeInfo>
    <AttributeInfo type="AKENTI">
            <AttrName> group </AttrName>
            <AttrValue> distrib </AttrValue>
            <Principal>
              <UserDN> /C=US/O=Lawrence Berkeley National Laboratory/OU=ICSD/CN=Srilekha Issuer </UserDN>
               <CADN> /C=US/O=Lawrence Berkeley National Laboratory/OU=ICSD/CN=IDCG-CA </CADN>
            </Principal>
    </AttributeInfo>
  </Condition>
  <Rights> read, write </Rights>
  <SubjectCA> /C=US/O=Lawrence Berkeley National Laboratory/OU=ICSD/CN=IDCG-CA </SubjectCA>
 </UseConditionCert>
 </SignablePart>
</AkentiCertificate>
```

Figure 2. UseCondition Certificate

Resources controlled by Akenti authorization may be grouped into a *resource realm.* A resource realm can be organized as a flat structure of resources such as instruments or compute platforms, or a hierarchical structure such as a file system or set of Web documents. Each resource realm has at least one Policy certificate which must be stored in a known and secure place. Normally it is on the same machine that controls access to the resource, but it could also be on the platform where the Akenti server is running, if they are different. Since a Policy certificate is centrally stored and may be administratively difficult to update there is a minimal amount of information in it. It contains information about the Certificate Authorities that are trusted to sign identity certificates, including a copy of their public keys and information about where they publish certificates and certificate revocation lists. It also lists the stakeholders (or stakeholder groups) for the resource and where they store the Use-condition certificates that they issue. It may optionally store URLs in which to search for Attribute certificates.

In the case of hierarchical resources, there must be at least one Policy certificate at the top of the tree (sometimes referred to as the root policy). Then there may be a Policy certificate at any level where there are new stakeholders, or restrictions on the allowed CAs. Levels without their own Policy certificates inherit policy from higher levels. Policy certificates are signed by one of the stakeholders listed in the certificate, making them self-signed certificates. As such they must be uploaded by a trusted method and kept in a secure location.

Each stakeholder group for a resource must create at least one and possibly more Use-condition certificates for its resource. A Use-condition certificate consists of a constraint which is a relational expression of the attributes a user must have to get a certain set of rights with respect to the resource. Components of the X.509 distinguished name can be used such as CN=Mary R. Thompson, or O=Diesel Combustion Collaboratory, or values of attributes that are defined in the context of the resource. For example, role = researcher or group = accounting. These attribute requirements can be combined with the boolean operators && or ||. It is also possible to specify real-time or system parameters such as time<=5PM && time>=9AM, or system_load < 2. If Akenti is unable to evaluate such system parameters it may pass them back to the resource gateway for evaluation. An attribute authority (consisting of an issuer and its CA) is specified as the signing authority for each attribute-value pair. Thus the stakeholder for a resource must specify who is trusted to attest to the attributes that are required.

The Policy certificate contains URLs to search for each stakeholder group's Use-condition certificates. A stakeholder may put Use-condition certificates in more than one place for reliability, but each directory must contain the complete set. Since Use-conditions restrict access to a resource, it is essential that either all or none of them are found. If no Use-conditions are found for a stakeholder group, all access to the resource is denied. This is not the case with Attribute certificates since they only serve to increase access. Thus a missing Attribute certificate may limit or deny a user's access, but will never allow an access that should be denied.

Attribute certificates contain an attribute-value pair and the subject name and issuer to whom it applies. They are signed by attribute authorities that have been specified in a Use-condition certificate. Attributes can apply to more than one resource, although they are likely to be applicable in only a single resource realm.

Creating policy

Since policy is contained in signed XML certificates which are interdependent, a stakeholder needs some tools to assist in their creation. A stakeholder starts by creating the root Policy certificate for the resource realm. The X.509 certificates of all the trusted CAs must be available from a trusted source and are placed in the root Policy certificate. This certificate also contains the URLs of the locations where these CAs publish the certificates that they issue and their certificate revocation lists. The first stakeholder must decide if other stakeholders for the resource are to be allowed and, if so, include their DNs and CAs in the root Policy certificate. In a hierarchical set of resources, only the top level stakeholders need to be known initially. They in turn, can delegate control to other stakeholders for resources lower in the hierarchy.

Akenti certificates can be created either by using a command line tool to sign an XML input certificate, or by a GUI program that steps a stakeholder though a menu of choices for each field in the certificate. The GUI program is supported by a Resource Definition Server running on the resource host which in turn reads a Resource Definition File and any existing Policy files to find stakeholder names, acceptable attributes and actions for a resource realm. The command line method is fine for very simple policy, and for the root Policy certificate, but as soon as the policy becomes hierarchical, or there are many stakeholders, the GUI interfaces which prompt the stakeholder with acceptable choices become preferable. The Resource Definitions File is only used to provide suggestions to the policy creation GUIs. It includes the names of the CAs, and their publishing directories,

principals that are acceptable for issuing specific attribute and values, and a list of actions that are relevant to the resource realm. Information that is used at access decision making time, such as the certificates of the CAs, must be stored in the root Policy certificate, since it is a signed document. In summary the two methods of getting started are:

- Create an XML version of a root Policy certificate, following one of the templates provided by the Akenti distribution, sign it using CertGen with the stakeholder's private key contained in a pkcs12 format file, and store it in the resource tree
- Create a Resource Definition File, start the Resource Definition Server, and then use the GUI, *PolicyCert.sh* to create, sign and store a Policy certificate.

The stakeholder must now create at least one Use-condition certificate for the resource. Anyone can create a Use-condition certificate, but it will only be used during the access control decision if it is issued and signed by one of the stakeholders currently listed in the resource's Policy certificate. As in the case of the Policy certificate, a Use-condition certificate can either be created by inputting an XML version of the certificate and private key to CertGen or can be generated and signed by a GUI program, *UseCondition.sh.* The GUI program uses the Policy certificate to determine the allowed stakeholders, and the Resource Definitions File to determine what attributes, values and actions have been defined for this resource realm. The stakeholder is led through a process to specify who he is, where his private key is, what resource the certificate applies to, what attributes and values are required, which attribute authorities should vouch for them, and what actions are to be granted. It also asks about such details as the length of time for which this certificate should be valid, the scope of the Use-condition (does it just apply to the one resource or to a hierarchy of resources), whether it is a critical Use-condition (it must be satisfied or the user gets no access to the resource even if he satisfies other Use-conditions). The Use-condition certificates must be stored in a directory that is specified in the Policy certificate.

Attribute certificates can also be created by either CertGen or a GUI program *Attribute.sh*. Attribute certificates are actually independent of a particular resource, but the GUI program will look at the Resource Definitions File associated with a particular resource to get a list of attribute names. Resource Policy certificates, and Use-condition certificates may specify where the Attribute certificates should be stored.

Once a set of Policy, Use-condition and Attribute certificates have been stored, the stakeholder can use a Web based interface to see what access they provide. The Resource Definition Server will execute the required CGI script.

Checking access

The Akenti authorization service can be called in several ways: It can be invoked as a function call by a gatekeeper program and thus run as part of the gatekeeper. It can be contacted as a server through an insecure protocol such a TCP. If the akenti server is running on the gatekeeper host, it can return the rights as a simple string. If it is running on another host, it can return a signed certificate. The gatekeeper process must have a copy of the Akenti servers's public key and verify the certificate, before it can trust the information. Or the Akenti server can be contacted as a server through a secure protocol such as SSL and the protocol will do the authentication of the Akenti server and encrypt the returned access string. Akenti returns an authorization answer in one of two ways: a list of strings representing unconditional actions; or a signed capability certificate which may include both conditional and unconditional rights. Conditional rights are rights that may have some conditions attached that only the gatekeeper can evaluate, such as current machine load, disk availability or the state of some related systems.

As has been mentioned previously, the Akenti policy engine finds all the Use-conditions by searching in the URLs specified in the Policy certificates and verifying the issuer and signature on each certificate. If a Use-condition certificate cannot be found for each stakeholder group, access to the resource is denied. Attribute certificates are searched by following URLs in either the Policy certificates and/or Use-conditions. Again, the issuer and signature of each certificate is verified. This signature verification requires that the Akenti policy engine be able to find the X.509 certificates for each issuer. If the CAs who issue certificates publish them in an LDAP server, Akenti will look there. Otherwise there must be some setup actions taken to put all the expected certificate issuers' X.509 certificates in a file system or a web browser where they can be found.

Mod_akenti module for Apache web server

Web-controlled sets of documents and services have rapidly grown from collections of read-only documents that are centrally administered to a vast array of remotely managed documents and services. In the scientific community such Web based systems have become known as portals, and are increasingly used to

provide a common interface to static documents, to allow shared authoring of documents, to allow access to legacy data bases, to allow execution of codes on shared server machines, and practically anything else an inventive scientist can think of. Authorization to perform such access is usually implemented by the http *Basic Authentication* mechanisms, (e.g. user/password or domain based) or by ad-hoc scripts based on the username. These passwords are passed across the internet in clear text and are thus deemed insecure.

In order to make Akenti authorization available for the widest range of distributed resources, we wanted to make it available to Web-accessed resources. There were several ways to accomplish this: referencing resources through CGI scripts that called Akenti, referencing resources through Java servlets or JSPs that called Akenti, or building Akenti authorization into a Web server. The first two methods, involve an indirection between the request and response which is both less efficient and requires more complicated URLs to refer to documents. Since the Apache Web server makes it straightforward to include new functionality, we decided to build a Akenti module for Apache.

The Apache [2] web server is a widely-used, high-performance freeware server which is built around an API [30] which allows third-party programmers to add new server functionality. Indeed, most of the server's visible features (logging, authentication, access control, CGI, and so forth) are implemented as one of several modules, using the same extension API available to third parties. The modules can be statically or dynamically linked to the server. [33]

How apache modules work

Apache divides the handling of requests into different phases:

- URI to file name translation
- Authentication and access checking
- Determining the MIME type of the requested entity
- Returning data to the client
- Logging the request

Each module can contribute to any of these phases. For each phase, a module can completely replace an existing module or can be added to a list of existing modules. The list of modules acts as a queue in which control is passed from one module to another. Each module can return one of three values: OK, DECLINE and FORBID. If a module returns OK, then the server passes the request on to other modules in the queue. A module returns DECLINE when it wishes to ignore a specific request. A FORBID return causes the server to forbid access to the resource requested. The FORBID return veto's other modules replies. Each module can declare a set of handlers to handle specific types of URI requests. The interface between the server core and the extension modules is through a module structure which consists of vector of callback routines. A module provides a callback for each phase that it wishes to handle and NULL for the rest. The module structure for Apache 1.3.x provides the option of defining one or more of the following callback routines.

module MODULE_NAME = {

STANDARD_MODULE_STUFF,
<module initializer routine>,
<per-directory config creator routine>,
<merge routine for directory config>,
<server config creator routine>,
<server config merge routine>,
<command table for defining directives>,
<list of handlers to handle specific requests>,
<filename-to-URI translation routine>,
<check/validate user_id routine>,
*<check user_id is valid *here* routine>,*
<check access routine>,
<MIME type checker/setter routine>,
<module specific fixup of header fields routine>,
<module specific logging activities routine>,
<header parser routine>,
<process initializerroutine>,
< process exit/cleanup routine>,
<post read_request handling routine>
};

Apache allows each module to read directives from the configuration file by specifying a command table structure. The entries in the command table include the name of the command, a pointer to the command handler, an argument which is passed to the command handler, items which tell the server core code where the command may appear (RSRC_CONF), what sort of arguments it takes (TAKE2 means two string arguments), and a description of what arguments should be supplied, in case of syntax errors.

There are three major classes of directives that can be defined in Apache. First Global directives which can occur inside server config files but must be outside virtual host sections. The second class is per-server directives which occur within the context of server config and the virtual host sections. The third class is the per-directory directives which can pretty much occur anywhere

(server config, virtual host, directory,.htaccess). These three classes are subsets of each other.

How mod_akenti works

Mod_akenti is an Apache module that provides Akenti authorization capabilities for the Apache web server. Mod_akenti is implemented as a Dynamic Shared Object module which can be loaded into the server at start-up or restart time. It currently works in Apache 1.3.x. Mod_akenti does not define any handlers as it serves as an access control mechanism for all requests to the web server unless otherwise specified.

Mod_akenti defines two global directives inside the server configuration, and defines a check access callback. So its interface consists of a call for per-directory configuration, a command table, and a callback for the check access routine.

The two Akenti directives are: AkentiConf, which supplies the name of the configuration file used to configure Akenti policy engine; and AkentiResources, which is used to specify what part of the document tree should be controlled by Akenti. The second directive is of interest as it allows other authorization mechanisms to coexist with that of mod_akenti. It accepts a set of resource names to be controlled, or 'ALL' to control the whole hierarchy or an empty argument to control none of the resources.

Configuration and installation

Mod_akenti is a C++ module, while the core Apache server is written in C. Hence the shared object standard C++ library (ex. libstdc++.so) must be linked at server start-up. This is done through the LoadFile command in httpd.conf. The other shared object libraries can be either in LD_LIBRARY_PATH or defined in the httpd.conf similar to standard c++ library. The Akenti module requires a secure Apache web server (Apache + mod_ssl., which in turn requires that the server be built with the Extended API), the OpenSSL libraries (an open source toolkit that implements SSL and TLS as well as general cryptography), the OpenLDAP libraries (open source library for LDAP suite of applications) and the Akenti suite of libraries. A special program apxs (APache eXtenSion) is used to insert mod_akenti into the web server before start-up. The mod_akenti distribution package [23] provides detailed information about how to build and configure the Akenti module.

Web authentication and authorization methods

Standard Web authentication and access control is based either on the domain in which the request originated, or something called *Basic Authentication* [15] where the user provides a user name and password which the Web browser matches against user information stored on the server machine. There are many authentication modules for Apache based on this mechanism [3]. Mod_auth is the basic module that matches a user and password with an entry in Web specific password and group files. Modules such as mod_auth_dbm and mod_auth_db provide greater scalability by looking up users in a data base. There are also modules available for authenticating users in ldap directories, Oracle, and msql data-bases, and Kerberos users. In all of these schemes the user name and password is passed over the network in plain text. There is one other form of user authentication which is not supported by many browsers called *Digest Authentication* which is implemented by mod_auth_digest. This protocol has the server send a nonce to the browser who then returns an MD5 hash of the nonce, the user name, password, http request and the URI. Thus the password is not sent in the clear.

Mod_ssl [21] which uses X.509 certificates to create encrypted channels between the browser and the server adds a whole new dimension to authentication and authorization. In the typical commercial use of SSL only the server is required to have an identity certificate and private key. This key is used to establish encrypted communication between the browser and server over which passwords can be passed securely. However, SSL can run in a mode that requires the browser to have a certificate and private key for the client. When this mode is used mod_ssl can provide access control based on the client certificate.

The mod_ssl directive SSLVerifiyClient can hold one of the three possible values: none, optional and require. If it is set to require, the browser must provide a certificate that identifies the user. If it is set to optional, the browser will look for a user certificate, but if none exists will attempt the access anyway. If it is set to none, no user certificate is sent.

Once mod_ssl has a client certificate, it provides several more types of access control. It can implement a *FakeBasicAuth* option where it uses the subject of the client's X.509 certificate as a user name, but no password needs to be obtained from the user. It also provides a directive called SSLRequire (see Figure 3.) which specifies constraints which need to be fulfilled in order to allow access. The requirement specification is an arbitrarily

complex boolean expression containing any number of access checks. The variables used in the expression include all the standard CGI/1.0 and Apache variables, plus a large number of variables defined by mod_ssl that refer to parts of both the server and client certificates: e.g. client subject's DN, the client issuer's DN and most components of the client's certificate. The syntax also allows an expression to be used from an arbitrary file. This method is used to match portions of distinguished name compared to the FakeBasicAuth where the whole DN is used.

While the SSLRequire directive is very powerful its main limitation is that the constraints are specified as part of server's configuration file. If many resources need to be controlled, the server configuration will expand to the point where it becomes difficult to manage. In distributed environments where policies for resource access are managed by multiple owners, a centralized access control list does not scale well. For example, WebDAV [16] has been implemented as Apache module, mod_dav, which allows extensions to HTTP protocol in order to provide a shared file system.

If several projects need to be managed by one server, there should be a a way to limit the writing of access policy for a set of resources to the project manager. But since all the policy is in one file, this is not possible.

Mod_akenti, on the other hand, stores all of its policy information outside of the Web server configuration file. The only information in the configuration file is the name of the resources which mod_akenti wishes to control and a pointer to Akenti's own configuration file. The Akenti configuration file points to where the root Policy certificate for each resource tree is. Akenti policy defines who the resource owners are and allows resource owners to express use-conditions on each resource. The use-conditions are signed and stored in a distributed fashion at the owner's convenience. The variables used in the use-conditions are defined by the stakeholder, rather than the Web server. Thus the same access policy can be used for resource referenced via the Web or by another remote method. At run-time Akenti collects all the use-conditions applicable for a certain resource in order to make access decisions. Akenti caches certificates in order to reduce search time. It also caches a Capability which has the access rights of a user for a resource, so that subsequent requests for the same resource require no decision making.

```
<Directory /foo>
   SSLRequireSSL
   SSLRequire %{SSL_CLIENT_S_DN_O} eq
                "LBNL" and
          %{SSL_CLIENT_S_DN_OU}
                in {"DSD", "ICSD", "NERSC"}
```

Figure 3 Example of SSLRequire

mod_akenti could also be used to provide access control for mod_dav which currently uses basic authorization provided by Apache. In this case, the use-conditions have to be specified for WebDAV methods (MOVE, COPY, PROPFIND, DELETE etc.). In addition, a few additional directives are required for mod_akenti inside the per-directory configuration.

Related Work

Policy representations

While there has been a great deal of work in formulating use requirements and standards for authorization protocols or data structures, no single standard has emerged. There is an IETF proposed Attribute Certificate profile [12] to carry attributes associated with an X.509 identity certificate. While the contents and purpose of this certificate are basically the same as an Akenti Attribute certificate, we chose not to use it in our implementation because it is difficult for users and applications to deal with ASN.1 structures. A major goal of Akenti was to make the policy as easy to understand as possible, so using ASCII files to represent policy and principals names consisting of a CA's DN and the user's DN was preferable to using a an ASN.1 structure that identified the holder as a CA and serial number. To understand the meaning of such a certificate, requires a program to print the contents in a readable form, and the ability to find the holder's X.509 certificate and extract the subject name.

KeyNote [5] is a trust management system, which provides a simple language for describing and implementing security policies, trust relationships, and digitally-signed credentials. The KeyNote defines *principal* as any convenient string which may include a cryptographic public key. Authorization policy is contained in *assertions* which consist of a sequence of fields. Each field is represented by a keyword and value. A *credential* asserts some attribute about a principal and is signed by a trusted authority. Both assertions and credentials are represented by the same keyword policy language. Akenti and KeyNote both provide a function call API for compliance-checking for a resource gatekeeper to call when making an access decision. Both systems return list of trusted actions. KeyNote is less tied to one form of authentication than Akenti. A KeyNote principal may be represented by a cryptographic key, or it may

just be an opaque string. They deliberately did not require X.509 certificates in order to separate the issues of secure naming and authorization. While this removes the need for maintaining a PKI, it means that the principals named in the authorization policy may be opaque making it harder for a stakeholder to read and evaluate the policy of a resource.

The mechanisms for creating and storing policy assertions and storing and marshaling certificates are left up to the installer of a KeyNote system. In contrast, one of the emphases of the Akenti system is to support remote creation and storage of policy certificates. It thus provides several tools to help in their creation and signing, while the policy engine supports gathering certificates from file systems, LDAP servers or Web servers. Other systems rely on the user being able to edit policy files on the resource gateway machine which does not meet our goal of accommodating distributed stakeholders.

In our original implementation of Akenti, we chose a simple keyword language for our certificates similar to that used by KeyNote. Eventually, expressing the constraints and trust relationships for all the attributes became increasingly awkward, with too much information being implicit in the ordering of fields or in relationships between fields. For our second implementation we switched to XML for greater flexibility and more precise definition of the semantics. We were also encouraged by the availability of XML parsing tools in a variety of languages and have made use of the Xerces parsers from the Apache XML Project [4]

A recent XML standard specification for security assertions named Security Assertion Markup Language (SAML) [17] has been published by the OASIS [29] consortium. This standard defines both XML protocols and assertion structures. Assertions come in three types: Authentication: the specified subject was authenticated by a particular means at a particular time; Authorization Decision: a request to allow the specified subject to access the specified resource has been granted or denied; Attribute: the specified subject is associated with the supplied attributes. Since Akenti is only supporting X.509 authentication, it does not need a general purpose Authentication structure. It just uses the X.509 certificate (or chain of certificates if delegation is involved) and assumes that the resource gateway has authenticated the certificate. Akenti will check for revocation, since the current implementations of SSL do not do this. The capability certificate returned by the Akenti server differs from the Authorization Decision assertion in that it does not contain the reasons (evidence) of why it made the decision, but may contain unresolved conditions on the actions, so that the gatekeeper can do further checks. Again the attribute assertion/certificate covers has the same purpose as the PKIX Attribute Certificate and the Akenti Attribute certificate: namely, a subject name, an associated attribute-value pair and the authority that attests to this. The SAML standards seem to be focused on letting various peers report security decisions. The focus in Akenti, is more on gathering and interpreting of policy (Use-condition) statements about the resource. The only real communication is the authorization request and reply between the resource gateway and the Akenti server.

Authorization models

The authorization model used by KeyNote is essentially the same as Akenti uses. A principal makes a request to a resource gateway, handing it an identity credential that can be authenticated. In Akenti this is normally just an X.509 certificate, while KeyNote supports other types of credentials. Then the gateway server makes an authorization request to the authorizer, e.g. Akenti or KeyNote. The current implementation of KeyNote only supports function calls, where Akenti will support function or server calls. The authorizer returns a list of allowed actions to the gate keeper for its interpretation or in the case of Akenti being called as a server, it returns a capability certificate signed by Akenti.

Shibboleth [11] is a cross-institutional authentication and authorization service for access control to Web-accessed resources. It is being specified by the InterNet2 middleware architecture committee. It has many of the standard goals of distributed authorization with one additional twist. It wants to be able to grant access to a user who can still maintain anonymity at the resource site. The major motivation for this goal is access to library materials by academics. Their authorization model entails a user making a request to a web server and providing a identity handle back to his home institution. The Web server then asks that institution for attributes about the user. It then checks those attributes against its local policy to allow or deny access. The user need only authenticate to his host site and may use whatever type of credentials that site recognizes. One difference between this trust model and that used by Akenti, is that in Akenti, the resource provider specifies a limited number of trusted authorities that it will accept for authenticating users and attributes. In the Shibboleth case, each member institute must trust all the sites at which any of its user's reside. So for a user to get access to a remote resource, its whole site must be trusted.

While in a more traditional PKI environment, a user only needs to get a credential for himself from an authority that the resource site trusts.

The Community Authorization Server (CAS) [28] is a new authorization service being developed by the Globus Project [13] for Grid environments. Their authorization model allows a resource site to grant a community access to resources and the authorization server for that community to grant access to the community members. This is implemented by having the user go to the CAS server and get a delegated proxy certificate [31] with the CAS server's identity, which includes a rights restriction extension that limits what resources can be accessed. The resource gatekeeper must interpret the restricted rights extension and verify that the community has such rights to the resource. Since the delegated proxy is a short-lived X.509 identity certificate it gets passed between the user and the resource gateway as part of the SSL connection. There is no additional information that needs to conveyed, as is the case when a user needs to hand attribute certificates to the gatekeeper. CAS differs from Akenti in that the examination of policy and granting of rights is done before the gatekeeper is contacted. This means the user must ask for all the rights he will need in advance of referencing the resource. In Akenti, all the gathering and checking of policy is done after the call to the gatekeeper to perform a certain action. Akenti does cache the rights that the user was granted, to deal with the common case of several calls in rapid succession for resources in the same realm.

Policy about resources is stored and managed by the CAS servers and so far mainly consists of lists of objects and allowed rights. This information is included in the rights restriction extension of the delegated proxy. The intent of the CAS project is to extend the policy language as the need arises. The CAS administrator is responsible for adding each community member to the appropriate groups. The CAS administrator may also delegate administration of subsets of the objects to additional people. In contrast, in Akenti, a new user would need to contact the stakeholder for the resource to be added to the policy files.

Conclusions

Akenti is an authorization service that uses authenticated X.509 identity certificates and distributed digitally signed authorization policy certificates to make access decisions about distributed resources. It supports authorization decisions based either on policy that is gathered by the resource gatekeeper, or on a rights-granting capability presented by the user. It supports Globus proxy identity certificates, and could be easily extended to handle restricted delegation credentials. We have implemented an Apache Web server module which allows the same authorization policy to be used to control access to Web accessed resources as well as resources accessed by other remote methods. Thus all the resources in a portal can use the same authorization mechanism.

Akenti differs from most of the other work that we have surveyed in the emphasis on using easily read policy statements that are independently created and signed by multiple stakeholders. This policy can be stored on the resource host or locally to the stakeholder and be gathered and evaluated by the trusted authorization server at the time of resource access. The Akenti distribution also includes several tools for displaying the combined authorization policy for a given resource and for tracking the steps in a user's authorization or rejection.

It has been used as part of the Diesel Combustion Collaboratory [26] to control access to Web-based documents and remote execution and is now being integrated with the Globus job manager to control access to legacy applications in the National Fusion Grid [19].

The code is freely available as C++ source code, or Linux and Solaris executables. (http://www-itg.lbl.gov/Akenti)

Acknowledgments

The original idea for Akenti came from William Johnston. Case Larsen did a large part of the original implementation. Maria Kulick, Guillaume Farret and Xiang Sun have also contributed to the current implementation.

Appendix A: XML definition for the Akenti policy language

```
<?xml version="1.0" encoding="US-ASCII"?>
<!-- This DTD is intended to define all the Akenti Policy elements:
    Policy Certificates, UseCondition Certificates, Attribute Certificates,
    Capability/Authorization Certificates, and Cache Certificates
 -->
<!-- Note: one or more (+), zero or more (*), or zero or one times (?)-->

<!ELEMENT AkentiCertificate (SignablePart, Signature)>

<!ELEMENT SignablePart ( Header, (PolicyCert  |  UseConditionCert  |  AttributeCert  |  CapabilityCert )  )>

<!ELEMENT Header ( Version, ID, Issuer, ValidityPeriod) >
 <!ATTLIST Header
          Type  (attributeCertificate  | cacheCertificate  |capabilityCertificate | policyCertificate | useCondCertificate ) #REQUIRED
          SignatureDigestAlg  (RSA-MD5 | RSA-SHA1 | DSA-MD5 ) #REQUIRED
          CanonAlg  (AkentiV1) #REQUIRED >


<!ELEMENT PolicyCert  ( ResourceName, CAInfo*, UseCondIssuerGroup+, AttrDirs*, CacheTime )>
<--
          ResourceName      Name of the resouce to which this policy applies
          CAInfo            The DN and X509 identity certificates of all the CAs we will trust.
                                    May include pointers  places where it publishes CRL's and identity certificates
          UseCondIssuerGroups Stakeholders and their Certificate directories
                                    At least one UseCondCert must be found from
                                    each group.
          AttrDirs          optional list of URLs in which to search for Attribute certificates
          CacheTime         Maximum time in seconds that certificates relevant   to this resource may be cached
-->
<!ELEMENT UseConditionCert ( ResourceName, Condition, Rights, SubjectCA*)>
<!--
          ResourceName      name of the resource to which the useCondition applies
          Condition         A boolean expression stating what attributes a user needs to satisfy the UseCondition and what users
                            and CAs are trusted to attest to specified attribute
          Rights            An opaque list of actions known to the stakeholder and the resource gateway
 -->
    <!ATTLIST UseConditionCert
          enable (true | false) #REQUIRED >
<!--
          scope     if sub-tree the UseCondCertificate applies to all the resources that are in the sub-tree named by the resource
                    if local, it applies just to the one resource named
          enable    if true, this UseCondition must be satisfied by anyone wanting to use the resource, if false it need not be satisfied
                    if a user satisfies other UseConditions.
-->
<!ELEMENT AttributeCert ( SubjectAndCA, AttrName, AttrValue, Condition*)>
<!--
          SubjectAndCA      Subject to which this attribute applies
          AttrName          name of attribute
          AttrValue         value of attribute
          Condition         An optional Constraint that is placed on how or when the attribute should apply
-->
<!ELEMENT CapabilityCert ( ResourceName, SubjectAndCA, Actions*, ConditionalActions*)>
<!--
          ResourceName      name of the resource to which the rights apply
          SubjectAndCA      user who has the rights
          UnConditionalActionsthe actions that have been authorized
          ConditionalActions  actions that still have some unevaluated constraints.
  -->
<!ELEMENT ConditionalActions ( Condition, Actions )>
  <!ATTLIST ConditionalActions
          critical (true | false) #REQUIRED   >
<!--
          Condition         Constraint that is placed on how or when the attribute should apply
```

```
        Actions             The access rights that are allowed if the condition  is true
        Critical            If this is false, the Condition must evaluate to  true, or even the UnConditionalActions do not apply
-->

<!ELEMENT CAInfo (CADN, X509Certificate+, IdDirs*, CRLDirs*)>
<!--
        CADN                the distinguished name of the CA
        X509Certificate     A chain of the X509 identity certificates of the CA,  includes its public key.
        IdDirs              an optional list of directories in which the CA stores the certs it issues
        CRLDirs         a list of 0 or more URLs to directory services  in which to search for certificate revocation lists
-->

<!ELEMENT Condition ( Constraint, AttributeInfo+)>
<!--    A Condition contains a boolean expression stating what attributes a user needs to satisfy the UseCondition and

        what users and CA  are trusted to attest to what attribute/value pairs.
-->
<!ELEMENT CRLDirs (URL+)><!-- list of 0 or more URLs to directory services   in which to search for certificate revocation lists-->
<!ELEMENT AttrDirs (URL+)> <!-- AttrDirs list of 0 or more URLs to directory services in which to search for attribute certificates.-->
<!ELEMENT IdDirs (URL+)>    <!-- list of 0 or more URLs to a directory services  in which to search for identity certificates.-->
<!ELEMENT UseCondIssuerGroup (Principal+,URL+)>  <!-- group of stakeholder and their certificate directories. -->
<!ELEMENT AttributeInfo (AttrName, AttrValue, (CADN | Principal), AttrDirs*, ExtArgs*) >
   <!ATTLIST AttributeInfo type (STANDARD | X509 | AKENTI | EXT_AUTH ) #REQUIRED>
<!--
        STANDARD            attributes if they are evaluated by some system call
        X509                attributes if they are part of an X509 Identity certificate,  e.g. O, OU, CN;
        AKENTI              attributes if there is an Attribute certificate to attest to a user's possession of the attribute
        EXT_AUTH            if some external authority is called to evaluate them

        AttrName            name of attribute used in constraint
        AttrValue           name of value required by constraint
        CADN                name of CA that issues the identity certificate that contains the x509 attribute we need.
        Principal           the name of the attribute issuer and CA for Akenti attr
                            or the name of an external authority that can evaluate  an attribute
        AttrDirs            an optional list of directories in which to search  for Attribute Certificates
        ExtArgs             optional  list of arguments that may be handed to an external authority.
-->

<!ELEMENT ValidityPeriod EMPTY><!-- Beginning and End date in UCTime of when the certificate is valid -->
  <!ATTLIST ValidityPeriod
     start CDATA #REQUIRED
     end CDATA #REQUIRED
  >
<!ELEMENT ExtArgs (String+)>
<!ELEMENT ID EMPTY>  <!--A unique ID assigned to every certificate when it is created   -->
        <!ATTLIST ID id CDATA #REQUIRED   >
<!ELEMENT Version EMPTY>  <!--  Certificate format version -->
        <!ATTLIST Version ver CDATA #REQUIRED    >
<!ELEMENT Issuer (UserDN,CADN,URL* )>
<!ELEMENT Principal (UserDN,CADN )>
<!ELEMENT SubjectAndCA (UserDN,CADN)>
<!ELEMENT URL (#PCDATA)>  <!-- protocol, host, port and file name -->
<!ELEMENT CADN (#PCDATA)>
<!ELEMENT SubjectCA (#PCDATA)>
<!ELEMENT X509Certificate (#PCDATA)>
<!ELEMENT UserDN (#PCDATA)>
<!ELEMENT ResourceName (#PCDATA)>
```

References

1. D. A.Agarwal, S. R. Sachs, W.E.Johnston *The Reality of Collaboratories* Computer Physics Communications, 1998, vol. 110, p. 134-141
2. Apache Software Foundation http://www.apache.org
3. Apache Module Registry, http://modules.apache.org/
4. Apache XML Project; http://xml.apache.org/
5. M. Blaze, J. Feigenbaum, J. Ioannidis, A. Keromytis. *The KeyNote Trust Management System, Version 2. RFC-2704.* IETF,September 1999. http://www.crypto.com/papers/rfc2704.txt
6. N. Damianou, N. Dulay, E. Lupu, M Sloman,: *The Ponder Specification Language Workshop on Policies for Distributed Systems and Networks* (Policy2001), HP Labs Bristol, 29-31 Jan 2001
7. T. Dierks, C. Allen, *The TLS Protocol, Version 1* IETF RFC 2246; http://www.ietf.org/rfc/rfc2246.txt
8. Diesel Combustion Collaboratory (DCC), http://www-collab.ca.sandia.gov/dcc/
9. DisCom^2, http://www.llnl.gov/asci/discom/
10. C. Ellison *SPKI Requirements*, IETF RFC 2692 1999, http://www.ietf.org/rfc/rfc2692.txt
11. M. Erdos, S. Cantor, *Shibboleth-Architecture DRAFT v04*, http://middleware.internet2.edu/shibboleth/docs/raft-internet2-shibboleth-architecture-04.pdf
12. S. Farrell, R. Housley, *An Internet Attribute Certificate Profile for Authorization*, <draft-ietf-pkix-ac509prof-09.txt>, June, 2001 http://www.ietf.org/internet-drafts/draft-ietf-pkix-ac509prof-09.txt
13. I. Foster, C. Kesselman, eds. *The Grid: Blueprint for a New Computing Infrastructure*, 1999, Morgan Kaufmann
14. I. Foster, C. Kesselman, S. Tuecke, T*he Anatomy of the Grid: Enabling Scalable Virtual Organizations.* International J. Supercomputer Applications, 15(3), 2001. http://www.globus.org/
15. J. Franks, et.al. *HTTP Authentication: Basic and Digest Access Authentication*, IETF RFC 2617, http://www1.ics.uci.edu/pub/ietf/http/rfc2617.txt
16. Y. Goland, et al., *HTTP Extensions for Distributed Authoring* -- WEBDAV, IETF RFC2518 http://www.ietf.org/rfc/rfc2518.txt
17. P. Hallam-Baker, E. Maler, eds. Assertions and Protocol for the OASIS Security Assertion Markup Language (SAML),draft-sat-core-25, http://www.oasis-open.org/committees/security/docs
18. R. Housley, W. Polk, W. Ford, D. Solo, *Internet X.509 Public Key Infrastructure Certificate and CRL Profile* <draft-ietf-pkix-new-part1-12.txt>, http://www.ietf.org/internet-drafts/draft-ietf-pkix-new-part1-12.txt
19. K. Keahey, et al., *Computational Grids in Action: The National Fusion Collaboratory*, submitted to Future Generation Computer System, 2001., http://www.fusiongrid.org
20. Launch Pad, Portal to the IPG, http://www.ipg.nasa.gov/launchpad/servlet/launchpad
21. modssl, http://www.modssl.org/
22. J. Myers, *Simple Authentication and Security Layer (SASL)*, IETF RFC 2222, 1997, http://www.ietf.org/rfc/rfc2222.txt
23. S. Mudumbai, *mod_akenti: Akenti Access Control module for Apache* http://www-itg.lbl.gov/Akenti/docs/mod_akenti.html
24. NASA's Information Power Grid, http://www.ipg.nasa.gov/
25. National Fusion Grid, http://www.fusiongrid.org/
26. C. Pancerella, L. Rahn, C. Yang, *The Diesel Combustion Collaboratory: Combustion Researchers Collaborating over the Internet*, Proceedings of ACM/IEEE SC99 Conference, November 13-19, 1999. Portland, Oregon, USA, http://www-collab.ca.sandia.gov/dcc/
27. Particle Physics Data Grid (PPDG), http://www.ppdg.net/
28. L Pearlman, V. Welch, I. Foster, C. Kesselman, S. Tuecke. *A Community Authorization Service for Group Collaboration.* Submitted to IEEE 3rd International Workshop on Policies for Distributed Systems and Networks, 2001, http://www.globus.org/research/papers.html#CAS-2002.
29. Oasis, www.oasis-open.org
30. R. Thau , *Apache API notes*, http://modules.apache.org/doc/API.html
31. S. Tuecke, et al., *Internet X.509 Public Key Infrastructure Proxy Certificate Profile* , IETF draft, http://www.ietf.org/internet-drafts/draft-ietf-pkix-proxy-01.txt
32. M. Thompson, et.al., *Certificate-based Access Control for Widely Distributed Resources*, Proceedings of the Eighth Usenix Security Symposium, Aug. '99
33. Wainwright P. *Professional Apache*,Wrox 2001, http://www.apache.org/