Certificate-based Access Control for Widely Distributed Resources

Mary Thompson, William Johnston, Srilekha Mudumbai, Gary Hoo,
Keith Jackson, Abdelilah Essiari

Information and Computing Sciences Division
Ernest Orlando Lawrence Berkeley National Laboratory
Berkeley, CA, 94720
pkidev@george.lbl.gov

Abstract

We have implemented and deployed an access control mechanism that uses digitally-signed certificates to define and enforce an access policy for a set of distributed resources that have multiple, independent and geographically dispersed stakeholders. The stakeholders assert their access requirements in use-condition certificates and designate those trusted to attest to the corresponding user attributes. Users are identified by X.509 identity certificates. During a request to use a resource, a policy engine collects all the relevant certificates and decides if the user satisfies all the requirements. This paper describes the model, architecture and implementation of this system. It also includes some preliminary performance measurements and our plans for future development of the system.

1. Motivation: Distributed Computing Environments

In distributed computing environments such as research collaborations spanning several institutions, there may be independent and geographically dispersed individuals with authority to control access to the resources [18]. We wish to provide an automated system to allow these *stakeholders* to assert their authority over a resource in a flexible manner, consistent with the scope of their authority. Our immediate motivation is to enable sharing over open networks of valued resources within the scientific community generally, and for distributed collaboratories in the DOE2000 project [8] in particular.

The Department of Energy (DOE) supports a number of collaborative research environments in which people from universities and companies work with DOE national laboratory personnel and resources. The Lab resources consist of large scientific instruments such as electron microscopes or high-energy light sources; supercomputers or other high-end compute servers; and large-scale storage systems. Researchers from any of the groups may contribute software and data to be shared by other members of the group. The owners of software and data want to be able to securely store data or use their software on hardware that is owned by another entity. The owners of hardware resources want to be able to control their uses. These stakeholders may want to share with some, but not all members of the collaboratory. For example, commercial members may want to use common compute hardware, but share their data and results only with other members of their own organization.

Such an environment requires that the stakeholders be able to enforce access policy on their resources even when those resources are physically controlled by a different administrative domain. Each stakeholder must be willing to trust that the resource server will enforce access control, but the stakeholder should be able to flexibly specify access requirements for its resources.

Traditionally, stakeholders have relied on access control lists (ACLs), stored on the resource server, to express access policy. However, such ACLs typically require a central administrator to make all changes, which means both that the administrator must be trusted by all stakeholders and that the administrator is potentially a bottleneck to rapid updating of the policy. Also, ACLs usually require the server domain to maintain accounts and other administrative support for both stakeholders and users. These problems are all exacerbated when some or all stakeholders and users are administratively and geographically remote from the server.

Another problem that arises in distributed research environments is that there may be multiple principals from different administrative domains who need to have input to the access control policy for a single resource. The attempted execution of proprietary code (e.g., a large scientific modeling program) owned by a third party on a remote supercomputer is an example of such a multi-

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. This document is report LBNL-42928

ply-controlled resource. The owner of the supercomputer may want to restrict the amount of run-time allotted to a user, and the author of the code may want to specify who may run the code. Getting permission to run the code, therefore, requires satisfying two separately administered policy requirements.

Multiple layers of management may wish to impose independent restrictions on the use of a large scientific instrument. For example, top-level administrators may have general restrictions based on nationality or membership in an organization, safety officers may require special training, and the principal investigator may have his own set of requirements for the project which has scheduled time on the instrument.

In these scenarios, the resource (data, instrument, computational or storage capacity) has multiple stakeholders and each stakeholder will impose conditions for access, called *use-conditions,* on the resource. All of the use-conditions must be met simultaneously in order to satisfy the requirements for access. Further, it is common that the principals in these scientific collaborations are geographically distributed and multi-organizational. Therefore, without reliable, widely deployed, and easily administered access control mechanisms it will not be feasible to administer a secure collaborative environment. The access control mechanism must allow secure, distributed management of policy-based access to resources and provide transparent access for authorized users and strong exclusion of unauthorized users, in an operating environment where stakeholders, users, and system/resource administrators may never meet face to face.

2. Goal: Policy-based Access Control

We want our access control mechanism to support, in a computer-based working environment, the same sort of distributed authority over resources that is used in non-computer group endeavors. Each stakeholder should be able to make its assertions (as it does now by signing a policy statement) without reference to a mediator, and especially without reference to a centralized system administrator who must act on its behalf. The mechanism must be dynamic and easily used by all concerned – stakeholders and users – while maintaining strong assurances. The solution should scale with the number of stakeholders, resources and users.

Specifically, the access control mechanism should be able to collect all of the relevant assertions (identity, stakeholder use-conditions, and corresponding user attributes) and make an unambiguous access decision requiring an absolute minimum of centrally administered configuration information. Once the policy-based decision is made, the resource server should be able to ensure compliance both on the part of the intended users and unauthorized parties. The mechanism should also be based on, and evolve with, emerging, commercially supplied, public-key certificate infrastructure components.

3. Approach: Certificate-based Distributed Security

Our approach to policy-based access control in a distributed environment is based on digitally signed documents, or *certificates*, that convey identity, authorization, and attributes. A digital signature can assert document validity without the physical presence of the signer or physical possession of documents signed in the author's handwriting. The result is that the digitally signed documents that provide the assertions of the stakeholders about a resource, or assertions of trusted authorities about attributes of a user, may be generated, represented, used, and verified independent of time or location.

Users are authenticated by presenting an X.509 identity certificate [17] and proving that they know the associated private key. These certificates are issued by *certificate authorities (*CA) that verify the connection between a person or system component and possession of a public key / private key pair. Stakeholders create and digitally sign *use-condition certificates* that define conditions that must be satisfied by a user before being given access to a resource. *User attributes* are asserted by "authorities" that provide assured information as digitally signed *attribute certificates* [11]. Both use-condition and attribute certificates may be stored local to the issuer as long as they can be provided by a server when they are needed to determine permissions during an access request.

Components that enable the use of these certificates include reliable mechanisms for generating, distributing, and verifying the digitally signed documents; mechanisms that match use-conditions and attributes to decide if access should be allowed; and access methods that enforce policy for the specific resource based on the access control decision. All of these mechanisms rely on public-key cryptography for digital signatures, a public-key infrastructure for certificate management and a protocol for secure, authenticated communication, such as the Secure Sockets Layer protocol (SSL) ([25], [26]). (For a general introduction to public-key technology see [12] or [24].)

A frequently asked question is how is the PKI-certificate approach is better than the well established DCE/Kerberos access control system. Kerberos and DCE allow remote users secure access to centralized resources.

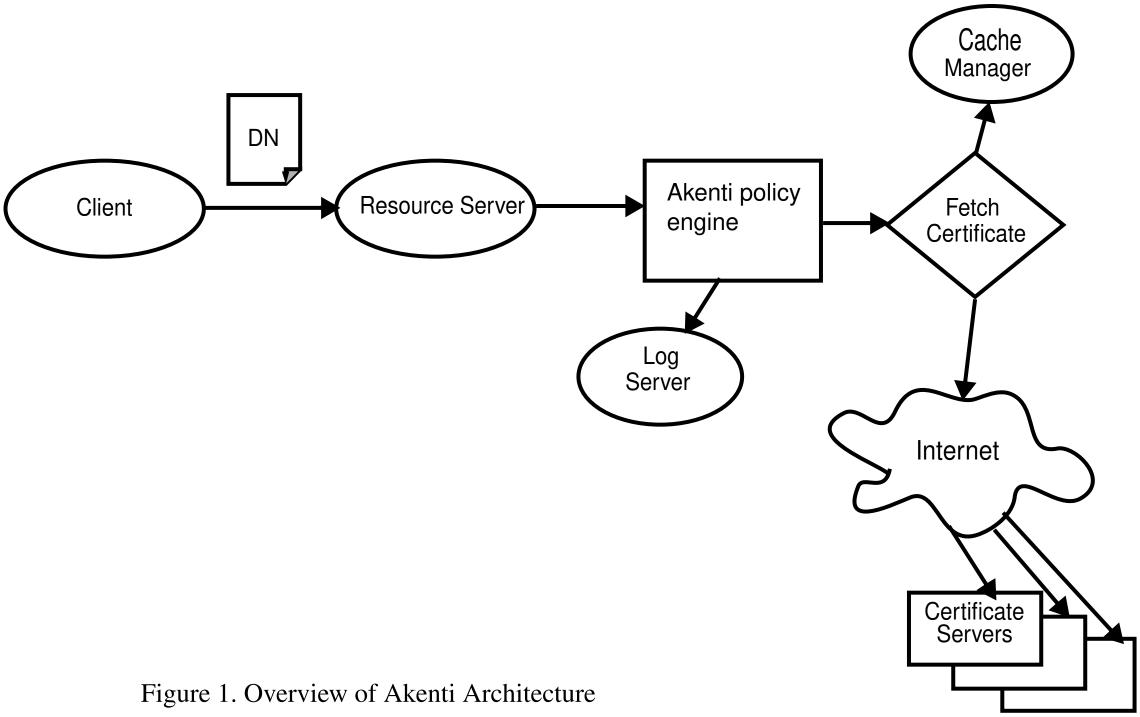


Figure 1. Overview of Akenti Architecture

Access by users from multiple administration domains can be addressed by establishing cross-realm trust. In the DCE environment the policy about a set of resources is defined by an ACL in the realm which controls the resource. Only user identities and group memberships are assigned by the other realms. Since the PKI approach allows pieces of the access control policy to be stored in distributed certificates, it enables distributed stakeholders to more easily control their resources. It is also our expectation that PKI identity certificates will become more widespread than Kerberos identities since many enterprises are beginning to use them for employee identification.

4. Architecture and Implementation

We are implementing a certificate-based access control system called Akenti [1], and initially deploying it in the DOE2000 Diesel Combustion Collaboratory [7]. Figure 1 shows the overview of the Akenti Architecture. The heart of this system is the *Akenti policy engine*, which gathers and verifies certificates and then evaluates the user's right to access to the requested resource based on these certificates. Figure 1 shows the major components of the run-time architecture. The *client* requests an operation on the resource and presents an identity certificate for authorization. The *resource server* authenticates the

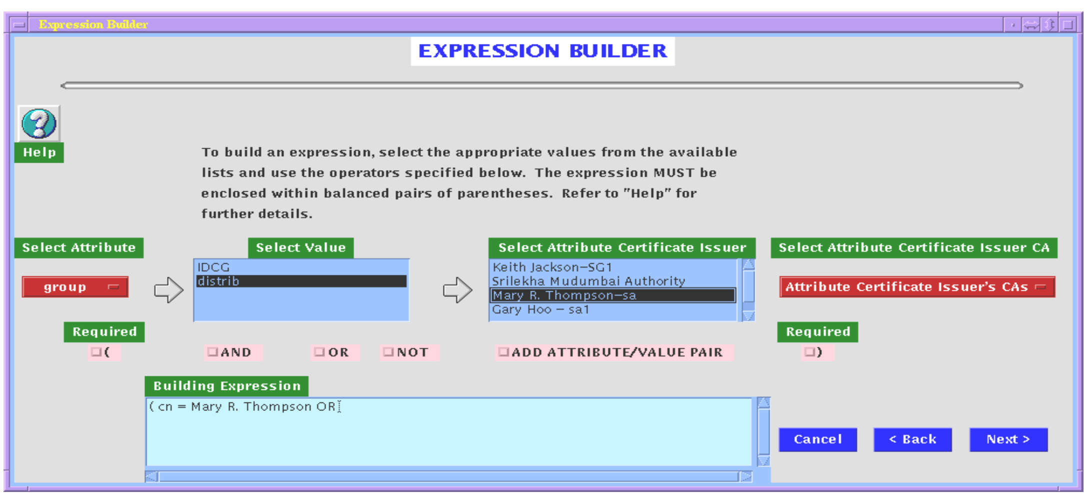


Figure 2. A screen from the use-condition certificate generator

certificate and then asks the Akenti *policy engine* for an access decision. Akenti checks with a *cache server* for possibly cached certificates and if that fails, searches *certificate directories* across the internet. Akenti also logs all of its actions with a *log server.* Once Akenti has all the necessary certificates, it returns its access control decision to the resource server. The resource server then acts on that decision to perform or deny the operation on the resource on behalf of the client.

One other essential component of the Akenti infrastructure is the set of tools that generate certificates, query the policy and display the run-time operation of the system.

Generating and Managing Certificates

Akenti uses three types of persistent certificates: X.509 user identity certificates, use-condition certificates, and attribute certificates. The identity certificates are generated and managed by certificate authorities, such as the Netscape CA server, Entrust server or Verisign. These CAs provide a Web interface that allows the creation or revocation of certificates; a directory server (usually an LDAP server) to provide the certificates for use by applications; and Web browser to manage the certificates for the user. CAs typically support a *certificate revocation list* (CRL) mechanism that can be queried when an application needs to verify a certificate. Our implementation uses the Netscape CA, which was the easiest to install and run in a research environment. It currently only checks with the directory server to verify that a certificate has not been revoked. Some minor code addition would be necessary to use a CA's CRL mechanism, rather than just relying on the directory server to provide only non-revoked certificates.

The format of the attribute and use-condition certificates is defined by Akenti and consists of a list of ASCII keyword and value tuples which are signed by the issuer. These values include a validity period and a unique ID for the certificate. We have written two Java applications to help the user generate and sign these certificates. An example of the use-condition certificate generator's interface for specifying a use-condition is shown in Figure 2. These applications know how to query the resource server and CA directory server to provide a menu of reasonable choices for the stakeholder or attribute issuer to use in creating a certificate. The generators use a configuration file to find the resource server and the user's signing keys. The resulting certificates can be stored in directories chosen by the user that are accessible via a Web server, an LDAP server, a file system, or an on-line MSQL database.

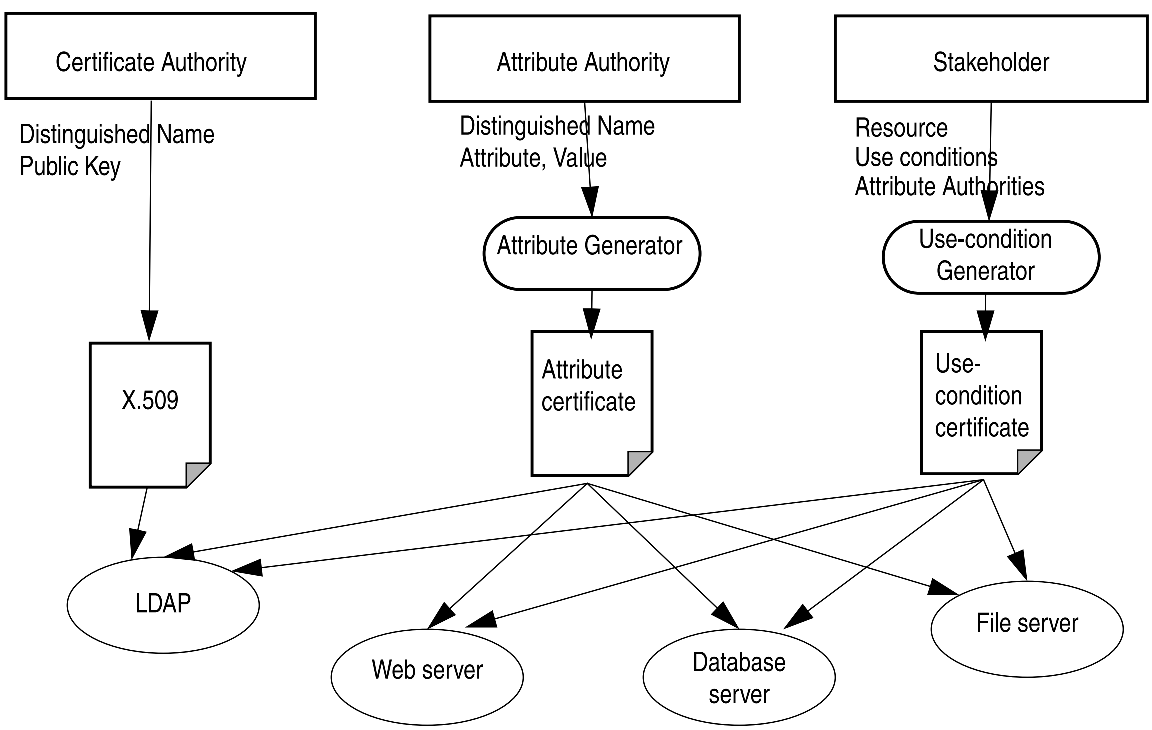


Figure 3. Certification Generation and Storage

```
-----BEGIN TEXT CERTIFICATE-----
-----BEGIN TEXT-----
use-condition
UID "portnoy.lbl.gov#1bea61fe#Mon Feb 01 00:17:11 PST 1999"
notValidBefore 981215014732Z
notValidAfter 991215014732Z
issuerAndCA "/C=US/O=Lawrence Berkeley National Laboratory/OU=ICSD/CN=IDCG-CA" "/
  C=US/O=Lawrence Berkeley National Laboratory/OU=ICSD/CN=Mary R. Thompson-sa"
resource http://imglib.lbl.gov/AkentiDist
scope sub-tree
attribute "( o : Diesel Combustion Collaboratory OR group : distrib )"
enable access read,execute
attributeIssuerAndCA o "Diesel Combustion Collaboratory" X509 "/C=US/O=Diesel Combustion
  Collaboratory/OU=SNL/CN=DieselCert.ca.sandia.gov" "/C=US/O=Diesel Combustion Collabo-
  ratory/OU=SNL/CN=DieselCert.ca.sandia.gov"
attributeIssuerAndCA group "distrib" Attribute "/C=US/O=Lawrence Berkeley National Labora-
  tory/OU=ICSD/CN=IDCG-CA" "/C=US/O=Lawrence Berkeley National Laboratory/OU=ICSD/
  CN=Mary R. Thompson-sa"
subjectCA "/C=US/O=Lawrence Berkeley National Laboratory/OU=ICSD/CN=IDCG-CA"
subjectCA "/C=US/O=Diesel Combustion Collaboratory/OU=SNL/CN=DieselCert.ca.sandia.gov"
-----END TEXT-----
-----BEGIN SIGNATURE-----
ZbO6puCmJGMY8Yz39RQ6Mf9Hx21+lC34suSH6onZ8Ml4CHVW+UHqQx6qShMe8D743+HR
  QPVDupsl
-----END SIGNATURE-----
-----END TEXT CERTIFICATE
```

Example 1. Use-condition Certificate

```
-----BEGIN TEXT ATTRIBUTE CERTIFICATE-----
-----BEGIN TEXT-----
attribute-certificate
attribute group
value distrib
notValidBefore 981215014732Z
notValidAfter 991215014732Z
subject "/C=US/O=Lawrence Berkeley National Laboratory/OU=ICSD/CN=Mary R. Thompson"
  "/C=US/O=Lawrence Berkeley National Laboratory/OU=ICSD/CN=IDCG-CA"
issuer "/C=US/O=Lawrence Berkeley National Laboratory/OU=ICSD/CN=Srilekha Mudumbai-
  sa" "/C=US/O=Lawrence Berkeley National Laboratory/OU=ICSD/CN=IDCG-CA"
-----END TEXT-----
-----BEGIN SIGNATURE-----
LedD9aawMkhpmW2dzt+o10Qb0Eanen0qMnYyAGYWPNL6DzbVqBIBXFesze40jPN6WelbV
  KL8SCP1Q/-----END SIGNATURE-----
-----END TEXT ATTRIBUTE CERTIFICATE-----
```

Example 2. Attribute Certificate

Figure 3 summarizes the different entities that create certificates, the tools that they use and the servers that manage the certificates. Not all types of certificate servers need to be available.

Let's look at an example of a use-condition and attribute certificate and what access they grant. The use-condition certificate in Example 1 is for the resource referenced by the name "*http://imglib.lbl.gov/AkentiDist*" and any directories under it. It was issued by the entity *Mary R. Thompson-sa*; it allows read and execute access to the resource. The user must either be a member of the organization *Diesel Combustion Collaboratory*, as attested to by presenting an identity certificate issued by the Diesel Lab CA, or be a member of the group *distrib,* as attested to by *Srilekha Mudumbai-sa*, whose signature must match the public one found in an ID certificate issued by the LBNL CA. The user must have an identity certificate issued by either the Diesel Lab CA or the LBNL CA.

The attribute certificate in Example 2 attests to *Mary R. Thompson* being a member of the group *distrib.* Since it has been issued and signed by *Srilekha Mudumbai-sa* it would satisfy the proceeding use-condition certificate.

Resource Server

Our model includes a resource server that interfaces to resources on behalf of the client. It is responsible for establishing a secure and authenticated connection with the user. Our current applications use an SSL-enabled Apache Web server ([2], [3]) and an Orbix SSL-enabled Object Request Broker (ORB) [22]. The SSL protocol is configured to require client side authorization. In this mode mutual authentication is performed and at the end of the handshake the server has an authenticated X.509 identity certificate for the user, which can be used to securely grant authorization to the entity at the other end of the encrypted connection.

After authentication, the server calls the Akenti policy engine with the DN of the user and the name of the resource that is being requested. The policy engine either returns "access denied" or a list of actions that are allowed. The resource server must know how to interpret and perform the named actions on behalf of the client.

Akenti Policy Engine

The policy engine is a library module that finds all the use-condition certificates that apply to a resource. It verifies that each use-condition certificate has been signed by a legitimate resource stakeholder and has not expired. Then for each use-condition that must be satisfied, the policy engine searches for the attribute certificates that attest to the required values for the user. Once all the use-conditions and attribute certificates are gathered and verified, the policy engine evaluates what actions, if any, the user is allowed to perform.

Since one of our goals is to avoid centralized access policy information, we immediately face the problem of where to look for use-condition certificates and how to know that they have all been found. Our solution introduces a minimal *authority file* that is stored with the resources. This file contains a list of servers which supply identity and attribute certificates; the list of the use-condition issuers (stakeholders); and where the use-condition certificates are stored. In addition, there is a *root authority file* that contains the list of trusted CAs, and their public keys, for the whole resource tree.

The Akenti policy engine searches each of the use-condition certificate directories listed in the authority file. It must find at least one use-condition certificate for each stakeholder. If a stakeholder supplies no use-condition certificate, Akenti denies all access to the resource. Several assumptions underlie this behavior. First, a stakeholder's use-condition certificates for a resource must all be stored in one place, so that if one is found, they all are. Second, each stakeholder intends to place a use-condition on the resource. (It is also possible to specify joint stakeholders, where a use-condition from either one will suffice.) Third, it is better to deny all access to a resource if the input from one of the stakeholders is missing, than to erroneously grant access that the missing use-condition would have denied. Finally, that the stakeholders will store their use-condition certificates in a secure and reliably accessible place. One option is for the resource server to provide a secure LDAP server on which the stakeholders may store their certificates.

Naming the resources is another issue that needs to be addressed. Our model assumes that the resources may form a hierarchy, such as a file system or a tree of Web documents. This model can obviously be reduced to a single level for something like a scientific instrument. Grouping resources into a hierarchy that reflects the desired protection reduces the number of use-condition certificates that must be issued. A use-condition has a scope of either local or sub-tree: Example 1, for instance, shows a sub-tree-scoped use-condition. A locally scoped use-condition only applies to the level named in the use-condition certificate; a subtree-scoped use-condition applies at that level and at any level beneath it. The name of the resource in the use-condition certificate is typically the name used to reference the resource. Hence, URLS are used for Web-accessed resources, CORBA object names can be used in the context of an ORB, etc.

There is one more non-obvious feature of use-conditions. Some use-conditions grant general access to a resource, as specified by "*enable access*" either as the only access or in conjunction with actions such as read or write. If a use-condition specifies access, a user must satisfy it before gaining any access to the resource. This feature allows stakeholders to exercise veto power over any subtree of resources. In particular, we envision this feature being used at the top level of a resource hierarchy, where a global use-condition might require that any user of the resource meet a condition specified by a high level authority, e.g., that all users must be member of some organization or group of organizations.

If a use-condition only grants actions, then any user who satisfies it is granted those actions in addition to whatever other actions she may be allowed. One use for such a certificate would be to grant "write" or "modify" privileges to a small subset of people while a larger group would be granted "read" access. For example there could be three use-conditions that apply to a resource: a subtree-scoped one at the root level that only grants access to everyone in the organization

LBNL; a second one at a local level that grants read permission to everyone in a group “readers”, and a third one at the local level that grants “modify” permissions to a group “writers”. Thus the user’s identity certificate would need to show her belonging to the organization LBNL, and she would need an attribute certificate placing her in either the “readers” or “writers” group before she would be allowed any access to the resource. If she also has a certificate placing her in the other group, she would get the additional access. Note that we use identity certificates in place of a separate attribute certificate to attest to the user’s values for selected components of a DN: organization, organizational unit and common name.

All use-conditions must be evaluated before a user’s access can be determined. All those that grant “access: must be satisfied; any that assert negative conditions, e.g., not from a proscribed country will take precedent and deny access; any of the positive ones may add more rights. It is certainly possible for multiple stakeholders to impose contradictory use-conditions, which may result in no access to the resource being granted. We believe that this mirrors the way stakeholders wish to impose control. The solution is for the stakeholders to be able to easily see what the combined results of all the use-conditions is, and to co-operate with the other stakeholders to create a set of use-conditions that satisfies everyone.

5. Implementation Refinements

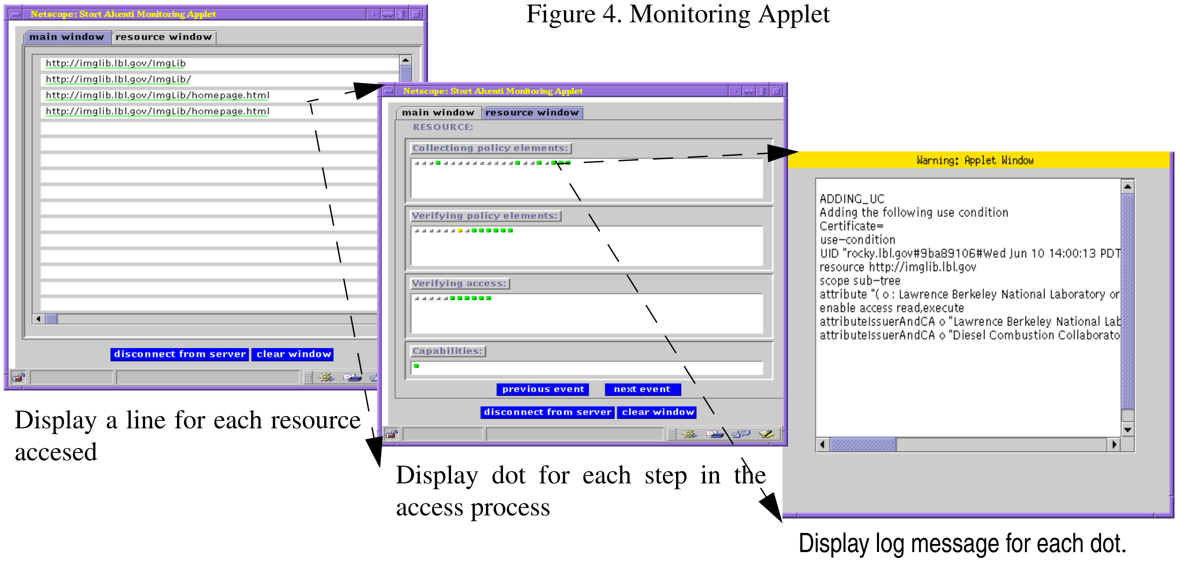
Caching

Since Akenti’s access control decisions are all based on the contents of signed certificates that are distributed across the network, an obvious performance bottleneck is the gathering of these certificates. One way to improve this performance is to make the searching as efficient as possible. We have done this by carefully choosing the parameters to the LDAP and database searches so as to maximize the amount of filtering on the server side and reduce the number of non-applicable certificates that are returned to Akenti. Certificates that are stored in file systems or Web directories are named by a hash index of the search attributes. Akenti’s search is based on these hash names.

The other obvious way to minimize searching over the network is to cache certificates locally once they have been found and verified. Since the Akenti policy engine is a library module without any persistent memory, we have implemented a cache server to store the certificates. In the case of the Web server used as an access control gateway, several processes may be checking access for the same set of resources in parallel and all talking to the same cache server. In our current implementation, the cache server runs on the same machine as the resource server, but it could easily run on a separate machine if desired. The cache manager caches use-condition certificates, identity certificates, and attribute certificates.

In addition to static certificates, Akenti creates, signs and caches a certificate containing the access rights of a user for a resource. This certificate is in effect a dynamic *capability certificate*. Capability certificates are especially useful for a hierarchical collection of Web docu-

Figure 4. Monitoring Applet



ments. A Web browser often makes several independent access requests for each logical document request, as when a page includes images. Also, in hierarchical collections of Web documents, related documents frequently all inherit access permissions from the same directory. Once the capability is cached, subsequent references short-circuit most of the Akenti policy engine.

The validity period for a cached certificate needs to be under the control of the stakeholders of the resource for which the certificate is going to be used. Since the same identity or attribute certificates may be used for several resources, checking the validity of a certificate requires two steps. When the certificate is first cached, the "not-valid-before time" is set to the current time, and the "not-valid-after time" is set to current time plus the validity period corresponding to the current resource. This value can be found in the authority file for the resource or can be the default value for the resource tree. If the cached certificate is going to be used to allow access to a different resource, the "not-valid-before time" stored with the cached certificate plus the validity period for the desired resource must be less than the present time. We make one exception to the above rule: since capability certificates represent the sum total of many different certificates, their lifetimes are kept very short (currently 5 minutes).

Monitoring

In order to provide a meaningful service, access control must be applied in such a way that the resources are protected as intended by the stakeholders. This involves understanding the structure of the resources and how they should be used, and developing a policy model that will support the intended access control. Akenti provides several services which are intended to help the stakeholder understand the policy implemented for the resource. In our prototype these services are available to anyone who has access to the resource tree. In a more restrictive environment, they could be limited to stakeholders of the specified resource.

First, a remote user can ask Akenti to statically display all the authority files and use-condition certificates applying to a resource. A stakeholder can use this facility to discover what use-conditions already apply, either those inherited from a higher level in the resource hierarchy or imposed by other stakeholders, before designing a new one.

Second, the Akenti policy engine performs extensive logging in order to provide dynamic information about the its behavior. The logging uses an existing logging library, NetLogger [27], that directs the logging messages to a log server and/or file or standard output depending on how the resource server has been configured. Directing the messages to a log server allows the information to be presented in real time to a user who is attempting to gain access. We have written a Java applet that graphically displays each step of the access-granting process. (See Figure 4.) The log is also saved to an audit file on the resource server so that the administrators or stakeholders may monitor the use of their resources.

6. Status

Over the past two years, we have implemented several Akenti-enabled servers. The prototype that has seen the most use is an Apache Web server with the SSLeay patches [3] and with Akenti replacing the standard access control module. It is being used as an access control gateway for a variety of Web-based resources within the Diesel Combustion Collaboratory [7]. This is a prototype collaboratory that involves two independent CAs and a number of government and commercial organizations scattered around the country. A single Akenti/Apache Web server is used to control access to two different image and data archives and a Web-based electronic notebook developed by Oak Ridge National Laboratory [14]. In these applications, the Akenti policy engine is called both by the Apache Web server and then again by the scripts that are used for fine-grained control of the resources. The Akenti policy engine is wrapped in a main program that is executed by scripts. Once the user has an identity certificate and the correct credentials, the access control is almost transparent.

We have also implemented a prototype CORBA ORB. Minor changes were made to the client side to find and present an identity certificate and to the server side to use one of the Object Management Group (OMG)-defined interceptors to call the Akenti policy engine.

Although we are just starting to evaluate Akenti, it appears to provide the sort of distributed management of access policy by multiple stakeholders that was our goal, while enforcing strong access control over the resources. In the case of remote references to resources, the additional overhead of Akenti does not seem to be unreasonable. We have invested considerable effort in creating user-friendly interfaces for the stakeholders. Currently we are rewriting the certificate generators. This is due partly to the rapid evolution of Java, and partly to our discovery that as we create policies for different sorts of resources and user populations, we find we need more flexibility and clarity in the certificate generator interfaces.

When problems occur, the logging facility is now both complete enough and sufficiently accessible that a practiced user can figure out what credential is missing, expired, etc. Probably the biggest problem with the logging information is that there can be too much of it, and sorting out the real cause of the problem is sometimes difficult. This is an area of ongoing development.

7. Vulnerabilities

The major vulnerability of the system derives from the fact that while stakeholders and their repositories are named in the authority file on the server, the use-condition and attribute certificates they depend on are maintained on distributed, "trusted" servers. If those certificate servers are not secure, then certificates could be deleted, resulting in an unintended access control policy. The type of failure depends on the type of certificates that are missing. If none of a stakeholder's use-condition certificates is available there will be a complete denial of access. If only some of a stakeholder's use-condition certificates are missing, the access could be greater than it should be. And if an attribute certificate is missing, specific users may get more or less access than they should.

The problem where a missing use-condition certificate allows greater permission than desired, can be solved by requiring the stakeholder to put all the use-conditions into one certificate. If that certificate is missing all access is denied. This constraint will produce more complicated use-condition certificates which may make the policy harder to understand.

The fact that a missing attribute certificate could permit too much access was revealed when comparing Akenti to KeyNote (see Section 9). The current use-conditions allow negative constraints, e.g., not belonging to some proscribed group. If that group certificate is missing, the user may get access to a resource that should be denied. To prevent this, use-conditions must always phrased in terms of positive conditions, so that attribute certificates will always increase access.

As we gain more operational experience, we will be better able to assess the importance of each of these vulnerabilities and the trade-offs required to address them.

8. Performance Measurements

The performance of a system composed of a client and a remote server using Akenti access control is often dominated by factors that are not controllable by the system designer. Nevertheless, it is valuable to measure the system performance in order to characterize the variables and to optimize those that can be influenced by the system.

Two variabilities arise from the fact that this is a distributed system: the network transmission time between the client and server, and the network transmission time plus the server response time between Akenti and the certificate servers. Performance factors under loose control by the stakeholder are the number of certificates required to make an access decision and the volume of data that is passed between the client and server. Finally, there is the overhead directly attributable to Akenti which includes the time associated with establishing an SSL connection and encrypting the data between the client and server, and the time spent in the Akenti policy engine gathering and verifying certificates.

The measurements in this paper are for file fetches between a Java SSL-enabled client an Akenti/Apache Web server. The client, server and all the certificate servers are on a 100 Mb/s LAN. The document sizes varied between 1KB and 1MB in order to evaluate the overhead of the SSL encryption. The SSL keys are 128-bit keys. The number of certificates that were required to permit access varied between a minimum of one use-

	No caching			Caching		
	Akenti (seconds)	**SSL/network (seconds)**	**Total (seconds)**	**Akenti (seconds)**	**SSL/ network (seconds)**	**Total (seconds)**
Min-acc						
1K	**0.86**	**0.65**	**1.51**	**0.20**	**0.65**	**0.85**
1M	**0.90**	**1.75**	**2.65**	**0.22**	**2.02**	**2.34**
Ave-acc						
1K	**2.26**	**0.73**	**2.96**	**0.115**	**0.646**	**0.762**
1M	**2.24**	**1.96**	**4.00**	**.188**	**1.77**	**1.96**

Table 1: Average times to fetch a document from a Secure Akenti server.

condition certificate and two identity certificates and a more average case that required two use-condition certificates, one attribute certificate, and four identity certificates. We also took measurements with and without Akenti server caching enabled.

On the server side, Akenti does extensive logging of each logical step in the policy engine. This measurement excludes the server side time spent in the Apache server and SSL encryption. The times in Table 1 are the times in the Akenti policy engine (Akenti), the total socket read time the client saw (Total) and the difference between the two which can mostly be accounted for by SSL overhead (SSL/network).

The test program fetched the same file 10 times and then calculated the mean fetch time. The data from the client program was combined with the matching server log entries to determine the values in Table 1. Each case was run several times in succession to average over variations in the network load. Thus each number is the average of about 30-40 file fetches.

Several observations can be made from this data. As the files get bigger, the SSL encryption times tend to dominate the overhead. However, SSL can be configured to do only authentication and message integrity checking if encryption is not needed, which would reduce this time. As more certificates are required to grant access, the times in the Akenti policy engine increase. We can see from the Akenti log files that the major categories of time in the policy engine are fetching certificates and verifying signatures. In the minimum certificate case about 79% of the total time was spent fetching certificates and 9% was spent on signature verification. In the average certificate case, about 83% of the time was in fetching certificates and 8% was spent in certificate verification. Failing to find a certificate, such as an optional attribute, took more than twice the time of successfully finding one. The rest of the time was split between reading authority files, parsing the use-conditions and general program overhead.

In the case when caching is enabled and a valid capability certificate is found, the time in the policy engine is about 0.11 seconds. The variation that appears in these times in Table 1 is the result of the capability timing out and having to be reestablished. The caching lifetimes of cached use-condition and identity certificates is generally longer than that for capabilities, so cached versions of those certificates may be used when reestablishing the capability.

For a Web server that is mainly fetching documents, caching by the Akenti policy engine provides a big performance benefit, since there are usually several clustered access to documents in the same general protection domain. If Akenti is being used by a server where the pattern of accesses is isolated, the caching may actually be a disadvantage, since cache misses and subsequent cache updates are relatively costly.

	With Akenti (seconds)	**No Akenti (seconds)**
1Kbyte	**0.76**	**0.02**
1Mbyte	**1.96**	**0.75**

Table 2: Document fetch with and without Akenti access control

The corresponding time that it took to fetch 1KB and 1MB files using the same client program but a standard Apache web server with no access control was about 0.02 seconds for the 1KB file and between 0.69 and 0.80 seconds for the 1MB file. (See Table 2.) For Web servers, that most meaningfully compares to the 0.76 and 1.96 second times for an average set of access constraints from a caching Akenti server. Obviously, the target applications for Akenti access control are ones where there is something important to protect and the granularity of the access checking is fairly large, e.g., a large document to be fetched, or a substantial process is to be started on the resource machine.

Another case where the Akenti overhead is not too severe is accessing a Web document that requires the parallel fetching of many secure components. For example, a document where all its parts are in the same protected tree. In this case the browser and the server fetch in parallel, and since Akenti has no trouble working in parallel and sharing the same cache, the net result of such a clustered fetch is not too much worse than the secure fetching of one document. For example, for a document containing 10 images, the html frame is retrieved in 4 seconds, the first three images appear after 8 seconds and the rest of the images appear at 10 seconds. The corresponding behavior for such a page with no access control is for all the images to appear after about 1 second.

9. Related Work

Currently there are several other projects working with signed certificates to enable access control decisions. One of the earliest attempts to define standards was the Simple Public Key Infrastructure (SPKI) [9] IETF draft proposal by Ellison, et al. This work proposed a standard for authorization certificates, name certificates and access control lists that are all represented by a formalized key-word, value syntax called S-expressions. An authorization certificate consists of an issuer, represented by a public key; a subject also represented by a public key; an optional delegation field; the actions that are authorized; and an optional set of real-time constraints on the certificate. We chose not to base Akenti on authorization certificates, but on policy and use-condition certificates instead. Our decision sidesteps the problem of revoking authorization certificates, and makes the policy more explicitly stated and thus easier for multiple stakeholders to understand. Our design also only allows only one level of delegation, rather than the possibility of delegation on each certificate. Again this makes the policy more restrictive, but easier to discover.

Pekka Nekander and Jonna Partanen of Helsinki University of Technology have used the SPKI-style certificates to carry access permissions with Java code, rather than relying on local access control configuration files [21]. They argue that maintaining a local security configuration on each machine on which a Java class might be executed does not scale to large numbers of machines and classes. This argument is parallel to our observation that requiring all users to have accounts and to be entered into ACLs on each resource server does not scale to separately administered distributed compute resources.

Nekander's and Partanen's system uses authorization certificates to grant a Java class (or JAR file) specific permissions. If the machine on which the class is to be executed accepts the signer of the certificate, the class is allowed to perform the actions. This type of access decision supports the Java security model which grants access to code on the basis of where it came from rather than who caused it to be executed. However, the focus in Akenti is on allowing access to resources, which are referenced through a resource server, by specified classes of users. Thus it is more natural to base access decisions on authenticated identity certificates matched against policy requirements at the time of resource use.

The PolicyMaker [5] and KeyNote [4] trust management systems present a very generalized approach to specifying and interpreting security policies, credentials and trust relationships. Both of these systems share with Akenti the goal of putting all the policy and credential information into signed certificates that can be stored in a distributed fashion. These certificates are then gathered together by a policy engine module or daemon at the time of resource access and interpreted to allow or deny the access. PolicyMaker certificates are much more generalized than Akenti's as they consist of programs written in a general programming language. Since an access policy is represented by the set of all such certificates, it may be very hard to understand the overall access policy for a resource. The KeyNote system settled on a C-like expression and regular expression syntax for describing conditions, similar to that which Akenti uses. However, in an Akenti application the policy for a resource use is spelled out in a few authority files and use-condition certificates which make explicit what actions users or classes of user have while in an authorization credential system such as KeyNote, all the assertions combine to imply a policy.

KeyNote certificates also differ from Akenti's in the principle of "assertion monotonicity" which means that each assertion will increase the permitted actions. Akenti permits a use-condition to specify negatives, e.g., that a person may not be a member of a given group. Then if the corresponding certificate that identifies the person as a member of the proscribed group is not found, the person may be granted more permissions than were intended by the stakeholder. Future versions of Akenti should close this loophole.

A more specific use of identity certificates for authorization can be found in the Globus Project [15], a joint research project between Argonne National Laboratory and the University of Southern California's Information Sciences Institute (USC/ISI). Globus is developing a software infrastructure for distributed computational and informational resources, and is experimenting with using X.509 identity certificates to provide a one-time per session sign-on to a distributed set of resources. [13]

The Globus infrastructure provides a gateway server at each site. The server on the client side accepts and verifies a user's identity certificate and creates a temporary proxy certificate to represent the user to other Globus servers. The server at the resource site authenticates the certificate that it receives and maps the identity into a local user ID. The resident operating system then performs access control as usual. This solution was motivated by the early use of Globus to provide remote access to supercomputers. The administrators of the supercomputing sites were not interested in relying on a new access control mechanism. The Globus solution allows a user to authenticate once per session by presenting an identity certificate. However, each user must still have a local user ID and account with each resource server. We are working with the Globus group to integrate Akenti as an alternative access control mechanism for those sites that want distributed policy management.

10. Future Work

There are two general directions for future work on Akenti. One is to implement the policy engine as a daemon in addition to the current library module approach. The second is to improve the syntax of the various certificates. Our intent is to define an XML-based (Extensible Markup Language) [10] format for our certificates. XML has the advantages of presenting self-describing documents and being widely used by various scientific disciplines. There are tools available for validating an XML document against its document type definition (DTD) which may be useful to the interface programs that are used to create the certificates. Our goal is to use a standard syntax which may be familiar to the people who have to write and understand the policy.

There are several applications which we intend to integrate with Akenti. We plan to use Akenti to make access decisions for a network-bandwidth quality of service (QoS) framework that is being developed at LBNL [16]. In order to conform to the standards of the QoS community, a policy server should communicate via the COPS (Common Open Policy Service) [6] protocol. This is one of the motivations for implementing Akenti as a server that will respond to requests for access control decisions. COPS is a statefull protocol that, among other things, allows the resource server to upload or download policy documents. This feature gives us the option of keeping the authority files with the resource tree and uploading them to the Akenti server, allowing it to be a stateless decision maker.

We are currently integrating Akenti security with a secure mobile agent system that is being developed at LBNL [19]. This system customizes the Java security model to enforce policy-based access control on mobile Java agents.

Another possibility is to implement the proposed standard Generic Authorization and Access control API [23] with Akenti. To do so would require adding some additional library interfaces.

Both COPS and GAA expect the policy module to be able to make decisions based on the time of day, the location of the requestor and, in the case of COPS, on some level of allowed quotas for the resource. In our current model, Akenti simply copies strings that represent actions from the use-condition certificates and returns them to the resource server which interprets them. The concepts of a time during which a user may use the resource, an allowed IP address or domain from which the request may come, and a quota for use of resources must be added to the policy engine. To accomplish this, Akenti will need to define a convention for naming the use-conditions for time, location and quotas and then checking for compliance. The last case requires getting a value for the amount of resources used from the resource server and storing it as an auxiliary certificate.

11. Conclusions

In a larger view, useful security is very much a risk management, deployment and user ergonomics issue. Once it has been determined what level of security is required, the hard problem is integrating that level of security into the end-user (e.g., scientific) environment so that it will be used, trusted to provide the protection that it claims, easily administered, and genuinely useful in the sense of providing new functionality that supports distributed organizations and operation. As large enterprises establish public key infrastructures and people become accustomed to using an identity certificate to authenticate themselves rather than a multitude of passwords, certificate-based access control will seem natural and be more easily understood.

Akenti facilitates the decentralized creation and management of policy certificates and the use of these certificates to make secure policy-based access control decisions. Distributed certificates permit the decentralized administration of shared resources which is an important goal in a collaborative research environment. We believe that our prototype Akenti implementation has demonstrated the viability of such a distributed system.

Acknowledgments

Case Larsen wrote many of the certificate-handling and cryptographic libraries used in the Akenti implementation. Bob Aiken and Mary Anne Scott of DOE/ ER/ MICS have been consistent supporters of this approach to security and access control.

Availability

An early version of the Akenti policy engine and certificate generators can be downloaded from the Akenti web site. (http://www-itg.lbl.gov/Akenti/)

References

[1] "The Akenti Approach", http://www-itg.lbl.gov/Akenti/

[2] "About the Apache HTTP Server Project", http://www.apache.org/

[3] "Apache-SSL", http://www.apache-ssl.org/

[4] M. Blaze, J. Feigenbaum, A.D. Keromytis, "The KeyNote Trust Management System", work in progress Internet Draft, March1999

[5] M. Blaze, J. Feigenbaum, J. Lacey, "Decentralized Trust Management System", *Proceedings of the 17th Symposium of Security and Privacy,* pp. 164-

175 IEEE Computer Science Press, Los Alamitos, 1996

[6] J. Boyle, R. Cohen, D. Durham, S. Herzog, R. Rajan, A. Sastry, *"The COPS (Common Open Policy Service) Protocol"*, Internet Engineering Task Force Draft, work in progress

[7] "Diesel Combustion Collaboratory Homepage", http://www-collab.ca.sandia.gov/

[8] "DOE2000 Homepage", http://www.mcs.anl.gov/DOE2000/

[9] C. Ellison, B. Frantz, B. Lampson, R. Rivest, B. Thomas, T. Yloenen, *"Simple Public Key Certificate",* Internet Engineering Task Force Draft, work in progress

[10] "Extensible Markup Language" http://www.w3.org/XML/

[11] S. Farrell, R. Housley, "An Internet AttributeCertificate Profile for Authorization", Internet Engineering Task Force Draft, work in progress

[12] W. Ford, *Computer Communications Security: Principles, Standards, Protocols, and Techniques*, Prentice-Hall, Englewood Cliffs, New Jersey, 07632, 1995

[13] I. Foster, C. Kesselman, G. Tsudik, S. Tuecke, "A Security Architecture for Computation Grids", Proc. *5th ACM Conference on Computer and Communications Security Conference*, pg. 83-92, 1998.

[14] A. Geist, N. Nachtigal, "ORNL Electronic Notebook Project", http://www.epm.ornl.gov/~geist/java/applets/enote/

[15] "The Globus Project", http://www-fp.globus.org/

[16] G. Hoo, W. Johnston, "QoS as Middleware: Bandwidth Brokering System Design", submitted to the *High Performance and Distributed Computing conference* 1999.

[17] R. Housely, W. Ford, W. Polk, D. Solo, "Internet X.509 Public Key Infrastructure", Internet Engineering Task Force Draft, PKIX Working group, work in progress. Also see Ford & Baum, *Secure Electronic Commerce*, Prentice-Hall, pp 214-230

[18] W. Johnston, S. Mudumbai, M. Thompson, "Authorization and Attribute Certificates for Widely Distributed Access Control", *IEEE 7th International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises-* WETICE '98

[19] S. Mudumbai, A. Essiari, W. Johnston, "Anchor - A Secure Mobile Agent Toolkit*",* - submitted to Mobile Agents '99

[20] B.C. Neuman and T. Ts'o, "Kerberos: An Authentication Service for Computer Networks", IEEE Communications Magazine, v.32, n.9, Sep 1994, pp. 33-38

[21] P. Nikander, J. Partanen, "Distributed Policy Management for JDK 1.2", *Proceedings of Network and Distributed System Security Symposium*, San Diego, CA, Feb 3-5, 1999

[22] OMG-CORBA homepage" http://www.omg.org

[23] T. Ryutov, C. Neuman, "Access Control Framework for Distributed Applications*",* Internet Engineering Task Force Draft, work in progress

[24] B. Schneier, *Applied Cryptography*, Second Edition, John Wiley & Sons, 1996

[25] "The SSL Protocol", http://home.netscape.com/eng/security/SSL_2.html

[26] "SSLeay FAQ", http://www.psy.uq.oz.au/~ftp/Crypto/

[27] B. Tierney, W. Johnston, B. Crowley, G. Hoo, C. Brooks, D. Gunter, "The NetLogger Methodology for High Performance Distributed Systems Performance Analysis", *Proceedings of the IEEE High Performance Distributed Computing -7,* '98