Extending the OSF DCE Authorization System
to Support Practical Delegation

Marlena E. Erdos
Joseph N. Pato

Hewlett-Packard Co.
Chelmsford, MA

ABSTRACT

In a simple client/server distributed environment, two principals are involved in most transactions - the initiator and the target of the operation. The target of the operation can reasonably make authorization decisions based on the identity of the initiator. This model is insufficient, however, when the server performs operations on other components on behalf of the initiator as is common in distributed object oriented environments. This paper will describe the need for a delegation facility in distributed object oriented systems and then present some elements of the delegation system we've proposed for inclusion in OSF’s Distributed Computing Environment (DCE).

INTRODUCTION

The need for a delegation facility

In a distributed object oriented environment, intermediate objects hide the details of complex system interactions. These intermediate objects receive high level requests from initiating clients and perform some series of low level operations on a number of other services. Unfortunately the interposition of the abstracting object prevents the target services from securely determining the identity of the initiator. All requests arriving at the target services appear to be the action of the intermediary rather than the true initiator.

The inability to determine the true initiator of a request has a chilling effect on the design of distributed systems. The designer of an intermediate service is forced into a set of unsatisfactory design choices. The service may be implemented as a local process that runs with the identity of the initiator, but loses the benefits of distribution. Alternatively, it may retain distribution but then it must run as a privileged principal that has full access to all services it abstracts. This solution has the disadvantage of forcing the abstracted services to trust that the privileged intermediary will make correct access control decisions on their behalf. A third unsatisfactory approach is the use of an alternate set of target service interfaces that allow an authorized principal to specify the principal on whose behalf the operation is really being performed. This solution comes at the cost of redundant interfaces that expose the details of privilege attributes to the application protocol. Finally, the service may be implemented in a way that impersonates the initiator - where the initiator transmits to the intermediary service the credentials (tickets and keys) necessary to be indistinguishable from the initiator. This final approach is much like a non-distributed application - but the mere fact of distribution (and in the DCE the high degree of location transparency) makes it so that this greatly increases the risk to the client of being compromised by a Trojan horse application.

To solve this problem adequately, some form of delegation is required. We've have proposed a delegation architecture and design to OSF for inclusion in the DCE [1]. This system is described below.

Delegation system

Our delegation architecture has three major components: First, we allow an intermediary to operate on other objects in a manner that reflects the initiator's identity as well as its own. A target server receiving such a chained request would see the privilege attributes of each participant in the chain.

Second, we extend the authorization model to allow target servers to make use of the distinction between initiators and intermediaries. Target servers may grant rights to principals acting as intermediaries on behalf of authorized initiators without granting rights for those principals to act on their own.

Lastly, we allow clients performing operations to place restrictions on the uses of their identity in chained calls. A client may choose to entirely disallow delegation or to limit which principals may use the client identity in a delegated manner.

The delegation design uses composition of privilege attributes to realize identity chaining, additions of new ACL entry types to reflect the initiator/intermediary authorization distinction, and extensions to the security API to allow clients to control delegation of their identity. This paper will further discuss the delegation architecture and design, and then present some low-level extensions to existing DCE
elements and mechanisms for accomplishing the implementation.

**ASPECTS OF THE EXISTING DCE**

In the basic DCE environment [2,3], access to a resource is managed by an application server which is the reference monitor for the resource. When a client attempts to perform an operation on the resource, the reference monitor examines the client's identity and compares it to control attributes associated with the resource. The client's identity is represented by a set of privilege attributes (PAs) and the control attributes are stored in an access control list (ACL).

Privilege attributes are the collection of information about a principal that is used by the authorization system when determining if access to a resource should be granted. These PAs are limited to the unique identifiers representing the initiating principal and the set of groups to which the principal belongs. A trusted system component, the Privilege Server (PS) [4], produces a tamper-proof privilege attribute certificate (PAC) that contains the PAs and is suitable for presentation by a client to a server. By relying on PACs for the identity of a caller, we place the same degree of trust in the Privilege Server that we have already placed in the authentication component of the distributed system. A compromised Privilege Server will be able to generate PACs that impersonate any legitimate user.

Collecting privilege attributes into a certificate that can be presented by the client to a server has the benefit of allowing servers to make authorization decisions locally without the need of contacting trusted system services to obtain privilege attributes for the client principal. In addition this model allows the client to choose the set of privileges to be used during a given session.

The emerging next generation of the DCE evolves the PAs supported to be closer to the capabilities found in ECMA [5,6]. This change allows for greater flexibility in the authorization models available to the distributed system, and of particular interest to this paper - provides a vehicle for recording delegation information.

**ARCHITECTURE & DESIGN OF DELEGATION**

**Intermediaries and Chained identities**

A server acts as an intermediary or delegate when in order to fulfill an operation a client made on it, it must perform one or more operations on other objects. We say that these subsequent operations are performed on behalf of the client as part of a chained call. All chained calls are performed with a chained identity.

We represent chained identities using the conventional notation [7,8,9]

```
foo FOR bar
```

This means that principal `foo` is acting on behalf of principal `bar`. When more than one intermediary is involved in the chained call, the identities of all participants are reflected in the chained identity. In general,

```
DelegateN FOR DelegateN-1 FOR ... FOR Initiator
```

means that principal `DelegateN` is acting on behalf of principal `DelegateN-1` which is acting on behalf of principal `DelegateN-1` extending back to the initiating client (aka `initiator`) which is acting on its own behalf.

Identity is represented by a set of privilege attributes (PAs). Logically, a chained identity is represented by an ordered array of PA sets, with the PA set of the initiator distinguished from those of intermediaries.

Each object acting as a client may choose whether or not it wishes to allow the immediate target to use its (the client's) PA set in a call chain. In other words, each client enables/disables delegation of its identity. However, the determination that a given call is part of a call chain is up to the intermediary and is dictated by the semantics of the situation. We'll discuss the consequences of a mismatch between what a client allows and what an intermediary needs to do in when discussing restrictions on the flow of identity below.

**Authorization model at the target**

Each server has one or more access control lists (ACLs). ACLs, as found in Posix or DCE [10], contain entries that identify the access rights granted to principals bearing certain PAs. To support delegation, the target server effectively grants one set of rights to initiators and another set to intermediaries.

We realize this distinction by extending the standard ACL entries for principals, groups, etc. with a corresponding set of entries that apply to principals and groups acting as intermediaries. These delegate entries grant intermediary rights i.e., the ability to act as an intermediary for an operation, but do not grant the ability to operate on the target object directly.

While authentication of an operation is done automatically by the security runtime at the server, authorization is only performed if the server application code explicitly invokes the authorization facilities. Whether a call is done with a chained identity or not is transparent to the server application code, however the authorization facilities go through additional checks when presented with a chained identity.

For both simple and chained identities the authorization facilities first determine whether or not the initiator is
authorized to perform the operation. They do this by examining the standard entries - and only these entries - on the ACL when calculating the initiator's rights. Initiators don't - and can't - expand their access through use of intermediaries; at best, intermediaries won't degrade access that the initiator has on its own. The initiator of a chained operation must have rights to perform the operation directly or the operation will be rejected.

If the initiator passes the authorization check and the call was chained, then the authorization facilities next check the PAs of each intermediary in the chain. Each delegate must have sufficient rights to act as an intermediary for the operation or the authorization facilities will return an authorization failure indication to the server application code. A delegate is deemed to be authorized if its PA set gives it either initiator rights or delegate rights for the operation. In other words, any principal that can perform an operation directly is implicitly authorized to be an intermediary for the operation. The access control algorithm is presented in Figure 1.

(i) Check Initiator:
    Apply standard algorithm
    IF access mode is denied THEN
    Deny Access
    ENDIF

(ii) Check Each Intermediary:
    FOR EACH Privilege Attribute Set IN Extended PAC DO
    Apply standard algorithm (allow delegate entries)
    IF access mode is denied THEN
    Deny Access
    ENDIF
    END

(iii) Grant Access

Figure 1: Access control algorithm for delegation

Note that the order of intermediaries, the topology of the call-chain, is not relevant in the access control decision.

Client restrictions on flow of identity

Before discussing client restrictions on identity flow we'd like to clarify some terminology. An object acts as a client when it sends an RPC to another object. We use the term target to refer to any object that is downstream in a call chain from a given client. Immediate target is the object a client performed an operation on directly. Direct Requester is the client that directly operates on a given target. As we've previously mentioned, initiator is the initial client in a call chain. Final target is the last object in a call chain.

In the absence of delegation it is simple to understand the flow of identity in the system. The identity of a client is projected to its immediate target. In this environment the client determines that it wants to perform an operation on the immediate target and its identity is not subject to further use by the target. In the presence of delegation, however, the immediate target gains the ability to project its caller's identity. In this environment we allow each client to protect itself by placing limitations on who may project its identity and to whom its identity may be projected.

As we've mentioned a client may simply allow or disallow use of its identity in a chained call. We also permit any object acting as a client, either as initiator or delegate, to place restrictions on the delegations it allows. The two types of delegation related restrictions are target restrictions and delegate restrictions.

Target restrictions set by a given client apply to all servers in a call chain that are downstream from the immediate client-intermediary pair. For example, in the following call chain

A -> B -> C -> D

target restrictions set by A apply to both C and D, but not to B. C and D are targets of a delegation through B. Though B is the immediate target of A's operation, it is not a delegation target.

Delegate restrictions set by a given client limit who may act as an intermediary. They apply to all servers that are downstream from the client that wish to acts as intermediaries. Again, given the above call chain, delegate restrictions set by A apply to B and C. The delegate restrictions are irrelevant to D simply because D is not acting as an intermediary.

The effect of either type of restriction being violated is the same. The identity of the party that placed the violated restriction will be replaced with the anonymous identity. Other identities already present in the chain are not affected.

If the client doesn't allow its identity to be delegated, then the server it calls will receive its identity - allowing the server to make an appropriate authorization decision. Any subsequent objects called on the client's behalf, however, will not see the client's identity. These objects will still see a chained identity but the security system will substitute the anonymous identity where the identity of the client would have appeared.

Here is an example of a chained identity with anonymous entries:

foo FOR anonymous FOR bar FOR anonymous
This says that principal _foo_ is acting on behalf of an anonymous intermediary which is acting on behalf of _bar_ which is acting on behalf of an anonymous initiator.

While it may seem much less important to allow intermediaries to place delegation restrictions than initiators, we feel such functionality is important for an extensible system.

**Extensible client restrictions**

Our system has two types of extensible restrictions on privilege attributes. These restrictions allow applications to implement a variety of security models and policies beyond those expressible through the supplied PA/ACL system. An example of an extensible restriction that an application might define is a time-of-day restriction.

Required attributes limit the activities that a target server can perform. A server receiving a required restriction must be able to understand it. If the application is unable to decode a required restriction it must reject access.

Optional restrictions differ only from required restrictions in that applications that are unable to decode a given optional restriction are free to ignore its presence.

**Example of the model**

Figure 2 provides a frequently used example of a compound document. In it a user is accessing a document which contains a graph that obtains its data from a spreadsheet. When this document is implemented in a distributed object environment, each component may run as an independent process with a distinct principal identity. The document, graph and spreadsheet are each reference monitors for their data and grant access based on the contents of their associated ACL.

In this example let the User process run as principal _U_, the Document as principal _D_, the Graph as principal _G_ and the Spreadsheet as principal _S_. The User process enables delegation and performs a view_document operation on the Document.

On receipt of the view_document operation, the Document consults its ACL and verifies that _U_ has the rights necessary for the view_document operation. The Document proceeds to compose a delegated identity of _D FOR U_ with delegation enabled and performs the view_graph operation on the Graph.

The Graph process receives the view_graph operation from the Document object. It consults the ACL shown in figure 3 and verifies that _U_ is authorized to initiate a view_graph operation and also verifies that _D_ is a legitimate delegate. As a component of completing the view_graph operation, the Graph composes the delegated identity _G FOR D FOR U_ and performs the obtain_range_data operation on the Spreadsheet.

```
<table>
<thead>
<tr>
<th>ACL Entry Type</th>
<th>PA Value</th>
<th>Permission</th>
</tr>
</thead>
<tbody>
<tr>
<td>User</td>
<td><em>U</em></td>
<td>view_graph</td>
</tr>
<tr>
<td>User_delegate</td>
<td><em>D</em></td>
<td>view_graph</td>
</tr>
</tbody>
</table>
```

*Figure 3. ACL for Graph Object*

Finally the Spreadsheet process receives the obtain_range_data operation. Applying the ACL shown in figure 4 it verifies that _U_ is a valid initiator matching the any_other entry and verifies that _G_ and _D_ are legitimate intermediaries since they also match the any_other entry in the ACL.

```
<table>
<thead>
<tr>
<th>ACL Entry Type</th>
<th>PA Value</th>
<th>Permission</th>
</tr>
</thead>
<tbody>
<tr>
<td>Any_other</td>
<td>N/A</td>
<td>obtain_range_data</td>
</tr>
</tbody>
</table>
```

*Figure 4. ACL for Spreadsheet Object*

**MECHANISM**

The DCE provides strong mechanisms for trustworthy transmission of identity between client and server. Delegation introduces changes in these identity transmission mechanisms. We discuss the existing mechanisms for projecting identity prior to considering the changes for supporting delegation.

Note that the DCE is designed to allow a number of different authentication and key distribution protocols to be used. DCE 1.0, however, only includes a concrete specification and implementation using the Kerberos V5 [11] protocol suite. Consequently, we will restrict the following discussion to the mechanisms used for identity flow that are used by the DCE in conjunction with the Kerberos V5 protocols.

**Overview of existing DCE system**

Of the features of the DCE security protocols, two are fundamental - the ability to provide integrity and
confidentiality protections to a communication session between a client and a server and the ability for the communicating agents to determine the identity of their partner. Kerberos V5 protocols provide the mechanisms for accomplishing both tasks. Conventionally, however, the notion of identity in a Kerberos environment is limited to the name of a given principal. As described above, a DCE identity is a set of privilege attributes that are active for a given principal during a given session. The DCE leverages the authorization data field of a Kerberos V5 ticket to carry the additional privilege attributes.

When a user session is created, normally through some variant of the local system login sequence, the DCE security runtime acquires the principal's ticket granting ticket (TGT). The TGT may then be used to obtain tickets to other server principals. These new tickets may then be used to exchange keys with those targets and establish protected communication. These tickets, however, are not suitable carriers for privilege attributes since the client is free to request any data in a ticket's authorization data field.

For the security runtime code at a server to trust a given set of privilege attributes it must believe that an authorized system service has constructed (certified) the data. In the DCE, the trusted component is the cell's Privilege Server. The server will trust a ticket bearing PAs and treat it as a PAC if the ticket is issued in the well-known name of the Privilege Server.

To get such a ticket, the DCE login code actually obtains two TGTs. The first TGT obtained in the client principal's name is generally only used to obtain a ticket to the cell's Privilege Server. Once this is obtained the runtime communicates with the Privilege Server to obtain a second TGT. The second TGT is issued in the name of the Privilege Server and contains the privilege attributes requested by and valid for the client principal in the ticket's authorization data area. This second TGT is referred to as the privilege TGT (PTGT). It is the PTGT that is then used by the client when obtaining tickets to target servers. The Kerberos V5 KDC will automatically transfer the PAs identifying the actual client from the PTGT into the authorization data field of the ticket for a given target server.

The Extended PAC

Our delegation model requires extensions to the contents of PACs. The content change is due to extending the notion of identity to include chained identities. In addition we have added delegation and extensible restrictions to the extended PAC.

Two other concerns have contributed to change in the DCE 1.0 PAC: performance and legal issues. The authorization field of a Kerberos V5 ticket is encrypted in the key of the target principal. This raises performance concerns as the number of privilege attributes is increased. In addition legal issues are raised with respect to the encryption of data. The extensible restrictions allow applications to provide arbitrary uncontrolled data in the EPAC. Encrypting this data may violate laws governing the export of encryption technology - and in some countries may violate laws controlling the transmission of encrypted data over public carriers.

The solution we have chosen to address these various issues was proposed by the European Sesame project [12]. The contents of a EPAC need not be confidential - therefore we have removed them from the authorization data field of the V5 ticket and simply placed a cryptographic hash of the EPAC in the PTGT. This seal\(^2\) serves to connect the EPAC to the ticket and provides the same guarantees of authenticity of the EPAC to the target server.

Becoming a Delegate

When a server needs to perform an operation on another target on behalf of its client, it must become a delegate for that client. The security runtime at the server possesses the EPAC for its caller and it has a PAC representing itself, but these two PACs are not directly usable to represent the new chained identity. The server must obtain a new EPAC (and PTGT) that represents the chained identity from the Privilege Server.

The mechanism described so far does not provide the delegate with the necessary data to submit to the Privilege Server to acquire the new EPAC representing the chained identity. The delegate does not possess proof can be presented to a third party that the incoming EPAC is legitimate. It needs some form of delegation token [13] that may be submitted to the Privilege Server when requesting a new chained EPAC.

The extended PAC, obtained from the caller, contains all of the information needed by the Privilege Server to determine if a given principal should be allowed to chain its identity to that of the caller. The only missing data is a seal protecting the integrity and authenticity of the extended PAC. We have, therefore, added a signature field to the extended PAC allowing it to become a true certificate and thus serve as the delegation token. The signature field supports both a seal - using a key known to the Privilege Server - and a signature using public key technology. This allows the same certificate to be used by the DCE's existing Kerberos environment and by public key based facilities.

\(^1\) The DCE's notion of an administrative domain, roughly comparable to a Kerberos realm.

\(^2\) We use the ISO definition for seal indicating a cryptographic checksum using symmetric keys. This should not be confused with other uses of the term that indicated confidentiality of the data.
Compatibility with existing servers

Whenever there is the introduction of a new revision of system, interactions with the prior revisions must be considered. In our case, it is possibly that a new intermediary will send an extended PAC to an old server that only understands the simple PAC format. To deal with this situation we provide three compatibility modes. We call refer to them as *initiator*, *direct requester* and *reject*.

When the intermediary requests initiator compatibility mode, the Privilege Server arranges the associated ticket so that it includes the PA set of the call chain's initiator in the authorization data field. Appended to this is the hash of the EPAC. In other words, the PA set of the initiator is placed where an old (DCE 1.0) server already expects a single PA set. The security runtime at old servers is set up to ignore extra bytes in a PAC, so the presence of EPAC hash does not have any untoward effect. With this compatibility mode, the current intermediary appears to the target server to actually have the identity of the initiator.

When the client requests direct requester mode, the PS arranges the ticket so that the intermediary's identity is in the authorization data field. Again, the EPAC hash is ignored as extra bytes by old servers. In this mode, in order for a DCE 1.0 server to authorize the operation, the intermediary must have appropriate rights.

In reject mode, the intermediary effectively asks the PS to set up the ticket so that an old server will know that it is dealing with a new client and will reject the call.

The current intermediary requests a particular compatibility mode when it requests a new login context that reflects a chained identity. Likewise the initiator has specified whether or not it will permit initiator compatibility mode in the course of obtaining a login context appropriate for an initiator. If the intermediary requests initiator compatibility mode but the initiator has not allowed it, the intermediary will receive an error indication immediately. Because of the possibility of this conflict, we allow the intermediary to effectively say 'I want initiator mode, but if it isn't permitted I'll take direct requester mode'. This form of request will generally not fail.

**Impersonation, aka Full Identity Forwarding**

While many of today's distributed systems (such as the DCE) lack a delegation facility, the need for delegation has existed for some time. Often, to accomplish a delegation, a server acting as an intermediary assumes the client's identity when performing operations on the client's behalf. In other words, the intermediary *impersonates* the client.

We feel that impersonation is dangerous, and that most uses of impersonation are better modeled as true delegations with chained identities. However, we acknowledge that impersonation is necessary for compatibility with existing administrative setups and particular application sets; hence our delegation facility includes a means for clients to permit servers to impersonate them (and a means for a server to act as an impersonator). This facility is part of the client programming model and will be discussed in that section.

**PROGRAMMING MODEL**

The programming model decouples the manipulation of identity from the details of the security protocols. The interface is logically divided into a portion of interest to clients and a portion used by servers.

Clients are primarily concerned with establishing their identity and the necessary controls on how that identity is used. The model provides a *login context* as an abstraction of the client's identity. A login context is an application level opaque handle to the data, including the EPAC and tickets, needed by the underlying protocols. The security protocols are enabled in the RPC communication system by associating a particular login context with a communication session between a client and a server.

The details of how login contexts are shared by application processes are dependent on the operating system on which the application is running. For most DCE environments, however, a default login context for a given principal is created when a process is created for that user through the OS greeting function. Applications will inherit this default login context, but they are also free to create new contexts that reflect a different set of allowable privileges and/or controls. An application may also create new contexts for a different principal if the application has access to that principal's key.

Servers are the reference monitors for the data they manage. In general they are concerned with extracting the privilege attributes associated with a given remote request. These attributes are then generally passed on to the standard access control algorithm to determine if the client is authorized to perform the requested operation. The caller's privilege attributes may also be used for auditing the operation or otherwise recording information about the participants in the call chain.

**Client Programming Model**

A client must decide for each remote call that it makes whether it is performing the operation on its own behalf or on behalf of a caller. This ought to be obvious from program context. Additionally, a client that acts on behalf of a caller must decide on whether to chain its identity with that of its caller (i.e. be a delegate), or (try to) assume the identity of the caller (i.e. be an impersonator).
Operationally, the client must setup (or reuse) a login context that is appropriate for its role as either initiator, delegate, or impersonator, and perform the operation under that login context.

We provide three calls that setup login contexts: become_initiator, become_delegate, become_impersonater. All three calls allow the setting of delegation-related restrictions, extensible restrictions etc. Each creates a new login context as a return value. The calls differ only with regard to the identity information passed in. Become_initiator takes an existing login context as an input parameter. Become_delegate takes an existing login context (i.e. the delegates identity) plus a reference to the identity to chain with. Become_impersonater takes only the identity to impersonate, but needs no existing login context.

\[
\text{new\_login\_context} = \text{become\_initiator} ( \\
\text{my\_login\_context}, \\
\text{delegation\_type\_permitted}, \\
\text{delegate\_restrictions}, \\
\text{target\_restrictions}, \\
\text{optional\_restrictions}, \\
\text{required\_restrictions}, \\
\text{permit\_initiator\_compat\_mode}, \\
\text{error\_status} ) ;
\]

Note that when you use an existing login context in the become_delegate call only the base identity from the login context is used. The restrictions that were present in the login context are replaced by those explicitly passed as parameters.

The compatibility_mode parameter is an enumeration with the following values: \text{initiator}, \text{direct\_requester}, \text{initiator\_if\_possible}, \text{none}.

\[
\text{new\_login\_context} = \text{become\_impersonater} ( \\
\text{callers\_identity}, \\
\text{delegation\_type\_permitted}, \\
\text{delegate\_restrictions}, \\
\text{target\_restrictions}, \\
\text{optional\_restrictions}, \\
\text{required\_restrictions}, \\
\text{error\_status} ) ;
\]

Note that there is no compatibility mode argument. If the direct requester's identity was actually a chained identity, whatever compatibility mode was used there is retained.

Server Programming Model

The server-side API is extended to allow applications to extract the privilege attribute set for each participant in a chained identity. We include calls to extract the PAs of the initiator of the operation the PAs of each delegate the delegation and extensible restrictions placed by each participant.

COMPARISON OF MODEL AND MECHANISM WITH OTHER WORK

The model for delegation proposed here has been developed independently of, but bears a striking resemblance to, the model proposed by Gasser and McDermott [7]. In both models composition the privilege attributes for all principals involved in an operation is combined with extensions to the authorization model to allow the expression of the role of intermediaries in that operation. Significant differences exist in the details of the design given that the DCE uses shared secret key authentication and uses the Privilege Server [4] as...
a delegation server [14] while the Gasser and McDermott model uses public key authentication methods.

Other workers have concerned themselves with mechanisms for trustworthy transmission of delegated identities. Varadharajan et. al. [15] proposes a method for chaining certificates in a shared-secret key environment as well as a mechanism for nesting delegation tokens in a public key environment. Karen Sollins [13] provides a mechanism for nesting shared-secret key delegation tokens. Both of these mechanisms for shared-secret key delegation tokens require target servers to contact the authentication service. This is inconsistent with the design goals of the DCE (as argued in [4]) which strive to reduce total system overhead by moving to a push model for privileges - thereby moving the collection of authorization data away from servers and to clients. Consequently we have developed our pushed token mechanism for nesting delegation information.

**SUMMARY**

We've presented a model and design for delegation in a distributed object-oriented environment. We were strongly motivated by the notions of protection of resources and protection of flow of identity:

We believe that each object in the system should have as much knowledge as is practically possible when making an access control decision in order to best protect its resources. Thus our delegation system permits a server to know all the participants in a chained call and to distinguish the rights granted to intermediaries from those granted to an operation's initiator.

We also believe that all principals in the system should have the ability to control the uses of their identities by objects that are not under their control. To that end, our system allows clients to place restrictions on the use of their identity in chained calls.

Our delegation system capitalizes on existing DCE trust mechanisms and is compatible with existing DCE applications. It is implementable with only modest changes to the DCE security system.

**BIBLIOGRAPHY**


