OSF DCE SIG
Request For Comments: 3.0

J. Pato (HP)

June 1992

EXTENDING THE DCE AUTHORIZATION MODEL TO SUPPORT PRACTICAL DELEGATION (Extended Summary)

1. INTRODUCTION

This paper describes the need for a delegation facility in a distributed computing environment. It then presents extensions to the DCE authorization model to support practical delegation. These extensions leverage and extend existing DCE protocol elements and mechanisms. Additions to the DCE programming model are outlined and shown to be small and well-contained.

1.1. Statement of the Problem

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.

In a distributed object oriented environment, intermediate objects are used to 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 of the operation. 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 which runs with the identity of the initiator, but loses the benefits of distribution. It may retain distribution and run as a privileged principal that has full access to all services it abstracts — but then it would have to implement the access control function for each of these services. 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 which 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) make 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. Delegation allows an intermediary to operate on other objects on behalf of an initiator in a manner that both reflects the initiator of the operation and is distinguishable from a normal request from the initiator.

The model for delegation proposed here [Pato 92] has been developed independently of, but bears a striking resemblance to, the model proposed by Gasser and McDermott [Gass 90]. In both models, a mechanism for composing 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 (a.k.a. delegates) 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 [Pato 90a] as a delegation server while the Gasser and McDermott model uses public key authentication methods.

2. OVERVIEW OF THE DCE DELEGATION MODEL

The delegation model consists of two components: composition of privilege attributes to accomplish a delegation; and extensions to the DCE access control model [Pato 90b] to check access of the composed principals.

Constrained or restricted delegations may also be achieved through an extension to the representation of privilege attributes (PAC's).

2.1. Composition of Principal Identities

Delegation information is an extension to the DCE privilege attribute certificate (PAC) data. When delegation is activated, a target server will receive an extended PAC that contains the privilege attributes for the initiator of the chain of operations as well as the privilege attributes for each intermediary involved in the chain. In the notation used in [Gass 90] and by Butler Lampson [Lamp 91]

Delegate **for** *Initiator*

indicates that the intermediary is operating on behalf of the initiator. When more than one intermediary is involved the PAC will contain the privilege attributes for all of the intermediaries and may be denoted as

DelegateN **for** *DelegateN-1* **for** ... **for** *Initiator*

2.2. Access Control List Entries for Delegation

DCE 1.0 access control lists contain entries that identify the access rights to be granted to principals bearing certain privilege attributes. When a server is presented an extended PAC, the authorization algorithm verifies that a privilege attribute for each principal involved in the operation grants the necessary rights to perform the operation. Unlike the method proposed by Paul Karger [Karg 86] the order of intermediaries is not considered significant. The standard authorization algorithm is extended as follows:

Delegation Access Control Algorithm:

```
 (i) Check Each Principal:
     FOR EACH Privilege Attribute Set IN Extended PAC DO
         Apply standard algorithm
         IF access mode is denied THEN
             Deny Access
         ENDIF
     END
(ii) Grant Access
```

The direct approach to delegation authorization checking applied above allows existing DCE services to support delegation without change to ACL format. While this provides the simplest migration path from the current system, it has the unfortunate consequence that every intermediary involved in an operation is granted the ability to perform the operation of their own initiative. It is preferable to be able to specify that a given principal or group can be an intermediary operating on behalf of an authorized initiating principal without granting the intermediary the ability to perform the operation on their own.

To support this extension for delegation, the standard entries are extended with a set of entries that only apply to principals acting as intermediaries. The extended *delegate* entries allow intermediaries to be listed on the ACL without granting those intermediaries the ability to operate on the target object directly. The modified authorization algorithm is now as follows:

Extended Delegation Access Control Algorithm:

```
 (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

2.3. An Example of the Model

Figure 1 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 the associated ACL.

```
+------+      +----------+
( User )----> ( Document )
+------+      +------+---+
             /      (ACL)
            /       +---+
           /
  +-------+     +----+
  ( Graph )---> ( SS )
  +---+---+     ++---+
     (ACL)      (ACL)
     +---+      +---+
```

Figure 1. Compound Document Components

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 also enables delegation and then proceeds to compose a delegated identity of *D* **for** *U* and performs the *view_graph* operation on the Graph.

```
User:           U  view_graph
User_delegate:  D  view_graph
```

Figure 2. ACL for Graph Object

The Graph process receives the *view_graph* operation from the Document object. It consults the ACL shown in figure 2 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.

```
Any_other:  obtain_range_data
```

Figure 3. ACL for Spreadsheet Object

Finally the Spreadsheet process receives the *obtain_range_data* operation. Applying the ACL shown in figure 3 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.

3. DETAILS OF THE DELEGATION MECHANISM

Gasser's system uses public key methods to chain delegation certificates. Vijay Varadharajan [Vara 91] 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 [Soll 88] 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 [Pato 90a]) 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 a different mechanism for nesting delegation information.

DCE 1.0 implements a shared-secret key KDC that supports the MIT Kerberos V5 protocol [Kohl 92]. This protocol contains mechanisms for authentication forwarding and proxy that provide some of the mechanisms necessary for delegation. These mechanisms are already employed in DCE 1.0 by the Privilege Server when constructing a PAC. The privilege server is a delegation service as described by Clifford Neuman [Neum 91].

We capitalize on the trust already present in the DCE privilege server (an online certification authority) by nesting delegated privilege attributes in an extended PAC. To enable delegation, a client obtains a delegation token from the privilege server and transmits it to the intermediary. The intermediary will then present this token to the privilege server to obtain a new ticket to a desired target server. This new ticket will contain the delegation information in the extended PAC.

3.1. The Delegation Token

A client that desires to delegate access to a server obtains a delegation token from the privilege server. The delegation token is signed by the privilege server to guarantee that the privilege attributes are not modified by either the client or server involved in the delegation. The flow of information, using authenticated RPC with DCE authorization enabled, is as follows:

C $\rightarrow$ PS : Delegate, Target
PS $\rightarrow$ C : { Delegate, Target, PA(C) } SKps

The client may identify the desired delegates and the eventual targets of the delegation. The privilege server seals that information along with the privilege attributes of the client, which are obtained from the PAC transmitted with the request, into the delegation token.

3.2. Enabling Delegation

A client process may enable delegation by annotating a login context with the allowable set of delegates and target principals. Once the login context is so annotated, operations using the context will transmit the appropriate delegation token to the server servicing the operation.

A server becoming a delegate transmits the delegation token received from its caller to the Privilege Server accompanying a request for a new extended PAC. The Privilege Server verifies that the server is a legitimate delegate and that the target principal is correct. If the eventual target is left unspecified in the delegation token, the intermediary will be able to serve as a delegate to all principals — otherwise it is restricted to being a delegate for the client to the specified target. The privilege server replies with the new privilege ticket granting ticket (PTGT) or privilege ticket (PTKT) as appropriate.

S $\rightarrow$ PS : { Delegate, Target, PA(C) } SKps, Target
PS $\rightarrow$ S : PTKT

3.3. Chaining Delegation

When an intermediary elects to delegate to another principal, it performs the same operations outlined for a client above. The intermediary performs the request for a delegation token only after establishing a login context that reflects the delegation from its predecessor. The chaining information is then derived by the privilege server from the PAC on the request for the delegation token.

To reduce the number of network operations, the two operations on the privilege server are combined into a single operation to retrieve both a delegation token and an extended PAC.

3.4. Examining Composite Privilege Attributes

The composite privilege attributes are certified by the privilege server and can be examined directly by a target server. The DCE programming model is extended with accessor functions to extract the privilege attributes for initiator and each of the intermediaries from the RPC runtime. The privilege attributes are sealed in a Kerberos ticket in a manner that allows DCE 1.0 servers to access the privilege attribute data of the immediate invoker of an extended PAC. This preserves inter-operability with DCE 1.0 servers.

3.5. Extensions for Restricted Delegation

The DCE 1.0 model of privilege attributes allows for extension to the data contained in a PAC — but provides no application level mechanisms for extending the data. Furthermore there is no policy articulated in the DCE design for how extensions of the privilege attribute data should be interpreted.

Adding a delegation facility to the DCE heightens the need for a mechanism to extend privilege attribute data with additional restrictions on the use of the privilege attributes. While we do not define any specific new restriction types for the DCE, we do provide a model and mechanism for adding required and optional application specific restrictions. This extension allows applications to implement a variety of security models and policies.

Restrictions are carried in both delegation tokens and in PAC's. Each restriction is encoded as an uninterpreted bag of bytes. It is the responsibility of application protocols to tag the bytes with enough descriptive information to be able to decode the restrictions.

3.5.1. Required restrictions

Adding required restrictions to a PAC or to a delegation token will restrict the activities that a server can perform. Examples of required restrictions that applications may define include: time-of-day restrictions; single or limited use tokens; interface selection; target object selection, etc.

Required restrictions must be understood by the application receiving the PAC. If the application is unable to decode a required restriction that appears in a PAC, it must reject access.

3.5.2. Optional restrictions

Optional restrictions are processed like required restrictions, but their effect may be limited to a given set of applications. Applications that are unable to decode a given optional restriction are free to ignore its presence.

4. EXTENSIONS TO THE DCE ACL MANAGER TO SUPPORT DELEGATED IDENTITIES

The reference implementation of the DCE ACL manager is extended to support the extended delegation access control algorithm described above. In addition it extracts any required restrictions from the PAC. If any required restrictions are present it will deny access. Optional restrictions are ignored as none are defined for base DCE components. Applications using optional restrictions will extend the reference implementation with the appropriate checks.

To fully support the extended access control algorithm, the following ACL entry types are defined:

(a) user_obj_delegate
(b) user_delegate
(c) foreign_user_delegate
(d) group_obj_delegate
(e) group_delegate
(f) foreign_group_delegate
(g) other_delegate
(h) foreign_other_delegate
(i) any_other_delegate
(j) extended_delegate

Supporting delegation through these extended type tags instead of modifying the data structures to define a delegate flag allows the reuse of existing data storage techniques. No format changes are needed since the delegate version of each of the access types has the same representation found in the non-delegate form.

5. ENCODING AND TRANSMISSION OF DELEGATED IDENTITIES IN THE DCE AUTHENTICATION/AUTHORIZATION PROTOCOL

As in DCE 1.0, identity information is transmitted in a PAC. The PAC is embedded in the *authorization data* field of a version 5 Kerberos ticket. An extended PAC is of the following format:

```
+-----------------+
| Header | Length |
+-----------------+
|      PA of      |
| Direct Invoker  |
+-----------------+
|      PA of      |
|    Initiator    |
+-----------------+
|     PAs of      |
|  Delegate List  |
+-----------------+
|    Required     |
|  Restrictions   |
+-----------------+
|    Optional     |
|  Restrictions   |
+-----------------+
```

Figure 4. Extended PAC

Delegation tokens are encoded in a version 5 Kerberos ticket. A delegation token is a ticket to the privilege server that contains an extended PAC as well as bearer and end-service control information in the authorization data field. The extra control information precedes the PAC data to prevent the delegation token from being used in normal authenticated RPC transactions.

```
+-----------------+
|   Legitimate    |
|  Delegate List  |
+-----------------+
|   Legitimate    |
|   Target List   |
+-----------------+
|    Invoker's    |
|  Extended PAC   |
+-----------------+
```

Figure 5. Delegation Token

Delegation tokens and tickets bearing extended PAC's are obtained from the privilege server through the new *rpriv_get_delegation* operation. This operation is invoked by the RPC runtime and is not intended for direct use by application code. Details of this protocol are described in the full paper.

```
void rpriv_get_delegation (
      [in]       handle_t         handle,
      [in]       unsigned32       authn_svc,
      [in]       unsigned32       authz_svc,
      [in]       rpriv_pickle_t   *del_token_req,
      [in]       rpriv_pickle_t   *del_tkt_req,
      [out]      rpriv_pickle_t   **del_token_rep,
      [out]      rpriv_pickle_t   **del_tkt_rep,
      [out,ref]  error_status_t   *st );
```

The *handle* argument is a normal rpc handle that allows the privilege server to access the caller's privilege attributes.

The *authn_svc* and *authz_svc* arguments specify the authentication and authorization protocols in use for the call. As in the other privilege server operation, only the shared-secret key authentication protocol and the DCE authorization protocol are supported by the implementation of the privilege server.

Tbe *del_token_req* and *del_tkt_req* arguments are byte streams that contain the Kerberos V5 KRB_TGS_REQ protocol to obtain either a new delegation token, a delegated ticket or both.

The *del_token_rep* and *del_tkt_rep* are the KRB_TGS_REP messages that provide the output data from the protocol.

6. CHANGES TO THE PROGRAMMING MODEL

The DCE programming model is expanded with a small number of operations to support this delegation model. The *sec_login* interface is extended with operations used by a client to enable and disable delegation and by a function used by servers to become delegates. A new interface *sec_cred* is created for access to privilege attributes. The absence of this interface as a set of accessor functions to PAC's was an oversight in DCE 1.0.

6.1. Changes to Sec_login Interface

```
void sec_login_enable_delegation (
      [in]   sec_login_handle_t   *login_context,
      [in]   sec_delegate_list_t  *legit_delegates,
      [in]   char                 *target,
      [in}   byte_list_t          *optional_restrictions,
      [in]   byte_list_t          *required_restrictions,
      [out]  error_status_t       *st );

void sec_login_disable_delegation (
      [in]   sec_login_handle_t  *login_context,
      [out]  error_status_t      *st );
```

The *sec_login_enable_delegation* operation is used by a client to enable delegation. Subsequent RPC calls that use DCE authorization will transmit a delegation token to the server implementing the remote call. Delegation is turned off for a given login handle via the *sec_login_disable_delegation* operation.

The *login_context* argument identifies the login handle to be operated on.

The *legit_delegates* argument provides a list of principals that are to be considered legitimate delegates. This information will be encoded into the delegation token obtained for this context. If this list is empty, any server principal will be allowed to be a delegate for the client. The *target* argument defines the end-server principal that can receive delegated requests. If this argument is not specified, the delegate will be able to operate on any target server principals. The *optional_restrictions* and *required_restrictions* are lists of the respective restrictions to be applied to the delegation request.

```
void sec_login_become_delegate (
      [in]       sec_login_handle_t  *login_context,
      [in]       rpc_authz_handle_t  *priv_attributes,
      [in]       char                *target,
      [in]       byte_list_t         *optional_restrictions,
      [in]       byte_list_t         *required_restrictions,
      [out,ref]  sec_login_handle_t  *new_login_context,
      [out]      error_status_t      *st );
```

The *sec_login_become_delegate* operation is used by server processes wishing to be a delegate for their caller. This operation will yield a new login handle which refers to the principal identity composed from the server's login handle and the privilege attributes delegated by the client.

The *login_context* argument identifies the server's identity.

The *priv_attributes* argument refers to the clients privilege attributes — including the needed delegation token as exported by the RPC runtime.

The *optional_restrictions* and *required_restrictions* are lists of the respective additional restrictions to be applied to this delegation.

The *new_login_context* argument is an output handle that refers to the composite principal information.

6.2. New Sec_cred Interface

The sec_cred interface provides accessor functions for extracting privilege attributes from the authorization data maintained by the RPC runtime.

```
void sec_cred_get_initiator (
      [in]   rpc_authz_handle_t  *priv_attributes,
      [out]  sec_id_pac_t        *dce_pac,
      [out]  error_status_t      *st );
```

Sec_cred_get_initiator extracts the privilege attributes for the initiator of an operation.

```
void sec_cred_get_delegate (
      [in]      rpc_authz_handle_t  *priv_attributes,
      [in,out]  sec_cred_cursor_t   *cursor,
      [out]     sec_id_pac_t        *dce_pac,
      [out]     error_status_t      *st );
```

Sec_cred_get_delegate is an iterator function to extract the privilege attributes of each delegate involved in an operation.

```
void sec_cred_get_optional_restrictions (
      [in]      rpc_authz_handle_t  *priv_attributes,
      [in,out]  sec_cred_cursor_t   *cursor,
      [out]     byte_list_t         *optional_restrictions,
      [out]     error_status_t      *st );

void sec_cred_get_required_restrictions (
      [in]      rpc_authz_handle_t  *priv_attributes,
      [in,out]  sec_cred_cursor_t   *cursor,
      [out]     byte_list_t         *required_restrictions,
      [out]     error_status_t      *st );
```

sec_cred_get_optional_restrictions and *sec_cred_get_required_restrictions* are iterator functions to respectively extract the optional and required restrictions from the authorization data maintained by the RPC runtime.

7. FULL IDENTITY FORWARDING (IMPERSONATION) FOR REMOTE SYSTEM LOGIN

Impersonation is often useful for implementing remote system login utilities. The mechanisms for accomplishing this are present in Kerberos V5 and can be found in extended versions of telnet and rlogin. It is sometimes also useful to allow RPC servers to impersonate their caller. To accomplish this we add to the *sec_login* interface extensions the following function:

```
void sec_login_enable_impersonation (
      [in]   sec_login_handle_t   *login_context,
      [in]   sec_delegate_list_t  *legit_delegates,
      [in]   char                 *target,
      [in]   byte_list_t          *optional_restrictions,
      [in]   byte_list_t          *required_restrictions,
      [out]  error_status_t       *st );
```

This operation behaves analogously to the *sec_login_enable_delegation* function, except that it transmits a delegation token that allows the server to be indistinguishable from the initiator.

8. ACKNOWLEDGEMENTS

The delegation model described in this paper was developed in collaboration with Marlena Erdos.

REFERENCES

[Gass 90] Morrie Gasser and Ellen McDermott, ‘‘An Architecture for Practical Delegation in a Distributed System’’, in *Proceedings of the 1990 IEEE Symposium on Security and Privacy*, IEEE Computer Society, 1990.

[Karg 86] Paul A. Karger, ‘‘Authentication and Discretionary Access Control in Computer Networks’’, *Computer Networks and ISDN Systems*, Vol. 10, No. 1, January 1986. This paper was reprinted in *Computers and Security*, 5 (1986).

[Kohl 92] John Kohl and B. Clifford Neuman, ‘‘The Kerberos Network Authentication Service’’, INTERNET-DRAFT RFC, revision 5, 17 April 1992

[Lamp 91] Butler Lampson, Martin Abadi, Michael Burrows, and Edward Wobber, ‘‘Authentication in Distributed Systems: Theory and Practice’’, in *Proceedings of the 13’th ACM Symposium on Operating System Principles*, October 1991.

[Neum 91] B. Clifford Neuman, ‘‘Proxy-Based Authorization and Accounting for Distributed Systems’’, Technical Report 91-02-01, Department of Computer Science and Engineering, University of Washington, 1991.

[Pato 90a] Joseph N. Pato, ‘‘DCE Authorization Services — Privilege Server’’, OSF DCE Specifications, 1990.

[Pato 90b] Joseph N. Pato, ‘‘DCE Access Control Lists (ACL’s)’’, OSF DCE Specifications, 1990.

[Pato 92] Joseph N. Pato and Marlena Erdos, ‘‘Authorization in the Face of Delegated Identities in a Distributed Object Oriented System’’, unpublished internal HP document, March 1992.

[Soll 88] Karen R. Sollins, ‘‘Cascaded Authentication’’, in *Procedings of the 1988 IEEE Symposium on Security and Privacy*, IEEE Computer Society, 1988.

[Vara 91] Vijay Varadharajan, Phillip Allen, Stewart Black, ‘‘An Analysis of the Proxy Problem in Distributed Systems’’, in *Proceedings of the 1991 IEEE Symposium on Security and Privacy*, IEEE Computer Society, 1991.

AUTHOR’S ADDRESS

Joseph N. Pato
Distributed Object Computing Program
Hewlett-Packard Co.
250 Apollo Drive
Chelmsford, MA 01824
USA

Internet email: pato@apollo.hp.com
Telephone: +1-508-436-4350