Open Software Foundation
Request For Comments: 8.2
T. Hunwick (EDS)
August 1996

SECURITY REQUIREMENTS FOR DCE

1. INTRODUCTION

The DCE Security Working Group — a joint subgroup of the DCE, Security, and Management SIGs — has proposed and voted on requirements for DCE security extensions. This RFC describes each of these requirements, the resultant voting process, and provides some analysis.

Initially, a laundry list of requirements was proposed by individuals; all suggestions which met three criteria were accepted:

(a) The requirement must be understandable (the precise explanation of each requirement, if not immediately obvious, was worked out in committee).

(b) The requirement must not duplicate another requirement on the list.

(c) The requirement must be within the scope of the OSF DCE effort.

Finally, a vote was taken among the working group members, on a one-organization/one-vote basis, for the priority of each requirement for subsequent DCE release(s) (2.0 and beyond).

In the sections that form the main body of this document, the ‘‘official’’ statement of each requirement, as agreed to by the working group, is indented and marked by a special stylized symbol (‘‘⇒’’). Much of the accompanying explanatory material is due to various authors/editors of this RFC (especially Bob Blakley), as an unofficial elaboration to aid the reader. In cases where a requirement has been partially satisfied in some release of DCE, its continued appearance here signifies a desire to finish the work (lift restrictions, whatever).

The identification code scheme (indicated by ‘‘tags’’ in curly brackets), and the classification of items into groups, were done after the fact and are artificial; they are retained for internal reference purposes, but each item should be understood to stand alone. The correspondence between ID tags and section numbers in this document can be gleaned from the Table of Contents (see below).

Note in particular that there are some ‘‘gaps’’ in the tags. These merely indicate various editorial manipulations, and have no other significance. For example, in the present document, the requirement ‘‘{PK24} Support for Multiple Cryptographic Algorithms’’ has actually been moved to an appendix, as an editorial decision — but it could equally have been simply deleted, without renumbering the subsequent requirements in the {PK} series.

This edition of RFC 8.2 includes votes received at the June 1996 meeting of the Open Group Security working group in Barcelona, Spain, and via email for several weeks before and after the meeting. The votes can be found in Appendix C along with a voting summary (annotated to indicate the ‘‘top ten’’ requirements).

1.1. Table of Contents

This document has been arranged in lexicographic (‘‘alphabetical’’) order, by tag, as indicated by the following Table of Contents:

2. {A} AUTHORIZATION

2.1. {A2} Authorization Based on Authentication Style

⇒ Permit a server to base authorization on ''authentication style'' (the method by which the user logged in), without changing the authorization model.

''Authentication style'' refers to the use of various technologies such as passwords, smart cards, biometrics, etc. A vote for this requirement explicitly prohibits a scheme where the ACL writer can specify an authentication style on the ACL, as it was felt that the ACL writer would not generally have enough information to make a useful decision and any such scheme would be too complex to use. One proposed method for implementing this without impacting any ACL model is to assign the user to different groups depending on the authentication method used.

Impact if requirement not satisfied: Without the proposed support, principals who use ''strong authentication'' mechanisms can't be granted additional authority via the authentication mechanisms of DCE. This significantly limits the value of strong authentication.

2.2. {A3} Authorization Based on Capabilities

⇒ Permit a server to control access based on the client's presentation of, and verifiable legitimate possession of, an unforgeable capability designating permitted rights to the information.

The ''capability model'' referred to here is one in which the client obtains, prior to accessing a server, a kind of ''ticket'' for some specific information (e.g., a specific file or set of them). This ticket could be in an extended PAC or separately transmitted. A server receiving and verifying such a ticket should not need to make further access control checks based on the client's identity or privileges (although it is permitted to). In principal, capabilities provide a great deal of flexibility for access control and may not be very difficult to implement within the DCE framework. However there is some controversy as to whether this flexibility can be managed in a practical manner.

Impact if requirement not satisfied: Authorization cannot be managed on a principal-by-principal basis without this support (in other words, the resource-oriented ACL model currently supported by DCE continues to be the only one available.)

2.3. {A4} Authorization Based on Location

⇒ Extend the authorization mechanism to permit restrictions based on the ''location'' of the user or delegates.

The server must be able to verify the locations. Location may be encoded as the (principal) name of the system into which the user logged in, but might more usefully be represented as the name of a group of systems, or another orthogonal naming scheme. A vote for this requirement neither requires nor prohibits the location to be specified on ACLs.

Impact if requirement not satisfied: Without this support administrators can't discriminate among laptops and mainframes when stating authorization policies.

Note: This functionality has been described as part of the DCE 1.1 Extended Login Feature (see ''DCE 1.1 Functional Specification, Extended Login and Password Management'', Revison 1.5, March 4, 1994, section 7.2 entitled ''Environmental Parameters and Registry Attributes Support''). Namely, the `sec_login_attach_env` flag and the `attach_environment` ERA (in the user's `login_set` attribute set) are used to construct a login context indicating ''user-with-machine-as-delegate''. This would respond to requirement {A4}. However, in DCE 1.1 this solution was not been completely implemented.

2.4. {A5} Anonymous Access

⇒ Permit an environment whereby a server can verify a client's right to access information without the server's gaining knowledge of the client's identity but with the possibility of maintaining individual accountability and auditability.

The term ''identity'' in this requirement refers to the ''individual identity'' that uniquely isolates the individual user, not group identities that pertain to a number of individuals. Authorization in such a case would be based on the client's other attributes in the PAC, rather than on the client's individual identity.

The accountability requirement implies an approach where the server receives the client's individual identity in a sealed form that the server cannot decipher, but which the server can write to an audit log for subsequent retrieval by some trusted authority.

Impact if requirement not satisfied: Without this support some European national privacy laws can't be obeyed by the administrators of a DCE system.

2.5. {A7} Authorization Based on Security Attributes

⇒ All security attributes of the user and delegates that the server can verify (obtained either from the client or by access to other information) should be useful as input to an authorization decision. The API should hide from the server application, as much as possible, the detailed syntax and semantics of this information. Authorization services should provide abstractions to track changes to security attributes of users in a way that is opaque to server applications.

In DCE today servers grant access based on the principal name and group memberships (and, in DCE 1.1, delegation), which servers can obtain in a verified manner from a PAC that the server verifies. In the future, as other attributes are added to the PAC or verified by the server through another mechanism (e.g., by querying a third party), a vote for this requirement is a vote to enhance the authorization services to permit access to be based on these attributes.

There is also a strong statement that the authorization API must be such that applications should not necessarily have to worry about the nature of these attributes. For example, it should be possible to add in the future a new user attribute called ''clearance'' and an ACL tag of ''minimum_clearance'', and existing applications that use the DCE ACL services should continue to work on objects with these new ACL entries.

The group deferred voting on this item as it was deemed too confusing and would require long discussion.

This requirement should be interpreted as follows: The DCE ''Access Decision Function'' implemented under the DCE API verb `sec_acl_is_authorized()` should be extended (or extensible by vendors or customers) to take into account information in extended PACs as described in DCE RFC 3. Existing DCE RPC servers should be able to get the benefit of the EPAC information support without any code changes (i.e., the EPAC-based authorization functionality should all be implemented ''under'', and hidden by, the `sec_acl_is_authorized()` API verb). A new administrative interface (for describing what information is to be considered by the access decision function, etc...) may be required.

Impact if requirement not satisfied: Without this support you can't really USE {A2}, {A3}, {A4} and {A8} functionality. Clearly groups aren't the only useful authorization information — though they are the only authorization information which can be used to make authorization decisions by the current DCE security code. Essentially this requirement wishes to have the DCE authorization checking algorithm USE the additional information which would be put into a PAC by mechanisms satisfying {A2}, {A3}, {A4}, and {A8}.

2.6. {A9} Mandatory Access Controls (Labeling)

⇒ Transmit labels associated with client principals and processes, for optional enforcement by a server. Provide labels on information against which the client labels can be compared.

This is traditional mandatory labeling and label checking, but without a requirement for NCSC or ITSEC evaluatability. Note the contrast with item {P7}.

Impact if requirement not satisfied: Without this support some government markets probably will not be able to deploy DCE.

2.7. {A10} Eliminate ACLs on Files in DFS

⇒ Currently, the DFS specification requires full POSIX-compliant ACLs on each file in DFS. Experience has shown that users find the interpretation of full ACLs on files to be difficult to understand. As a result, users do not understand what protections actually exist on a file, nor how to set the ACLs to describe the desired access. This requirement is to simplify the authorization model used on files in DFS. This requirement does not suggest removing ACLs on directories, nor does it recommend a specific replacement for file authorization. (A potential replacement which has shown itself useful in practice is the AFS file access control model.)

Impact if requirement not satisfied: User confusion on file protection will likely result in appropriate authorization be set on files. Some information will be inappropriately released and other information will not be accessible when intended.

(Sponsor: Mark Sherman.)

2.8. {A11} IP Addresses on ACLs in DFS

⇒ Permit IP addresses, with optional wildcards, to be placed on directory and file ACLs in DFS.

Impact if requirement not satisfied: <<TBD>>

One common use of DFS is for shared software storage, that is, to be a place where commonly used software is kept for use on many machines in a network. Many commercial software packages are licensed to be used on a single machine or a collection of machines. These machines are distinct from the file server storing the file. If one uses IP addresses in ACLs, one can restrict the access to a file to principals from a specific collection of machines. Hence, by allowing IP addresses on ACLs, an administrator can satisfy the software vendor requirement that a program can be used only from a collection of machines, regardless of the user on that machine.

(This customer requirement came from the AFS customer base and has already been implemented in Transarc's AFS.)

(Sponsor: Mark Sherman.)

3. {D} DELEGATION

3.1. {D1} Full Delegation

⇒ Permit a client to explicitly specify that its current access rights should be transferred to a server, so that the server can access remote resources using the client's rights. Delegation should be available through all authentication mechanisms and APIs, including RPC. It must be possible for the user to control whether or not delegation takes place.

In this scheme, the delegate is permitted to fully impersonate the user (for some period of time), and this delegation should be transferable through RPC and any other authentication protocol. For any delegation scheme, there is a strong requirement that the user should be able to specify that delegation must not happen.

Note: PARTIALLY satisfied by RFC 3.0 work in DCE 1.1. However, 1.1 has the restriction that the chain of delegates must lie entirely in the client's cell (though the final target server can be in a different cell).

3.2. {D3} User-to-User Authentication

⇒ A server without access to a long term key should be able to authenticate itself. For example, servers that do not have long term keys are those activated on diskless nodes by users who log into them. This requirement is one that permits a user's process to act as a server.

Note: PLANNED for DCE 1.2.

Description of the Problem: The goal of this requirement was to enable applications that do not have access to a principal's long term key to be the receivers of a protected RPC. Client applications already have this insulation from access to the user's long-term key. These applications only need access to the login context for the user. Servers, however, need current access to the key.

When a ‘‘server’’ is a named, long running system resource — like the name server or the security server — it seems natural that the server is running on a machine with access to the long term key to the identity of that server. This kind of server is not normally associated with a human user, but rather with a pseudo-user identity belonging to the server itself.

This model fits with the well with the type of core DCE services distributed by the OSF today. It will not, however, map well onto all application domains. In particular, some applications need a ‘‘messaging’’ model. While the model may be achieved by using RPCs — it is still necessary for the application’s (architecturally) ‘‘client’’ code to play the role of a DCE ‘‘server’’. This client code performs an RPC to the system service — and the output parameters to the RPC indicate that some activity has been queued. The client code then enters a wait state to get the real reply to the operation (a callback). This kind of messaging model can be found in many Object-Oriented environments as well as some existing communication protocols.

Using today’s security model, there are two ways to accomplish this form of asynchronous activity: The application can force users to make their long term keys available (either in a keytab file {forcing the key to be stored in the clear on the local machine} — or via an application provided key acquisition function {requiring that the user type their key to an arbitrary application}); the application does not perform a protect RPC operation for the callback — but simply uses an unprotected RPC to induce the client code to call the server using protected RPC to retrieve the data. Neither of these two solutions are desirable — the first exposes the user’s long term key to arbitrary applications and potentially to clear-text storage on the machine, and the second contorts the application into an extra communication exchange.

Proposal: Server applications need to be provided with the same sort of insulation that is available for client applications. In particular it should be possible to direct a protected RPC to an application that has access to the login context for its principal rather than just access to the long-term key for that principal. This proposal is a bit more narrow than the way the requirement is currently worded — it would require that authentication be possible if login context be available to the ‘‘server’’ code, while the original statement of the requirement would have admitted other solutions. This narrower scope, however, may be more understandable and covers the needs of many asynchronous or object-oriented environments.

It should also be noted that this requirement can be met using the existing Kerberos V5 protocol. Kerberos allows authentication using a ‘‘user-to-user’’ protocol that involves communicating with the TGS (Ticket Granting Service) with two TGTs (Ticket Granting Tickets). The primary change to DCE would be to detect that this protocol should be used rather than the conventional use of the TGS.

Public Key based protocols can also accommodate this requirement. All that is necessary is the generation of a temporary certificate signed by the principal. The user makes this certificate and corresponding keys available to the application rather than the certificate that identifies the user directly.

The best example of an application which requires this function is the ‘‘talk’’ utility: Hoyt types ‘‘talk Belinda’’, and is given the read end of one pipe and the write end of another pipe. Belinda, if she accepts the ‘‘talk’’ session, is given the other ends of both pipes. Now they are able to write messages to one another and read the answers. In order for Hoyt and Belinda to feel comfortable that their conversation is not overheard, it is necessary for them to ‘‘authenticate one another’’; i.e., for Hoyt to authenticate Belinda and Belinda to authenticate Hoyt. Note that it’s not good enough for Hoyt to authenticate Belinda’s server and Belinda to authenticate Hoyt (which is what would happen if the talk daemon were implemented naively as a standard DCE server) — in this case Belinda is confident that Hoyt is at the other end of the conversation, but all Hoyt knows is that ‘‘someone at Belinda’s machine’’ is talking to him.

3.3. {D3b} Delegation While Absent

⇒ It must be possible for a server to be authorized to acquire a user’s credentials, using the server’s password, and without the user being present to validate the credential acquisition in real time.

Impact if requirement not satisfied: Today’s DCE security model makes it quite difficult to deal with two situations:

(a) A user submits a request to a system service for an action to be performed by the service at a later time. At that later time (potentially much later — maybe a week, or a month, or even a year), the service dequeues the user’s request and executes it, possibly trying to access some resources on the user’s behalf during the execution of the request.

This is the classic ‘‘batch processor’’ scenario. Allowing the batch job executed by the batch processor to execute under the user’s identity is very difficult with today’s DCE architecture.

(b) A user logs on to her machine’s operating system but not to DCE. Later, the user runs an application, which needs to issue a DCE call to get some work done. The user may or may not still be logged on: an example case in which the user will not be logged on is if she executed `at 1145 lpr -Pmyprt foo.txt`, where `at` is the Unix `at` command, and `foo.txt` is a file in the DCE DFS (which will of course have to be read in order to be printed). The operating system needs to acquire a DCE ticket in the identity of the user in order to execute the application and get the requested work done. But it may not have access to the user’s DCE password — especially if the user is not logged on.

Solution: Allow a user or system administrator to designate certain servers as ‘‘trusted’’; allow a ‘‘trusted’’ server to obtain TGTs for users, using the server’s long-term key in place of the user’s password.

(Sponsor: Bob Blakley.)

3.4. {D4a} Integrity Protection of Requests

⇒ Provide a service that permits a client to protect a request in a way that a server can detect if the request was modified by a delegate.

One implementation of this would sign (or encrypt) the client’s request under a key specified in the PAC, thereby preventing a delegate from modifying the request in an undetectable manner. For the purposes of such ‘‘locked’’ requests, the delegate is performing mostly a routing function.

Impact if requirement not satisfied: Without this support, it may not be possible for a server to determine whether delegates have faithfully performed the operation requested by the initiator.

Note: This appears to require PK-like signing; see {PK15}.

3.5. {D4b} Integrity Protection of Responses

⇒ Provide a service that permits a server to protect a response in a way that a client can detect if the request was modified by a delegate.

Note: See {D4a} (as noted there, this appears to require PK-like signing).

4. {E} EXTENSIONS TO OTHER APPLICATIONS AND MECHANISMS

4.1. {E1a} Secure Remote Login

⇒ Remote login should be secured using DCE mechanisms and offered as part of the DCE package.

Candidate applications that fall into this category include `telnet`, `rlogin` and terminal emulation.

Impact if requirement not satisfied: See {E1b}.

Note: See {E1b}.

4.2. {E1b} Secure Remote File Transfer

⇒ Remote file copy should be secured using DCE mechanisms and offered as part of the DCE package.

This category includes `ftp` and `rcp`.

Impact if requirement not satisfied (applies to {E1a} and {E1b}): Users will need to log in separately to DCE and to other popular Unix and Internet utilities. Lack of an enterprise-wide single signon capability is a significant source of customer dissatisfaction with all vendors’ products.

Note: (applies to {E1a} and {E1b}): PARTIALLY PLANNED for DCE 1.2. Namely, the implementation planned for 1.2 will use simple ‘‘Kerberization’’, instead of DCE RPC and/or PACs/ACLs. (An implementation using GSS-API and DCE PACs/ACLs exists, and will potentially be made available through OSF, but there are no plans to include it in DCE 1.2.)

4.3. {E3} Other User Authentication Methods

⇒ Provide interfaces that permit arbitrary site-selected user authentication methods, including smart cards and biometrics devices.

This calls for ‘‘non-password-based’’ authentication technology support. This means mostly extensions to the registry and to the login API to support the additional information and interaction required to allow integration of advanced authentication technologies.

4.4. {E4} More Options for Communications Security

⇒ Permit site and/or application-selectable options for protecting communications, such as choice of encryption algorithms.

Note that DCE today already contains hooks that to identify alternative encryption algorithms in the protocols. This item refers to items such as the provision of additional algorithms, registration of multiple algorithms, additional flexibility in the choice of algorithms by applications, and negotiation in the handling of multiple algorithms.

See also {E7}.

4.5. {E7} Algorithms

⇒ Permit independent selection of algorithms for confidentiality and integrity.

The requirement is that, in the case where messages are protected with both confidentiality and integrity, there be independent selection of the algorithms used for the two mechanisms.

Impact if requirement not satisfied: Without this requirement it may be difficult to export strong-integrity, weak-confidentiality DCE configurations.

See also {E4}.

4.6. {E8} GSSAPI Framework

This requirement proposes a set of ‘‘framework’’-oriented enhancements to the DCE GSS-API implementation, related to incorporation and installation of alternate supporting GSS-API security mechanisms (without recompiling DCE), and to criteria to be employed in mechanism selection. The DCE implementation should also be re-done so that DCE’s protected RPC takes advantage of this framework (probably by layering the protected RPC over GSS-API).

Sub-requirements are listed in the following subsections.

(Sponsor: Bob Blakley.)

4.6.1. {E8a} Service provider interface for GSSAPI

⇒ The DCE implementation of GSS-API shall support an explicit service provider interface to underlying mechanisms, enabling developers and integrators of DCE systems to incorporate separately-developed implementations of those mechanisms in a modular fashion.

4.6.2. {E8b} Dynamic installation of mechanisms

⇒ In addition to {E8a}, the DCE implementation of GSS-API shall enable administrators to install GSS-API mechanism modules within an end system’s installation, and thereby reconfigure the set of GSS-API mechanisms available within that end system.

To some extent, and depending on implementation questions, this may be a vendor/ISV binary issue, having to do with shared dynamically-loadable libraries (which some operating systems might not even support). However, the DCE reference implementation(s) should give vendors/ISVs as much assistance as possible along these lines.

4.6.3. {E8c} Share credentials among mechanisms

⇒ It must be possible for several security mechanisms with different cryptographic technologies and different available combinations of message confidentiality and message integrity to share the same key distribution and credential distribution mechanisms. It must be possible in particular for several different mechanisms to have access to the same copy of a user’s credentials (for example, a user’s DCE PTGT).

Note that satisfying this requirement will entail exposing DCE interfaces through which a DCE PTGT can be retrieved from the credential cache. A possible way to meet this requirement would be to implement a ‘‘credential cache manager’’ with a published interface, and modify the DCE security code to use this interface.

4.6.4. {E8d} Separate cryptographic functionality

⇒ It must be possible to write mechanism code without having to implement crypto function; crypto function should be provided as a system service under an API. Crypto code should be separable from mechanism code so that crypto code can be replaced for exportable versions and so that hardware crypto support can be introduced. Replacement of crypto code and introduction of hardware crypto capability should both be possible without requiring changes to mechanism or application code.

Note that the X/Open GCS-API work should be examined for appropriateness.

4.6.5. {E8e} Mechanism selection by QoP

⇒ The DCE implementation of GSS-API shall enable callers to determine the set of underlying mechanisms which satisfy a specified set of ‘‘quality of protection’’ parameters (e.g., caller-specified requirements for per-message confidentiality and integrity services).

4.7. {E9} Secure DFS Data Transfer

⇒ Provide a way for users and administrators to specify that packet privacy be used for file data transfer.

Impact if requirement not satisfied: Users who wish secure environments for their function shipping capabilities also wish the same security for their data shipping applications. In the absence of this feature, users write their own special purpose secure fine-transfer utilities or programs.

One motivation for adopting a DCE environment is the added security provided. One aspect of the security is the ability to encrypt data as it goes across the network. Currently, there is no interface within DFS to allow this ability to be exploited within DFS for transferring data between servers or from server to client.

Note: PLANNED for DCE 1.2.

(Sponsor: Mark Sherman.)

5. {F} LOGOFF

5.1. {F1} Termination of Login Sessions

⇒ It should be possible for a DCE principal to log off through specific action, or to be gracefully logged off through inactivity time-out (following suitable warning), or to be forced to log off by security administration action.

‘‘Log off’’ in this context means:

(a) An audit record is produced of the act of logging off.

(b) If it maintains state information about principals’ login status, the primary authentication server to which the principal passed authentication information is informed that the principal is no longer present in the system.

(c) Subsidiary authentication services to which authentication has been automated are informed of the principal’s departure, as appropriate (i.e., if they expect this information).

(d) Any applications running on the principal’s workstation under that principal’s responsibility and rights should be terminated. This may cascade to the termination of remote applications depending on individual application functionality.

(e) The workstation from which the principal is accessing the system is purged of all credentials, and of security context information relating to that principal.

The following subsidiary functions should be able to be supported as a consequence of these actions:

(a) It should be possible (for example primarily through the audit record generated, above), to provide the information to support a security manager being able to see who is currently logged in to (or has not properly logged off from) the system.

(b) It should be possible for an auditor to retrospectively identify when principals logged in and logged off from the system.

(c) It should be possible to support a ‘‘no double login’’ security policy.

If an active principal simply pulls the plug on his or her workstation, graceful log off is not possible. However it should be possible for an authentication server supporting the ‘‘no double login’’ policy, on getting a login request from the principal from a second workstation, to inquire of the original workstation whether the principal is still present, get no reply (or a reply indicating that the principal is not known), conclude that the principal is not present there, and initiate its internal log off of the principal from the inoperative workstation.

Impact if requirement not satisfied: <<TBD>>

(Sponsor: Tom Parker.)

Note: It is believed that the requirement as stated above subsumes other proposed requirements, which were the following:

(a) ‘‘{F2} DCE Logoff’’

DCE should implement a ‘‘logoff’’ function which will invalidate the logged-off user’s credentials and generate an audit log entry.

(Sponsor: Bob Blakley.)

(b) ‘‘Detection of Active Connection’’

It should be possible to configure the system so that any given principal must keep an active connection with a watchdog process and that process would attempt to cause the user to be terminated if the active connection is lost.

This requirement is intended to provide automated termination of idle client-server associations established using DCE credentials.

(Sponsor: Chii-Ren Tsai.)

(c) ‘‘{F4} Idle Session Timeout’’

DCE login sessions should have idle session timeouts. Timeout of the session should cause a complete termination of client security data.

This requirement is intended to provide automated invalidation of the DCE credentials associated with idle terminal sessions on the user’s DCE client machine.

(Sponsor: Chii-Ren Tsai.)

(d) ‘‘{F5} Complete Termination on Disconnect’’

Termination of a login session should cause all security data on a client machine to be destroyed.

No data should be retained that would allow a serial user of the machine to impersonate a user that had previously used the machine.

This requirement is intended to provide automated invalidation of the DCE credentials associated with idle terminal sessions on the user’s DCE client machine.

(Sponsor: Chii-Ren Tsai.)

6. {I} IDENTITIES AND GROUPS

6.1. {I1} Global Groups

⇒ Permit the ability of a group to include members in any cell while retaining the shorthand, accountability and auditing capabilities of current DCE groups.

This requirement calls for four things:

(a) The registry administrator of a group's home cell should have control over group membership (i.e., should be able to add or delete members at will and should be able to prevent others from doing so).

(b) Any group, from any cell, should be able to be added to a DCE ACL.

(c) Each DCE cell's Privilege Server should seal a user's active DCE groups into that user's PAC when the user issues a DCE request from within the cell.

(d) DCE Privilege Servers receiving cross-cell ticket-granting requests should preserve the originating cell's group membership information but should be able to seal additional DCE groups (specified in the receiving cell's Registry Service) into the generated ticket.

Impact if requirement not satisfied: Without this support administration of DCE authorization based on groups is difficult in multi-cell environments.

Note: PARTIALLY PLANNED for DCE 1.2. However the implementation planned for DCE 1.2 has the characteristic that in any given protected communication, the ‘‘global’’ group must be in the client's or the server's cell — it can't be in an arbitrary third-party cell.

6.2. {I2} Preserve Rights When Principal Changes Cell

⇒ It should be possible to set up an environment whereby a user registered in a cell does not necessarily lose access to everything when changing registration to another cell.

Given the facts that a user identity consists of a single cell UUID and user UUID, that ACLs contain the client's cell UUID, and that there is no easy way to find and then change all the ACLs in all remote cells on which a given user is listed, a user who moves to a different cell and authenticates himself under that new UUID will lose access to all files protected by DCE ACLs. Satisfying this requirement to retain access requires either the client's PAC to name all the client's old identities, or the server to support or interrogate a facility that maps old names on ACLs to new names.

Impact if requirement not satisfied: Without this support, organizational changes cause large security administration headaches.

6.3. {I3} Role Based Administration

⇒ DCE should provide the facilities to allow for ‘‘role based’’ security administration. This requirement is not meant to create another type of non-discretionary access control mechanism. Role based administration should be looked at as an ease-of-use issue for security administrators. (The work involved in meeting this requirement must include agreeing on an appropriate definition of ‘‘roles’’!)

Impact if requirement not satisfied: Administration of user privileges will remain a complicated and time consuming task.

6.4. {I4} Privileges

⇒ DCE must implement a notion of ‘‘privilege’’ and a means of administering assignment of privileges to users through the Registry.

The notion of privilege here is not the one defined, for example, in the Compartmented Mode Workstation specification. Instead, what is meant here is that special administrative roles (examples might include GUEST, USER, OPERATOR, ADMINISTRATOR, AUDITOR, SUPERUSER, INSTALLER) should be defined and some mechanism defined for permitting users with certain privileges to perform some system operations regardless of DAC (ACL-based) authorization.

(Sponsor: Bob Blakley.)

6.5. {I5} rgy_edit Sub-Administrator Support

⇒ The `rgy_edit` program must be modified to support the concept of ‘‘sub-administrators’’ (a well-known concept in mainframe environments). In a large enterprise, the job of user ID administration is typically divided among many people. Each person is given a certain subset of all the users to administer. The `rgy_edit` program must be modified to allow the security administrator to define subgroups of users that a ‘‘sub-administrator’’ can do basic user ID administration for, while preventing that person from doing administrative functions on users in other sub-domains.

This requirement is an interface requirement. It is not necessary to change underlying administrative functionality as long as the user’s view of administration is simplified (shielding the administrator from the more unsavory aspects of hierarchical names, aliases, etc.).

Impact if requirement not satisfied: If this requirement is not met it means that the task of user ID administration in a large enterprise will remain an overly complex task to do securely and efficiently.

Note: Here and throughout this document, it would be more correct to speak about ‘‘the registry-editing functionality of `dcecp`’’, since `rgy_edit` has been superseded by `dcecp`.

Note: The following 7 requirements ({I6}−{I12}) also relate to the sub-administration problem.

(Sponsor: Charles Blauner.)

6.6. {I6} Allow Additional Namespace Layers

⇒ The DCE registry should allow additional namespace layers.

The general technique for sub-administration is to introduce an additional level of directories above key items in both the `cdsd` and `secd` namespaces in order to allow distinct ACLs to be used to control the administration of subdomains. For example, instead of `/.:/hosts/machine_1` and `/.:/hosts/machine_2`, use `/.:/hosts/dept_x/machine_1` and `/.:/hosts/dept_y/machine_2`. Ideally, the additional directories should be placed as high as possible in the tree. For example, `/.:/sec/dept_x/group/my_group` would be preferable to `/.:/sec/group/dept_x/my_group`. With the first structure, the administrator for `dept_x` sees everything that belongs to him in the same place, rather than spread across the namespace.

Currently, not all components will support this type of scheme. For example, some installation scripts insist that their be no subdirectories between `/.:/hosts` and the hostname directories. The Registry requires that `/principal` be directly below `/.:/sec`, etc. All of these restrictions should be removed.

If it is too expensive to remove some of the restrictions, e.g., putting a layer immediately above a junction (`/.:/dept_x/sec`), the restrictions should be fully documented.

(Sponsor: Hal Lockhart, Dieter Mack.)

6.7. {I7} Allow Empty Directories in the Registry

⇒ The DCE registry should allow empty directories.

Currently, the creation of directories in the `/.:/sec` namespace is a side effect of creating objects. It is not possible to create an empty directory and if a directory becomes empty, it is automatically deleted, regardless of permissions! When creating additional directory levels to support sub-administration it is desirable to allow empty directories with associated object and initial creation ACLs. Empty directories should be allowed in the registry namespace.

(Sponsor: Hal Lockhart, Dieter Mack.)

6.8. {I8} Prohibit Aliases as a Cell-Wide Policy

⇒ It should be able to prohibit aliasing, on a cell-wide policy basis.

Aliases for principals and groups share the same UUID as the primary name, and thus are considered to be the same principal or group when ACL evaluation is performed. Therefore, the use of Aliases can cause various kinds of user confusion and security risks. For this reason, some organizations prefer not to allow the use of aliases at all. The ability to prohibit the use of aliases should be a policy on a cell-by-cell basis.

(Sponsor: Hal Lockhart, Dieter Mack.)

6.9. {I9} Allow a Principal to Belong to No Group

⇒ It should be possible for a principal to be free-standing, and not a member of any group.

Currently, when the registry is created, ‘‘built-in’’ principals such as `cell_admin`, `dce-rgy` and `dce-ptgt`, are put into the group and organization `none`. There are two problems with this. First, they should not be in the same group. Second, `none` does not signify ‘‘no group’’, it is simply a group named `none` that can be specified in an ACL just like any other group.

DCE should allow a principal to belong to zero groups. Also, the process to create the registry should put the various ‘‘built-in’’ principals in distinct groups.

(Sponsor: Hal Lockhart, Dieter Mack.)

6.10. {I10} Facilitate Integrated Login with Sub-Administration

⇒ Sub-administration should be integrated with system login facilities.

In order to permit sub-administration it is desirable to allow principal names like `dept_x/hal`. Many existing login programs do not support this. At a minumum, an integrated login should support all legal DCE syntax, including: `dept_x/hal`, `/.:/sec/principal/dept_x/hal` and `/.../big_cell.osf.org/sec/principal/dept_x/hal`. (Perhaps some abbreviation of the last two should be permitted.)

Ideally, the machine should determine any prefix to its own principal name and apply it by default to the name typed. This would allow the user to type just `hal` when logging into his own machine and others in the same administrative domain, while allowing logins elsewhere in the cell by typing `dept_x/hal`. Since the login program is provided by the platform vendor, the Open Group should provide detailed instructions and sample code to do this.

(Sponsor: Hal Lockhart, Dieter Mack.)

6.11. {I11} Control Integrated Login Using an ACL

⇒ Host identity should be integrated with system login facilities.

With integrated login, a user can login anywhere in the network, based on the principal identity maintained in the registry. However, many organizations wish to control which machines can be used by which users. This can be done using passwd overide, but this is unsatisfactory and inconsistent with the DCE authorization model.

Integrated login should check an ACL (or perhaps an ERA?) to determine if the user is allowed to login to a particular machine. Note that this requires obtaining a PAC. Since the login program is provided by the platform vendor, the Open Group should provide detailed instructions and sample code to do this.

(Sponsor: Hal Lockhart, Dieter Mack.)

6.12. {I12} DFS Support for Sub-Administration

⇒ DFS should fully adopt the DCE security model for administration.

Currently DFS has many adminstrative mechanisms left over from AFS, which do not conform to the DCE authorization model and make sub-administration difficult. Administrative capabilities are controlled by an assortment of mechanisms including bos user lists, admin groups in FLDB server entries and file exporter groups. Another barrier to sub-administration is that the fileset namespace is currently constrained to be flat.

DFS should implement a consistent ACL-based scheme for controlling server operations, fileset manipulation and aggregate managment. DFS should permit a hierarchical namespace for filesets.

(Sponsor: Hal Lockhart, Dieter Mack.)

6.13. {I13} Split Local and Remote Client Configuration Steps

⇒ DCE security should decouple the local and remote steps of client host configuration.

Currently the host configuration scripts combine the steps of configuring the local host and configuring the DCE infrastructure. As a result the local configuration can not be done unless the machine is online to the DCE servers. DCE should cleanly separate these steps, so that local and remote configuration can be done at different times and/or by different individuals.

(Sponsor: Hal Lockhart, Dieter Mack.)

7. {N} AUTHENTICATION

7.1. {N1} Alternate Authentication Mechanism Support

⇒ Support ''traditional'' authentication clients (other than Kerberos) in a DCE client system.

This includes support for integrated management of DCE and non-DCE credentials; for example, this could include extending the DCE login context to provide pointers to Lotus Notes ID files, Netware credentials, etc..., in addition to DCE tickets.

Note: An earlier, incomplete, statement of this requirement called for ''the addition (or at least the support) of other (non-Kerberos) Authentication Services under GSS-API in a DCE environment''. The ''(or at least the support)'' clause, as applied to GSS-API, is superseded by the {E8} series.

7.2. {N2} dce_login Error Feedback

⇒ The `dce_login` program must be modified so that it does not provide any error feedback that could help someone that is attacking a system in a DCE cell. The `dce_login` program should solicit a password whether or not a valid user ID has been entered. The same error message should be returned for both invalid user IDs and invalid passwords.

Note: It is understood that `dce_login` is a ''sample'' implementation for the guidance of vendors/ISVs, not a DCE ''conformance requirement''. Nevertheless, the DCE reference implementation should give vendors/ISVs the best (security) guidance possible, hence the relevance of this requirement.

Impact if requirement not satisfied: If the `dce_login` program is not modified, its use provides an attacker with additional information that can help attackers to penetrate a system.

7.3. {N3} dce_login Password Expiration Handling

⇒ The `dce_login` program must be modified to properly handle password expiration. The program currently allows users to login with an expired password. This means that DCE is not enforcing the password expiration policy set in the cell's registry. The `dce_login` program must also provide a mechanism for changing a password when one has expired.

Impact if requirement not satisfied: Failure to meet this requirement means that the `dce_login` program will continue to fail to enforce the passwords expiration policy set by the cell administrator.

Note: PARTIALLY satisfied in DCE 1.1 (the part about the lockout is satisfied, but not the part about changing an expired password).

7.4. {N5} Login Policy Enhancements

7.4.1. {N5a} Successive failures lockout

⇒ DCE must give the administrator the ability to prevent a principal from logging on to a cell after N successive login failures. The number N needs to be configurable (specified by the Registry policy).

Note: PARTIALLY satisfied in DCE 1.1. Namely, it is satisfied *separately* at the master registry and at its non-master replicas (thus increasing the desired number N by a number-of-replicas factor).

7.4.2. {N5b} Active session indication

⇒ DCE must be able to notify a user at login time if he currently has a session active within the cell (this should be an option).

This calls for a modification to `sec_login` and `dce_login` to notify the user of the last login time and to tell the user if any of those logins is still active (i.e., credentials haven't yet expired).

7.4.3. {N5c} Password expiration date

⇒ DCE must be able to notify user that his password will expire in N days. The number N needs to be configurable (specified by the Registry policy).

This is a requirement on `dce_login`.

7.4.4. {N5d} Periodic lockout

⇒ DCE must allow administrators to prevent users from logging in during specified time periods.

Note: This is satisfied when `sec_login_policy_check()` is implemented (this function appears in the DCE 1.1 extended login spec but will not be implemented in DCE 1.1).

7.5. {N6} Current PAC for Credential Refresh

⇒ The DCE credential refresh process should obtain a ''truly updated'' (not just a ''partially updated'') PAC.

Currently, credential refresh obtains a ''partially updated'' PAC which reflects only group *deletions* since last login. A new login is required to get a PAC which reflects deletions AND *additional* groups.

Impact if requirement not satisfied: Applications which wish to renew credentials via a refresh (as opposed to a re-login), will NOT allow principals to gain the benefits of additional privileges due to new group memberships acquired after login. From a usability perspective, refreshing credentials is preferred over re-login because it allows the same credential file to be used. Thus, applications can offer a credential refresh process which allows periodic re-authentication without interrupting application work. Re-login on the other hand, requires users to exit out of an application and re-enter, as it changes the credential file, and Unix restricts processes from changing the environment variables (credentials are pointed to by the `KRB5CCNAME` environment variable) of their siblings or parents.

Note: There are currently (as of DCE 1.2.2) no plans to address this requirement.

(Sponsor: Jeannine Lonski.)

7.6. {N7} Role-Based Login

⇒ DCE login information should include a username, password, and rolename. Access control should take role-based privileges into account, but accountability (auditing) should be done based on the username or corresponding ID.

This would help implement policies based on least privilege, and/or categories, for example.

(Sponsor: Joe Wallin.)

7.7. {N8} Log-Over

⇒ A ''log over'' function should be provided which requires username, password, and rolename as input. This function should change active username and rolename as specified, and accountability and auditing should take place under the new identities. Execution of log-over should not require termination of any applications running under the original identity.

Impact if requirement not satisfied: Today's DCE makes shift-change operations difficult, since there is no way to change the active user identity associated with a console without shutting down the applications running at that console. Without a log-over function, this problem will continue to exist.

(Sponsor: Joe Wallin.)

7.8. {N9} Multicell Authentication

⇒ Users should be able to authenticate in multiple cells.

Administrators in one cell may not trust the administrators in another cell but they may trust an individual. Therefore, each cell administrator who wishes to allow access to a user would like to provide a login to the user. But the user would like simultaneous access to all authorized cells and current PAC allows only a single login. Therefore, DCE should allow multiple logins.

Impact if requirement not satisfied: <<TBD>>

(Sponsor: Mark Sherman.)

7.9. {N10} Cross Cell Authentication with Local Identity

⇒ In the collaborative research environment, foreign users are given accounts in the local cell, so they can used traditional Unix commands such as telnet and FTP. These users may also have accounts on their own cell. DCE needs a way to allow authentication to the local account based on being authenticated in a foreign cell even when the user IDs do not match.

Kerberos 5 handles this problem via the use of the `.k5login` file and/or the `aname` database. This allows a user to define in the local cell what foreign identities are acceptable for authentication.

DCE needs an equivalent capability, and needs to be able to accept a Kerberos TGT for this purpose. Maybe this information could be stored as an extended registry attribute, and the security server could use this during authentication.

Impact if requirement not satisfied: DCE's inability to provide this mapping and its current method of mapping all foreign users to a single local user are unacceptable in the Unix environment. Even if the Unix system could allow this, many organizations may still insist that the locally assigned account be used when operating with in the local cell. This forces the user to abandon the use of cross cell authentication by either:

(a) Using the standard Unix commands and logon over the network from the foreign cell to the local cell, thus sending the DCE user ID and password over the network in clear text.

(b) Destroying the DCE context or maintaining two DCE contexts, on his/her workstation one for each cell, thus defeating the purpose of single signon.

(Sponsor: Doug Engert.)

8. {P} ATTRIBUTES: PERFORMANCE, ROBUSTNESS, EXTENSIBILITY, INTEGRATION, STANDARDS

Note: The {P} tag originally stood for ''performance'', and the other attributes were subsequently added.

8.1. {P4} Recovery from Security Compromise

⇒ After a detected security compromise, it must be possible to continue operating with an acceptable level of security and functionality. What is considered ''acceptable'' should be configurable by the security administration (e.g., ''hard lockout'' vs. ''graceful degradation'', etc.).

An example of one of the most serious compromises is disclosure of the secret key of a cell's privilege server. A less serious compromise is disclosure of one user's password. DCE must recognize that commercial requirements vary considerably — some organizations want to shut down completely when a compromise is detected, and others are willing to operate completely insecurely, if necessary to keep the business running. Therefore, a configurable range of emergency modes of operation should be provided. Note that this item does not place any requirement on the ability to detect a security compromise.

Note: Even though this requirement remains on the ballot, it needs more clarification. Here are some outstanding questions for people who feel this requirement is important:

(a) Who is the sponsor of this requirement?

(b) What behavior might satisfy this requirement?

(c) What policies for continuity of service should be supported?

(d) How serious must a compromise be before the system should not continue?

(Sponsor: <<TBD>>.)

8.2. {P5} Detection of Denial of Service

⇒ Maximize the ability to detect and diagnose denial of service attacks.

Note: This requirement is known to require clarification, but will be included in the ballot. A sponsor for this requirement is needed.

Impact if requirement not satisfied: <<TBD>>

8.3. {P6} Algorithms for Security Protocol Messages

⇒ Retain flexibility in the choice of algorithms and techniques for protecting messages in the security protocol.

Compare this with {E7}, which asks for the ability to select algorithms to protect the *application data*.

Note: This requirement is known to require clarification, but will be included in the August ballot. It is believed that Denis Pinkas is the sponsor of this requirement; we need confirmation of this, or an alternative sponsor. We also need additional explanatory text.

8.4. {P10b} Kerberos Version 5 Interoperation

⇒ Interoperate with Kerberos version 5. In particular, the DCE security server should be tested to insure that it is a fully RFC 1510 compliant provider of Kerberos 5 services.

Note: What other requirements exist for interoperability? (The old text stated a general requirement for interoperability with secure, non-DCE environments.)

Note: PLANNED for DCE 1.2.

(Sponsor: Joe Pato.)

8.5. {P14} Defaults

⇒ Permit establishment of an environment (e.g., on a per-system or per-cell basis) whereby an application not otherwise interested in security can make maximum use of security features as configured (e.g., in a per-system or per-cell configuration file) by an administrator.

In designing APIs to support security, it should be possible for an application to choose (or automatically use) administrator-configured defaults wherever practical, as most applications, especially the client side of applications, have little basis on which to make a rational security-relevant decision. For example, one default configured by an administrator or user might be whether or not to allow delegation on outbound RPC calls. To enable this capability, the RPC delegation mechanism should allow an application to specify ''default'' as a delegation option.

Note: Some, if not all, of this functionality is supported in DCE 1.1, with the server initialization facility for servers (`dce_server_*()`), and the `binding_callout` ACF attribute (in conjunction with automatic binding) for clients. If additional facilities are required, they should be specified (presumably by a sponsor if one could be indentified).

Impact if requirement not satisfied: Without this support, the strong security features of DCE may typically remain unused. ''Unsophisticated'' users should get some benefit from all OSF's and the suppliers' hard work by default, out of the box.

8.6. {P15} Registry Fractionalization

⇒ To provide local administrative control at remote sub-cell sites within a large geographically-dispersed cell, the DCE security registry should be capable of being selectively replicated, such that the remote site's security replica has a *subset* of the master registry, as needed to provide service at the remote site.

Impact if requirement not satisfied: (1) Local sub-cell control of the DCE cell could be lost in the event of a network service interruption. (2) The remote site's copy of the master registry could be corrupted, causing total cell compromise.

See also {P16}. Compare this requirement with {I5}.

(Sponsor: Bill Estrem.)

8.7. {P16} Security Server Affinity

⇒ The DCE client should be configurable to bind to a specified security server replica (or group of security server replicas).

Impact if requirement not satisfied: In large DCE cells, users logging in have no control over which security server replica(s) they use to get their tickets. This can lead to undesirable performance characteristics in large geographically dispersed cells.

See also {P15}.

(Sponsor: Bill Estrem.)

9. {PK} PUBLIC KEY

9.1. {PK1} Public-Key Authentication

⇒ Allow DCE security to operate without on-line trusted servers.

This is (a strong form of) the requirement for public-key authentication in DCE (as opposed to the current private-key authentication mechanisms).

Currently, the database maintained by ''secd'' contains much information that must be kept confidential (e.g., principal keys) as well as information that must be integrity-protected (e.g., principal privileges). This makes ''secd'' a valuable target for attack, and the presence of confidential data within `secd` makes recovery from a compromise expensive (requiring every principal within the cell to be issued a new key).

This requirement is that such sensitive security data to be either removed completely from DCE, or where that is impossible, that it be moved into off-line agents, which can be better physically protected than the current on-line servers.

Impact if requirement not satisfied: Without this feature, system administrators must make a trade-off between availability of the security service (more security servers), and security (few, well-protected security servers). Recovery from a compromise of any security server will remain a very costly process.

(Sponsor: John Wray.)

9.2. {PK2} Persistent Non-Repudiation

⇒ The ability for a message recipient to prove at a (perhaps substantially) later date the identity of a message's originator.

This requirement assumes the initial co-operation of the message originator in attaching a signature to the message. The message recipient must be able, at a later date, to verify the signature, in such a way as to prove to a third party the message originator's identity. The ''perhaps substantially'' clause in the requirement statement implies that any keys necessary for verifying signatures must be retained in an integrity-protected repository. This requirement differs from {PK4} below in that it would be acceptable here for message originator/content verification to require information that is known only to the message recipient.

Impact if requirement not satisfied: If this feature is not provided by the base DCE services, trusted messaging applications (e.g., expense approval) must implement this functionality themselves. Without a DCE framework for storage and retrieval of signature keys, each such application is likely to solve the problem in its own, incompatible way, resulting in greatly increased management complexity.

(Sponsor: John Wray.)

9.3. {PK2a} Disposable Signing Keys

⇒ This is a specific instance of {PK2}, with modifications for explicit long-term protection. If only symmetric (secret) key technology is used for long-term protection of data, then the keys used to apply integrity protection must be retained for the life of the data, increasing the potential for key disclosure and unauthorized data modification.

Using public-key technology, however, it would be possible for the signature-generating key to be deleted once the signature has been applied (retaining only the signature-verification key). Re-certification for long-term storage (longer than the projected life-time of a given key) may be accomplished without knowledge of the original signature key.

Impact if requirement not satisfied: Keys used to apply signatures to data must be retained for the life of the data, increasing the possibility of undetected modification of audit data.

(Sponsor: Ellen McDermott.)

9.4. {PK3} Establishment of Inter-Cell Trust

⇒ <<TBD>>

Note: Obviously this needs clarification. Does it refer to public-key cross-registration of cells? Can it be subsumed in {PK10}?

(Sponsor: Joe Pato.)

9.5. {PK4} Signatures for Broadcast Integrity/Identity.

⇒ Users should be able to attach signatures to documents or namespace entries that can be verified at a (perhaps substantially) later date, providing proof of both integrity of document content and origin.

This is a variation on {PK2}, requiring ''non-targeted'' signatures (i.e., signatures that may be verified by any principal, rather than only by the principal to whom the message is addressed).

Impact if requirement not satisfied: Distribution of trusted (integrity-protected) information to multiple targets will require that each target be individually contacted so that the information may be transmitted using DCE session-oriented integrity protection mechanisms. If document signatures were provided, the information could be distributed simply by placing it in a widely-accessible place (e.g., the name-service).

(Sponsor: John Wray.)

9.6. {PK5} Role-Based Persistent Non-Repudiation

⇒ An extension of {PK2}, explicitly requiring that the role(s) in which the message signatory was operating be established as part of the signature verification process.

This extends {PK2}, so that the role(s) under which the message signer (or ''originator'') is currently operating become attached to (encoded in) the signature in a non-forgeable way, and that the subsequent verification of the signature reveals those roles (not just the set of all authorized roles potentially available to the signer).

<<This requirement is an enhancement of {PK2}; should there be a corresponding enhancement of {PK4}, or can we take this requirement as applying to both {PK2} & {PK4}?>>

Impact if requirement not satisfied: If this feature is not provided by the base DCE services, trusted messaging applications (e.g., expense approval) may be forced to implement this functionality themselves. If roles are introduced at an application level, different applications' role models are likely to be incompatible with one another, resulting in increased management complexity.

(Sponsor: John Wray.)

9.7. {PK6} 2-Party Confidentiality

⇒ Allow two parties to establish a shared session key without entrusting that key to the security server.

Note: It would be good to have some explanatory text for this requirement, especially since it could be satisfied by non-public-key means (e.g., Kerberos tickets can be used to share client-specified or server-specified session keys).

(Sponsor: <<TBD>>. Maybe Denis Pinkas?)

9.8. {PK7} Serverless Verification of Foreign EPACs

⇒ The ability for an application server to directly verify an EPAC generated in a foreign cell, without requiring intermediate privilege servers to ''translate'' or otherwise approve the EPAC.

Application servers should be able to directly verify the signatures/checksums that protect EPACs, even when those signatures/checksums were applied by a foreign cell's privilege server. This implies that DCE should include a trusted means of obtaining any necessary verification keys for foreign privilege servers.

This is another aspect of ''offline-ness'' ({PK1}). It might (perhaps depending on implementation) be incompatible with global groups ({I1}).

Impact if requirement not satisfied: <<TBD>>

(Sponsor: John Wray.)

9.9. {PK8} Support for Secure Store-and-Forward Messaging

⇒ This would enable a client to sign or seal a message that can be transferred to a target peer/server via an arbitrary number of (untrusted) message forwarders or queues. The target peer/server should be able to verify or unseal the message on receipt, so that end-to-end secure communication can be maintained.

For example, if Citibank in New York decides to send a financial message via the Internet to Citibank in Hong Kong to transfer fund. In this case, the message must be signed and sealed in New York since it will go through various untrusted queues of intermediate nodes before it arrives Citibank in Hong Kong. Thus, end-to-end security must be maintained. Otherwise, it could cause an unmeasurable loss. Currently, data encryption in DCE is based on DES, which is not strong enough to maintain data confidentiality of the message, especially the message is exposed to a number of untrusted nodes and is vulnerable to brute-force attacks because each node can hold the message for a certain amount of time which may be long enough to compromise the key used to encrypt the message.

Impact if requirement not satisfied: Under this circumstances, a proprietary solution must be sought to enhance end-to-end security for asynchronous messaging; therefore, DCE security service cannot be used as an enterprise-wide security solution.

Note: This requirement was originally entitled ''Asynchronous Messaging (Non-Trusted Store & Forward)''. It was re-titled to show that there is no expectation for DCE itself to include a store-and-forward messaging facility, but just that DCE security should provide *support* for such a facility.

(Sponsor: Chii-Ren Tsai.)

9.10. {PK9} Support Secure E-Mail

⇒ DCE should support secure email systems (e.g., SMIME, MOSS).

Note: This requirement was originally entitled ''Secure E-Mail''. It was re-titled to show that there is no expectation for DCE itself to include an email facility, but just that DCE security should provide *support* for such a facility. To reflect this thinking, two other requirements have been deleted (or rather, subsumed into this one). They were:

(a) ''Support Privacy-Enhanced Mail Integration (PEM)''

Provide some integration between the DCE registry and the PEM infrastructure for registering users to minimize the additional work system managers and users have to do to participate in both environments, with the caveat that such integration should not be mandatory for those administrations choosing to keep them separate.

This item is similar to the generic requirement in {E5}, except it relates to PEM in particular. An example of integration might be the storage of PEM certificates and/or corresponding private keys in the registry, or the automatic initiation of a certificate creation process for PEM as a user is registered in DCE. Desirable points of integration are left to further study.

Impact if requirement not satisfied: Without this support, PEM users will have to log on twice to send and receive mail.

(b) ''X.400 Security Integration''

As above, but for X.400.

Impact if requirement not satisfied: Without this support, X.400 users will have to log on twice to send and receive mail.

(Sponsor: Gary Gaskell.)

9.11. {PK10} Direct Trust Relationships

⇒ The ability for a principal (registered in DCE cellA, say) to directly establish a trust relationship(s) with a principal from a foreign cell (DCE cellB), without involvement of the respective cell administrators.

The requirement of establishing direct trust does not necessarily imply that there was never *a priori* security knowledge exchanged between the two organizations (intra- or inter-company). Interpretation of an organization's security policy will determine if it is acceptable for principals in these two organizations to use an established trust relationship. Security policy statements might include items like:

(a) A principal (other than an administrator) cannot establish a trust relationship for its organization.

(b) A principal can only establish a trust relationship with approved principals from approved cells.

Currently, the method of establishing trust between cells is to either allow everyone in the cell (the whole cell) to be trusted, or no-one in the cell is trusted.

This requirement is asking for the ability to establish a direct trust relationship at session connect time, given that the security policy allows for the session to be established.

A couple of assumptions can be stated:

(a) Every principal must know and trust at least one (their own) certification authority.

(b) Each principal can obtain a digitally signed list of other trusted certification authorities from its own.

Impact if requirement not satisfied: Cell administrators will have to continue managing expanding matrices of inter-cell keys.

(Sponsor: Tom Hunwick.)

9.12. {PK11} Non-Composable Key Change

⇒ Compromise of a principal's key (including long-term key) should not compromise that principal's future keys, as long as the ''attacker'' in possession of the compromised key does not change the key before the legitimate owner.

It is possible to devise key-change protocols which:

(a) Use the principal's existing key to validate the change, and

(b) securely distribute a new key to the principal, but

(c) do not encrypt the new key under the old key, or do anything else which would allow an attacker in possession of the original key to learn the new key.

These key-change protocols have a property which is called ''perfect forward secrecy'' in the academic literature. This property should be exhibited by the DCE key-change and password-change protocols.

(Sponsor: Bob Blakley.)

9.13. {PK13} Support Virus Detection (Software Certification)

⇒ DCE should support the attachment of signatures to executable files to allow detection of unauthorized modification of those files.

Note: This requirement was originally entitled ''Virus Detection (Software Certification)''. It was re-titled to show that there is no expectation for DCE itself to include a virus detection facility, but just that DCE security should provide *support* for such a facility.

(Sponsor: Gary Gaskell.)

9.14. {PK14} Support Software License Management

⇒ DCE should support software licensing management.

Note: This requirement was originally entitled ''Software License Management''. It was re-titled to show that there is no expectation for DCE itself to include a software licensing facility, but just that DCE security should provide *support* for such a facility.

(Sponsor: Bob Blakley.)

9.15. {PK15} Public-Key Support for Delegation through Untrusted Intermediates

⇒ A target server receiving a delegated request should be able to verify the original client's request even if it does not trust any intermediate in the delegation chain to properly represent the client's intentions. Similarly, the client should be able to verify the target server's response even if it trusts none of the intermediates.

An ''on-line'' version of {PK8} (see also {D4a}, {D4b}).

(Sponsor: Joe Pato.)

9.16. {PK16} Support for Notary Services

⇒ DCE should support notary services.

A notary service is one whose function is to apply signatures to documents. These signatures bind the document content to the identity of the principal who presented the document for signature, as well as to the date of the signature.

This service differs from {PK2} in that, instead of verifying only document content and origin, the service validates (at least):

(a) Content.

(b) Principal presenting the document.

(c) Date presented.

(d) Who notarized it.

A Notary service has its own keys which are used to apply signatures to documents (possibly using services provided in response to {PK4}). A notarized signature (as opposed to a self-generated signature, which {PK2} and {PK4} address) is one whose date cannot be falsified by the originator. This means that a notarized document need not be disseminated by the originator until proof of the document's existence and date is required.

Note: This requirement was originally entitled ''Notary Services''. It was re-titled to show that there is no expectation for DCE itself to include a notary service facility, but just that DCE security should provide *support* for such a facility.

On the other hand, if OSF (not necessarily DCE) does not provide a standard notary service, then non-standard notary services will be provided by applications. If standard rules don't exist for notary signature formats, key usage etc., these application-provided services will be incompatible with one another, resulting in increased management complexity. [Of course, this same argument applies to all the other PK requirements with ''Support for'' clauses ...]

(Sponsor: John Wray.)

9.17. {PK17} Support for Workflow Computing

⇒ <<TBD>>

Note: This requirement was originally entitled ''Workflow Computing''. It was re-titled to show that there is no expectation for DCE itself to include a workflow facility, but just that DCE security should provide *support* for such a facility.

(Sponsor: Joe Pato.)

9.18. {PK18} Interoperability with Other PK Environments

⇒ Interoperate securely with PK-based distributed computing environments including OSI, Netware, ONC+.

Note: Clarification needed. Is it covered by the current work being done on PKI architecture (Blauner, Blakley, et al.)?

(Sponsor: <<TBD>>.)

9.19. {PK19a} Revocation of the Private Key Component

⇒ The ability of a trusted authority to notify all parties within the computing environment (cell or multiple cells, if appropriate) that specified private keys are no longer valid as of a stated time.

A means for recovering from compromised keys must be available, before introducing public key technology into DCE. Currently in DCE when a key is compromised it is changed at the Security Server at soon as the compromise is recognized. Since the Security Server is an on-line trusted server which is involved in the initiation of any new sessions, recovery from the compromised key starts when sessions based on the compromised key time out. Public key based systems do not require an on-line trusted server thus making the current recovery model inappropriate for services based on public key technology.

Note: The NSA calls this ''CKL (Compromised Key List)'', and it connotes a high-priority push model. This is to be contrasted to ''CRL (Certificate Revocation List)'', which connotes a low(er)-priority pull model.

Impact if requirement not satisfied: Compromised keys could be used indefinitely, in perpetrating masquerading/impersonation attacks and breaching confidentiality.

(Sponsor: Ellen McDermott.)

9.20. {PK19b} Revocation of the Identity Binding

⇒ The ability of a trusted authority to notify all parties within the computing environment (cell or multiple cells, if appropriate) that specified couplings/affiliations of identities and organizations and/or roles are no longer valid as of a stated time.

A means for notifying all parties within a cell (and communicating cells) that the association between a key, username/uuid and an organization is no longer valid must be available before introducing public key technology into DCE. Currently, the Security Server vouches for users' affiliations in a cell by allowing users to start new sessions under their present identity/affiliation. If a user's affiliation has changed, the Security Server will refuse to grant TGTs which are needed to create new sessions under the current identity coupling. Public key based systems do not require an on-line trusted server thus making the current means of revoking affiliations inappropriate for services based on public key technology.

Note: Is there also here a connotation of push vs. pull model (see {PK19a})?

Impact if requirement not satisfied: Invalid identity bindings could be used indefinitely to perpetrate fraud and breach confidentiality.

(Sponsor: Ellen McDermott.)

9.21. {PK20} Trust Hierarchy

⇒ Establish a well defined structure of authority that can be recognized by users and applications.

Public key based services must be able to use the current DCE hierarchical trust model ({P15}). Public-key services are not obliged to incur the overhead of pairwise key exchanges between cell administrators. Any other trust model used by DCE must be published.

Impact if requirement not satisfied: Multiple trust models could develop between cells. These models may have flaws or used in conjunction with other trust models may introduce flaws. Flaws in trust models result in: inappropriate or misplaced trust, and lack of containment in cases of compromise.

(Sponsor: Ellen McDermott.)

9.22. {PK22} Interoperability with Recognized Standards

⇒ Public key services must be interoperable with relevant standards, per service area.

For example, if the service area is secure electronic mail the public key electronic mail system must be compliant with at least one of the following mail systems: X.400, PEM, PGP. This implies compatible certificate and revocation list formats and processing procedures. OSF will have to decide on the minimal set of standards a service area must meet in order to be accepted as a part of an OSF offering.

Impact if requirement not satisfied: Implementations of multiple versions of service based on public key technology will emerge making interoperability improbable and possibly weakening the level of security of the system. Implementations of public key services that do not meet widely accepted standards have had less exposure, which means that they may contain weakness that could have been found due to the rigorous scrutiny of the standards process.

(Sponsor: Ellen McDermott.)

9.23. {PK22a} X.509v3 Certificate Format

⇒ Public key certificates used within DCE should be X.509v3 compliant. (Throughout this RFC, unless otherwise specified or there are good reasons to the contrary, the phrase ‘‘X.509 certificate’’ always means ‘‘version 3’’.)

The majority of currently available public key products use X.509(v1) compliant certificates, and they plan to migrate to X.509v3 as it becomes available (as this is written, X.509v3 is still at ‘‘draft amendment’’ status). X.509 is well defined and is the only widely accepted public key certificate standard. As DCE integrates public key services they must be X.509(v3) compliant to allow interoperability between non-DCE public key services and DCE public key services at the application level.

X.509 is used by both PEM and X.500 (and the latter is used in X.400), so this item can be seen as an enabler for the extension of DCE to support of those services. An X.509 capability in DCE, even without PEM or X.400 support, may be useful in that vendors could provide the added value in a manner that is interoperable with other implementations.

DCE should permit acquisition of X.509 certificates by DCE users. This may, but does not necessarily, include the ability to use X.509 for user authentication. Similarly, DCE should permit servers to be registered with X.500 names and should permit servers to get X.509 certificates.

Impact if requirement not satisfied: Implementations of public key services using non-compliant X.509 certificates will not be able to interoperate with services using X.509 certificates or other non-compliant certificate formats. Without this support, users and servers will have to log on twice to get strong authentication to X.500 directories.

(Sponsor: Ellen McDermott.)

9.24. {PK22b} Standardized Formats for Signed Data

⇒ Signed data must adhere to an agreed upon format.

OSF must endorse a format for signed data. This will enable DCE signature services to be interoperable in verifying signed data.

Impact if requirement not satisfied: Multiple implementations of signature services using different formats may result, making verification of signatures and data integrity difficult at best.

(Sponsor: Ellen McDermott.)

9.25. {PK27} Attribute/Key Binding

⇒ Before public key services are integrated into DCE a statement of attribute/key binding, and alignment with secret key services in DCE must be made.

OSF must make an explicit statement of attribute/key binding and alignment for both public key and secret key services. (Here, ‘‘attributes’’ means client-side authorization data of various forms, such as identity, role, etc.) This statement should highlight the relationship between services of different cryptographic bases, and their binding definitions.

Impact if requirement not satisfied: DCE could end up with redundant or insufficient service infrastructure in an attempt to enable interoperability between cells using different bindings for different cryptographic systems. This scenario would result in increased complexity in cell management.

(Sponsor: Ellen McDermott.)

9.26. {PK28} No Mandatory Dependence on Public Key

⇒ None of the core DCE services may be provided solely by a public key service.

DCE security services have been developed using DES, a widely available secret key system, which is accepted in many countries. Public key cryptography is not efficient for the core security services found in DCE. Encryption (for confidentiality) using public key systems is on the order of ~1000 times slower than DES. The overhead of signatures for data integrity can be overkill in many circumstances. Efficiency of authentication systems based on public key vs. secret key systems is arguable depending on the desired qualities.

Impact if requirement not satisfied: Versions of DCE based solely public key security services will not gain wide acceptance in all of countries in which DCE is present.

(Sponsor: Ellen McDermott.)

9.27. {PK29} Key Escrow and/or Key/Data Recovery

⇒ Schemes for key escrow and/or key/date recovery must be defined (and accepted as an OSF standard).

Public key products, such as secure mail and secure electronic documents, have signed and/or encrypted data that can have a long life. A means of validating signatures or decrypting messages at a period of time well into the future must exist. Therefore, a way of securely storing the keys required for validation and decryption must be available.

Impact if requirement not satisfied: Multiple, possibly non-secure, key storage sites may be created, which will lead to increased management burden and possible security holes.

(Sponsor: Ellen McDermott.)

9.28. {PK30} Digital Signature API

⇒ An API enabling applications to create and verify unforgeable (public-key-based) digital signatures.

This requirement includes the provision of a trusted public-key storage mechanism, which the signature verification API can use to retrieve verification keys. Additionally, explicit application-provided key management should be supported.

The API should employ a standard format for signatures, and support common mechanisms (e.g., the PKCS series of standards).

Impact if requirement not satisfied: Applications requiring public key technology will have to implement the key-store and user private-key acquisition in incompatible ways. Public-key applications will also have to implement and ship include their own cryptographic code, resulting in export issues for these applications.

(Sponsor: John Wray.)

9.29. {PK31} Key Establishment

⇒ It should be possible to distribute long-term keys to DCE servers over the network in a secure fashion without requiring the receiving server to have a secret key in its possession.

Impact if requirement not satisfied: It will continue to be necessary to have a trusted administrator physically present at each server's machine to install that server's initial long-term key.

(Sponsor: Bob Blakley.)

9.30. {PK32} Multiple Public Keys

⇒ DCE Security should allow multiple public key-pairs per user.

This requirement allows a user to have multiple public key pairs. This faciliates a user using different key pairs for different situations. It is envisaged that a user may commonly use one key pair in their everyday duties, and reserve another key pair for ‘‘higher security situations’’. This follows the philosophy of X.509 version extensions that allows the purpose of a key pair to included in a certificate.

It is anticipated that one key pair may be used in the authentication process (that is now being described in RFC 68.x). Other key pairs might be used for signatures on documents or transactions. It should be possible for the user to have multiple key pairs and hence provide multiple signatures for different security levels.

An example might be an expenditure authorisation system, where small amounts are authorised using one key, and that signature function is performed on a host computer. The local policy might then require that signatures for large amounts be via a different key (it is expected that this key may more protected using more rigourous technology, e.g., the private key only exists on a token). DCE should provide for this by providing for multiple key pairs in the services that store and distribute the public keys (i.e., the public key certificate distribution service).

Another issue is that for full X.509 compliance, all attribute/value pairs can be multi-valued. This is another source for the requirement of allowing users to have multiple key pairs.

Impact if requirement not satisfied: Providing only a single public key pair will limit DCE PK deployment. It is the experience of the electronic mail communities that multiple key pairs should be provided for.

(Sponsor: Gary Gaskell, Bob Smart.)

9.31. {PK33} Support for Smartcards

⇒ DCE security should support smartcards.

Note: See the RFC 57 series (which remains incomplete at this writing). <<It would be good to get some more descriptive text than this.>>

(Sponsor: Roger Merckling.)

10. {R} INFRASTRUCTURE

10.1. {R1} Registry and Naming Convergence

⇒ <<TBD. Definition deferred until we find out the results of the naming working group.>>

10.2. {R2} Trust Relationships

⇒ Implement some kind of trust relationship between cells so that it is not necessary for all potentially interacting cells to directly communicate in advance. There should be at least one default implied relationship (e.g., a hierarchy based on names) throughout DCE to minimize the number of explicit trust relationships that a manager of a new cell must specify. In addition, a security manager should have the ability to establish additional explicit trust relationships within a cell that override or augment the default relationships.

This requirement was created mostly for management ease in an environment of many cells. It is unlikely that a security manager of a cell can make a rational decision as to which authorities to trust for all possible remote cells with which the users and servers in his cell need to communicate. It is unlikely that security manager will even know, in advance, what those cells are. Therefore, some well-defined (and of course, secure) default relationship needs to be consistently supported throughout the environment.

On the other hand, there is also the requirement that security managers cognizant of trust relationships be able to override the default where they have direct pairwise agreements with each other.

Note: PARTIALLY satisfied by DCE 1.1, but parts of the work remain to be completed. Namely, while every cell pair no longer needs to share a key, all cells still must have prior knowledge of the name/UUID of every cell they may want to communicate with.

10.3. {R8} Client-Server Security Negotiations

⇒ A DCE client must be able to determine the security characteristics of a DCE server before making an RPC call. The client should be able to determine whether or not the server supports the level of security that the client requires. If the server does not support the level of security the client requires, the client could then import the binding handle from a different server that provides the same service and check for the security provided by the new server. It would then be up to the client to find a server with adequate security or knowingly make the RPC call to a server with a lower level of security.

Impact if requirement not satisfied: Currently, if a client makes an RPC call to a server without knowing the level of security supported by that server, there are three possible outcomes to that attempted RPC call:

(a) Client-Server mismatch. The RPC call fails and the application dies quietly or produces a runtime error depending on certain parameters.

(b) Client-Server mismatch. The RPC call succeeds, but the level of security provided is less than the client expected. In this case the client does not know that it is not getting the expected level of security.

(c) Client-Server match. The RPC call succeeds with the correct level of security being requested by the client being provided.

Of the three possible outcomes, the first two are not acceptable when maintaining a secure applications environment. Failure to meet the requirement means maintaining status quo in this unacceptable situation.

A likely implementation, for RPC-based applications, is that servers should advertise their security characteristics in their CDS entries, and this information should be returned by CDS to clients in the binding.

An alternative implementation would be (this would require a large change to the DCE RPC protocol):

(a) A return code from the server indicating failure due to impermissible security level ALONG WITH an indication of what security level would be acceptable.

(b) A mechanism to permit the client code to retry the request with the specified security level — after passing a check to insure that the specified level is acceptable to the client — automatically.

(Sponsor: Charles Blauner.)

10.4. {R10} Workstation Lockout Capability

⇒ The System Security Administrator (or Officer) needs the ability to lock a workstation out from DCE, so that no access to DCE services is possible from that machine.

Note: Clarification (and a sponsor) is needed. For example: What level of ‘‘no access’’ is acceptable? Is ‘‘unauthenticated’’ access permissible from a locked workstation? How will machines be identified so that the administrator can specify them through the lockout interface? By machine principal name? IP address? Some other way? Is this capability intended for use in conjunction with some kind of intrusion detection? Are there operating system or other prerequisites for implementation of this requirement?

(Sponsor: <<TBD>>.)

11. {S} SYSTEM INTEGRITY

11.1. {S1} Better Support for Security Policy Data Update Integrity

⇒ Fix the ‘‘security policy data update integrity problem’’.

This problem is best illustrated using an EXAMPLE (of a problem which can arise during ACL administration): Two users, A and B, both have administrative authority with respect to a resource R. User A accesses ACL on resource R, intending to add an entry for Y (a user or group). User B accesses same ACL, intending to update permissions for X. User A completes the addition. User B completes the change. The new entry for Y no longer exists; this results in two errors:

(a) User B deleted an entry (not B’s intent).

(b) User A's completed (and confirmed) update is not in place.

DCE DACLs have no mechanism to prevent ACL update collisions like those illustrated above. This problem is particularly severe for ACLs with many entries, since granularity of DACL manipulation is the entire ACL (i.e., `rdacl_lookup()` returns the entire ACL and `rdacl_replace()` replaces the entire ACL).

Note that this problem also exists for Registry data; two users with admin authority with respect to the same registry object (say, an account) can make changes in such a way that one set of changes is inadvertently destroyed. (E.g., if adminA and adminB's `rgy_edit` sessions both read the data for a particular account at the same time, then they each change something within that account data and do a replace, one of the two changes may be lost. The chances of it happening are minimal — even more so than with `acl_edit`, since only administrators run `rgy_edit` to make changes like this, but everybody uses `acl_edit`).

At the Paris SIG meeting a document was circulated which proposed some solutions to this problem, which the group discussed. The (quite correct!) conclusion was that my solutions were far too drastic. On further reflection it seems that the two most reasonable solutions are:

(a) Add an interface like the Unix command line `chmod` interface, whereby an administrator can specify `+rw` or `-r`, etc.... This eliminates many of the worst side-effects without radical changes to code.

(b) A slightly more involved solution would be a ''test-and-update'' API verb which would commit a change only if the ACL matched an ACL specified in an input parameter. The typical use of the API set would then be:

```
read remote object's ACL into a local copy, say ACL*
make a copy of ACL*, call it ACL**
change the copy (ACL**)
call test-and-update (ACL*, ACL**)
```

The changes found in `ACL**` would be committed to the object's ACL only if the object's ACL ''still'' matched `ACL*`.

(Sponsor: Bob Blakley.)

12. {SS} SINGLE SIGN-ON (SSO)

12.1. {SS1} Single Signon

⇒ DCE security should support single signon.

There are two issues which must be addressed to achieve ''single signon'' in a heterogeneous distributed network:

(a) Integration of DCE login with local OS login, particularly in the case where the local OS is NOT a Unix-type OS. This is the issue of adding extra attributes to the DCE user registry for operating systems that need extra information stored — for example for IBM's MVS. This is addressed by requirement {P10a} and is satisfied by the DCE 1.1 RFC 6.0 work. Note that additional work by DCE vendors is needed to exploit this support and produce an integrated OS/DCE login.

(b) Integrating DCE login with `telnet`, `rlogin`, `ftp`, `rsh`, etc. The issue here is that Kerberized commands (see {E1b}) would not necessarily use the same tickets as those used by DCE, particularly on nodes that don't otherwise support DCE. Morrie Gasser raised this issue in the SIG many months before the extra attributes issue was raised; it can be summarized as follows:

''If I am on a machine using DCE and I quote my password appropriately, so that I can use authenticated RPC, can I also use the tickets/credentials/whatever to do `rlogin/rsh/telnet/ftp`/etc... to other systems on the network that speak Kerberos, but do NOT speak DCE? If I have single signon, then I should only have to login once, and that login should work with both DCE and non-DCE Kerberos. The issue is one of human interface, not what particular flavor of tickets are used by what applications. I don't really care whether DCE uses vanilla tickets or chocolate tickets, as long as DCE and regular Kerberos can interoperate securely.''

To properly support single signon, both issues have to be solved. A user expects single signon to be just that — login a single time and have that authentication used everywhere. This means that DCE has to be aware of non-Unix operating system attributes, at least to some extent, that DCE Kerberos should interoperate with regular Kerberos, even with machines that don't support DCE itself, and finally that a local OS should support local login reliably, even if network links go down.

For single signon to really work with DCE and other Internet services, the following should be possible for a user who does NOT have access to source code:

(a) Login once to talk to other DCE applications on other machines.

(b) Login once (same login as in (a)) to do `telnet` and `ftp` sorts of things to Internet machines that run DCE.

(c) Login once (same login as in (a)) to do `telnet` and `ftp` sorts of things to Internet machines that run Kerberos, but do not run DCE.

(d) Receive Kerberized telnet and ftp from Internet machines that run Kerberos, but do not run DCE.

(e) Integrate DCE login with non-Unix operating system login, so that I can connect to or from non-Unix operating systems and still have things all work right (i.e.: the extra operating system-specific attributes stuff).

(f) Sufficient local login capability (perhaps by caching or whatever) so that I can login locally to my ‘‘home’’ system(s), even if the network links are broken.

(g) Login once to talk to local applications which have their own security subsystems.

(h) Login once to talk to both ‘‘Kerberos-based’’ (DCE and other Kerberos) remote servers, and remote servers which use other secure protocols (e.g., Novell NetWare, Lotus Notes, etc...).

(a)–(d) include the requirements called for in {E1a} and {E1b}; (e) is addressed by the DCE 1.1 RFC 6.0 work, and (f) is a vendor issue. (g) and (h) are not addressed by other requirements herein; some DCE enabling may be desirable for these capabilities.

Note: Nick Mansfield is working with X/Open to produce a good list of SSO requirements, driven by business needs. That document is still in draft form, but its requirements should be considered to be hereby ‘‘incorporated by reference’’.

(Sponsor: Maryanne Hondo.)

13. {T} AUDITING

13.1. {T2} Remote Auditing

⇒ Record security-relevant events on remote centralized audit systems.

(Sponsor: Chii-Ren Tsai.)

13.2. {T4} Separation of Roles

⇒ An auditor principal should be created to solely perform audit system management to enforce ‘‘separation of roles’’. Audit system management includes modification of the event selection list (ESL), invocation of audit daemon, audit trail analysis and audit trails backup and restore. See {I3}, {I4}, {T4a}.

(Sponsor: Chii-Ren Tsai.)

13.2.1. {T4a} Audit policy protection

⇒ The audit event filters must not be modifiable by system administrators other than auditors. See {T4}.

(Sponsor: Chii-Ren Tsai.)

13.3. {T6} New Audit Event Class — Object Creation

⇒ A finer level of granularity should be applied to ‘‘Object Creation’’, similar to ‘‘Object Deletion’’, rather than generalizing it under ‘‘Controlled Access’’ events.

(Sponsor: Chii-Ren Tsai.)

13.4. {T8} Define Audit-Event Specific Information

⇒ New audit events can be added to the audit system and be recognized by the `auditd`. The mechanism to expand audit events including how to define event specific information in audit records must be specified.

(Sponsor: Chii-Ren Tsai.)

13.5. {T9} Audit Daemon Status Alert

⇒ Whenever the `auditd` is killed, an alarm must be sent to the auditor.

(Sponsor: Chii-Ren Tsai. Ivan Milman has agreed to add some explanatory text.)

13.6. {T10} Audit Trails Storage Full

⇒ Make the ‘‘audit log full’’ behavior of DCE auditing more robust.

The following kinds of behavior are supported in DCE 1.1 to address the audit log full occurrence:

(a) (Default) — `auditd` stops writing audit records to the log.

(b) (Override) — `auditd` discards old audit logs.

Neither of these behaviors are acceptable. To resolve this issue, two provisions need to be done:

(a) The auditor must be alarmed by the `auditd` whenever a configurable threshold for the audit trail file system is reached, e.g., the file system is 90% full. Optionally, the alarm can trigger a preconfigured program to backup audit trails automatically in order to preempt space.

(b) Several audit trail servers are configured, so that whenever a server is full, audit trails will be transferred to another server automatically. In the mean time, an alarm is sent to the auditor via a preconfigured method. The `auditd` must be aware of the configuration of audit trail servers. Consequently, audit trails distributed on different servers can be reconstructed to ensure that audit records are stored in order.

(Sponsor: Chii-Ren Tsai.)

13.7. {T11} Audit Trails Protection

⇒ Audit trails must be protected against unauthorized examination and modification. This protection should not rely upon security characteristics of the underlying operating system (this probably means that the audit trail will have to be encrypted to satisfy this requirement, probably by public-key methods).

(Sponsor: Chii-Ren Tsai.)

Note: This requirement subsumes two requirements formerly placed in the {PK} series. They are recorded here, for archival purposes:

(a) ‘‘Long-Term Protection of Audit Data’’

The ability to determine the originator, verify the integrity and if required, maintain the confidentiality of audit data over a long period of time (perhaps many years).

This requirement assumes that the message originator must sign all audit data submitted to the audit repository. Authorized reviewers of audit data must be able to verify signatures in a manner that proves the identity of message originators to a third party, at any time after the submission of audit data. The requirement is for ‘‘long-term’’ protection which implies that keys required for verifying signatures must be kept in an integrity protected cache, at a minimum. If the same key is used to sign and seal audit data, then the key must be confidentiality protected as well to ensure only authorize people may review audit data.

(Sponsor: Ellen McDermott.)

(b) ''Protect Audit Log Against Non-Auditor Examination''

The use of public key technology would allow multiple sources of audit data to write to a single audit repository, in such a way that the resulting audit trail cannot be read by anyone other than a designated auditor. Even audit sources would be unable to read their own data.

Impact if requirement not satisfied: Audit data assembled from a variety of sources requires an active audit data collection agent (in addition to the sources of audit data) so that audit sources may use DCE session-oriented security services to protect data in transit.

(Sponsor: Ellen McDermott.)

13.8. {T14} Audit Reduction Tool for Audit Trail Analysis

⇒ It is desirable to provide a tool for audit trail analysis. The tool can be used to trace security or access violation, select audit records by a combination of attributes from the audit header, and generate audit reports.

(Sponsor: Chii-Ren Tsai.)

13.9. {T18} Hierarchical Audit Events Filtering

⇒ To support hierarchical audit event management, the interpretation of an ESL must go through the whole list instead of first match. In addition, if there is an overlap in terms of a user and the user within a group or multiple groups, the union of actions should be taken by the `auditd`.

For instance, the user `micky@gftpd.citicorp.com` belongs to two groups, namely `tpd` and `odessa`, and the ESL is shown below:

```
princ:micky@gftpd.citicorp.com:Authentication:FAILED:LOG
group:tpd:Authentication:FAILED:ALARM
group:odessa:Authentication:FAILED_ACCESS:ALARM
realm:gftpd.citicorp.com:Authentication:ALL:LOG
```

In this example, `FAILED` and `FAILED_ACCESS` attempted by `micky` should be logged and alarmed, and `SUCCESS` by `micky` should by logged in the context of Authentication event class.

(Sponsor: Chii-Ren Tsai.)

14. {X} MISCELLANEOUS

14.1. {X1} Loading and Booting Security

⇒ Provide interfaces and an infrastructure whereby it is possible for a user to have increased confidence that he is interacting with legitimate software on his node, free from Trojan horses and viruses. This includes mechanisms permitting the hardware to verify the booted image on a diskless node, as well as the operating system being able to detect corruption of applications software. To the extent possible, this infrastructure should be integrated with the existing DCE registration infrastructure.

Providing secure booting of systems requires both hardware and software cooperation. To boot securely, a workstation (e.g., the boot ROM) must verify the integrity of a software image it receives before being willing to execute it. Similarly, for secure loading of applications (e.g., at the time a Unix process calls `exec()`), the operating system must be able to verify the integrity of the executable file. While major portions of the solution to this problem are outside the scope of DCE, DCE can provide an important part of the management infrastructure and support tools that allow vendors (or a subsequent OSF offering) to provide the full service in an interoperable way.

14.2. {X7} Maker/Checker for Security Administration

⇒ It must be possible to control updates to the DCE security database using a ''separation of duties'' authorization model. A ''Maker/Checker'' model requires that two principals be involved in any policy related or authorization data change to the DCE security database. One principal with ''Maker'' privileges can enter the data and describe the changes to be made; a second principal with ''Checker'' privileges can authorize or deny the changes described by the maker, but is not allowed to modify the data.

Note: Since the DCE ACL manager source code is already included with binaries, it is already possible for application-writers to modify the ACL manager code to do this. However, this requirement asks for (optional) *native* DCE support for this capability.

(Sponsor: Chii-Ren Tsai.)

14.3. {X8} Emergency Patch Mechanism

⇒ OSF should institute an emergency DCE security patch mechanism.

Like any software system, DCE will from time to time contain bugs with serious security implications. Because of the way DCE is implemented and productized it has frequently taken a long time for fixes to bugs such as Alias Creation and DFS Integrity to reach the field. End users and vendor field staff have no easy way of communicating with OG or even determining if a fix exists, determining how to identify the fix and obtain it and finding out what fixes currently exist.

The Open Group should create a special process for making key security-related fixes available to the field rapidly, outside of the normal distribution channels. This should also include a scheme for assigning an id to each patch to facilitate requesting the fix through a vendor. It should also include a way to query the list of existing fixes and descriptions. Possibly DCE could be used to control access to this information via the Internet.

Note: This requirement was instigated by the observation that a certain bug fix was still not fixed in the latest release of at least one volume distributor of DCE, even though it was a CERT advisory over a year earlier!

(Sponsor: Hal Lockhart.)

A. APPENDIX A. PROPOSED REQUIREMENTS REMOVED FROM LIST

The following requirements were initially proposed by at least one individual from the working group, but were subsequently deleted, for the reasons given.

A.1. {A6} Authority Identity

⇒ Provide functions that permit authorization to be determined based on the identity of the user's authority or authorities (trusted registry) instead of the user's identity.

A vote for this requirement is a vote for a scheme whereby the server has the option of looking only at the authority's identity, plus the attributes that the authority claims for the client, in order to evaluate access. It is similar to {A5} in that the client's individual identity is not relevant to the access control decision, but unlike {A5} does not require the client's identity to be hidden from the server.

The group decided not to vote on this as a separate item, as it is viewed to be subsumed by item {R2}. That is, DCE already provides the ability for an ACL to list ''any principal in a cell'', and if the only types of authorities supported are those for the entire cell, then trusting any principal in a cell is semantically equivalent to trusting the authority for the cell. Therefore, the only new requirement this imposes is a way of deciding who the trusted authority is for the cell.

A.2. {E6} Interoperability with Selected Non-DCE Systems

⇒ Consider interoperability with popular non-DCE environments such as PC LANs.

This proposed requirement was considered out of scope for DCE.

A.3. {N11} Support for Roles

⇒ It should be possible for a principal to be allocated an organizational role reflecting the principal's job in the organization, and for that principal to be able to specify (or default to) that role at authentication time. The required level of authentication for that user in that role, and subsequent access rights of the user should be able to be affected by the choice of role. This should be able to include tailoring the principal's view of the system through the desktop presented to the principal (who should only see the applications that are accessible in that role).

Impact if requirement not satisfied: <<TBD>>

Note: This requirement is covered by other requirements (see {I3}, {I4}, {N7}).

(Sponsor: Tom Parker.)

A.4. {P1} Performance

⇒ Authentication and authorization should not significantly degrade access. Performance degradation due to communication security should be minimized.

This item was deleted because good performance is a generally accepted principal for any software engineering effort and need not be spelled out explicitly.

A.5. {P2} Robustness to Prevent Denial-of-Service

⇒ Enabling of security should not result in a significantly increased risk of denial of service, either due to bugs, random environmental events, or malicious attacks.

This item was deleted for reasons similar to {P1}.

A.6. {P3} Interoperation with Operating System Security

This item is subsumed in {P10a}.

A.7. {P7} B1 Portability

If functional requirements which move the DCE code toward these goals are formulated, this will be reconsidered.

A.8. {P8} C2 Portability

If functional requirements which move the DCE code toward these goals are formulated, this will be reconsidered.

A.9. {P9} ITSEC Evaluation

If functional requirements which move the DCE code toward these goals are formulated, this will be reconsidered.

A.10. {P11} Interoperability Among All DCE Variants

Deleted because this requirement is not unique to security.

A.11. {P12} Extensibility Mechanisms for Future DCE Versions

Deleted because this requirement is not unique to security, and in any case it is good software engineering.

A.12. {P14} Standards

⇒ Accommodate security standards, or draft standards, where appropriate. Candidates include those from ISO, ECMA, IETF and POSIX.

Deleted because this requirement is not unique to security and is a given OSF goal.

A.13. {PK21} Performance Profile

⇒ The performance of base services using public key technology must at least as fast as base services using the current implementation of DES.

The current DCE security services are based on the Data Encryption Standard (DES), which is a notably faster mechanism (~1000 times faster) for confidentiality than well known public key algorithms, such as RSA and ElGamal. Public key based services must be able to provide confidentiality and integrity with the same or better efficiency as that found in DCE v1.

Impact if requirement not satisfied: Implementations of public key based services that are significantly slower than the current version of DCE will result in commercially unacceptable products.

Note: This requirement was deleted because it mostly applies to vendors/ISVs, not to OSF.

(Sponsor: Ellen McDermott.)

A.14. {PK24} Support for Multiple Cryptographic Algorithms

⇒ Services based on public key cryptography must be algorithm independent.

Services based on public key cryptography must be able to be implemented using a set of public key cryptographic algorithms. The goal of this requirement is to allow the consumer of the product to choose an algorithm(s) based on their needs and their government's restrictions.

Impact if requirement not satisfied: Products will be built that cannot be exported out of the country where they are developed and/or imported to consuming countries. Multiple implementations of the same service may result from customers' demand for services with different algorithms, which will lead to complexity in key management.

Note: This requirement is already subsumed under such requirements as {E7} and {E8d}.

(Sponsor: Ellen McDermott.)

A.15. {PK26a} Statement of OSF's Role with Regard to Licensing Issues

⇒ OSF must state its role with regard to cryptographic algorithm licensing.

OSF must decide whether it will: negotiate with the patent holders of public key algorithms and broker the use of these algorithms for use with DCE; or have vendors negotiate their own licensing agreements.

Impact if requirement not satisfied: Confusion over the proper way to proceed when building public key services for DCE will result, causing a delay in the release of products.

A.16. {PK26b} Statement of OSF's Role with Regard to Export Issues

⇒ OSF must state its role with regard to exporting cryptographic material outside of the U.S.

OSF must decide whether it will: submit applications for export on behalf of vendors adhering to approved implementation of a public key service; or have vendors submit applications directly with the State Department.

Impact if requirement not satisfied: Confusion over the proper way to proceed when building and shipping public key services for DCE will result, causing a delay in release of products.

A.17. {PK26c} Statement of OSF's Role with Regard to Certifying Authorities

⇒ OSF must state its role with regard to setting up and participating in a certifying authorities' hierarchy.

OSF must decide whether it will: build, administer and participate in a hierarchy of certifying authorities; or have vendors and end-users find other means of provision of such a hierarchy.

Impact if requirement not satisfied: If OSF does not provide an approved certification authority hierarchy, disjoint and untrustworthy certification authority hierarchies may result.

A.18. {T12} Performance

⇒ The performance impact of `auditd` on a system must be measured and published.

(Sponsor: Chii-Ren Tsai.)

Note: This is considered to be a requirement on vendors, not on OSF.

A.19. {T13} Internationalization

⇒ Internationalization of audit records should be supported, because event specific information may contain different languages originated from geographically dispersed regions within a given organization.

We agreed to remove this requirement because it is viewed as outside the mission of the Security SIG, and within the mission of the i18n SIG, to assert this requirement.

A.20. {T17} Suppress Sensitive Data in Audit Logs

⇒ It should be possible to designate data ''sensitive'' so that the audit subsystem will not write it into the audit log.

This requirement was withdrawn with the permission of the sponsor.

B. APPENDIX B. REQUIREMENTS SATISFIED BY DCE

As requirements are satisfied over time by successive releases of DCE, they are moved out of the main body of this RFC and into this section, for archival purposes.

B.1. {A1} Authorization Based on Delegation

⇒ Given a delegation model that permits the server to know which principals were involved in a delegated request, build an authorization model that permits the server to enforce an authorization decision based on those identities. Extend the authorization mechanism to allow an administrator (e.g., the ACL writer) to specify permitted delegates in a practical way. The idea is fundamentally to permit the delegates (when or if delegation is supported in DCE) to be named in the ACL.

Note: Satisfied by RFC 3.0 work in DCE 1.1

B.2. {A8} Other Security Attributes

⇒ Through an extensible mechanism, permit the server to obtain from the client verifiable security attributes of the user other than principal or group memberships.

This requirement is a prerequisite to {A7}. Examples of new attributes that might be supported are clearance and role.

Note: Satisfied by RFC 6.0, RFC 3.0 work in DCE 1.1

B.3. {D2} Delegation with Restrictions

⇒ Where delegation is specified by a client, permit the client to restrict delegated rights in some useful fashion to be a subset of those rights currently possessed by the client.

This requirement does not necessarily imply extensions to the delegation mechanism. An acceptable implementation is one where the client can restrict its current rights (as in DCE today), and then delegate all of its current rights.

Note: Satisfied by RFC 3.0 work in DCE 1.1.

B.4. {E2} Security API Through Protocols Other Than RPC

⇒ Applications that choose not to use RPC for communications should have the ability to make use of all the DCE authentication and communications security mechanisms through a portable API.

This is the GSS API requirement.

Note: Satisfied by RFC 5.0 work in DCE 1.1

B.5. {E5} Application Security Services

⇒ Permit an application to gain access to enough of the underlying DCE security infrastructure to provide security services such as digital signatures, data confidentiality and nonrepudiation without the need for the application to implement another user and system registration facility and authentication mechanism.

Many application-level security mechanisms, for applications such as privacy enhanced mail and electronic document interchange, require facilities to permit registration of users, installation of user secrets (passwords or private keys), and check of user names in order to make authorization or authentication decisions. These facilities are conceptually similar to those already provided by DCE but differ in most details. This is a requirement to install services in DCE to access the existing DCE mechanisms so that applications will not need to implement their own. It may also require augmenting the DCE facilities to better service varying needs of applications. (See requirements {PK9b} and {PK9c} for specific applications to be supported.)

Note: Except for the request for *augmenting the DCE facilities* (which are adequately covered in other requirements), the SIG believes this requirement is satisfied by the GSS-API functionality in DCE 1.1.

B.6. {N4} Site Definable Password Complexity

⇒ The password complexity policies supported by `rgy_edit` and `dce_login` must be extended.

More specifically, the extensions should allow for the following two additional policy choices:

(a) Complex passwords that require a mix of alpha, numeric and special characters.

(b) A site definable policy. This policy would require securely branching to site provided code that defines specific local password complexity rules. Since some sites will not want to reveal those rules, the ability to securely execute their own password checking code is a must.

Impact if requirement not satisfied: DCE will continue to allow trivial passwords that do not meet the standards and policies of many of the organizations that want to use DCE.

Note: Satisfied by DCE 1.1.

B.7. {P10a} OS Integration

⇒ Integrate DCE with certification/registration environments of other operating systems. This includes a generic ability to centralize registry databases for multiple systems, to permit flexibility in storing OS-specific user attributes in the registry, and to permit adjusting the frequency of registry updates based on local system needs.

This is an extension to DCE to make it more portable to environments supporting other than Unix-based hosts.

Note: Satisfied by RFC 6.0 work in DCE 1.1.

B.8. {R7} Pull Model

⇒ Permit an environment where a server can obtain and verify all of the user's security attributes relevant to an authorization decision without action by or cooperation of any of the clients or delegates.

A vote for this item is a statement that there will be important cases where the client (more precisely, the user, the workstation or the user's privilege server) does not necessarily know, at the time a PAC is created, all the privileges (especially group memberships) that will be necessary to satisfy a subsequent access request. This requirement becomes much more relevant if DCE is extended to permit groups to name members outside their cell, as there is no way for any system to create a PAC naming all groups of which the user is a member. Since the server knows exactly what attributes are required to satisfy a request, a vote for this item permits an option whereby the server can verify the user's attributes even if the user does not supply them.

Note: Insofar as the SIG can determine, this requirement has always been satisfied by DCE, because DCE supports facilities (APIs and protocols) whereby servers can query registry servers (the client's and/or others, as the paragraph above envisions) to determine the client's privilege attributes (provided the server has read access to them). Doing so does not respect the client's responsibility to invoke ‘‘least privilege’’, but that's an application-level policy matter, not a DCE-level mechanism problem.

B.9. {R9} acl_edit Command Language Extension

⇒ The `acl_edit` command language needs to be extended to allow a user to do all the necessary processing for setting-up and maintaining ACLs without having to exit and re-execute `acl_edit`.

Impact if requirement not satisfied: Currently, a user must execute the `acl_edit` program three separate times using different options in order to complete the simple task of giving a user full access to a directory. This is not user-friendly or an acceptable overhead on administrators. If the requirement is not met, this problem will continue to exist.

Note: This is satisfied in DCE 1.1 with the `dcecp` command (successor to `acl_edit` and other administrative commands).

B.10. {T1} Local Auditing

⇒ Record security-relevant DCE events on the local system.

Note: Satisfied by RFC 28.0, 29.0 work in DCE 1.1.

B.11. {T3} Application Access to Audit Functions

⇒ Extend the auditing functionality introduced in DCE 1.1 to allow applications to use the audit APIs and associated mechanisms. The extensions should allow application audit data to be sent to the same centralized audit repository used internally by DCE. Additionally, any audit based alarm mechanism or reporting/reduction tools that may be provided by DCE should be usable with application based auditing.

Impact if requirement not satisfied: Applications will have to develop their own auditing mechanism. This implies that a security administrator will have to look at many different audit files of possibly different formats to get an accurate picture of what is happening in the system. The potential for different audit record formats also means that it will be very difficult to develop a single audit reduction/analysis tool. In summary the lack of a centralized audit for both DCE and applications will result in an increase in the complexity of the security administrators job.

Note: Satisfied by DCE 1.1.

B.12. {T5} Administrative Tool for Audit Events Selection

⇒ It is necessary to provide an administration tool for audit events selection, since to modify ESL in terms of large number of principals and groups is not an easy effort. Also, the tool should be able to provide information about principals, groups, realms and audit events, and detect any errors.

(Sponsor: Chii-Ren Tsai.)

Note: Satisfied in DCE 1.1, by `dcecp`. The second sentence of the requirement is solved by writing a ‘‘`dcecp` script’’ (based on the tools command language `tcl`).

B.13. {T7} Audit Events Registration

⇒ To ensure lack of collisions between audit records from different servers audit events definitions, namely set-ID and event-ID, are preassigned by major vendors with preagreement. This is a policy issue instead of a technical issue. If OSF can coordinate the definitions, it would be much easier for vendors and users.

(Sponsor: Chii-Ren Tsai.)

Note: OSF now does this. See RFC 81 (‘‘DCE Assigned Values’’).

B.14. {T16} Before/After Audit Events

⇒ It should be possible for applications to log the contents/value of a system resource before an auditable action is performed on that resource, and to record the modification requested and the success or failure of the modification. The idea here is to be able to reconstruct the original state of the resource from the information in the audit log.

(Sponsor: Chii-Ren Tsai.)

Note: Satisfied by the auditing facilities in DCE 1.1. (There aren’t configurable audit hooks inside the DCE components themselves, but that isn’t what this requirement asks for.)

B.15. {T19} Default Configuration of Auditable Events

⇒ If no filter is specified, the `auditd` should send an alarm to the auditor instead of assuming all events are logged, which may generate too many audit records. So, it is good to have some default audit events specified in a default event list, which should not be overridden. However, if there is any filter specified in the ESL, the default list would be ignored.

(Sponsor: Chii-Ren Tsai.)

Note: Satisfied in DCE 1.1.

B.16. {X2} Limit Password Dictionary Attacks

⇒ A site or cell should have the option to configure a mechanism that prevents dictionary attacks on poorly chosen passwords by wiretappers or nodes masquerading as legitimate clients. A tool to avoid picking bad passwords is not by itself an adequate solution.

The Kerberos protocol used in DCE returns recognizable messages (e.g., the TGT) encrypted only by the user's password, which is subject to capture and analysis. Satisfying this requirement requires a change to the Kerberos protocol or use of an intermediary to communicate with the Kerberos server that is trusted not to guess passwords. Public key techniques are useful in simplifying the solution to this problem.

Note: Satisfied by RFC 26 work in DCE 1.1.

B.17. {X3} GSSAPI Changes for Exportability

⇒ The DCE 1.1 version of GSSAPI doesn't come in exportable and non-exportable versions (i.e., GSS_Seal is not split out into two routines: one that works, and one that doesn't). OSF should provide exportable and non-exportable versions of GSS_Seal.

The requirement here is to provide a version of the GSS_Seal verb which just turns cleartext into cleartext, so that vendors can produce an exportable package in which the ''Seal'' interface is present but not functional.

Note: Satisfied in DCE 1.1.

C. APPENDIX C. VOTE RESULTS

This section presents the votes received from OSF member companies on the requirements described in this document. The votes are arranged by item category.

Key to company abbreviations: <<TBD>>

D. AUTHOR'S ADDRESS

Thomas R. Hunwick
Electronic Data Systems
700 Tower Drive, MS 3103
Troy, MI 48098
USA

Internet email: MSUSTR02.THUNWI01@eds.com
Telephone: +1-810-265-0105