OSF DCE SIG
Request For Comments: 28.1

S. Luan (VDG)
R. Weisz (IBM)

November 1993

DCE SERVER AUDITABLE-EVENT IDENTIFICATION AND A PROPOSED AUDIT LOGGING API

1. INTRODUCTION

Audit is one of the high-priority security features to be supported in DCE 1.1 and later versions voted by the DCE Security Working Group [RFC 8.0]. Extant audit requirements include those provided in the Department of Defense Trusted Computer System Evaluation Criteria [NCSC 85] (commonly known as the Orange Book, or TCSEC), the Trusted Network Interpretation [NCSC 87] (commonly known as the Red Book, or TNI, which is an interpretation of the TCSEC for trusted networks), and the NCSC audit guideline [NCSC 88]. In this RFC, we identify the auditable events in the DCE Servers by interpreting the Class C2 Requirements of the above evaluation criteria in the context of DCE Release 1.0. Class C2 is the lowest security class that includes any audit requirement, and it is also widely accepted as the minimum security class for commercial and government organization uses. An audit-record header structure that satisfies the C2 Requirements is proposed. We also define some audit data structures and a set of audit logging API functions. The API functions can be used by DCE servers and audit-trail analysis tools to log and retrieve audit records. These data-structure and API-function specifications are derived from those of the POSIX 1003.6. However, significant differences exist since DCE is built upon distributed, heterogeneous systems, which the POSIX standard does not currently address.

The rest of this document is organized into seven sections and two appendices. In Section 2, we provide a review of C2 audit requirements of TCSEC, TNI, and the NCSC audit guideline. In Section 3, we define the DCE audit header structure that satisfies the C2 audit requirements. In Section 4, we define C2 auditable events in Security and Time Services. In Section 5, we describe the data types defined for DCE audit. In Section 6, we describe the audit logging API functions proposed to be used by DCE servers. In Section 7 we demonstrate an example of the inclusion of the proposed DCE audit hooks in the Security Server code. Section 8 contains acknowledgements. Appendices A and B contain a list of proposed auditable events in Security and Time services respectively.

2. TCSEC AND TNI C2 REQUIREMENTS

TCSEC and TNI C2 Audit Requirements include those on auditable events, event information, and audit functionalities. This RFC only addresses the auditable events and event information requirements for the DCE Security and Time Services. The audit-functionality requirements (e.g., to selectively audit the actions of any one or more users based on individual identity, selection of auditable events) are addressed in a companion RFC [RFC 29.1].

2.1. Auditable-Event Requirements

The following excerpt is from the Orange Book on the C2 auditable-events requirement:

> ''The TCB shall be able to record the following types of events: use of identification and authentication mechanisms, introduction of objects into a user's address space (e.g., file open, program initiation), deletion of objects, actions taken by computer operators and system administrators and/or system security officers, and other security relevant events.''

The TCSEC-required events can be partitioned into the following classes:

(a) OE1 — Use of identification and authentication mechanisms.

(b) OE2 — Introduction of objects into a user's address space.

(c) OE3 — Deletion of objects.

(d) OE4 — Actions taken by computer operators and system administrators and/or system security officers.

(e) OE5 — Security-relevant event: Any event that attempts to change the security state of the system, (e.g., change discretionary access controls, change the security level of the subject, change user password, etc.). Also, any event that attempts to violate the security policy of the system (e.g., too many attempts to login, attempts to violate the mandatory access control limits of a device, attempts to downgrade a file, etc.).

The TNI provides a C2 interpretation of ''other security-relevant events'' as the following:

(a) TE1 — Identification of each access event (e.g., establishing a connection or a connectionless association between processes in two hosts of the network) and its principal parameters (e.g., host identifiers of the two hosts involved in the access event and user identifier or host identifier of the user or host that is requesting the access event), and identification of the starting and ending times of each access event using local time or global synchronized time.[1]

(b) TE2 — Identification of security-relevant exceptional conditions (e.g., potential violation of data integrity, such as misrouted datagrams) detected during the transactions between two hosts.

(c) TE3 — Utilization of cryptographic variables.

(d) TE4 — Changing the configuration of the network (e.g., a component leaving the network and rejoining).

2.2. Audit-Record Information Requirements

The TCSEC-Required Event Information includes:

(a) OEI1 — Date and time of the event.

(b) OEI2 — User.

(c) OEI3 — Type of event.

(d) OEI4 — Success or failure of the event.

(e) OEI5 — The origin of request (e.g., terminal ID) — Identification/authentication events only.

(f) OEI6 — Name of the object — events that introduce an object into a user's address space and object deletion events only.

TNI does not specify additional required event information beyond that implied by the event requirement TE1. We label this required information as TEI1 below:

(a) TEI1 — Principal parameters of access events (e.g., host identifiers of the two hosts in the access event and user identifier or host identifier of the user or host that is requesting the access event), and identification of the starting and ending times of each access event using local time or global synchronized time.

Some of the above requirements need interpretations for the DCE context. The required ''user'' information (OEI2) should include UUID's of a principal and the principal's home cell. The principal could be a human user or a non-human entity (e.g., a machine principal, a server principal). When multiple principals are involved in an access, the UUID's for all these principals should be recorded; e.g., the UUID's of delegates.

The required information on the origin of request (OEI5) should include the host address and the port number. Note that such information is not authenticated in DCE 1.0 but can be authenticated when delegation (a DCE 1.1 feature) is supported.

The naming issue of DCE objects has been addressed for the DACL interface, and the same object names or corresponding UUID's can be used to satisfy OEI6 and TEI1 (the object name is often one of the principal parameters of an access).

---

1. This RFC defines an access event based on TE1 if and only if the server ''controls'' the access; i.e., if the server enforces some form of access control policy (e.g., DACL-based access control). Otherwise, an event definition is unnecessary, because an event should be access irrelevant if it does not involve access control.

In order to satisfy TEI1, each event record should include fields for storing event-specific information for that event.

3. DCE AUDIT-RECORD HEADER SPECIFICATION

Based on the requirements on audit-record contents from the TCSEC, TNI, and POSIX 1003.6 specification on audit record contents, we define the structure of DCE audit records to consist of two parts: (1) a header and (2) zero or more items of event specific information. The header describes the event and subject information, reflecting requirements OEI1 to OEI5. The event specific information items reflect TEI1. These items could also be object names, reflecting requirement OEI6. The proposed audit-record header is illustrated below.

```
Data Type   size Name          Description
------------------------------------------------------------
unsigned16    2  format     format of the record
uuid_t       16  server     uuid of the server principal
unsigned32    4  event      event number
unsigned32    2  outcome    event outcome
unsigned16    2  authz_st   authorization status
uuid_t       16  client     uuid of the client principal
uuid_t       16  cell       uuid of the client cell
unsigned16    2  num_groups number of client's local groups
uuid_t      *16  groups     client's local groups
utc_t        16  time       time when server commits record
char         *1  addr       the address of the client
```

Explanation of header contents:

(a) The `format` field specifies a version number of the event's tail format used for the event specific information. With this format version number, the audit analysis tools can accommodate changes in the formats of the event specific information. For example, the event-specific information of an event may initially be defined to be a 32-bit integer, and later changed to a character string. Format version 0 (zero) is assigned to the initial format for each event.

(b) The `server` field specifies the UUID of the server which generates the audit record.

(c) The `event` field specifies the event number. This field is suggested by the Orange-Book Requirement OEI3. A proposed scheme for event-number assignment is described in the companion RFC [RFC 29.1].

(d) The `outcome` field is suggested by the Orange-Book Requirement OEI4. This field indicates whether the event failed or succeeded, and, if failed, the reason of the failure.

(e) The `authz_st` field indicates how the client is authorized: by a name or by a DCE PAC (Privilege Attribute Certificate).

(f) The `client` and `cell` fields are the UUID's of the client and its cell. These fields are suggested by the Orange-Book Requirement OEI2.

(g) The `num_groups` and `groups` fields can be used to record the local group privileges that are used by the client to get the access permission. The group information will, by default, not be included in the header (num_groups is set to 0 in this case), to minimize the size of the audit records. If the group information is deemed as important, it can be included. Foreign groups (global groups that do not belong to the same cell where the client is registered) information is not included in this version of audit header but may be included in later versions when global groups are supported.

(h) The `time` field is suggested by the Orange-Book Requirement OEI1. This is a timestamp of `utc_t` type recording the time when the server commits the audit record; i.e., after providing the event information through audit API function calls. To maintain the actual sequence of events, recording this time is more appropriate than recording the time when the audit record is actually appended to an audit trail, because the implementation of the audit subsystem may involve communication between the server and a remote audit daemon. Indefinite delays may be incurred by network problems or intruders. The inaccuracy in the utc_t timestamp may be useful for correlating events. When searching for events in an audit trail that occur within a time interval, if the results of comparisons between the time of an event and the interval's starting and ending times is maybe (due to the inaccuracies), then the event should be returned.

(i) The addr field can be used to record the client's address (port address of the caller). Port addresses are not authenticated in DCE1.0; i.e., a caller could provide a fake port address to a DCE server. However, if this unauthenticated port address is deemed to be useful information, a DCE server can record this information using this field, reflecting the Orange-Book Requirement OEI5.

The identity of the server cell is not recorded in the header, since we assume that all audit records in an audit trail are for servers within a single cell, and implicitly, the server cell is the local cell.

4. DEFINING AUDITABLE EVENTS FOR DCE SECURITY AND TIME SERVICES

For audit records to be trustworthy, they have to be generated by components of a well-defined TCB; i.e., a TCB with its boundary delineated. Without first delineating the TCB boundary, the identification of auditable events and their code points would be more difficult and error prone. For example, one might think the sec_login_valid_and_cert_ident() function performs authentication and should be audited. However, this observation offers no clue as to at which *TCB boundary(ies)* auditable events are occurring due to a call of this function, and by whom these events should be audited. In fact, there are two events that are occurring at two TCB boundaries. The first event is a *login* event to the local OS (a TCB) and should be audited by the local OS using the traditional local audit calls. The second event is a *ticket-granting* event to the DCE Security Server (a cell's TCB), and it should be audited by the Security Server at its ticket-granting interface.

We define the TCB boundary of a DCE cell to be the network interfaces of the DCE servers. The DCE security server provides two types of network interface to clients; i.e., DCE RPC interfaces and a UDP/IP port interface. The DCE time server provides only RPC interfaces.

Security Server RPC interfaces include krb5rpc, rdaclif, rdacliftmp, rpriv, rs_acct, rs_query, rs_rpladmn, rs_update, rsec_cert, and secidmap. All the RPC interfaces are offered using the rpc_c_authn_dce_secret authentication service. The security server's RPC runtime uses dce-rgy as its authentication identity. Within the same process, the security server's UDP/IP interface provides Kerberos AS/TGS functions, with krbtgt/cell_name as its authentication identity.

Time Server RPC interfaces include those for managing time service and requesting/providing time: time_control, time_service, gbl_time_service, time_provider.

We can derive the list of interfaces and their operations from the idl files in DCE 1.0 security-server and time-server source trees (local idl-interface definitions are discarded since these interfaces are not used by clients to interact with the security server). For each interface operation, auditable events should be defined when appropriate. In the following we classify the events that should be auditable by interpreting the TCSEC and TNI requirements in the context of DCE. We have identified the auditable events within the security and time servers using this classification (see Appendices A and B).

(a) Authentication (OE1).

Auditable event(s) should be defined for an attempt at invoking Authentication Server/Ticket-Granting Server/Privilege Server (AS/TGS/PS) operations. These events should record the source address of the request.

(b) Object Deletion (OE3).

Security-Server objects include:

(i) *ACLs*
(ii) *accounts*
(iii) *pgo items*
(iv) *registry properties*
(v) *registry/organization policies*
(vi) *registry master key*

Time-Server objects include:

(i) *time service processes*
(ii) *clock readings*
(iii) *global-set membership* (in the Cell Service Profile)
(iv) *time service attributes*

(c) Changing Security State (OE5).

Auditable event(s) should be defined for an attempt at invoking an operation that modifies security-server objects or updates an ACL.

(d) Controlled Access (TE1).

Auditable event(s) should be defined for an attempt at invoking operations which involve access control. This is the most general event class since almost all server interface operations involve access control.

(e) Network Exceptional Conditions (TE2).

Requirement TE2 is irrelevant to auditing the RPC interfaces of the DCE security server and the time server, since network security-relevant exceptional conditions are handled by the RPC runtime, not within the server code. This requirement is only relevant to auditing in the RPC runtime (e.g., a failed client response to the server's challenge, detected replays) and the TGS's UDP interface (e.g., invalid ticket requests). Auditable event(s) should be defined for these network security-relevant exceptional conditions.

(f) Cryptographic (TE3).

Auditable event(s) should be defined for the usage of cryptographic keys in the RPC runtime. Authentication (AS/TGS/PS) events usually involve cryptographic operations. In this case these events belong to the Cryptographic event class too.

(g) Configuration (TE4).

Auditable event(s) should be defined for an attempt at changing the maintenance/operation states of the registry server and time server.

The definition of the auditable events that are required according to the above classification/criteria also allows the security-relevant exceptional conditions to be captured in the same events. This reflects Requirement OE5. The DCE audit design must provide the functionality to select only failed events.

Note that Requirement OE2 is irrelevant to DCE security-server and time-server audit since any access to this interface affects neither the caller's address space, nor the address space on the server employed on behalf of a remote caller. Also note that OE4 must be satisfied by subject-oriented audit selection functionality of the DCE audit design.

Classifying auditable events properly allows system administrators to efficiently select events to be recorded. However, the event classification presented above only reflects the requirements on auditable events and is not necessarily efficient. The audit subsystem design should allow additional event classes to be defined. For example, it is useful to define event classes that include all events for a specific type of objects; e.g., event class Account Operation would include the following events:

(a) `ACCT_Add`
(b) `ACCT_Delete`
(c) `ACCT_Rename`
(d) `ACCT_Lookup`

5. DATA TYPES

This section describes the data types used by the DCE audit interfaces. Types and structures in this section shall be defined in <dce/audit.h>

5.1. dce_aud_ev_info_t Type

The event information structure `dce_aud_ev_info_t`, as illustrated in Table 5-1, is used to hold the event-specific information which a server wants to include in the audit record for an event.

```
    Table 5-1 -- dce_aud_ev_info_t data structure
  ==================================================
      typedef struct {
          unsigned16 format;
          union {
              idl_small_int small_int;
              idl_short_int short_int;
              idl_long_int long_int;
              idl_hyper_int hyper_int;
              idl_usmall_int usmall_int;
              idl_ushort_int ushort_int;
              idl_ulong_int ulong_int;
              idl_uhyper_int uhyper_int;
              idl_short_float short_float;
              idl_long_float long_float;
              idl_boolean boolean;
              uuid_t uuid;
              utc_t utc;
              sec_acl_t * acl;
              idl_byte * byte_string;
              idl_char * char_string;
          } data;
      } dce_aud_ev_info_t;
  ==================================================
```

The `format` field defines formatting information that is used to determine the type of the data referenced by the `data` field. Possible values of the `format` field are:

(a) `aud_c_evt_info_small_int` (1 byte)
(b) `aud_c_evt_info_short_int` (2 bytes)
(c) `aud_c_evt_info_long_int` (4 bytes)
(d) `aud_c_evt_info_hyper_int` (8 bytes)
(e) `aud_c_evt_info_usmall_int` (1 byte)
(f) `aud_c_evt_info_ushort_int` (2 bytes)
(g) `aud_c_evt_info_ulong_int` (4 bytes)
(h) `aud_c_evt_info_uhyper_int` (8 bytes)
(i) `aud_c_evt_info_short_float` (4 bytes)
(j) `aud_c_evt_info_long_float` (8 bytes)
(k) `aud_c_evt_info_boolean` (1 byte)
(l) `aud_c_evt_info_uuid` (16 bytes)
(m) `aud_c_evt_info_utc` (16 bytes)
(n) `aud_c_evt_info_acl` (variable length)
(o) `aud_c_evt_info_byte_string` (variable length)
(p) `aud_c_evt_info_char_string` (variable length)

Byte strings and character strings are terminated with a 0 (zero) byte. New data types may be added to this list if they are used frequently. Servers could use pickling service of the IDL compiler to encode complex data types into byte strings to be included in an audit record.

5.2. dce_aud_hdr_t Values

The header structure `dce_aud_hdr_t` is used to store header information from an audit record. The structure represents the header contents specified in Section 3.

```
typedef struct {
        unsigned32       format;    /* event tail format */
        uuid_t           server;    /* server uuid */
        unsigned32       event;     /* event number */
        unsigned16       outcome;   /* event outcome */
        unsigned16       authz_st;  /* authorization status */
        uuid_t           client;    /* client principal */
        uuid_t           cell;      /* client cell */
        unsigned16       num_groups;/* no. of client groups */
        utc_t            time;      /* record commit time */
        [string, ptr]
        char             *addr;     /* client address */
        [size_is(num_groups), ptr]
        uuid_t           *groups;   /* client groups */
} dce_aud_hdr_t;
```

5.3. dce_aud_rec_t Values

This typedef defines an opaque, implementation-defined audit record descriptor. An audit record descriptor is created/manipulated/disposed by the following functions (specified in Section 6): The functions `dce_aud_start()` and `dce_aud_next()` return an audit record descriptor; the `dce_aud_get*()` and `dce_aud_put*()` functions manipulate the audit record via this descriptor. The `dce_aud_commit()` function logs (or raises an alarm using) the record associated with this descriptor. The `dce_aud_discard()` function disposes of this descriptor.

5.4. dce_aud_trail_t Values

This typedef defines an opaque, implementation-defined audit trail descriptor. For example, the descriptor would contain RPC binding information if a local or remote audit daemon is designated to collect audit records before they are written to a trail. An audit trail descriptor is created/manipulated/disposed by the following functions (specified in Section 6): `dce_aud_open()` returns an audit trail descriptor, `dce_aud_next()` and `dce_aud_commit()` gets or commits an audit record from and to an audit trail via this descriptor. The `dce_aud_close()` function disposes of this descriptor.

6. PROPOSED AUDIT LOGGING API

The following proposed audit logging API functions are provided to create and alter audit records in internal buffer storage, to commit those records to an audit trail, and to free up the buffer storage. A suggested implementation is described in the companion RFC [RFC 29.1]

Summary of DCE audit logging API functions:

(a) `dce_aud_start()` — Initializes an audit record using an RPC binding handle.

(b) `dce_aud_start_with_name()` — Initializes an audit record using the client's name.

(c) `dce_aud_start_with_pac()` — Initializes an audit record using the client's PAC.

(d) `dce_aud_put_ev_info()` — Adds event specific information to an initialized audit record.

(e) `dce_aud_commit()` — Commits a specified audit record (appends it to a specified audit trail).

(f) `dce_aud_discard()` — Discards (frees the allocated memory of) a specified audit record.

(g) `dce_aud_next()` — Reads next audit record from a specified audit trail.

(h) `dce_aud_get_header()` — Returns a pointer to the header structure of a specified audit record.

(i) `dce_aud_get_ev_info()` — Returns a pointer to the next event-information item in the tail of an audit record.

(j) `dce_aud_length()` — Returns the length of a specified audit record.

(k) `dce_aud_print()` — Transforms a specified audit record into human readable form.

(l) `dce_aud_open()` — Opens a specified audit trail for read or write.

(m) `dce_aud_close()` — Releases the data structures associated with a specified audit trail descriptor.

6.1. Audit Record Creation/Initialization

Function `dce_aud_start()`: Determines whether a specified event should be audited given the client-binding information and the event outcome. If the event should be audited or if it is uncertain whether the event should be audited (because the event outcome is still unknown), then an audit record descriptor is initialized and returned to the caller.

6.1.1. Synopsis

```
#include <dce/audit.h>

void
dce_aud_start (event, binding, options, outcome, ard, status)
unsigned32 event;
rpc_binding_handle_t binding;
unsigned32 options;
unsigned32 outcome;
dce_aud_rec_t *ard;
unsigned32 *status;
```

6.1.2. Parameters

Input:

(a) `event` — Specifies the event to be audited. This is a 32-bit event number. The encoding scheme of the event numbers is described in the companion RFC [RFC 29.1]. The `event` field in the audit-record header will be set to this number.

(b) `binding` — Specifies the client's RPC binding handle from which the client identification information is retrieved to set the client, cell, num_groups, groups, addr fields in the audit-record header.

(c) `options` — This parameter can be used to specify the optional header information desired (`aud_c_evt_all_info`, `aud_c_evt_group_info`, `aud_c_evt_address_info`). It can also be used to specify the options of always logging an audit record (`aud_c_evt_always_log`) or always sending an alarm message to the standard output (`aud_c_evt_always_alarm`). If any of these two options is selected, then the filter mechanism is bypassed. The value of the `options` parameter is the bit-wise OR of any selected combination of the following option values.

(i) `aud_c_evt_all_info`
(ii) `aud_c_evt_groups_info`
(iii) `aud_c_evt_address_info`
(iv) `aud_c_evt_always_log`
(v) `aud_c_evt_always_alarm`

(d) `outcome` — The event outcome to be stored in the header. The following values for status are supported:

(i) `aud_c_esl_cond_unknown` — The event outcome (denial, failure, or success) is unknown yet.

(ii) `aud_c_esl_cond_success` — The event completed successfully.

(iii) `aud_c_esl_cond_denial` — The event failed due to access denial.

(iv) `aud_c_esl_cond_failure` — The event failed due to reasons other than access denial.

Output:

(a) `ard` — Returns a pointer to an audit record buffer. If the event does not need to be audited because it is not selected by the filters (described in RFC 29.1), or if the environment variable `DCEAUDITOFF` has been set, then a NULL pointer is returned. If the function is called with `outcome` set to `aud_c_esl_cond_unknown`, then it is possible that the function cannot determine whether the event should be audited. In this case, a non-null `ard` is still returned, and a known event outcome has to be provided when the `ard` is used in calling `dce_aud_commit()` (to commit an audit record).

(b) `status` — Returns the status code from this function. The possible status codes and their meanings are as follows:

(i) `aud_s_invalid_binding` — The input binding does not contain valid identification information.

(ii) `aud_s_ok` — The call was successful.

6.1.3. Description

The `dce_aud_start()` function determines if an audit record should be generated for the specified event. The decision is based on the event filters (described in [RFC 29.1]), an environment variable (`DCEAUDITOFF`), the client's identity provided in the `binding` parameter, and the event outcome (if it is provided in the `outcome` parameter). If this event needs to be audited, the function allocates an audit record descriptor and returns (a pointer to) it, i.e., `ard`. If the event needs not be audited, invalid parameter(s) is/are provided, or some internal error(s) has occurred, then a NULL pointer is returned in `ard`. If the `aud_c_evt_always_log` or `aud_c_evt_always_alarm` option is selected, then an audit record descriptor will always be created and returned.

This function obtains the client identity information from the RPC binding handle and record them in the newly created audit record descriptor.

Event-specific information can be added to the record by calling `dce_aud_put_ev_info()` function. This function can be called multiple times after calling `dce_aud_start()` and before calling `dce_aud_commit()`. A completed audit record can be either appended to an audit-trail file or sent to the audit daemon. This is done through calling `dce_aud_commit()`.

Since `dce_aud_start()` requires the client's RPC binding handle as an input parameter, it cannot be used by servers which offer non-RPC interfaces. Two extended functions, `dce_aud_start_with_pac()` and `dce_aud_start_with_name()`, are included for use by these servers.

```
void
dce_aud_start_with_pac (event, client, address, options,
    outcome, ard, status)
unsigned32 event;
sec_id_pac_t *pac;
unsigned_char_t *address;
unsigned32 options;
unsigned32 outcome;
dce_aud_rec_t *ard;
unsigned32 *status;
```

```
void
dce_aud_start_with_name (event, client, server, cell, address,
    options, outcome, status)
unsigned32 event;
unsigned32_char_t *client, *server, *cell, *address;
unsigned32 options;
unsigned32 outcome;
dce_aud_rec_t *ard;
unsigned32 *status;
```

6.2. Adding Event-Specific Information

Function `dce_aud_put_ev_info()`: Adds event specific information to a specified audit record buffer.

6.2.1. Synopsis

```
#include <dce/audit.h>

void
dce_aud_put_ev_info(ard, dce_aud_ev_info_t, status)
dce_aud_rec_t ard;
dce_aud_ev_info_t info;
unsigned32 *status;
```

6.2.2. Parameters

Input:

(a) `ard` — A pointer to an audit record descriptor initialized by `dce_aud_start()`.

(b) `info` — A data structure containing an event-specific information item that is to be appended to the tail of the audit record identified by `ard`.

Output:

(a) `status` — Returns the status code from this functions. The possible status codes and their meanings are as follows:

(i) `aud_s_invalid_descriptor` — The input audit-record descriptor is invalid.

(ii) `aud_s_invalid_ev_info` — The `info` parameter contains illegal values.

(iii) `aud_s_ok` — The call was successful.

6.2.3. Description

The `dce_aud_put_ev_info()` function adds event-specific information to an audit record. The event-specific information is included in an audit record by calling `dce_aud_put_ev_info()` one or more times. The order of the information items included by multiple calls shall be preserved in the audit record by the system, so that they may be read in the same order by the `dce_aud_get_ev_info()` function. This order shall also be observed by the `dce_aud_print()` function. The `info` parameter is a pointer to an instance of the self-descriptive `dce_aud_ev_info_t` structure (defined in Section 5).

6.3. Read an Audit Record

Function `dce_aud_next()`: Reads next audit record from a specified audit trail into a buffer.

6.3.1. Synopsis

```
#include <dce/audit.h>

void
dce_aud_next (at, predicate, ard, status)
dce_aud_trail_t         at;
char                    *predicate;
dce_aud_rec_t           *ard;
unsigned32              *status;
```

6.3.2. Parameters

Input:

(a) `at` — A pointer to the descriptor of an audit trail previously opened by the function `dce_aud_open()`.

(b) `predicate` — There are three possible forms of predicates — attribute=value, attribute>value, attribute<value, where the attribute is one of the audit-record attributes listed below. (Note that the attribute names are case sensitive, and no space is allowed in a predicate expression.) Multiple predicates are specified as a comma separated list of the form attribute1=value1,attribute2>value2, ... (no space is allowed). Note that when multiple predicates are defined the values are logically ANDed together.

(i) SERVER — The UUID of the server principal which generated the record. The attribute value must be an UUID string. Operators allowed: =.

(ii) EVENT — The audit event number. The attribute value must be an integer. Operators allowed: =.

(iii) OUTCOME — The event outcome of the record. The possible attribute value are SUCCESS, FAILURE, or DENIAL. Operators allowed: =.

(iv) STATUS — The authorization status of the client. The possible attribute values are DCE for DCE authorization (PAC based), and NAME for name-based authorization. Operators allowed: =.

(v) CLIENT — The UUID of the client principal. The attribute value must be an UUID string. Operators allowed: =.

(vi) TIME — The time when the record was generated. The attribute value must be a null-terminated string that express an absolute time. Operators allowed: =, <, and >.

(vii) CELL — The UUID of the client's cell. The attribute value must be an UUID string. Operators allowed: =.

(viii) GROUP — The UUID of one of the client's group(s). The attribute value must be an UUID string. Operators allowed: =.

(ix) ADDR — The address of the client. The attribute value must be the string representation of an RPC binding handle. Operators allowed: =.

(x) FORMAT — The format version number of the audit event record. The attribute value must be an integer. Operators allowed: =, <, and >.

If the predicate parameter is a NULL pointer, the next record in the trail is returned.

Output:

(a) ard — A pointer to the audit record descriptor containing the record returned.

(b) status — Returns the status code from this function. The possible status codes and their meanings are as follows:

(i) aud_s_invalid_descriptor — The descriptor of audit trail is invalid.

(ii) aud_s_invalid_predicate — The predicate is invalid.

(iii) aud_s_end_of_trl — End of the trail is reached without finding a matching record.

(iv) aud_s_ok — The call was successful.

6.3.3. Description

This function attempts to read the next record from the audit trail specified with the audit trail descriptor at. This function also defines the predicate to be used to search for the next record and returns a matching record if one exists. The dce_aud_next() function can be used to search for successive records in the trail that match the defined predicate. By default, if no predicate is explicitly defined, the function returns the next record read from the audit trail.

The value returned in ard can be supplied as an input parameter to the functions dce_aud_get_header(), and dce_aud_get_ev_info().

Storage allocated by this function must be explicitly freed by a call to dce_aud_discard() when the application is finishing using the dce_aud_rec_t object, or before the dce_aud_rec_t object is used for another call to dce_aud_next().

If the function successfully reads an audit trail record, the cursor associated with the audit trail descriptor at will be advanced to the point at the next record in the audit trail.

If no appropriate record can be found in the audit trail, a value of zero is returned and the cursor is advanced to the end of the audit trail. If a call is unsuccessful, the position of the cursor is not changed.

6.4. Examine an Audit Record

Only the synopses are provided for these functions because the meanings of the parameters are obvious, and a description for these functions are unnecessary for the purpose of this RFC.

Functions:

(a) `dce_aud_get_header()`

(b) `dce_aud_get_ev_info()`

(c) `dce_aud_length()`

6.4.1. Synopsis

```
#include <sys/audit.h>

void
dce_aud_get_header (ard, header, status)
dce_aud_rec_t   ard;
dce_aud_hdr_t   **header;
unsigned32      *status;

void
dce_aud_get_ev_info (ard, event_info, status)
dce_aud_rec_t     ard;
dce_aud_ev_info_t **event_info;
unsigned32        *status;

unsigned32
dce_aud_length (ard, status)
dce_aud_rec_t     ard;
unsigned32        *status;
```

6.5. Write a Completed Audit Record

Function `dce_aud_commit()`

6.5.1. Synopsis

```
#include <dce/audit.h>

void
dce_aud_commit (at, ard, options, format, outcome, status)
dce_aud_trail_t at;
dce_aud_rec_t ard;
unsigned32 options;
unsigned16 format;
unsigned32 outcome;
unsigned32 *status;
```

6.5.2. Parameters

Input:

(a) `at` — Designates an audit trail to which the completed audit record should be written.

(b) `ard` — Designates an audit record descriptor which was returned by a previous, successful call to the `dce_aud_start()` function. The content of this record buffer will be appended to the audit trail specified by `at`.

(c) `options` — Bitwise OR of option values described below. A value of 0 for `options` shall result in the default operation.

(d) `format` — Event's tail format used for the event specific information. With this format version number, the servers and audit analysis tools can accommodate changes in the formats of the event specific information, or use different formats dynamically.

(e) `outcome` — The event outcome to be stored in the header. Possible event-outcome values are:

(i) `aud_c_esl_cond_unknown` — The event outcome (denial, failure, or success) is unknown yet.

(ii) `aud_c_esl_cond_success` — The event completed successfully.

(iii) `aud_c_esl_cond_denial` — The event failed due to access denial.

(iv) `aud_c_esl_cond_failure` — The event failed due to reasons other than access denial.

Output:

(a) `status` — Returns the status code from this routine. The possible status codes and their meanings are as follows:

(i) `aud_s_invalid_trl_descriptor` — The audit-trail descriptor is invalid.

(ii) `aud_s_invalid_descriptor` — The audit-record descriptor is invalid.

(iii) `aud_s_daemon_unreachable` — The audit record is not committed because the audit daemon is unreachable.

(iv) `aud_s_storage_failure` — The audit record cannot be written to stable storage. This code is meaningful only when `aud_c_evt_commit_sync_no_wait` option (described below) is selected.

(v) `aud_s_ok` — The call was successful.

6.5.3. Description

The `dce_aud_commit()` function determines whether the event should be audited given the event outcome. If it should be audited, the function completes the audit record identified by `ard` and writes it to the audit trail designated by `at`. If any of the `aud_c_evt_always_log` or `aud_c_evt_always_alarm` options is selected, then the event is always audited (logged or an alarm message sent to the standard output).

If one of the `aud_c_evt_commit_sync` and `aud_c_evt_commit_sync_no_wait` options is selected, the function attempts to flush the audit record to stable storage. If the stable-storage write cannot be performed for some reason, the function will either continue to try until the stable-storage write is completed (if option `aud_c_evt_commit_sync` is selected) or return an error status (if option `aud_c_evt_commit_sync_no_wait` is selected). In the former case, the server's thread that calls `dce_aud_commit()` will be forced to wait, i.e., it will not return to the remote client without auditing. If the server itself wants to handle the failure of the immediate stable-storage write, it should use the `aud_c_evt_commit_sync_no_wait`. This would be the case, for example, if the server needs to inform its remote client that the service cannot be performed because of the auditing failure.

6.6. Discard an Audit Record

Only the synopsis is provided since the meaning of the parameters is obvious, and a description of these functions is unnecessary for the purpose of this RFC.

Function: `dce_aud_discard()`

6.6.1. Synopsis

```
#include <dce/audit.h>

void
dce_audit_discard (ard, status)
dce_aud_rec_t ard;
unsigned32 *status;
```

6.7. Format an Audit Record

Function: `dce_aud_print()`

6.7.1. Synopsis

```
#include <dce/audit.h>

void
dce_aud_print (ard, options, buffer, status)
dce_aud_rec_t ard;
unsigned32 options;
char **buffer;
unsigned32 *status;
```

6.7.2. Parameters

Input:

(a) `ard` — An audit record descriptor.

(b) `options` — The options governing the transformation of the binary audit record information into a a character string. The value of the `options` parameter is the bitwise OR of any selected combination of the following option values.

   (i) `aud_c_evt_all_info`
   (ii) `aud_c_evt_groups_info`
   (iii) `aud_c_evt_address_info`
   (iv) `aud_c_evt_specific_info`
   (v) `aud_c_evt_raw_info`

Output:

(a) `buffer` — Returns the pointer to a a character string converted from the audit record specified by `ard`.

(b) `status` — Returns the status code from this routine. The possible status codes and their meanings are as follows:

   (i) `aud_s_invalid_descriptor` — The descriptor of the event record buffer is invalid.

   (ii) `aud_s_ok` — The call was successful.

6.7.3. Description

The `dce_aud_print()` function shall transform the audit record designated by `ard` into a character string and place it in a buffer. (allocated using `malloc()`, to be freed by the caller).

The `options` parameter is set to the bitwise OR of flag values defined in the <dce/audit.h>. A value of 0 for flags shall result in default operation, i.e., none of the group, address, and event specific information is included in the output string.

6.8. Select an Audit Trail

Function `dce_aud_open()`: Open a specified audit trail for read or write.

6.8.1. Synopsis

```
#include <dce/audit.h>

void
dce_aud_open (flags, description, first_evt_num,
              num_of_evts, status)
unsigned32 flags;
char *description;
unsigned32 first_evt_num;
unsigned32 num_of_evts;
dce_aud_trail_t *at;
unsigned32 *status;
```

6.8.2. Parameters

Input:

(a) `flags` — Specifies the mode of opening. The flags parameter is set to the bitwise OR of the following flag values defined in the <dce/audit.h> header.

  (i) `aud_c_trl_open_read`
  (ii) `aud_c_trl_open_write`

(b) `description` — A character string specifying an audit-trail file to be opened. If `description` is NULL, the default audit-trail is opened. When the audit trail is opened for write, the default audit trail is an RPC interface to a local audit daemon.

(c) `first_evt_num` — The lowest assigned audit event number used by the calling server. The scheme for event-number assignment is described in [RFC 29.1].

(d) `num_of_evts` — The number of audit events defined for the calling server.

Output:

(a) `at` — A pointer to an audit trail descriptor. When the audit trail descriptor is no longer needed, it should be released by calling `dce_aud_close()`.

(b) `status` — Returns the status code from this routine. The possible status codes and their meanings are as follows:

  (i) `aud_s_no_permission` — The caller does not have the permission to open the audit trail or to write audit records through the audit daemon.

  (ii) `aud_s_invalid_description` — The description parameter designates an invalid audit-trail file.

  (iii) `aud_s_daemon_unreachable` — The audit daemon is unreachable.

  (iv) `aud_s_ok` — The call was successful.

6.8.3. Description

The `dce_aud_open()` function shall open the audit trail specified by the `description` parameter, or, if `description` is NULL, shall use the default audit trail, which could represent (1) an RPC interface to a local audit daemon (current implementation), (2) an RPC interface to a remote audit daemon. A remote audit daemon would be useful if the calling server did not have local storage for audit trails.

If the `flags` parameter contains `aud_c_trl_open_read`, the specified file (`description` cannot be null in this case) is opened for reading audit records with `dce_aud_next()`. If `flags` contains `aud_c_trl_open_write`, the specified file or the default audit-trail devices (remote or local RPC) is opened/initialized for appending audit records with `dce_aud_commit()`. Exactly one of `aud_c_trl_open_read` and `aud_c_trl_open_write` flags needs to be specified in any call to `dce_aud_open()`.

If the audit trail specified is a file and the calling server does not have the permission to open the file for read/write, a NULL pointer should be returned in `at`, and `status` should be set to `aud_s_no_permission`. The same values will be returned if the default audit trail is used (e.g., through an audit daemon) and if the calling server is not authorized to use the audit daemon to log records.

6.9. Finish Using an Audit Trail

Function `dce_aud_close()`: Release data structures of file openings, RPC bindings, and other memory associated with the audit-trail access designated by a specified audit trail descriptor.

6.9.1. Synopsis

```
#include <dce/audit.h>

int
dce_aud_close (at, status)
dce_aud_trail_t at;
unsigned32 *status;
```

6.9.2. Parameters

Input:

(a) `at` — A pointer to an audit trail descriptor returned by a previous call to `dce_aud_open()`.

Output:

(a) `status` — Returns the status code from this routine. The possible status codes and their meanings are as follows:

  (i) `aud_s_invalid_descriptor` — The audit trail descriptor is invalid.

  (ii) `aud_s_ok` — The call was successful.

7. A SAMPLE PROGRAM

In this section we demonstrate how the event `ACL_Replace` we defined in Appendix A can be captured in the security-server code. The lines that are added for audit instrumentation are marked with a + in the beginning.

(a) Event Type — `ACL_Replace`

(b) Event Info —

```
char string     component_name  (item 1)
uuid_t          manager_type    (item 2)
unsigned16      acl_type        (item 3)
sec_acl_t       old_acl         (item 4)
sec_acl_t       new_acl         (item 5)
```

7.1. File src/security/server/rs/rs_dacl.c

```
PUBLIC void rdacl_replace
(h, component_name, manager_type_p, sec_acl_type, sec_acl_list_p,
 st_p)
handle_t                    h;
sec_acl_component_name_t    component_name;
uuid_t                      *manager_type_p;
sec_acl_type_t              sec_acl_type;
sec_acl_list_t              *sec_acl_list_p;
error_status_t              *st_p;
{
  char                     buf[64];
  char                     acl_type_buf[20];
```

```
   sec_id_t                user_obj, group_obj, *user_p, *group_p;
   rsdb_acl_ids_t          ids;
   sec_acl_permset_t       requested_perms;
   sec_acl_permset_t       parent_perms = NULL;
   sec_rgy_foreign_id_t    client_ids;
   sec_rgy_name_t          *object_name_p;

+  dce_aud_rec_t           ard;
+  sec_acl_list_t          *sec_acl_list_old_p;
+  dce_aud_ev_info_t       ev_info;
+  unsigned32              aud_error_status;

   /* owner permission required on object named by component name */
   requested_perms = sec_acl_perm_owner;
   object_name_p = (sec_rgy_name_t *)component_name;

   user_p = &user_obj; group_p = &group_obj;
   READ_INTEND_WRITE_LOCK(lock_db) {
       switch(sec_acl_type) {
       case sec_acl_type_object:
           sprintf(acl_type_buf, "object");
           break;
       case sec_acl_type_default_object:
           sprintf(acl_type_buf, "initial object");
           break;
       case sec_acl_type_default_container:
           sprintf(acl_type_buf, "initial container");
           break;
       default:
           sprintf(acl_type_buf, "unknown");
           break;
       }

+      /* Allocate an event audit record buffer and initialize the
+         buffer using the identity, address information provided
+         in the RPC binding pointed by "h". */
+      dce_aud_start(ACL_Replace, h, aud_c_evt_all_info,
+                    aud_c_esl_cond_unknown, &ard,
+                    &aud_error_status);

+      /* Prepare event tail data only if the event needs
+       * to be logged */

+       if (ard) {
+          ev_info.format = aud_c_evt_info_char_string;
+          ev_info.data.char_string = object_name_p;
+          dce_aud_put_ev_info(ard, ev_info, &aud_error_status);
+          ev_info.format = aud_c_evt_info_uuid;
+          ev_info.data.uuid = manager_type_p;
+          dce_aud_put_ev_info(ard, ev_info, &aud_error_status);
+          ev_info.format = aud_c_evt_info_ushort_int;
+          ev_info.data.ushort_int = sec_acl_type;
+          dce_aud_put_ev_info(ard, ev_info, &aud_error_status);
+          ev_info.format = aud_c_evt_info_acl;
+          sec_acl_mgr_lookup(sec_acl_mgr, (sec_acl_key_t)&ids,
```

```
+                              manager_type_p, sec_acl_type,
+                              &sec_acl_list_old_p, st_p);
+           ev_info.data.acl = sec_acl_list_old_p->sec_acls[0];
+           dce_aud_put_ev_info(ard, ev_info, &aud_error_status);
+           ev_info.data.acl = sec_acl_list_p->sec_acls[0];
+           dce_aud_put_ev_info(ard, ev_info, &aud_error_status);
+        }

       if(rs_is_authorized(h, requested_perms, parent_perms,
                           *object_name_p, manager_type_p,
                           false, &client_ids, st_p)) {
           rsdb_acl_get_ids_from_name(component_name, sec_acl_type,
                                      manager_type_p, &user_p,
                                      &group_p, &ids, st_p);
       {
           if (GOOD_STATUS(st_p)) {
               READ_INTEND_WRITE_PROMOTE(lock_db);
               sec_acl_mgr_replace(sec_acl_mgr, (sec_acl_key_t)&ids,
                                   manager_type_p, sec_acl_type,
                                   sec_acl_list_p, st_p);
               if(GOOD_STATUS(st_p)) {
+                  if (ard) dce_aud_commit(at, ard, 0, 0,
+                                          aud_c_esl_cond_success,
+                                          &aud_error_status);

                   rs_log_acl_replace(sec_acl_mgr,
                      (sec_acl_key_t)&ids, manager_type_p,
                      sec_acl_type, sec_acl_list_p,
                      (rs_replica_master_info_t *) NULL);
               }
           }
       }
   } END_READ_INTEND_WRITE_LOCK;
   /* return only sec_acl status codes */
   if (STATUS_EQUAL(st_p, sec_rgy_not_authorized)) {
       SET_STATUS(st_p, sec_acl_not_authorized);
+      if (ard) dce_aud_commit(at, ard, 0, aud_c_esl_cond_denial,
+                              &aud_error_status);
   } else if (STATUS_EQUAL(st_p, sec_rgy_object_not_found)) {
       SET_STATUS(st_p, sec_acl_object_not_found);
+      if (ard) dce_aud_commit(at, ard, 0, aud_c_esl_cond_failure,
+                              &aud_error_status);
+  }
+  dce_aud_discard(ard, &aud_error_status);
 }
```

8. ACKNOWLEDGEMENTS

We would like to thank Virgil D. Gligor (University of Maryland, College Park), Narayanan Vasudevan, Ping Lin, Phillip Rogaway, Bob Blakley and R.K. Aditham (IBM), for their comments on the earlier drafts of this RFC.

APPENDIX A. AUDITABLE EVENTS IN DCE SECURITY SERVER

This appendix contains a complete list of auditable events (satisfying NCSC C2 requirements) for DCE Security Services (Version 1.0). A few updates in the list for 1.0.x releases are expected. Code points are represented with a file path-name followed by a function name, separated by a colon. The common prefix,

`/dce/V1.0/src/security/`, of the file path-names has been omitted.

A.1. Authentication Interface (krb5rpc) Operations

(a) `rsec_krb5rpc_sendto_kdc`: This is an RPC interface operation for accessing Kerberos AS/TGS services. Ticket-Granting tickets and application tickets are requested and returned. There is no access control on this interface other than what is within the Kerberos ticket-granting mechanism itself; i.e., the TGS request verification. Note that the control flow of this interface follows the same path that is used by the UDP/IP port interface of AS/TGS. Thus, the events defined here are for both the RPC interface and the UDP/IP interface.

(i) Event Type: `AS_Request` (Authentication, Cryptographic)

Event Specific Info: None

Code Point:

```
krb5/kdc/do_as_req.c: process_as_req()
```

(ii) Event Type: `TGS_TicketReq` (Authentication, Cryptographic)

Event Specific Info: None

Code Point:

```
krb5/kdc/do_tgs_req.c: process_tgs_req()
```

(iii) Event Type: `TGS_RenewReq` (Authentication, Cryptographic)

Event Specific Info: None

Code Point:

```
krb5/kdc/do_tgs_req.c: process_tgs_req()
```

(iv) Event Type: `TGS_ValidateReq` (Authentication, Cryptographic)

Event Specific Info: None

Code Point:

```
krb5/kdc/do_tgs_req.c: process_tgs_req()
```

A.2. DACL Management Interface (rdaclif) Operations

[Unauthenticated callers and callers authenticated by name are attributed a pac that serves as the anonymous unauthenticated user. All fields in this PAC will be zero which maps to the `UUID_NIL` for identities. The authenticate bit will also be false. The scoping cell is set to the local cell — this causes the unauthenticated user to match `other` entries (for the local cell) in an ACL.]

(a) `rdacl_lookup`: Retrieve an acl of an object in the security server. Review of acl associated with an object in security server is allowed if the caller has any access to the object.

Event Type: `ACL_Lookup` (Controlled Access)

Event Specific Info:

```
char string    component_name
uuid_t         manager_type
unsigned16     acl_type
```

Code Point:

```
server/rs/rs_dacl.c: rdacl_lookup()
```

(b) `rdacl_replace`: Replace the acl of an object in the security server. The client must have the `sec_acl_perm_owner` permission for the update to be carried out.

Event Type: `ACL_Replace` (Changing Security State, Controlled Access)

Event Specific Info:

```
char string   component_name
uuid_t        manager_type
unsigned16    acl_type
sec_acl_list_t old_acl_list
sec_acl_list_t new_acl_list
```

Code Point:

```
server/rs/rs_dacl.c: rdacl_replace()
```

(c) `rdacl_get_access`: Determine the caller's access to a specified object. This call is authorized if the caller has any access to the object.

Event Type: `ACL_GetAccess` (Controlled Access)

Event Specific Info:

```
char string    component_name
uuid_t         manager_type
sec_acl_permset_t net_rights
```

Code Point:

```
server/rs/rs_dacl.c: rdacl_get_access()
```

(d) `rdacl_test_access`: Determine if the caller has the requested access. The return value of the call indicates whether the caller has the requested access to the object.

Event Type: `ACL_TestAccess` (Controlled Access)

Event Specific Info:

```
char string    component_name
uuid_t         manager_type
sec_acl_permset_t(unsigned32) desired_permset
```

Code Point:

```
server/rs/rs_dacl.c: rdacl_test_access()
```

(e) `rdacl_test_access_on_behalf`: Return true if the caller has any access and the subject has the requested access to the object.

Event Type: `ACL_TestOnBehalf` (Controlled Access)

Event Specific Info:

```
char string   component_name
uuid_t        manager_type
```

Code Point:

```
server/rs/rs_dacl.c: rdacl_test_access_on_behalf()
```

(f) `rdacl_get_manager_types`: List the types (UUID's) of ACLs protecting an object. The caller must have some permissions on the object for each of the manager types that is defined for the object. Otherwise, no manager type is returned.

Event Type: `ACL_GetMgrTypes` (Controlled Access)

Event Specific Info:

```
char string component_name
unsigned32 (sec_acl_type_t) acl_type
```

Code Point:

```
server/rs/rs_dacl.c: rdacl_get_manager_types()
```

(g) `rdacl_get_printstring`: Returns an array of printable representations for each permission bit or combination of permission bits a specified ACL manager will support. The ACL manager type specified must be one of the types indicated by the ACL handle. This call does not involve access control. No auditable event is defined for this call.

Event Type: No event defined for this operation

(h) `rdacl_get_referral`: Obtain a referral to an acl update site, This function is used when the current acl site yields a `sec_acl_site_readonly` error. Some replication managers will require all updates for a given object to be directed to a given replica. Clients of the generic acl interface may know they are dealing with an object that is replicated in this way. This function allows them to recover from this problem and rebind to the proper update site. Client is required to have execute access on the parent of the object named by `component_name`.

Event Type: `ACL_GetReferral` (Controlled Access)

Event Specific Info:

```
char string  component_name
uuid_t  manager_type
unsigned32(sec_acl_type_t) sec_acl_type
```

Code Point:

```
server/rs/rs_dacl.c: rdacl_get_referral()
```

A.3. Privilege Server Interface (rpriv) Operations

(a) `rpriv_get_ptgt`: Return a privilege certificate to the ticket-granting service. The caller supplies the group set, and the privilege server seals the group set in the authorization portion of a privilege ticket-granting ticket, after first rejecting any groups that are not legitimately part of the caller credentials. A group will be rejected if the caller is not a member of the group, or the group is not allowed on project lists (the projlist_ok flag is not set). There is no access control on this interface other than what was within the Kerberos ticket-granting mechanism itself; i.e., the TGS request verification. This call may result in growth of potential access set.

Event Type: `PRIV_GetPtgt` (Controlled Access, Cryptographic)

Event Specific Info:

```
char string client_address
unsigned16 num_groups  (number of local groups in PAC)
uuid_t  groups (local groups in PAC)
```

Code Point:

```
server/rs/rpriv.c: rpriv_krb_get_ptgt()
```

A.4. Registry Server Account Interface (rs_acct) Operations

(a) `rs_acct_add`: Add an account with a specified login name. The caller needs to have m, a, and u (mgmt_info, auth_info, and user_info) permissions on the principal of the account that is to be added. The constituent principal, group, and organization (PGO) items for an account must be added before the account can be created. Also, the principal must have been added as a member of the specified group and organization.

Event Type: `ACCT_add` (Controlled Access, Changing Security State)

Event Specific Info:

```
char string login_name
```

Code Point:

```
server/rs/rs_acct.c: rs_acct_add()
```

(b) `rs_acct_delete`: Delete an account with a specified login name. The caller needs to have m, a, and u (mgmt_info, auth_info, and user_info) permissions on the principal of the account that is to be deleted.

Event Type: `ACCT_Delete` (Object Deletion, Changing Security State, Controlled Access)

Event Specific Info:

```
char string login_name
```

Code Point:

```
server/rs/rs_acct.c: rs_acct_delete()
```

(c) `rs_acct_rename`: Change the account group or org. The caller has to have the m (management) permission on `old_login_name.pname`.

Event Type: `ACCT_Rename` (Controlled Access)

Event Specific Info:

```
char string old_login_name
char string new_login_name
```

Code Point:

```
server/rs/rs_acct.c: rs_acct_rename()
```

(d) `rs_acct_lookup`: Returns data for a specified account. The caller must have the read right according to the ACL of the account's principal.

Event Type: `ACCT_Lookup` (Controlled Access)

Event Specific Info:

```
char string login_name
```

Code Point:

```
server/rs/rs_acct.c: rs_acct_lookup()
```

(e) `rs_acct_replace`: Replaces both the user and administrative information in the account record specified by the input login name. The administrative information contains limitations on the account's use and privileges. The user information contains such information as the account home directory and default shell. The administrative information can only be modified by a caller with the admin_info (a) privilege for the account's principal. The user information can be modified by a caller with the user_info (u) privileges for the account's principal. Two events are defined for this operation.

(i) Event Type: `ACCT_UserReplace` (Controlled Access)

Event Specific Info:

```
char string login_name
```

Code Point:

```
server/rs/rs_acct.c: rs_acct_replace()
```

(ii) Event Type: `ACCT_AdmnReplace` (Controlled Access, Changing Security State)

Event Specific Info:

```
char string login_name
```

Code Point:

```
server/rs/rs_acct.c: rs_acct_replace()
```

(f) `rs_acct_get_projlist`: Returns members of the project list for the specified account. This operation requires the caller to have the read (r) permission on the account principal for which the project list data is to be returned.

Event Type: `ACCT_GetProjlist` (Controlled Access)

Event Specific Info:

```
char string login_name
```

Code Point:

```
server/rs/rs_acct.c: rs_acct_get_projlist()
```

A.5. Registry Miscellaneous Operation Interface (rs_misc) Operations

(a) `rs_login_get_info`: Returns login information for the specified account. This information is extracted from the account's entry in the registry database. This operation requires the caller to have the read (r) permission on the account's principal from which the data is to be returned.

Event Type: `LOGIN_GetInfo` (Controlled Access)

Event Specific Info:

```
char string login_name
```

Code Point:

```
server/rs/rs_misc.c: rs_login_get_info()
```

(b) `rs_wait_until_consistent`: not implemented, no event defined.

A.6. Registry PGO Interface (rs_pgo) Operations

(a) `rs_pgo_add`: Adds a PGO item to the registry database. This operation requires the caller to have the insert (i) permission on the parent directory in which the PGO item is to be created.

Event Type: `PGO_Add` (Changing Security State, Controlled Access)

Event Specific Info:

```
signed32(sec_rgy_domain_t) name_domain
char string name (item name)
```

Code Point:

```
server/rs/rs_pgo.c: rs_pgo_add()
```

(b) `rs_pgo_delete`: Deletes a PGO item from registry database. Any account depending on the deleted PGO item is also deleted. The deletion operation requires the caller to have the delete (d) permission on the parent directory that contains the PGO item to be deleted and Delete_object (D) permission on the PGO item itself.

Event Type: `PGO_Delete` (Object Deletion, Changing Security State, Controlled Access)

Event Specific Info:

```
signed32(sec_rgy_domain_t) name_domain
char string name (item name)
```

Code Point:

```
server/rs/rs_pgo.c: rs_pgo_delete()
```

(c) `rs_pgo_replace`: Replaces the data associated with a PGO item in the registry database. The caller needs to have the mgmt_info (m) permission on the PGO item, if `quota`, `flags`, or `unix_num` is being set. (Only a cell principal's `unix_num` is modifiable.) The caller needs to have the fullname (f) permission to modify the fullname of the PGO item.

Event Type: `PGO_Replace` (Changing Security State, Controlled Access)

Event Specific Info:

```
signed32(sec_rgy_domain_t) name_domain
char string name (item name)
```

Code Point:

```
server/rs/rs_pgo.c: rs_pgo_replace()
```

(d) `rs_pgo_get`: Returns the name and data for a PGO item. The desired item is identified by a query key, which can be a name, a uuid, a unix_num, or a sequential-search flag. The caller needs to have the read (r) permission on the PGO item to be viewed.

Event Type: `PGO_Get` (Controlled Access)

Event Specific Info:

```
signed32(sec_rgy_domain_t) name_domain
uuid_t item (the uuid of the item being searched)
unsigned32(rs_pgo_query_key_t) key  (the query key)
```

Code Point:

```
server/rs/rs_pgo.c: rs_pgo_get()
```

(e) `rs_pgo_key_transfer`: Performs a specified key transfer between the uuid, unix_num, and name of a PGO item. The caller needs to have some permission on the PGO item for id -> name and unix_num -> name transfers.

Event Type: `PGO_KeyTransfer` (Controlled Access)

Event Specific Info:

```
signed32(sec_rgy_domain_t) name_domain
uuid_t item (the uuid of the item)
unsigned32(rs_pgo_query_key_t) key (query key)
unsigned32(rs_pgo_query_t) requested_result_type
```

Code Point:

```
server/rs/rs_pgo.c: rs_pgo_key_transfer()
```

(f) `rs_pgo_add_member`: Add a member to a group or an organization. The caller must have the Member_list (M) permission on the group or organization. Additionally, if this call is for adding a group member, the caller must have the group (g) permission on the principal to be added.

Event Type: `PGO_AddMember` (Changing Security State, Controlled Access)

Event Specific Info:

```
signed32(sec_rgy_domain_t) name_domain
uuid_t princ  (principal's uuid)
uuid_t go   (group or organization's uuid)
```

Code Point:

```
server/rs/rs_pgo.c: rs_pgo_add_member()
```

(g) `rs_pgo_delete_member`: Delete a principal from a group or an organization in the registry database. The caller must have the Member_list (M) permission on the group or organization. Note that the caller does not need to have the group (g) permission on the principal to be deleted for deleting the principal from a group.

Event Type: `PGO_DeleteMember` (Changing Security State, Controlled Access)

Event Specific Info:

```
signed32(sec_rgy_domain_t) name_domain
uuid_t princ (principal's uuid)
uuid_t go  (group or organization's uuid)
```

Code Point:

```
server/rs/rs_pgo.c: rs_pgo_delete_member()
```

(h) `rs_pgo_is_member`: Test whether a specified principal is a member of a specified group or organization. The caller must have test permission on the group or organization.

Event Type: `PGO_IsMember` (Controlled Access)

Event Specific Info:

```
signed32(sec_rgy_domain_t) name_domain
uuid_t princ    (principal's uuid)
uuid_t go   (group or organization's uuid)
```

Code Point:

```
server/rs/rs_pgo.c: rs_pgo_is_member()
```

(i) `rs_pgo_get_members`: If the specified domain is group or organization, list the members of a specified group or organization. If the domain is principal, list the groups in which the principal is a member. The caller must have the read (r) permission on the principal, group, or organization.

Event Type: `PGO_GetMembers` (Controlled Access)

Event Specific Info:

```
signed32(sec_rgy_domain_t) name_domain
uuid_t pgo (PGO's uuid)
```

Code Point:

```
server/rs/rs_pgo.c: rs_pgo_get_members()
```

A.7. Registry Policy Interface (rs_policy) Operations

(a) `rs_properties_get_info`: Return a list of registry properties. The caller must have the read (r) permission on the policy object from which the property information is to be returned.

Event Type: `PROP_GetInfo` (Controlled Access)

Event Specific Info: None

Code Point:

```
server/rs/rs_policy.c: rs_properties_get_info()
```

(b) `rs_properties_set_info`: Set the registry properties. The caller must have the mgmt_info (m) permission on the policy object for which the property information is to be set.

Event Type: `PROP_SetInfo` (Controlled Access)

Event Specific Info: None

Code Point:

```
server/rs/rs_policy.c: rs_properties_get_info()
```

(c) `rs_policy_get_info`: Return the policy for a specified organization or the registry (if no organization name is specified). The caller must have the read (r) permission on the policy object or organization item from which the data is to be returned.

Event Type: `POLICY_GetInfo` (Controlled Access)

Event Specific Info:

```
uuid_t   org
```

Code Point:

```
server/rs/rs_policy.c: rs_policy_get_info()
```

(d) `rs_policy_set_info`: Set the policy for a specified organization or the registry (if no organization name is specified).

Event Type: `POLICY_SetInfo` (Changing Security State, Controlled Access)

Event Specific Info:

```
uuid_t   org
```

Code Point:

```
server/rs/rs_policy.c: rs_policy_set_info()
```

(e) `rs_policy_get_override_info`

(f) `rs_policy_set_override_info`

The override design is not in DCE 1.0. NO access control has been implemented for these two operations.

(g) `rs_auth_policy_get_info`: Return the authentication policy for a specified account or the registry (if no account is specified). The caller must have the read (r) permission on the policy object or account/s principal from which the data is to be returned.

Event Type: `AUTHPLCY_GetInfo` (Controlled Access)

Event Specific Info:

```
char string acct_name
```

Code Point:

```
server/rs/rs_policy.c: rs_auth_policy_get_info()
```

(h) `rs_auth_policy_get_effective`: Return the effective authentication policy for an account. If no account is specified, the authentication policy for the registry is returned. The caller must have the read (r) permission on the policy object of the registry. If an account is specified, the caller must also have the read (r) permission on the account's principal.

No new event defined for this operation. `AUTHPLCY_GetInfo` is used here.

Code Point:

```
server/rs/rs_policy.c: rs_auth_policy_get_effective()
```

(i) `rs_auth_policy_set_info`: Set the authentication policy for an account or the registry (if no account is specified). The caller must have the auth_info (a) permission on the account's principal or policy object of the registry.

Event Type: `AUTHPLCY_SetInfo` (Changing Security State, Controlled Access)

Event Specific Info:

```
char string acct_name
```

Code Point:

```
server/rs/rs_policy.c: rs_auth_policy_set_info()
```

A.8. Registry Administration Interface Operations

(a) `rs_rep_admin_stop`: Direct the registry server to stop servicing remote procedure calls. The caller must have Admin (A) permission on the registry policy object.

Event Type: `REPADMIN_Stop` (Controlled Access, Configuration)

Event Specific Info: None

Code Point:

```
server/rs/rs_rpladmn.c: rs_rep_admin_stop()
```

(b) `rs_rep_admin_maint`: Direct the registry server into (checkpoint the database, close files, etc.) or out of maintenance state. The caller must have Admin (A) permission on the registry policy object.

Event Type: `REPADMIN_Maint` (Controlled Access, Configuration)

Event Specific Info:

```
boolean       into_maintenance
```

Code Point:

```
server/rs/rs_rpladmn.c: rs_rep_admin_maint()
```

(c) `rs_rep_admin_mkey`: Direct the registry to change its master key and re-encrypt account keys using the new master key. The caller must have Admin (A) permission on the registry policy object.

Event Type: `REPADMIN_Mkey` (Controlled Access, Cryptographic)

Event Specific Info: None

Code Point:

```
server/rs/rs_rpladmn.c: rs_rep_admin_mkey()
```

(d) `rs_rep_admin_info`: Awaiting new design in DCE 1.x.x. No access control in DCE 1.0.

A.9. Certification Interface (rsec_cert) Operations

(a) `rsec_login_validate_cert_auth`: No event defined

A.10. Identifier Mapping Interface (secidmap) Operations

(a) `rsec_id_parse_name`: Translate a global name into principal and cell names and UUID's. If the principal's uuid is requested, the caller must have at least one permission of any kind on the principal item.

Event Type: `SECID_ParseName` (Controlled Access)

Event Specific Info:

```
char string global_name
```

Code Point:

```
server/rs/rs_secidmap.c: rsec_id_parse_name()
```

(b) `rsec_id_gen_name`: Generate a global name from cell and principal UUID's. The caller must have at least one permission of any kind on the specified principal.

Event Type: `SECID_GenName` (Controlled Access)

Event Specific Info:

```
char string global_name
```

Code Point:

```
server/rs/rs_secidmap.c: rsec_id_gen_name()
```

A.11. Registry Interfaces not Implemented

```
rs_query, rs_query_are_you_there,
rs_update, rs_update_are_you_there
```

APPENDIX B. AUDITABLE EVENTS IN TIME SERVICE

This appendix contains a complete list of auditable events (satisfying NCSC C2 requirements) for DCE Time Service (Version 1.0). A few updates in the list for 1.0.x releases are expected. Code points are represented with a file path-name followed by a function name, separated by a colon. The common prefix, `/dce/V1.0/src/time/`, of the file path-names has been omitted.

B.1. Control Interface (time_control) Operations

(a) `CreateCmd`: Create the Time Service as a server or a clerk. The caller must have write access to the management interface.

Event Type: `CNTRL_Create` (Controlled Access, Configuration)

Event Specific Info: signed32 servType

Code Point:

```
service/mgtrpc.c: CreateCmd()
```

(b) `DeleteCmd`: Delete the Time Service. This command stops the process. The caller must have write access to the management interface.

Event Type: `CNTRL_Delete` (Object Deletion, Controlled Access)

Event Specific Info: None

Code Point:

```
service/mgtrpc.c: DeleteCmd()
```

(c) `EnableCmd`: Enable the Time Service. This command makes the server available to the network. The `clockSet` argument tells the Time Service whether or not to set the clock after the first synchronization. The caller must have write access to the management interface.

Event Type: `CNTRL_Enable` (Controlled Access, Configuration)

Event Specific Info: signed32 clockSet

Code Point:

```
service/mgtrpc.c: EnableCmd()
```

(d) `DisableCmd`: Disable the Time Service by making it unavailable to the network. In the case of servers, it makes it unavailable to RPC client trying to talk to it. For clerks, stop synchronizing with servers. The caller must have write access to the management interface.

Event Type: `CNTRL_Disable` (Controlled Access, Configuration)

Event Specific Info: None

Code Point:

```
service/mgtrpc.c: DisableCmd()
```

(e) `UpdateCmd`: This command causes the Time Service to synchronize with the time provided in `timeVal`. The caller must have write access to the management interface.

Event Type: `CNTRL_Update` (Controlled Access)

Event Specific Info:

```
utc_t   old_time
utc_t   new_time
```

Code Point:

```
service/mgtrpc.c: UpdateCmd()
```

(f) `ChangeCmd`: This command changes the epoch number on the server and optionally sets the time to a new time. These values are passed in the argument `changeDir`. The caller must have write access to the management interface.

Event Type: `CNTRL_Change` (Controlled Access)

Event Specific Info:

```
signed32        old_epoch
signed32        new_epoch
utc_t           old_time
utc_t           new_time
```

Code Point:

```
service/mgtrpc.c: ChangeCmd()
```

(g) `SynchronizeCmd`: This command causes the Time Service to synchronize immediately. If the argument `clockSet` is true, the clock is set to the new value after a synchronization. The caller must have write access to the management interface.

Event Type: `CNTRL_Sync` (Controlled Access)

Event Specific Info: signed32 setClock

Code Point:

```
service/mgtrpc.c: SynchronizeCmd()
```

(h) `AdvertiseCmd`: This command adds (Advertises) this Time Server node as a member of the global set in the Cell Services Profile. The caller must have write access to the management interface.

Event Type: `CNTRL_Advt` (Controlled Access)

Event Specific Info: None

Code Point:

```
service/mgtrpc.c: AdvertiseCmd()
```

(i) `UnadvertiseCmd`: This command removes (Unadvertises) this Time Server node as a member of the set of global servers in the Cell Services profile. The caller must have write access to the management interface.

Event Type: `CNTRL_Unadvt` (Controlled Access)

Event Specific Info: None

Code Point:

```
service/mgtrpc.c: UnadvertiseCmd()
```

(j) `SetDefaultCmd`: This command sets an attribute to its default value. The attribute type is passed in the `setAttr` argument. The caller must have write access to the management interface.

Event Type: `CNTRL_SetDflt` (Controlled Access)

Event Specific Info:

```
byte useDefault (true, set attribute using default)
signed32 attribute (attribute id)
```

Code Point:

```
service/mgtrpc.c: SetDefaultCmd()
```

(k) `SetAttrCmd`: This command sets an attribute to a value given by the manager. The attribute type is passed in `setAttr` argument and the attribute value in `AttrValue` argument. The caller must have write access to the management interface.

Event Type: `CNTRL_SetAttr` (Controlled Access)

Event Specific Info:

```
signed32 attribute (attribute id)
signed32 attribute_type
```

Code Point:

```
service/mgtrpc.c: SetAttrCmd()
```

(l) `ShowAttrCmd`: This command queries the time service for the value of a particular attribute. The attribute value is passed back in the argument `attrValue`. The caller must have read access to the management interface.

Event Type: `CNTRL_ShowAttr` (Controlled Access)

Event Specific Info:

```
signed32 attribute (attribute id)
signed32 attribute_type
```

Code Point:

```
service/mgtrpc.c: ShowAttrCmd()
```

(m) `ShowAllCharsCmd`: This command queries the time service for the values of all the characteristic attributes. The caller must have read access to the management interface.

Event Type: `CNTRL_ShowAllCrs` (Controlled Access)

Event Specific Info: None

Code Point:

```
service/mgtrpc.c: ShowAllCharsCmd()
```

(n) `ShowAllStatusCmd`: This command queries the time service for the values of all the status attributes. The caller must have read access to the management interface.

Event Type: `CNTRL_ShowAllSts` (Controlled Access)

Event Specific Info: None

Code Point:

```
service/mgtrpc.c: ShowAllStatusCmd()
```

(o) `ShowAllCntrsCmd`: This command queries the time service for the values of all the counters. The caller must have read access to the management interface.

Event Type: `CNTRL_ShowAllCntrs` (Controlled Access)

Event Specific Info: None

Code Point:

```
service/mgtrpc.c: ShowAllCntrsCmd()
```

(p) `ShowLocServersCmd`: This command queries the time service for the servers in the local set. A variable conformant array is used to return the set of local servers available. The size of the array transmitted over RPC is determined at run-time. The caller must have read access to the management interface.

Event Type: `CNTRL_ShowLocServers` (Controlled Access)

Event Info Structure: None

Code Point:

```
service/mgtrpc.c: ShowLocServersCmd()
```

(q) `ShowGblServersCmd`: This command queries the time service for the servers in the global set. A variable conformant array is used to return the set of global servers available. The caller must have read access to the management interface.

Event Type: `CNTRL_ShowGblServers` (Controlled Access)

Event Specific Info: None

Code Point:

```
service/mgtrpc.c: ShowGblServersCmd()
```

B.2. Time Provider Interface (time_provider) Operations

Auditable events in the RPC-based Time Provider Program (TPP) interfaces are defined here. These events are invoked by a Time Service daemon running as a server (in this case it makes an RPC client call to the TPP server).

(a) `ContactProvider`: Send initial contact message to the TPP. The TPP server responds with a control message. This operation may cause modification of the time server's (not the provider's) clock and should be defined to be an auditable event in the time server. There is no access control in the provider for this operation, but the integrity of the messages is protected.

Event Type: `PROVIDER_Contact` (Controlled Access)

Event Specific Info: None

Code Point: DCE 1.0 does not contain complete client code in the time server to use the time provider interface.

(b) `ServerRequestProviderTime`: The client sends a request to the TPP for times. The TPP server responds with an array of time stamps obtained by querying the Time Provider hardware that it polls. There is no access control in the time provider for this operation, but the integrity of the message is protected.

Event Type: `PROVIDER_ReqTime` (Controlled Access)

Event Specific Info: None

Code Point: DCE 1.0 does not contain complete client code in the time server to use the time provider interface.

B.3. Global DTS Time Server Interface (gbl_time_service) Operations

(a) `ClerkRequestGlobalTime`: This routine handles RPC time requests sent by a clerk to a DTS global server. The server returns two arguments to the clerk: its current utc time and a processing delay. The processing delay is expressed in nanoseconds. Currently there is no mechanism in RPC that can give an accurate measure of the time it takes the server to process the request, but this field is reserved here for when such mechanism exists. This operation may cause modification of the time clerk's (not the server's) clock and should be defined to be an local auditable event in the time clerk. There is no access control in the server for this operation, but the integrity of the messages is protected.

Event Type: No auditable event is defined for this operation

(b) `ServerRequestGlobalTime`: This routine handles RPC time requests sent by a server to another DTS global server. The server returns four arguments. The current utc time, a processing delay and the server's current epoch. This operation may cause modification of the calling time server's (not the servicing time server's) clock and should be defined to be an auditable event in the calling time server. There is no access control in the servicing time server for this operation, but the integrity of the messages is protected.

Event Type: No auditable event is defined for this operation

B.4. DTSS Server Interface (time_service) Operations

(a) `ClerkRequestTime`: This routine handles RPC time requests sent by a clerk to a DTSS server. The server returns two arguments to the clerk: its current utc time and a processing delay. Currently there is no mechanism in RPC that can give an accurate measure of the time it takes the server to process the request, but this field is reserved here for when such mechanism exists. This operation may cause modification of the calling time clerk's (not the servicing time server's) clock and should be defined to be an auditable event in the calling time clerk. There is no access control in the servicing time server for this operation, but the integrity of the messages is protected.

Event Type: No auditable event is defined for this operation.

(b) `ServerRequestTime`: This routine handles RPC time requests sent by a server to another DTSS server. The server returns four arguments. The current utc time, a processing delay, the server's current epoch and its courier role. This operation may cause modification of the calling time server's (not the servicing time server's) clock and should be defined to be an auditable event in the calling time server. There is no access control in the servicing time server for this operation, but the integrity of the messages is protected.

Event Type: No auditable event is defined for this operation.

REFERENCES

[NCSC 85] National Computer Security Center, ''Department of Defense Trusted Computer System Evaluation Criteria,'' DOD 5200.28-STD, December 1985.

[NCSC 87] National Computer Security Center, ''Trusted Network Interpretation of the Trusted Computer System Evaluation Criteria,'' NCSC-TG-005, July 31, 1987.

[NCSC 88] National Computer Security Center, ''A Guide to Understanding Audit in Trusted Systems,'' NCSC-TG-001, June 1, 1988.

[RFC 8.0] M. Gasser, ''DCE SIG Security Requirements,'' July 1992.

[RFC 29.1] S. Luan, R. Weisz, ''A Design of an Audit Subsystem for DCE — Implementation Specification'', November 1993.

AUTHOR'S ADDRESSES

Shyh-Wei Luan
VDG Inc.
6009 Brookside Drive
Chevy Chase MD 20815
USA
Internet email: luan@vnet.ibm.com
Telephone: +1-301-240-7385

Robert Weisz
IBM Canada Laboratory
1150 Eglinton Ave. East
North York, Ontario M3C1H7
CANADA
Internet email: weisz@torolab6.vnet.ibm.com
Telephone: +1-416-448-3131