OSF DCE SIG
Request For Comments: 29.0

S. Luan (VDG)
R. Weisz (IBM)
December 1992

DESIGN OF AN AUDIT SUBSYSTEM FOR DCE — IMPLEMENTATION SPECIFICATION

1. INTRODUCTION

Audit is one of the high-priority security features to be supported in DCE 1.1 and later versions as voted by the DCE Security Working Group [RFC 8.0]. In a companion RFC [RFC 28.0], we identify the auditable events in the DCE Security and Time Servers by interpreting the NCSC Class-C2 Requirements [NCSC 85, NCSC 87, NCSC 88], and also provide a functional specification of an audit-logging API used by servers to log the identified auditable events.

In this RFC, we present a proposed implementation design for the DCE audit subsystem, which includes the audit-logging API specified in [RFC 28.0] and an audit daemon. The audit daemon will be run on each server host. The daemon maintains event-selection information for the servers on the same host and provides an RPC interface to the servers for event logging.

The rest of this document is organized into 7 sections. In Section 2, we describe the proposed DCE-audit functionalities. In Section 3, we explain why local-OS audit subsystems cannot be used to provide the proposed DCE-audit functionalities. In Section 4, we describe the encoding schemes of event types and classes. In Section 5, we specify the format of the files used by the proposed DCE-audit subsystem. In Section 6, we explain how these files are maintained by the audit daemon. In Section 7, we describe how the proposed audit API interacts with the audit daemon. Section 8 contains a design for object-audit selection.

2. PROPOSED FUNCTIONALITIES

The proposed functionalities of the DCE audit subsystem include:

(a) *Event Logging:* An audit logging API used (1) to record auditable events at some specified or default destination, (2) to retrieve and print audit records that match certain specified criteria (predicates). The API has been specified in [RFC 28.0].

(b) *Event Selection (Filtering):* A command interface through which a system administrator can select the events of a subject (i.e., principal, group, or realm) to be logged. A filtering mechanism shall determine whether an event should be logged, given the event type and the identity of the client[1] which invokes the event, and the event outcome (success or failure).

(c) *Configurable Event Classes:* Event classes are defined to provide an efficient mechanism by which sets of event types can be logically grouped and selected in an event list by means of a single value. Event classes shall be configurable; i.e., a system administrator shall be allowed to add/remove event types of an existing event class or define new event classes. Non-configurable event classes (e.g., code-point hardcoded event classes suggested in [RFC 25.0][2]) would make event-selection mechanisms impractically inefficient. System administrators will have to select multiple event classes even if they are only interested in a small subset of events in each of the selected classes.

---

1. The functionality to allow a system administrator to selectively audit the actions of any one or more users based on individual identity is a TCSEC Class C2 requirement.

2. [RFC 25.0] suggests the use of facility tables of DCE 1.1 Serviceability design for event-class definitions. Each event class has an entry in a facility table, which specifies whether that event class is selected to be audited. At an event's code-point in the server program, checks are done to determine whether any of a set of *compile-time assigned* event classes to which the event belongs, is selected. Thus, event classification is fixed at compile time. To reconfigure event classes, a system administrator has to ask the vendor to have the server modified and recompiled!

(d) *Critical Event Notification (Alarm Raising):* The event selection interface should also allow a system administrator to direct selected classes of critical events to the system console.

3. SEPARATION OF DCE AUDIT FROM LOCAL AUDIT

A DCE-specific audit subsystem should be designed to record DCE auditable events. Local-OS auditing subsystems cannot be used for DCE audit for the following reasons:

(a) Local-OS auditing subsystems are only able to offer audit event selection based on local subject/object identities. Thus, the DCE audit event selection cannot be based on the event selection mechanisms of local audit subsystems. For example, a local audit subsystem would not allow events to be selected on a per DCE principal, group, realm, or object basis.

(b) The separation of DCE audit from local audit simplifies the porting efforts that are required on different platforms. If the DCE audit-subsystem relies on local-OS audit sub-systems, it has to use a generic interface that can be translated into audit-system calls on all platforms where DCE servers may be installed. This is a non-trivial task, if at all possible.

(c) It would be redundant to include all local audit header information in DCE-audit event records. The header information includes local audit specific information such as the process id, local user ids, group ids, effective (local) subject privileges, etc. Recording such information for each DCE-audit event is unnecessary because DCE-audit events occur as a result of the DCE TCB interface access, not a local TCB interface access. For example, an access to a p/g/o item in the registry database generally would not cause the registry server to access any local TCB interface, because the entire registry database is read into the registry server's address space when the server starts. In case a DCE TCB interface access results in any local TCB interface access, the local TCB access will be audited by the local audit subsystem. Correlation of accesses to local TCB interface and DCE TCB interface can be established through an analysis of the two audit trails together. Recording local-audit information when there is no access to the local TCB interface is not useful.

4. ENCODING OF EVENT TYPES AND EVENT CLASSES

The goal of the event-type and event-class encoding schemes is to allow extensibility and flexibility in defining and classifying DCE auditable events. Event types and classes are represented with 32-bit integers, instead of UUID's, which require additional 12 bytes. [Also, using UUID's would make it difficult to map an event number to the identity of the server for which the event is defined. A simple mapping from event numbers to server identities facilitates the reading of event class configuration files described in Section 5.]

The schemes (described in the following two sub-sections) used for event-number and class-number assignments are similar to the IP address-assignment scheme.

4.1. Event Numbers

An event number encodes the identification of an event set as well as the identification of the unique event. An event set is assigned to an organization or a vendor. The organization or vendor then becomes the authority on defining events within that event set.

Conceptually, each event number is a pair (`set-id, event-id`), where `set-id` identifies an event set, and the `event-id` identifies an event within the event set. In practice, each event number must have one of the following formats.

```
             0 1 2 3 4    8      16      24      31
 ------------------------------------------------
| Format A |0|  set-id  |        event-id        |
 ------------------------------------------------
| Format B |1|0|    set-id       |   event-id    |
 ------------------------------------------------
| Format C |1|1|0|       set-id         |event-id|
 ------------------------------------------------
| Format D |1|1|1|0|          event-id           |
 ------------------------------------------------
| Format E |1|1|1|1|          reserved           |
 ------------------------------------------------
```

Given an event number, its format can be determined from its four high-order bits. Format-A event numbers, which are allocated to organizations such as the OSF itself and major vendors which need more than 16 bits for event-number assignment, devote 7 bits to `set-id` and 24 bits to `event-id`. Format-B event numbers are allocated to intermediate-size vendors which need 8 to 16 bits for event-number assignment. Format-C event numbers are allocated to small-size vendors which need less than 8 bits for event number assignments. Format-D event numbers are not administered by the OSF and can be used freely for DCE cell-specific events. The use of these event numbers might not be unique across cells and should be avoided by servers which may be installed in more than one cells. Format-E event numbers are reserved for future use.

Format-A event numbers with 0 (zero) as `set-id` are assigned to OSF. That is, all events numbers used by the OSF have a zero in the most significant byte.

```
             0 1 2 3 4    8      16      24      31
 ------------------------------------------------
| Format A |0|0 ...... 0|     OSF event-id       |
 ------------------------------------------------
```

The following is a sample of the proposed event number assignment for DCE base events. The event numbers are defined in server's include files.

```
/* Event numbers 0x00000100 to 0x000001FF are assigned to the
   Security Server. */

#define AS_Request        0x00000100
#define TGS_TicketReq     0x00000101
#define TGS_RenewReq      0x00000102
#define TGS_ValidateReq   0x00000103

/* Event numbers 0x00000200 to 0x000002FF are
   assigned to the Time Server. */

#define CNTRL_Create      0x00000200
#define CNTRL_Delete      0x00000201
#define CNTRL_Enable      0x00000202
#define CNTRL_Disable     0x00000203
```

The following is a hypothetical example of the event-number assignment by a vendor, where `vn` stands for the name of the vendor.

```
#define ET_vn_BANK_SERVER_ACCT_OPEN     0x01000000
#define ET_vn_BANK_SERVER_ACCT_CLOSE    0x01000001
#define ET_vn_BANK_SERVER_ACCT_WITHDRAW 0x01000002
#define ET_vn_BANK_SERVER_ACCT_DEPOSIT  0x01000003
```

Organizations/Vendors shall administer the event numbers assigned to them and maintain the unique assignment of event numbers to events in their products.

4.2. Event Classes and Event Class Numbers

The goal of event classes is to provide an efficient mechanism by which sets of event types can be logically grouped and specified in an event list by means of a single value. Usually, event classes are defined for events that have some commonality; e.g., Authentication event class and Account-Operation event class.

Similar to event numbers, event-class numbers encode the identification of an event-class set as well as the identification of a unique event class within that set. An event-class set is assigned to an organization or a vendor. The organization or vendor then becomes the authority on defining the event classes within that set.

Conceptually, each event class number is a pair (`set-id`, `class-id`), where `set-id` identifies an event-class set, and the `class-id` identifies an event class within in the set. In practice, each event-class number must have one of the following forms.

```
           0 1 2 3 4    8      16      24      31
 -----------------------------------------------
| Format A |0|     set-id      |  class-id      |
 -----------------------------------------------
| Format B |1|0|        set-id        |class-id|
 -----------------------------------------------
| Format C |1|1|0|        class-id              |
 -----------------------------------------------
| Format D |1|1|1|        reserved              |
 -----------------------------------------------
```

Given an event-class number, its format can be determined from its three high-order bits. Format-A class numbers, which are allocated to organizations/vendors which need more than 8 bits for event-class number assignment, devote 14 bits to `set-id` and 16 bits to `class-id`. Format-B class numbers are allocated to vendors which need not more than 8 bits for event-class number assignment. Format-C class numbers are not administered by the OSF and can be used freely for DCE cell-specific event classes. Format-D numbers are reserved for future use.

Some event classes will be defined by vendors for product-specific events, thus: EC_*VendorA_ProductX_EventClass1*. Event classes which encompass multiple vendors will be defined by the OSF (e.g., `EC_OSF_C2_Configuration`), or by cell administrators who can tailor the definition to satisfy the specific requirements of their cells.

Format-A class numbers with 0 (zero) as the `set-id` are assigned to OSF itself. That is, all event-class numbers used by the OSF have a zero in the two most significant bytes.

```
           0 1 2 3 4    8      16      24      31
 -----------------------------------------------
| Format A |0|0 .............. 0|  class-id      |
 -----------------------------------------------
```

The following is the proposed event-class number assignment for DCE C2 event classes [RFC 28.0], which are defined for TCSEC C2 requirements.

```
EC_OSF_C2_Authentication     0x00000000
EC_OSF_C2_Object_Deletion    0x00000001
EC_OSF_C2_Security_State     0x00000002
EC_OSF_C2_Controlled_Access  0x00000003
EC_OSF_C2_Network_Exception  0x00000004
EC_OSF_C2_Cryptographic      0x00000005
EC_OSF_C2_Configuration      0x00000006
```

5. FILE FORMAT

5.1. Event Class Configuration Files

Each event class is defined in a separate event class configuration file, which always contains an event-class number and a list of events. All event class configuration files reside in directory

/opt/dcelocal/etc/security/audit/ec/. Servers need to read these files once in order to initialize an event table at runtime (described in Section 7). Optionally, an event class configuration file also contains a list of event number prefixes of the events in this class. This information is useful if the event list is very long. Servers which do not have events with one of the prefixes listed need not scan the event list. If the prefix list is not provided in the file, servers will have to read the entire file to find out if the class contains any of their events.

The following example is the configuration file of EC_OSF_C2_Authentication event class. The contents of file /opt/dcelocal/etc/security/audit/ec/EC_OSF_C2_Authentication are listed below.

```
# Event-class number of EC_OSF_C2_Authentication:

ECN = 0x00000000

# Event numbers in this event class have one of the
# following prefixes.  "Prefixes" information is optional.

Prefixes = 0x100

# The following is a list of events which
# belong to this event class.

# Security Daemon Events:

# AS_Request
0x00000100
# TGS_TicketReq
0x00000101
# TGS_RenewReq
0x00000102
# TGS_ValidateReq
0x00000103
```

5.2. Event Selection List Files

Two files are maintained by an audit daemon for event selection: an event-selection-list (ESL) file, /opt/dcelocal/var/security/audit/esl, and an update log (ESL-update) file, /opt/dcelocal/var/security/audit/esl-update.

The event-selection list represents the selected event classes for principals, local groups, and realms. Each entry in this list contains five fields: subject type, subject name, event class, audit level, and action specifier. Subject types include principal, local group, or realm. Event class field specifies which event class is being selected to be audited for the subject. Audit levels include ALL (log both successful and failed events), FAILED (log only failed events), and FAILED_ACCESS (log only events that failed due to access denial). The action specifier can be one of LOG (log only) and ALARM (log and raise alarm). The audit daemon maintains the event selection list both in its address space and in the ESL file. The audit daemon does not always overwrite the ESL file when there is an update. Instead, it logs the updates in the ESL-update file. Periodically, the audit daemon incorporates the updates in the ESL file, and clears the update file.

The following is an example of the contents of an ESL file and an ESL-update file that are maintained by an audit daemon on a server machine. The string names of principal, group, realm, event class, audit level, and action specifier are used here for clarity. In actual ESL and ESL-update files, the UUID's of principals, groups, and realms, and event class numbers are used (see Section 4.2). Audit levels and action specifiers are represented with integers.

```
       Event Selection List File (Example)
 ------------------------------------------------------------
|  princ:luan@gburg.ibm.com:Authentication:ALL:LOG             |
|  princ:weisz@toronto.ibm.com:Controlled Access:FAILED:LOG    |
|  local grp:acct-admin:Account Operation:ALL:LOG              |
|  realm:austin.ibm.com:Autentication:ALL:LOG                  |
|  realm:austin.ibm.com:Controlled Access:FAILED:ALARM         |
 ------------------------------------------------------------

   Event Selection List Update File (Example Cont.)
 ------------------------------------------------------
|  ADD:realm:gburg.ibm.com:Authentication:FAILED:LOG     |
|  DELETE:realm:austin.ibm.com:Autentication:ALL:LOG     |
 ------------------------------------------------------
```

The following is the effective ESL initialized by the audit API in a server after reading the ESL and ESL-update files.

```
   Server Audit Runtime ESL (Example Cont.)
 ---------------------------------------------------------
|  PRINCIPAL:                                               |
|     luan@gburg.ibm.com:Authentication:ALL:LOG             |
|     weisz@toronto.ibm.com:Controlled Access:FAILED:LOG    |
|  LOCAL GROUP:                                             |
|        acct-admin:Account Operation:ALL:LOG               |
|  REALM:                                                   |
|        austin.ibm.com:Controlled Access:FAILED:ALARM      |
|        gburg.ibm.com:Authentication:FAILED:LOG            |
 ---------------------------------------------------------
```

6. AUDIT DAEMON

An audit daemon runs on each machine where some server(s) (including DCE components or applications) may generate audit records. The audit daemon provides two RPC interfaces: an event-selection interface provided to control programs used by the system administrator, and an audit-logging interface provided to the server processes.

6.1. Event-Selection RPC Interface

The Event-Selection RPC interface of an audit daemon allows a (local or remote) system administrator (through a command interface) to control the event filtering behavior of the servers on the machine where the audit daemon runs. This RPC interface maintains ESL and ESL-update files (described in Section 5.2).

```
interface audit_select {

typedef unsigned32  audit_subject_type_t;
const unsigned32  audit_subject_principal     = 0;
const unsigned32  audit_subject_group         = 1;
const unsigned32  audit_subject_organization  = 2;
const unsigned32  audit_subject_realm         = 3;

typedef unsigned32  audit_level_type_t;
const unsigned32  audit_level_all             = 0;
const unsigned32  audit_level_failed_only     = 1;
const unsigned32  audit_level_failed_access   = 2;

typedef unsigned32  audit_action_type_t;
const unsigned32  audit_action_log_only       = 0;
const unsigned32  audit_action_raise_alarm    = 1;
```

```
void audit_select_add_eventclass (
[in]            handle_t              h,
[in]            audit_subject_type_t  type_name,
[in] [string] char                  *subject_name,
[in] [string] char                  *event_class,
[in]            audit_level_type_t    audit_level,
[in]            audit_action_type_t   audit_action,
[out]           error_status_t        *st
);

void audit_select_remove_eventclass (
[in]            handle_t              h,
[in]            audit_subject_type_t  type_name,
[in] [string] char                  *subject_name,
[in] [string] char                  *event_class,
[in]            audit_level_type_t    audit_level,
[in]            audit_action_type_t   audit_action,
[out]           error_status_t        *st
);
}
```

6.2. Audit Logging RPC Interface

The audit daemon provides the following RPC interface for audit logging to the server's audit runtime (i.e., the audit-logging API implementation) in server processes.

```
interface audit_log {

typedef struct {
unsigned32        event;       /* event number */
unsigned16        outcome;     /* event outcome (failed?) */
unsigned16        auth_st;     /* authentication status */
uuid_t            client;
uuid_t            cell;        /* client cell */
unsigned16        num_groups;  /* no. of client groups */
[size_is(num_groups)]
uuid_t            groups[];    /* client groups */
utc_t             time;        /* record commit time */
[string] char     *addr;       /* client address */
} audit_log_header_t;

void audit_log_append (
[in]          handle_t              h;
[in]          audit_log_header_t    header_info;
[in]          unsigned32            tail_len;
[in] [length_is(tail_len)]
              idl_byte              tail_info;
[out]         error_status_t        *st;
);
}
```

The audit-logging API passes audit records (including parts of the header information) to the audit daemon through calling the `audit_log_append()` operation. Note that some parts of the audit header information (specified in [RFC 28.0]) are not provided as call parameters. The `drep`, `version`, `format`, `hlen`, `tlen`, and `server` fields are provided by the audit daemon itself before an audit record is logged in a trail file.[3] The audit_log_append() call has to be authenticated so that the identity of the caller (i.e., server) can be securely determined.

7. AUDIT LOGGING API

A server utilizes the audit facility through calling the audit logging API,[4] which performs event filtering and logging. Function dce_aud_open() reads ESL and ESL-update files to initialize the ESL in the server's address space. The function also creates a thread to maintain the freshness of the server's ESL. The server's audit runtime (the created thread) maintains the timestamps for the last modification times of the ESL and ESL-update files. Periodically, the runtime checks whether the files have been modified. If neither of the two files has been modified, the server's copy of the ESL is up-to-date. Otherwise, the function takes the following actions.

(a) If the ESL file has been modified (the ESL-update file must have also been modified in this case), the runtime discards the server's copy of the ESL, reads the new ESL file, and applies the updates from the new ESL-update file.

(b) If only the ESL-update file has been modified, the runtime applies the updates from the file directly on its copy of the ESL.

The dce_aud_open() function also reads the event class configuration files (described in Section 5.1) to initialize an event table in the server address space. The event table contains an entry for each of the server's events specifying the event classes each event belongs to. The event class configuration files are normally static. Thus, we do not provide a mechanism to update the event table. The servers will have to be restarted if event classes are reconfigured.

The dce_aud_start() function determines if an audit record should be logged for a specified event. From the input parameters, the function obtains the UUID's of the client principal and the client cell, and the group privileges used by the principal for the event. Based on this information and the server's ESL, the function derives the selected event classes. [This list of event classes is then cached by the server in a hash table keyed by the principal's UUID for future use.] The function then checks if the event being recorded by the server belongs to any of the selected event classes, and if the event should be logged, given the event's outcome (if known) and the audit level specified in the ESL. If so, an audit record buffer is initialized and returned.

The dce_aud_put_ev_info() function fills the event-specific information in the audit record buffer. The dce_aud_commit() function writes the audit record to the audit daemon. Figure 7-1 illustrates the runtime scenario of the audit-logging API. The data structures maintained by the server's audit runtime are displayed on the right side of the figure.

---

3. The current design only addresses *local audit*; i.e., audit records are always written to a local file. To perform *remote audit*, the audit daemon will ship the audit records to a central audit machine.

4. The audit logging API functions (dce_aud_*()) are specified in [RFC 28.0].

```
                                     |Server's Audit
                                ---->|Runtime Data Structures

                                           Audit-Trail
                                           Descriptor
                                          +----------------+
      +-----------------------------------| Audit daemon's |
      |                                   |  RPC binding   |
      |                                   +----------------+
      |
      |                                     Event Table
      |                                   +----------------+
      |                                   | event  classes |
dce_aud_open() -----> +--------------+    |   1:   A, B    |
      |        read   | Event Class  |--> |   2:   C       |
      |               | Config Files |    | ...            |
      |               +--------------+    +----------------+
      |               +--------------+    +----------------+
      +-------------> |   ESL Files  |--> |      ESL       |
           read       +--------------+    +----------------+
                                                  |
                                                  |
                                                  |
    filter events by checking Event Table and ESL |
      +-------------------------------------------+
      |
      |
      |                                   Audit Record Buffer
      V         initialize record buffer  +-----------------+
dce_aud_start() ------------------------> |   event name    |
                 name, header(, outcome)  |   header Info.  |
                                          |   event Info.   |
dce_aud_put_ev_info() ------------------> |   event outcome |
                       event-specific     +-------+---------+
                        information               |
                                                  |
                                                  |
       read audit record                          |
      +-------------------------------------------+
      |
      |
      V          audit_log_append() RPC to audit daemon
dce_aud_commit() ---------------------------------------->
                  binding info. is in the trail descriptor
```

Figure 7.1 Runtime scenario of the audit-logging API

8. OBJECT AUDIT SELECTION

The ability to select audit events by the object-access types of the events is useful for some applications. The following object-audit mechanism is presented to address this requirement. [This is not a C2 requirement.] We do not propose immediate implementation of this mechanism, since it requires a new field in the DACL data structure that would break the compatibility with DCE 1.0.x releases. However, we still present the design in this RFC so that the new field can be considered in the future versions of the DACL data structure.

In order to provide the object-audit function for a server, an access audit selection bit-vector is proposed in the DACL data structure. The selection bit-vector is 32-bit long, and each bit in the vector is used to indicate whether the granting of a specific permission should be audited. The new DACL structure is illustrated below.

```
typedef struct {
    sec_acl_id_t            default_realm;
    uuid_t                  sec_acl_manager_type;
    unsigned32              object_audit_permset;;
    unsigned32              num_entries;
    [ptr, size_is(num_entries)]
    sec_acl_entry_t         *sec_acl_entries;
} sec_acl_t;
```

Example: The following is the standard declaration of a permission set.

```
const unsigned32 sec_acl_perm_read            = 0x00000001;
const unsigned32 sec_acl_perm_write           = 0x00000002;
const unsigned32 sec_acl_perm_execute         = 0x00000004;
const unsigned32 sec_acl_perm_owner           = 0x00000008;
const unsigned32 sec_acl_perm_insert          = 0x00000010;
const unsigned32 sec_acl_perm_delete          = 0x00000020;
const unsigned32 sec_acl_perm_test            = 0x00000040;
const unsigned32 sec_acl_perm_unused_00000080 = 0x00000080;
```

If the `object_audit_permset` field is set to 0x0000003A, then all the accesses that require any of the write, owner, insert, and delete permissions shall be audited.

8.1. Object-Audit Selection API (Extension to the sec_acl_*() API)

Function `sec_acl_set_object_audit()`: Sets up the audited permissions in an object's ACL.

8.1.1. Synopsis

```
void sec_acl_set_object_audit(h, manager_type,
  object_audit_permset, status)
sec_acl_handle_t h;
uuid_t *manager_type;
unsigned32 object_audit_permset;
unsigned32 *status;
```

8.1.2. Parameters

Input:

(a) `h` — A handle referring to the object whose ACL is to be accessed. Use `sec_acl_bind()` to create this handle.

(b) `manager_type` — A pointer to the UUID identifying the type of the ACL manager in question. There may be more than one type of ACL manager protecting the object whose ACL is bound to the input handle. This argument is used to distinguish them. The function `sec_acl_get_manager_types()` is used to acquire a list of the manager types protecting a given object.

(c) `object_audit_permset` — The permissions that are to be audited.

Output:

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

   (i) `sec_acl_unknown_manager_type` — Manager type selected is not an available option.

   (ii) `sec_acl_invalid_permset` — The `object_audit_permset` input parameter contains some undefined permission bit(s).

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

8.2. Object-Audit Filtering API (Extension to sec_acl_mgr_*() API)

Function `sec_acl_mgr_get_obj_audit()`: Determines if an object access that requires certain permissions to the object should be audited.

8.2.1. Synopsis

```
boolean sec_acl_mgr_obj_audit(sec_acl_mgr, sec_acl_key,
                              required_perms, status)
sec_acl_mgr_handle_t sec_acl_mgr;
sec_acl_key_t        sec_acl_key;
unsigned32           required_perms;
unsigned32           *status;
```

8.2.2. Parameters

Input:

(a) `sec_acl_mgr` — A pointer to a `handle` for an ACL manager and the storage configuration chosen with `sec_acl_configure()`. Having a distinct handle for the combination of ACL manager and configuration allows the application to use different storage strategies for different types of objects.

(b) `sec_acl_key` — A pointer to the object's ACL. The type of the pointer is dependent on the ACL manager involved.

(c) `required_perms` — The ACL permissions that are required for the access to succeed.

Output:

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

  (i) `sec_acl_invalid_permset` — The `required_perm` input parameter contains some undefined permission bit.

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

8.2.3. Description

The function `sec_acl_mgr_obj_audit()` returns `true` if the use of any of the specified required-permissions needs to be audited. Otherwise, it returns `false`. The servers which call this function would subsequently call `dce_aud_*()` functions with an `object-audit` option being set if the `sec_acl_mgr_obj_audit()` returns `true`.

9. ACKNOWLEDGEMENTS

We would like to thank Virgil D. Gligor (University of Maryland, College Park), Ping Lin (IBM), Joe Pato (HP), Denis Pinkas (Bull), Pau-Chen Cheng (IBM), Chii-Ren Tsai (VDG), and Rajashekar Kailar (University of Maryland, College Park), for their comments on the earlier drafts of this RFC and suggestions on the design.

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 24.0] R. Salz, ‘‘DCE 1.1 Serviceability Proposal’’, November 1992.

[RFC 25.0] E. McDermott, ‘‘DCE 1.1 Auditing Strategy and Design Proposal’’, December 1992.

[RFC 28.0] S. Luan, R. Weisz, ‘‘A DCE Server Auditable-Event Identification and a Proposed Audit Logging API’’, December 1992.

AUTHOR’S ADDRESS

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

Robert Weisz
IBM Canada Laboratory
1150 Eglinton Ave. East
North York, Ontario M3C1H7
CANADA
Internet email: weisz@torolab5.vnet.ibm.com
Telephone: +1-416-448-3131