OSF DCE SIG **E. McDermott (DEC)**
Request For Comments: 25.0 **December 1992**

DCE 1.1 AUDITING STRATEGY AND DESIGN PROPOSAL

1. INTRODUCTION

Originally, the three heads of Kerberos signified authentication, authorization and auditing. DCE's security core is based on the Kerberos design, however it has only two of the three heads, authentication and authorization. This paper proposes a design and a strategy for adding the third head, auditing. First we shall discuss what shall be audited and why, and then we shall present a plan for adding this technology to DCE.

1.1. Statement of the Problem

It is all too often the case that a principal that has been authorized to perform an action, performs the action for an unauthorized purpose. One of my favorite examples of this is the old bank scam of tellers sending pennies (or fractions thereof) to their accounts during every transaction they perform. Tellers are authorized to move money from one account to another, however they are not usually authorized to take a percentage of each transaction for their own personal gain. Auditing records these inconsistencies, allowing a perpetrator to be uncovered.

For DCE, we shall use the term auditing to mean the noting of successes and/or failures of certain events. These events have been deemed relevant from a security perspective by virtue of applying the criteria set forth later in this document. Code points shall be identified by the provider's of DCE components utilizing the selection criteria described in this document. Calls to an audit log shall be made from the identified code points; the format of these calls and the format of the data passed in these calls is an issue that overlaps with the serviceability project.

Serviceability encompasses error and event logging, tracing and debugging. While the method for collecting auditing data versus serviceability data may differ (e.g., auditing data has stricter requirements to come from an authenticated source and be integrity protected), the interface to the filtering mechanism shall be the same for ease of management.

This paper addresses the issues such as auditing criteria, data representation and format, user interfaces and APIs. For clarity, we start with the Requirements section for DCE 1.1 auditing, then we continue with sections discussing a functional definition of the proposed auditing system, formats and interfaces. Finally, we shall close this paper with sections on applicable standards, management issues and open issues.

Please keep in mind while reading this document that auditing for DCE 1.1 shall be done on a local basis only, i.e., no central collection mechanism. Further, as with the rest of DCE, it is assumed that the underlying operating system is not malicious.

2. REQUIREMENTS

Many of the requirements listed here overlap with those for serviceability. For sake of completeness, I have listed those requirements here along with requirements that are unique to auditing. The following requirements are grouped according to: collection, storage and analysis.

2.1. Collection

2.1.1. Interface

This section specifies requirements on the selection of auditable events.

(a) Auditing shall be configurable by event class across all DCE components. This requirement provides for a consistent view across an environment. The mechanism used to select the event classes to be audited should be the same mechanism used to select serviceability parameters.

(b) The user interface must be rich enough to allow the selection of events to be audited, as well as the selection of information desired in relation to the selected events. (The selection of specific auditable data within an event shall not be provided in DCE 1.1.)

(c) Turning DCE auditing on or off must result in the event being noted in the audit log with the appropriate information.

2.1.2. Format

This section specifies requirements on the representation of events collected and the order they are placed in the call to the audit log.

(a) The audit data format must be extensible; meaning that new audit event types and event data can be easily added to DCE auditing.

(b) In order to accurately record the identity of the principal initiating an auditable event and to enable right to privacy laws to be met, audit IDs shall be used. An audit ID shall be assigned to a principal upon registering with the DCE and stay with that principal until the termination of the principal. This audit ID shall be included with all audit events recorded. Further, in the case of delegation the audit IDs identifying the initiator and the intermediaries of the event being audited, must be the found in the audit record. OSF shall use the UUID of a principal as the principal's audit ID; however, other methods of assigning audit IDs may be used.

(c) A header of audit data that is consistent with the minimal acceptable level of information needed to isolate the principal, the event, the location and the time must accompany the audited event.

(d) Text that may accompany audit data shall be kept in a message catalogue to facilitate internationalization, as well as vendors' tailoring of messages to reflect specific hardware environments.

2.1.3. Code-points

(a) Providers shall be responsible for identifying and adding audit calls according to the criteria specified in this document. Providers are only responsible for adding auditing to components for which they are responsible. A list of proposed code points and the rationale for why they were chosen must be submitted by the providers to OSF for approval.

2.1.4. Documentation

(a) A document explaining how to create new audit event types and classes shall be provided by OSF; it shall explain the process by which a vendor creates and registers new event types and event classes. This information shall be placed in the application development reference manual.

(b) A document noting how code points were selected for auditing shall be provided by the providers; it shall note which points of the criteria found in this document were used to identify each audit point. (Each audit point shall correspond to the list provided to OSF by requirement noted in 2.1.3. This information shall be placed in the administration reference manual.)

2.2. Storage

2.2.1. Interface

(a) The interface used to access the local audit log must be designed with the expectation that auditing will become distributed in the near future. OSF plans to use DME's Event Service (EVS) to support auditing in a future release of DCE. However the EVS interfaces are not well defined at this time, therefore the interface between the audit daemon and log is defined in this document. (Remember auditing in DCE 1.1 is local only, however flexibility to migrate easily to distributed auditing should be maintained.)

(b) Audit data must be logged where the auditable event occurs whenever possible, i.e., at the server. For example, suppose that a client is in the process of authenticating itself to a server and the server crashes. The event that a crash occurred while client X (and any other pertinent information) was logging in must be captured at the server if at all possible.

(c) The audit log must reside on a stable medium, i.e., not in local memory. It would be desireable to have shadow or duplicate audit logs in the future; these logs would be backup logs, residing in a different physical location, that would be updated periodically (for example, every X units of time).

(d) An auditable event deemed to be critical (policy defined), shall generate an alarm to the appropriate administrator, as well as logged in the audit log. DCE shall send this audit information to a critical events function; it is up to the underlying operating system to read the file and raise the alarm.

(e) Confidentiality of audit data must be maintained. ACLs shall be used on the management, user and application interface. This use of ACLs shall protect the confidentiality of the audit data without resorting to more complex cryptographic solutions. The audit log must rely on the protection provided by the underlying filesystem; users/applications accessing the audit log should be privileged.

(f) Authenticity of audit data must be verified. Any entity reading or writing the audit log should be authenticated and have its authorization checked. Again, we shall use ACLs to protect the DCE interface to the log (only the audit daemon can write to the log), however we must rely on the underlying file system's mechanisms to protect the log from other means of corruption.

2.2.2. Format

This section specifies requirements on the physical representation of the bits and the order in which these bits are to be stored.

(a) Audit data shall conform to a format that adheres to a widely accepted standard; adherence to the POSIX auditing (see [POSIX]) standard for the type of data captured and its order would be desirable if possible. (POSIX does not address the issue of physical representation of bits.) The goal of this requirement is to facilitate the creation of many reduction tools, as well as ease of the integration of DCE auditing with any underlying auditing system.

2.2.3. Management of storage facility

(a) A tool for management of local audit file, capable of being able to rollover or flush the audit log when it becomes full shall be provided by OSF. The management action taken must be noted in the log.

(b) The auditing system for DCE shall be ''fail-safe''; this means that if the auditing system should fail for any reason, the appropriate actions (policy defined) shall be taken. However, the auditing system encompasses may elements, such as the audit daemon, the audit log and the communications paths between the audit point and the audit daemon. Policies must define actions to be taken in relation to the different failures the auditing system may experience. For example, if there is a failure in the communication path between the code point and the audit daemon, the policy might dictate that the event be sent to the critical events audit function (see following paragraph). Whereas if the audit daemon crashed, the policy may dictate that all computation stop.

DCE shall send critical audit information to a separate critical events function. Critical events are those events that the administrator of a system deem critical via a specific policy; these events appear in one of the other event classes as well. Therefore, it is recommended that the event class containing the critical event be audited, resulting in the critical event being sent to both the critical events function and the audit log. Again, an example critical event audit function will be supplied by OSF.

2.2.4. Documentation

(a) Documentation describing the format in which the audit data is stored must be provided to enable reduction tools to be written. This document shall be written by OSF. This information shall be placed in the application development reference manual.

(b) Documentation explaining how one might manage the local audit log and the effects of certain decisions on the system. (Example: Rollover the audit log and potentially lose critical information or decide not to allow rollover and risk the machine crashing.) This document shall be written by OSF. This information shall be placed in the administration guide.

2.3. Analysis

2.3.1. Reduction tool

(a) One basic reduction tool with a well defined interface, shall be provided by OSF for use with DCE audit data. This allows other companies the opportunity to pursue enhanced reduction tools as a market.

2.3.2. Documentation

(a) Documentation explaining how to use the OSF's reduction tool and what the events displayed indicate shall be provided by OSF. This information shall be placed in the administration reference manual.

3. FUNCTIONAL DEFINITION

3.1. Criteria

The decision to audit events within DCE components shall be made after carefully considering each DCE components' impact on the security mechanisms. For example, if the time server could be manipulated, PACs could be created with or checked against incorrect expiration times, allowing spoofing attacks. This example is clearly an attack on what DCE claims to protect against, therefore auditing shall be added to the time server.

A process similar to the one described in the preceding paragraph has been carried out for DCE 1.1. It identified the security and time servers to be the components for which successful attacks would be the most devastating. While other components would yield useful audit data, we have a limited amount of time in which to add auditing to DCE for the 1.1 release; therefore, auditing shall be added only to the security and time daemons at this time.

The security and time servers provide services throughout DCE; for example, the security service is used every time there is an authenticated RPC call. Not all security and times services shall be audited; an analysis of the DCE library calls shall be done by OSF. The result of this analysis will be a list of calls to be audited (which are security or time services). An example of a call that shall require auditing is `sec_login_valid_and_cert_ident()`. Auditing this call could help detect dictionary attacks on the local system.

The ''Orange Book'' ([OBook]) and ''Red Book'' ([RBook]) shall be used as guides, however common sense should prevail; we must not get carried away in literal interpretation of these references. (POSIX offers no guidance on audit selection criteria.) A metric other than these references needs to be employed to identify auditable events for DCE, which are geared more toward the commercial working environment. This implies we choose or develop a metric by which we deem an event auditable in DCE. The criteria for the selection of auditable events in DCE shall be:

(a) Use of authorization and authentication mechanisms. This requirement comes from the Orange Book; its goal is to record any entities attempting to gain access to or utilize resources for which they are not entitled. It is presumed that resources that warrant protecting utilize authentication and authorization services. Therefore, recording all uses of the authentication and authorization services, shall yield a record of both legitimate and attempted illegitimate uses of protected resources.

(b) Addition, deletion, modification of objects. This requirement is an expansion of the Orange Book requirement to record deletion of objects. The goal of this requirement for DCE is to record changes or deletion of ''crucial objects''. The phrase ''crucial objects'' refers to objects that utilize security services (such as a principal), or security objects (such as a registry object).

(c) Security related actions taken by the computer operators and system administrators and/or system security officers. This requirement comes from the Orange Book; its goal is to keep a record of the actions privileged people perform. This record can then be scrutinized for inconsistencies between policy and actual performance.

(d) Identification of each access event (e.g., any time a ticket is acquired) and its principal and identification of the starting and ending times of each access event using local time or global synchronized time. Principals may be users or servers in DCE which can be identified by an UUID. (Derived from the Red Book.) (Note: This criterion should be applied with a good deal of consideration with regard to performance, i.e., use common sense.)

(e) Identification of security relevant exceptional conditions detected during the transactions between two processes (e.g., potential violation of data integrity, such as replayed messages). (Derived from the Red Book.)

(f) Utilization of cryptographic variables. This requirement comes from the Red Book; its goal is to record the generation/destruction/use of cryptographic keys. Improper key management can result in successful masquerading attacks.

(g) Changing the configuration of the cell (e.g., a component leaving or rejoining a cell). The goal of this requirement is to keep track of DCE servers (security and time servers for DCE 1.1) as they start up and shut down within a cell; thus enabling identification of the time spent and perhaps potential damage a rogue or faulty server may have caused while in a cell. (Note: This criterion introduces a potential overlap of events recorded by the serviceability and auditing systems.)

(h) Any event that tries to circumvent, violate or break (whether or not successfully) the security mechanisms in DCE. Finally, this last criterion's goal is to have common sense used when choosing auditable events. Auditing systems are of value only when they are used; an audit system that drains too great a percentage of the system's resources is typically turned off.

4. CHARACTERISTICS AND REQUIREMENTS OF AUDITING

4.1. Audit Interface (Interface to the Audit Daemon)

Unlike serviceability, auditing has the requirement that all entities writing to the audit log be authenticated and authorized. This requirement shall be filled by utilizing a privileged/''trusted'' daemon, the audit daemon, which shall reside on each node. After code points have been identified, audit calls to the audit daemon shall be put in place. An audit call shall be sent from the service in which the event occurs via authenticated RPC (the authenticated RPC in this case shall NOT be audited), and shall contain the information conveyed in the following data structure. (Note: The authenticated RPC call in this case must not cause any auditable events, thus avoiding a recursive loop. A communications mechanism other than RPC may be used providing that the mechanism provides authentication and authorization services and OSF approves it.)

```
interface audit_daemon
{
    typedef struct {
        uuid_t      aud_event;  /* event name code */
        unsigned16  aud_event_status;
                                /* success or failure of
                                   event */
        unsigned32  aud_tail_format;
                                /* Format of tail of this
                                   record.  The format must be
                                   defined in a catalogue.  The
                                   default for this shall be 0,
                                   for no additional
                                   data. */
        utc_t       aud_call_to_daemon_utc;
                                /* time at which the call to
                                   the audit daemon is made */
        byte        *aud_tail_data;
                                /* tail data; default = NULL */
    } appl_aud_t;

    unsigned32
    audit_daemon_log_event (
        [in] handle_t h,
        [in] appl_aud_t *audit_info );
}
```

The event class, event name code and tail format shall be determined when the codepoints are created; these parameters must be chosen from the OSF approved lists of: events, event classes and tail formats. The `aud_status` shall be a function of the code executed causing the event to be logged. For example, if an unauthorized principal attempted to access a file, this attempt would cause an audit call with a status of failure to be sent to the audit daemon. The authentication status may be determined by extracting that information from the return values of a `rpc_binding_inq_auth_info()` call. (This call requires the binding handle of the initiating principal.)

The return status returned from the audit daemon to the calling process must be one of the following:

(a) `AUD_CODE_RECEIVED` — Received and accepted.

(b) `AUD_CODE_UNRECOG_EVENT` — Received, but not a recognized event.

(c) `AUD_CODE_UNRECOG_EVENT_CLASS` — Received, but not a recognized event class.

(d) `AUD_CODE_DATA_TOO_LONG` — Received, but opaque data too long.

(e) `AUD_CODE_CRITICAL_EVENT` — Received, critical event: raise alarm (policy defined).

(f) `AUD_CODE_LOG_FULL` — Rejected, log full.

Vendors may add to this list by registering new status codes with OSF. Further, any failure status codes returned from the audit daemon must be sent to the critical event function. This function shall be provided by OSF, however licensees may replace this function with their own local security facility. OSF shall also provide a function that applications may use to log audit data and check the return status; this function will have the same parameters as the audit interface (`audit_daemon_log_event()`).

Authenticated DCE RPC shall be used between the code point and the audit daemon to take advantage of the authentication and authorization services provided. (Again, this use of authenticated RPC shall not be audited.) The audit daemon must create an audit log for the local system with exclusive write privilege for the daemon process. (This aspect of the audit design may change when DCE migrates to distributed auditing.)

Additionally, you shall notice that there is a timestamp for when the audit call was made and one for when the audit record gets written to the log. The reason for both timestamps is to determine when events truly occurred, since calls coming into a multi-threaded audit daemon may not be processed in true sequence.

4.2. Log Format

The data representation for the data stored in the audit log shall be human readable form, i.e., text, to be more consistent with serviceability and to avoid schedule dependencies on other work slated for DCE 1.1. As DCE gains experience with pickling, auditing may change its data representation to pickles.

4.3. User Interfaces

A single reduction tool, capable of displaying all of the data in the audit log shall be provided by OSF. The tool must be able to display single records within the audit log in a human readable and clear format. Vendors may wish to embellish the OSF supplied reduction tool or supply their own reduction tools as value added features.

4.4. Audit Log Interface

Auditing in DCE 1.1 shall be local; therefore there shall be no remote interfaces for log storage defined in this design document. Auditable events shall be logged on the system that the event occurred.

The interface to the local audit log may have to be modified in the future to accommodate the addition of distributed audit capability. However, the interface to the local log shall be kept as generic as possible to allow the migration to distributed auditing to be as easy as possible.

The following is the C language representation of the structure of an audit call from the audit daemon to the audit log.

```
typedef struct {
unsigned16   aud_format;      /* format of the audit log */
unsigned32   aud_hlen;        /* length of header */
unsigned32   aud_tlen;        /* length of tail */
unsigned16   aud_tformat;     /* format of tail */
uuid_t       aud_server;      /* UUID of the server
                                 (audit daemon) */
uuid_t       aud_server_cell;/* UUID of the server cell */
unsigned16   aud_event;       /* event_name_code*/
```

```
unsigned16   aud_status;      /* success or failure of event */
unsigned16   aud_authn_st;    /* indicates whether the client is
                                 unauthenticated, or method by
                                 which it was authenticated */
uuid_t       aud_client;      /* AUUID of client principal */
uuid_t       aud_client_cell;/* UUID of client cell */
utc_t        aud_log_time;    /* time event was written to
                                 buffer/log */
utc_t        aud_cdpt_time;   /* the time at which the call to
                                 the audit daemon is made */
byte         *aud_tail_data; /* tail data; default = NULL */
} aud_log_t;
```

Note: The format of the tail is defined in `aud_tformat`.

Message catalogues shall be utilized wherever possible, for example the tail format definitions and event names shall be implemented via message catalogues.

4.5. Management Interfaces

Filters allow authorized users of the audit service to specify what events they wish audited. DCE 1.1 shall provide a limited filter capability. Filtering shall be carried out by specifying which event classes are to be audited and whether or not both successes and failures, or just the failures for each event in the event class is to be audited.

Audit filters shall be set using the audit component of serviceability (see the [RFC 24.0]). The audit component within serviceability is privileged and shall be protected by an ACL. Event classes are equivalent to serviceability's facilities and the following levels are defined for use with facilities within the audit component.

(a) `AUD_OFF` — Do not audit event class.

(b) `AUD_FAILURES` — Audit failures of events within this event class.

(c) `AUD_SUCC_AND_FAIL` — Audit both successes and failures of events within this event class.

The audit mask is set via the serviceability system and is stored in a file which is read at start time. It can only be set by privileged users, i.e., system/security administrator, and can be set per workstation or per component. Once the auditing system is started, privileged users can change the audit mask settings at any time via the remote management interface described in the serviceability document. Any changes in the audit mask must be recorded in the audit log. (Note: This is not an event in an event class. It is not configurable; it must be logged.)

Filtering of audit data at the code point by checking if a specific event class is to be audited, reduces the amount of audit data collected and transported.

POSIX has not agreed upon event classes; therefore OSF has taken it upon itself to define the following classes which DCE shall support: authentication, authorization, addition/modification/deletion of objects, cryptographic, configuration, administrative and critical. The authentication event class must be turned on to make the audit data yielded from the other event classes meaningful.

Events and event classes must be registered at OSF, thereby keeping events and event classes consistent across all vendors. The initial event classes are:

(a) Authentication: Events that either use authentication or are directly related to authentication.

(b) Authentication Critical: These are authentication events that would raise alarms or follow other policy defined procedures specified by vendors.

(c) Authorization: Events that either use authorization or are directly related to authorization.

(d) Authorization Critical: These are authorization events that would raise alarms or follow other policy defined procedures specified by vendors.

(e) Addition/Modification/Deletion of Objects: While there may appear to be some overlap between authorization and addition/modification/deletion of objects, these are two distinct events. A principal may have authorization to access an object, but the event class's purpose is to record whether accesses were for the policy defined purpose.

(f) Add/Mod/Del Critical: These are additions/modifications/deletions of objects that would raise alarms or follow other policy defined procedures specified by vendors.

(g) Cryptographic: Events that are directly relate to the use and manipulation of cryptographic variables.

(h) Cryptographic Critical: These are cryptographic related events that would raise alarms or follow other policy defined procedures specified by vendors.

(i) Configuration: Additions or deletion of servers to a cell (only security and time servers for DCE 1.1).

(j) Configuration Critical: These are configuration related events that would raise alarms or follow other policy defined procedures specified by vendors.

(k) Administrative: Events that can be only initiated by a DCE administrator.

(l) Administrative Critical: These are administrative related events that would raise alarms or follow other policy defined procedures specified by vendors.

5. THE INITIAL AUDIT EVENTS

The initial audit events are listed in the following subsections. (This is just an initial cut at the auditing events. Modifications can be expected!)

5.1. Event Class: Authentication

(a) `dce_login`
(b) password change
(c) password export
(d) password import
(e) password override
(f) `sec_login_certify_identity()`
(g) `sec_login_export_context()`
(h) `sec_login_get_pwent()`
(i) `sec_login_import_context()`
(j) `sec_login_newgroups()`
(k) `sec_login_purge_context()`
(l) `sec_login_refresh_identity()`
(m) `sec_login_set_context()`
(n) `sec_login_setup_identity()`
(o) `sec_login_valid_and_cert_ident()`
(p) `sec_login_validate_first()`
(q) `sec_login_validate_identity()`
(r) `sec_login_validate_first()`
(s) `sec_login_validate_identity()`

5.2. Event Class: Authorization

(a) object access
(b) registry database access
(c) `rdacl_replace()`
(d) `sec_acl_get_error_info()`
(e) `sec_acl_mgr_configure()`
(f) `sec_acl_mgr_get_access()`
(g) `sec_acl_replace()`

5.3. Event Class: Addition/Modification/Deletion of Objects

(a) security relevant file deletion

(b) creation of a group
(c) addition to a group
(d) deletion from a group
(e) time (`dtscp`) change
(f) time (`dtscp`) set
(g) time (`dtscp`) synchronize
(h) time (`dtscp`) update

5.4. Event Class: Cryptographic

(a) `sec_key_mgmt_change_key()`
(b) `sec_key_mgmt_delete_key()`
(c) `sec_key_mgmt_delete_key_type()`
(d) `sec_key_mgmt_free_key()`
(e) `sec_key_mgmt_garbage_collect()`
(f) `sec_key_mgmt_gen_rand_key()`
(g) `sec_key_mgmt_get_key()`
(h) `sec_key_mgmt_get_next_key()`
(i) `sec_key_mgmt_get_next_kvno()`
(j) `sec_key_mgmt_initialize_cursor()`
(k) `sec_key_mgmt_manage_key()`
(l) `sec_key_mgmt_set_key()`

5.5. Event Class: Configuration

(a) security server addition
(b) security server deletion
(c) time server addition (`dtscp` create)
(d) time (`dtscp`) `dtsd` (restart daemon)
(e) time (`dtscp`) enable
(f) time server deletion (`dtscp` delete)
(g) time (`dtscp`) disable
(h) `sec_clientd`
(i) `sec_create_db`
(j) `auditd` (starting up/shutting down the audit daemon)
(k) `secd`

5.6. Event Class: Administrative

(a) `rgy_edit`
(b) `sec_admin`
(c) `sec_rgy_acct_add()`
(d) `sec_rgy_acct_admin_replace()`
(e) `sec_rgy_acct_delete()`
(f) `sec_rgy_acct_passwd()`
(g) `sec_rgy_acct_rename()`
(h) `sec_rgy_acct_replace_all()`
(i) `sec_rgy_acct_user_replace()`
(j) `sec_rgy_auth_plcy_set_info()`
(k) `sec_rgy_pgo_add()`
(l) `sec_rgy_pgo_add_member()`
(m) `sec_rgy_pgo_delete()`
(n) `sec_rgy_pgo_delete_member()`
(o) `sec_rgy_pgo_is_member()`
(p) `sec_rgy_pgo_rename()`
(q) `sec_rgy_pgo_replace()`
(r) `sec_rgy_plcy_set_info()`

(s) `sec_rgy_properties_set_info()`

5.7. Event Class: All Critical Classes

EVENTS TO BE DETERMINED ACCORDING TO LOCAL POLICY.

6. IMPLEMENTATION STRATEGY

The implementation of this design shall happen in stages, allowing a workable portion of the auditing system to be delivered in DCE 1.1 if the entire design cannot be completed in time.

The first stage shall consist of three parallel activities: OSF's implementation of the stage one portion of the design, OSF's documentation of this work (as noted in the requirements section), and the providers' identification, documentation and instrumentation of their components.

The second stage shall consist of three parallel activities: OSF's implementation of the stage two portion of the design, OSF's documentation of this work (as noted in the requirements section), and OSF's investigation and documentation of issues surrounding migration to distributed auditing.

6.1. Stage One

The management interface shall be provided via the serviceability project. It shall have an ACL on the interface to restrict access to authorized users. Authorized users can turn DCE auditing off or on for successes or failures of specified event classes via this interface.

An audit routine that simply extracts, formats and writes audit data to the audit log shall be written as an interim measure. Unlike the audit daemon described in Section 4, it does not authenticate and check authorization of entities writing to it. The audit routine shall be called from a selected code point in the following manner.

```
if (AUDITING(EC_AUTHORIZATION)) {
    status = dce_audit(h, audit_info);
    if (status != AUD_CODE_RECEIVED)
        dce_aud_critical_events(h, status, audit_info);
}
```

The audit routine shall then extract, format and write audit information to the audit log. The audit log shall be simply a file that shall reside in `<dcelocal>/var/security/admin/...` and is accessable by privileged principals. The permissions of the local system should be set to allow the security and time servers write permission to the audit log. Principals allowed to utilize the reduction tool must have read permission for the audit log. (Note: These principals must be privileged due to the sensitivity of audit data.)

The handle that is passed to the audit routine shall be used to extract: `aud_auth_status`, `aud_client_auuid`, `aud_cl_cell_uuid`. The audit routine shall compute/lookup the: `aud_format`, `aud_hlen`, `aud_tlen`, `aud_server`, `aud_cell`, `aud_cdpt_time`. The audit routine shall format the data passed explicitly in the audit call (`aud_event`, `aud_event_status`, `aud_tail_format` and `aud_call_to_daemon_utc`), the data passed implicitly via the handle and the data looked-up or computed. Finally, the audit routine writes the formatted data to the audit log in the following manner.

```
aud_log_t   *audit_data; /* see section 4.4 for typedef */

if (fwrite(audit_data) != SUCCESS)
     return(AUD_CODE_LOG_FULL);
else
     return(AUD_CODE_RECEIVED);
```

Events in the critical events class shall be sent to the critical events function by the audit routine. The critical events function shall be a routine that writes critical event data to the console. Vendors may want critical event data handled in a different way, and are free to use their own critical events function.

The reduction tool provided by OSF shall be an ``emacs-like'' utility. The tool shall be accessible by all users that have read privilege to the audit log.

6.2. Stage Two

During stage two the audit daemon shall be written. It shall authenticate audit data, by using authenticated calls and checking the authorization of the caller to write to the audit log. The audit daemon shall verify authorization by checking the ACL on the audit log. If a principal successfully completes an authentication exchange, but is not authorized to write to the audit log, the data sent by the principal shall not be formatted. It shall be sent to critical events function as received.

Once the audit daemon has verified that the caller is authorized to submit audit data, it shall use the handle that is passed to the audit routine to extract: `aud_auth_status`, `aud_client_auuid`, `aud_cl_cell_uuid`. The audit daemon shall compute/lookup the: `aud_format`, `aud_hlen`, `aud_tlen`, `aud_server`, `aud_cell`, `aud_cdpt_time`. The audit daemon shall format the data passed explicitly in the audit call (`aud_event`, `aud_event_status`, `aud_tail_format` and `aud_call_to_daemon_utc`), the data passed implicitly via the handle and the data looked-up or computed. Finally, the audit daemon writes the formatted data to the audit log. The audit log shall be simply a file that shall reside in `<dcelocal>/var/security/admin/....` (Note: The audit daemon replaces the audit routine from stage one.)

7. OTHER COMPONENT DEPENDENCIES

The serviceability project shall supply the auditing project with the management interface for turning auditing on/off, selecting the audit mode and event classes.

8. COMPATIBILITY

DCE 1.1 auditing shall write to a DCE audit log (a simple file) on the local system. Compatibility with the auditing subsystem residing on the local system is NOT guaranteed.

9. STANDARDS

POSIX 1003.6.12 draft 12e.

10. OPEN ISSUES

International issues surrounding importability of DCE auditing to non-US countries. OSF is incorporated in the U.S., and is therefore legally bound by U.S. export laws, not other countries' import laws; however we should be aware of problems our licensees may have trying to sell OSF technologies overseas. It would be useful to identify the areas of auditing that may cause importability problems and perhaps `ifdef` portions out if possible.

11. ACKNOWLEDGEMENTS

This work was done while the author was on sabbatical at OSF.

REFERENCES

[OBook] National Computer Security Center, Orange Book, ''Department of Defense Trusted Computer Security Evaluation Criteria'', December 1985.

[POSIX] POSIX 1003.6.12 draft 12e, September 1992.

[RBook] National Computer Security Center, Red Book, ''Department of Defense Trusted Network Interpretation'', July 1987.

[RFC 24] R. Salz, DCE-RFC 24.0, ''DCE 1.1 Serviceability Proposal'', November 1992.

AUTHOR'S ADDRESS

Ellen McDermott
Open Software Foundation
11 Cambridge Center
Cambridge, MA 02142
USA

Internet email: emcd@osf.org
Telephone: +1-617-621-7266