A Specification of
Trusted NFS (TNFS) Protocol Extensions

1. Status Of This Memo

This document is an Internet Draft. Internet Drafts are working documents of the Internet Engineering Task Force (IETF), its Areas, and its Working Groups. Note that other groups may also distribute working documents as Internet Drafts.

Internet Drafts are draft documents valid for a maximum of six months. Internet Drafts may be updated, replaced, or obsoleted by other documents at any time. It is not appropriate to use Internet Drafts as reference material or to cite them other than as a ''working draft'' or ''work in progress''.

Please check the I-D abstract listing contained in each Internet Draft directory to learn the current status of this or any other Internet Draft.

This draft document specifies extensions to RFC 1094 [1] which support network file access in a multilevel secure (MLS) network environment[1]. This draft was approved by the Trusted Systems Interoperability Group (TSIG), whose charter is to promote multi-vendor trusted system interoperability.

2. Abstract

Additional functionality has been developed for UNIX® systems to address the TCSEC [2] requirements for trusted systems. New requirements are driving efforts to develop interoperable, networked solutions for trusted UNIX environments. A specific approach for addressing TCSEC MLS requirements is identified in the CMW requirements document [3]. Developing support for network interoperability among MLS classified systems is a primary goal of the trusted UNIX community.

Sun Microsystem's Network File System (NFS†) V2 protocol is an industry (de facto) standard network file access mechanism, and represents one of the key components of system interoperability in the current UNIX networking market. This draft document describes extensions to the NFS V2 protocol which support network file access in a MLS network environment. It will be submitted to the RFC editor as a protocol specification. Distribution of this draft document is unlimited. Please send comments to the author at the address identified in section 7 below.

3. MLS Extensions for NFS

MLS functionality includes discretionary access control (DAC), subject and object security labeling, mandatory access control (MAC), authentication, auditing, and documentation. Exchanging information between MLS systems requires communicating additional security information along with the actual data.

The primary goal of this specification is to describe extensions to the NFS V2 protocol which support network file access between MLS systems with a minimal impact on the existing NFS V2 environment[2]. It is also intended that this MLS environment will permit unmodified NFS clients and servers to continue to be fully supported.

The general approach used in extending the NFS V2 protocol is to transport additional user context in the form of an extended NFS UNIX style credential between a Trusted NFS (TNFS) client and server, and to map that context into the appropriate server security policies which address file access. In addition, security file attributes are returned with each NFS (TNFS) procedure call. Otherwise, the NFS V2 protocol remains essentially unchanged.

---

[1] Multilevel Secure systems include, for example, support for B1 (and above) and CMW security policies.

® UNIX is a registered trademark of UNIX Systems Laboratories (U.S.L.)

† NFS is a trademark of Sun Microsystems, Incorporated

[2] Revisions to the NFS V2 protocol have been specified and presented for comment to the NFS community; this document addresses extensions to the V2 protocol only.

Two companion documents [4][5] complete the set of documentation describing the TNFS environment.

3.1. The Extended User Context

The Sun RPC protocol [6][7] includes two authentication parameters in a request message:

an authentication credential - used to identify or present a client subject's credentials to a server along with a given request for access or information, and

an authentication verifier - used to validate the subject's credentials,

and an authentication verifier in the RPC response message.

An NFS server uses the client subject's credentials to perform appropriate access checks prior to servicing the request. The verifier parameter in the RPC request message is used to authenticate the client subject's credentials[3].

Several styles of authentication are currently defined for NFS[4], and an NFS server may elect to support multiple authentication styles concurrently. A new RPC authentication style, AUTH_MLS, is defined for use in the TNFS environment. The definition of the AUTH_MLS credential combines the information in the AUTH_UNIX credential with extensions for the additional security attributes:

- audit id - immutable subject (user) identifier, not affected by modifications to either the real or effective user or group identifiers,
- sensitivity label - used with a MAC policy; a subject generally has a static, top-level clearance, but is permitted to execute processes at a sensitivity level different from (i.e. lower than) his/her actual clearance,
- information label - used with a CMW dual level labeling policy; dynamically adjusted based upon the information content associated with the subject (or object),
- integrity label - used with commercial, multi-party security policy (e.g., Clark-Wilson [8], Biba [9]),
- privilege mask - used to identify privileges (e.g., chown, chmod) or ''rights'' granted to a given subject, generally to override an existing security policy,
- vendor label - used to accommodate additional, vendor specific policies,
- subject clearance - used to support the CMW requirement that a subject cannot write above the clearance, and
- audit information - used to convey additional audit information, such as:
  - audit preselection information including the audit event type, the subject and object identity, and the subject and object sensitivity label, and
  - additional subject information such as a subject's terminal or process identifier.

The additional security attributes will actually be represented within the AUTH_MLS credential by fixed size *tokens*, which can support multiple translation schemes through the use of an appropriate translation mechanism [5]. For instance, mechanisms such as M.I.T. Project Athena's Hesiod/BIND or Sun Microsystem's NIS[5] lookup service could be used to support the translation of tokens and security attribute information.

There are several advantages to the use of a token translation model. One major advantage is that the actual security attribute information may be defined within the translation service, while the attribute representation may be defined by a small, fixed sized token within the relatively small amount of unallocated space in the credential structure. A second advantage of a translation model is that it may accommodate multiple security policies and translations. Finally, a token translation model permits security policies to be developed independently from the translation mechanism. Tokens are transferred within the

[3] Authentication of client and server identities will not be addressed in this specification.

[4] Styles currently defined are AUTH_NONE, AUTH_UNIX, AUTH_SHORT, and AUTH_DES.

[5] Network Information Service, known previously as the Yellow Pages Service

AUTH_MLS credential as opaque objects which are given context by the security policy mechanisms implemented by the TNFS clients and servers.

Note that although tokens are defined as opaque objects, tokens which represent the same security attribute and which reside within the same translation scheme may be compared for equality. This characteristic permits tokens representing a specific security attribute to be referenced in comparisons without requiring the tokens to be translated.

3.2. Discretionary Access Control

A Discretionary Access Control (DAC) policy provides for the restriction of subject access to objects based on the identity of subjects and/or the groups for which they are members. Most secure systems address DAC requirements through the use of access control lists. Associated with each file is a list which identifies the set of user and group combinations authorized to access the file, along with the access privileges associated with each combination.

The information contained in the AUTH_MLS credential of a TNFS client request includes user and group identification sufficient to permit the server to apply appropriate DAC policies in controlling access to its shared, local file objects. For example, the subject represented by the user and/or group identifiers contained in the client request may be checked against the access control list information associated with the referenced file on the server. Access control list information is not required to be transmitted from the client to the server in support of a server based access control policy. Client based support for access control of server based file objects is discussed below in the section which describes the extended attribute cache.

3.3. Mandatory Access Control

A Mandatory Access Control (MAC) policy provides for the restriction of subject access to objects based on the sensitivity of the information contained in the objects. MAC policies thus include assigning levels of trust or clearance to system users (subjects), and levels of sensitivity to system objects, and then ensuring that only users with sufficient clearance can access the classified information.

3.3.1. Sensitivity Labels

When MAC policies are enabled, each system subject and object is created with a sensitivity label, and the system MAC policies compare the labels when determining access.

The AUTH_MLS credential contains the sensitivity label and clearance information associated with the TNFS client subject (application) making the access request. This information is sufficient to permit the MAC policy checking mechanism on the server to determine whether to permit access to the requested object or information.

3.4. Information Labels

Information labels represent the actual sensitivity of a given subject or object, and permit the additional identification of control markings for a given piece of information. The information label is dynamically adjusted on both subjects and objects to the highest sensitivity level reflected by a subject/object pair: if a subject issues a write request to an object, the information label of the object will be adjusted (if necessary) to the level defined by the information label of the subject; if a subject issues a read request to an object, the information label of the subject will be adjusted to the level defined by the information label of the object. Note that information labels are adjusted upwards as a result of these actions.

The AUTH_MLS credential in the RPC request message contains the current information label associated with a TNFS client application (subject), and permits a remote file's object information label to be adjusted (if necessary) as a result of a client generated *write* operation. The TNFS reply message includes a field for the information label associated with an accessed file object, permitting the subject's information label to be adjusted (if necessary) as a result of a client generated *read* operation.

3.5. Privilege

The TCSEC/CMW concept of *least privilege* is an integral part of the MLS environment. Fine grained privileges are granted to subjects (and associated processes), and executable objects (files) according to a strict set of rules. All subjects are limited with respect to the system actions they may perform. An executable object is also limited to a specific set of actions, regardless of the subject which executes the object. Privilege sets associated with a file object are used to adjust a process's privileges during the execution of that object. Thus, at any given time, a subject will possess only those privileges necessary to support the completion of its current task.

Note, however, that the privileges associated with a subject on a client system might not be extended to that subject on a given remote server system. Although most subjects will likely retain their privileges on the server, a client administrator, for example, might not be granted administrative privileges on the server.

For TNFS, subject privileges are defined within the AUTH_MLS credential, and file privileges are defined within the security file attributes.

3.6. File Name Attributes

UNIX file names may vary in length from 1 to 255 characters, and represent an additional data storage mechanism which must be protected by appropriate MLS policies. Generally, the contents of a file may be classified, but the name of the file or knowledge of its existence is not. In some cases, however, the name of the file as well as its contents may require classification and protection. For example, consider the following file name:

codeword.SAND_AIRDEF.is.the.TOP-SECRET.DESERT_STORM.air.defense.project

The association of sensitivity and information labels with directory file name entries provides the support necessary to protect the use of classified file names.

3.7. Additional MLS Extensions for NFS

In an MLS environment, both DAC and MAC access control policies are applied in determining access to a given object. In a network environment of MLS systems participating in TNFS file access, the AUTH_MLS credential permits a TNFS server to apply both DAC and MAC policies in consideration of a request from a remote NFS client subject. Thus, MLS based network file access using the NFS V2 protocol can be supported through the use of the AUTH_MLS credential as described.

Listing or modifying the DAC and MAC security attributes of a server's file or file name from a client, however, requires additional protocol extensions. Identifying additional security access restrictions when a request is made to open a remote file is also considered to be a requirement. Extensions designed to satisfy these requirements are addressed by TNFS, and are described in the next subsections.

3.7.1. Remote Access to Extended File Attributes

The DAC and MAC security attribute information includes MAC and information labels, and access control list information (ACLs). Supporting remote access to this information is more difficult to address in the network environment, since:

- it requires transmitting additional file security attribute information (or its representation)
- additional file attribute information cannot be accommodated in the existing NFS V2 protocol file attribute data structures; additional support for setting and getting the extended security attributes is required.

Thus, extensions to the NFS V2 protocol procedures have been defined to support access to the extended attributes of served files and file names. The complete set of NFS protocol procedures and security extensions are referred to in this document as the TNFS protocol.

3.7.2. File Open Enhancement

Using the NFS V2 protocol, a client request to *open (2)* a remote file on the server may be translated by the client into a GETATTR procedure call for the current directory[6], followed by a LOOKUP procedure call for the file to be opened. If valid responses from these procedure calls are returned, the client's NFS file attribute cache is updated, and an open file descriptor may be returned to the requesting application.

Since the NFS V2 protocol does not transmit an actual open request to the server, however, an MLS server will not be able to apply the appropriate DAC and MAC policy at the time of the open request, and the application may find that it has successfully opened the file, but that it cannot access the file due to stronger access control policies being applied by the server in response to specific client access requests.

An *access* protocol procedure would permit the client to determine whether access to the file would be supported by the server, based on the application's open request type and the associated extended security attribute information. An *ACCESS* TNFS protocol procedure has been defined to address this issue. Thus, if file attributes are being cached on the client, and the security attributes of a client process issuing a request to open a remote file have been modified since the last time it issued an open request for that file, then an *ACCESS* procedure call shall be made to the server to revalidate the access rights of that client process.

3.7.3. File Name Enhancement

Supporting the retrieval of the security attributes associated with each file name requires an extension to the directory result structure returned by the NFS directory procedures: LOOKUP, CREATE, and MKDIR. This data structure extension is defined in section 3.4.5.1.

The ability to modify file name security attributes independently from file data security attributes is also required. A new TNFS procedure, *SETNAMELABEL*, has been defined to support this capability.

3.7.4. MultiLevel Directory Enhancement

Directories are files which contain file names and pointers to the data associated with the file names. The files contained in a directory include both regular files as well as other subdirectory files. Directories are used to group files, and to support the file system hierarchy.

In an MLS environment, files and directories are labeled with specific classifications; security policies limit the access of a given file to a user with a classification which dominates the file's classification. MLS implementations must continue to maintain the basic file system directory hierarchy, and also support the MLS access policies. They must support the creation, storage, and access of files and data of different security classifications, and also provide some accommodation for the use of commonly shared directories, such as */tmp* and */usr/tmp*.

One implementation approach is to use file name security attributes, as described previously. The TNFS file name attributes and *SETNAMELABEL* procedure support this approach. An alternative is to create a set of diversion directories below the actual MultiLevel Directory. Each diversion directory is associated with a specific classification level, and user access is directed into the appropriate diversion directory in a transparent, pass-through manner. The TNFS *MLD* procedure supports diversion directory implementations. Additional information is provided in [4].

3.7.5. TNFS Protocol Extensions

Extensions to the NFS V2 protocol are defined in this section of the specification. These extensions are designed to support remote access to the security file attribute extensions, and to support the file open, file name, and MultiLevel Directory enhancements.

---

[6] Depends on the presence of valid attributes in the lookup cache (DNLC).

3.7.5.1. Data Structure Definitions

The definitions which support the MLS extensions are described in this section. Since the definitions for the TNFS protocol are an extension of the original NFS V2 protocol, this specification will include all of the extended data structure definitions, and a few of the original definitions for clarity. Note that the arguments and results are defined using the RPC language.

The following RPC constants are used to identify the TNFS extensions which support MLS security policies. The TNFS program will be registered as a separate service with the RPC port mapping service.[7] Registration as a different service distinguishes the TNFS service from the original NFS V2 service. The use of a different program number distinguishes each request/response message.

```
PROGRAM 390086  /* TNFS Program Number */
VERSION      1  /* TNFS Version 1 */
```

The **stat** type is returned from every procedure call. A value of NFS_OK indicates the call completed successfully. Other values indicate that an error occurred during the servicing of the request. Note: this structure is unchanged from the NFS V2 Protocol Specification. It is (partially) reproduced here for clarity.

stat

```
enum stat {
        NFS_OK = 0,
        NFSERR_PERM = 1,
        NFSERR_NOENT = 2,
        . . .
        [other NFS errors as defined in the V2 protocol specification]
};
```

The credential parameter is included in each RPC request message, and is used to supply the client subject's credentials to the server. The AUTH_MLS credential will be used with the TNFS procedure calls and is defined as follows:

```
#define AUTH_MLS 200000     /* decimal */
#define MLS_TOKEN_SIZE 4    /* 4 octets or 32 bits */
typedef opaque t_token[MLS_TOKEN_SIZE]; /* tokens are opaque */
struct authmls_cred {
        u_long  auc_stamp;          /* arbitrary ID */
        char    auc_machname<255>; /* machine name */
        t_token auc_ids;            /* token for effective uid, gid, gids */
        t_token auc_aid;            /* audit id token */
        t_token auc_privs;          /* subject privileges token */
        t_token auc_sens;           /* sensitivity token */
        t_token auc_info;           /* information token */
        t_token auc_integ;          /* integrity token */
        t_token auc_vend;           /* vendor specific policy token */
        t_token auc_clear;   /* subject clearance token */
        t_token auc_audinfo;        /* audit information token */
};
```

[7] TNFS server implementations may elect to share a common UDP [13] port number with the original NFS V2 service, or to make use of a different port number.

Note that if a given security attribute is not being exchanged, then the corresponding credential token value shall be set to ''all bits on''. A given security policy may require that only a subset of the security attributes provided for in this specification be exchanged. For example, a C2 network security policy may require the support of privileges, and may also require support for Access Control Lists (ACLs). In that case, the sensitivity, information, integrity, vendor specific, and clearance token values shall be set to ''all bits on'' in the exchange messages.

The **fattr** structure defines the complete set of file attributes of a file. The extended **fattr** structure combines the NFS V2 **fattr** structure with additional fields for a file's security attributes. The security attributes are represented by tokens.

```
struct fattr {
        ftype   type;       /* file type */
        u_long  mode;       /* encoded access mode */
        u_long  nlink;      /* number of hard links */
        u_long  uid;        /* file's owner id */
        u_long  gid;        /* file's group id */
        u_long  size;       /* file size in bytes */
        u_long  blocksize; /* number bytes/block */
        u_long  rdev;       /* device number of the file */
        u_long  blocks;     /* current number of blocks */
        u_long  fsid;       /* file system id */
        u_long  fileid;     /* unique file identifier */
        timeval atime;      /* time of file's last access */
        timeval mtime;      /* time last modified (written) */
        timeval ctime;      /* time of last attribute change */
        t_token privs;      /* file privileges token */
        t_token sens;       /* sensitivity token */
        t_token info;       /* information token */
        t_token integ;      /* integrity token */
        t_token acl;        /* access control list token */
        t_token vend;       /* vendor specific policy token */
        t_token audinfo;    /* audit info token */
};
```

Note that if a given security attribute is not being exchanged, then the corresponding file attribute token value shall be set to ''all bits on''. Note also that the audit information token in the file attribute structure is included to permit the server to return optional information, such as the set of privileges used by the server in processing the client's request, and/or whether the request resulted in a change of the state of the accessed object, such as the adjustment of the information label.

The **sattr** structure defines the file attributes which can be set from the client. The extended **sattr** structure combines the NFS V2 **sattr** structure with additional fields for the security attributes, which are represented by tokens. A token value of is to be ignored.

```
struct sattr {
        u_long  mode;     /* encoded access mode */
        u_long  uid;      /* file's owner id */
        u_long  gid;      /* file's group id */
        u_long  size;     /* file size in bytes */
        timeval atime;    /* last access time */
        timeval mtime;    /* last data modify time */
        t_token privs;    /* file privileges token */
        t_token sens;     /* sensitivity token */
```

```
        t_token info;    /* information token */
        t_token integ;   /* integrity token */
        t_token acl;     /* access control list token */
        t_token vend;    /* vendor specific policy token */
};
```

The **sattrargs** structure is used by the SETATTR procedure. It contains the extended **sattr** structure definition.

```
struct sattrargs {
    fhandle file;
    sattr attributes;
};
```

The **attrstat** structure defines a common procedure result containing the status of the procedure call. It is returned with the results of GETATTR, SETATTR, and WRITE procedure calls. If the call was successful, **attrstat** contains the results for the specific procedure called, and the complete set of file attributes for the file on which the procedure was executed.

```
union attrstat switch (stat status) {
        case NFS_OK:
            fattr attributes;
        default:
            void;
};
```

The **diropargs** structure is used in directory operations. The fhandle **dir** is the directory containing file name **name**.

```
struct diropargs {
        fhandle dir;
        filename name;
};
```

The **diropres** structure defines the results of a directory procedure call. If the call was successful, **diropres** contains a new file handle **file**, the complete set of associated file **attributes**, and the file name attributes: **sens**, **info**, and **vend**.

```
union dirores switch (stat status) {
        case NFS_OK:
            struct {
                fhandle file;
                fattr attributes;
                t_token sens;
                t_token info;
                t_token vend;
            } diropok;
        default:
            void;
};
```

The **readlinkres** structure defines the results of a READLINK procedure call. If the call was successful, **readlinkres** contains the **data** in the symbolic link of the file identified by the file handle argument, and the complete set of associated file **attributes**. File attributes are returned with the READLINK procedure call to support the information label adjustment policy.

```
union readlinkres switch (stat status) {
    case NFS_OK:
        struct {
            path data;
            fattr attributes;
        } readlinkok;
    default:
        void;
};
```

The **readdirres** structure defines the results of a READDIR procedure call. If the call was successful, **readdirres** returns a variable number of directory **entries**, with a total size of up to the amount specified in the argument **count** of the **readdirargs** structure. Each entry contains a unique file identifier, the name of the file, and an opaque entry in the directory, which is used in a subsequent READDIR procedure call to obtain additional entries starting at that ''point'' in the directory. The **eof** flag has a value of TRUE if there are no more directory entries. For TNFS, file attributes are returned with the READDIR procedure call to support the information label adjustment policy.

Note that in responding to a READDIR procedure call, the server shall return only those directory entries which the requesting client process dominates. Thus, security attribute tokens are not required to be returned with each entry, and the directory information which is returned may be passed to the requesting process without additional processing by the client TCB.

```
union readdirres switch (stat status) {
    case NFS_OK:
        struct {
            entry *entries;
            bool eof;
            fattr attributes;
        } readdirok;
    default:
        void;
};
```

3.7.5.2. TNFS Protocol Procedure Definitions

The TNFS Protocol Definition integrates the use of:

- the extended **fattr** and **sattr** structures,
- an AUTH_MLS authentication style RPC credential,
- a new TNFS protocol program number to differentiate between NFS V2 and the security extended TNFS protocol,
- a new protocol procedure, ACCESS, to support the file open enhancement,
- a new protocol procedure, SETNAMELABEL, to support the modification of the file name security attributes, and
- a new protocol procedure, MLD, to support diversion directories

Other than these changes, however, the syntax and semantics of TNFS remain the same as in the original NFS V2 specification.

3.7.5.2.1. Access Procedure

The following descriptions are used to define the new ACCESS procedure.

Definitions used to identify the access request type:

```
#define READ    0x001
#define WRITE   0x002
#define EXEC    0x004
#define SEARCH  0x008
#define APPEND  0x010
#define STAT    0x020
```

Arguments for the remote access procedure:

accessargs

```
struct accessargs {
        fhandle  file;
        u_long   flag;
 };
```

Response from the remote access procedure:

accessres

```
union accessres switch ( stat status ) {
    case NFS_OK:
        struct {
            bool_t status;   /* access status: TRUE or FALSE  */
            fattr  attributes;  /* standard file attributes */
        }  accessok;
    default:
        void;

};
```

Procedure definition for checking remote access permission:

accessres
NFSPROC_ACCESS(accessargs) = 18

Description:

Determine if access as described by **flag** will be permitted on the remote served object **file** by the requester. Flag values are bit encoded as defined previously. READ access means that the data in **file** can be read, WRITE access means that the data in **file** can be modified (written), EXEC access means that **file** can be accessed and executed (local execution of a remote file), SEARCH access means that the directory **file** can be used as the argument to a LOOKUP operation, APPEND means that the file size can be extended, and STAT access means that the requester has permission to stat the file. If **status** is NFS_OK:

accessok.status will be set to **TRUE** if the access request would be allowed, and set to **FALSE** otherwise, and

attributes will contain the complete set of file attributes.

Otherwise:

the NFSERR error number returned identifies the error condition.

Implementation:

The ACCESS procedure provides a means for checking file access permission prior to issuing a subsequent set of file operations. For example, a TNFS client may issue an access procedure as a result of an application's file *open (2)* request to determine if subsequent file *reads* and/or *writes* by the application would be denied by the server as a result of the server's extended file access security policies. Note that the processing of an *open (2)* request for a remote file shall include an *ACCESS* procedure call if the security attributes of the issuing client process have been modified since the last time that process issued an open request for that file. Note also that the information returned by the server in response to an ACCESS procedure call is not static; subsequent file administrative procedures may result in the modification of the file's security attributes.

3.7.5.2.2. Set Name Label Procedure

The following descriptions are used to define the new SETNAMELABEL procedure.

Arguments for the set name label procedure:

setnmlabargs

```
struct setnmlabargs {
        struct diropargs dirargs;
        t_token  sens;
        t_token  info;
        t_token  vend;
 };
```

Response from the set name label procedure:

diropres

```
union diropres switch ( stat status ) {
   case NFS_OK:
       struct {
          fhandle file;
          fattr attributes;
          t_token sens;
          t_token info;
          t_token vend;
   } diropok;
   default:
       void;

};
```

Procedure definition for setting file name security attributes:

diropres
NFSPROC_SETNAMELABEL(setnmlabargs) = 19

Description:

Set the file name security attributes: the sensitivity label **sens**, the information label **info**, and the vendor specific policy label **vend** on the file name **name** in the parent directory **dir**. If **status** is NFS_OK:

then the reply **file** and reply **attributes** are the file handle and attributes for the file **name** in the directory given by **dir** in the argument, and the reply **sens**, reply **info**, and reply **vend** are the sensitivity, information, and vendor specific policy labels for the file name **name**.

Otherwise:

the NFSERR error number returned identifies the error condition

Implementation:

The SETNAMELABEL procedure provides a means for modifying the file name security attributes: the sensitivity, information, and vendor specific policy labels associated with the file name object. When a file is created, the file name sensitivity label will be set equal to the sensitivity value identified in the credential structure, and the file name information label will be set to the information value identified in the credential structure. Once the file is created, however, the sensitivity and information labels of the file name and the file data are maintained independently. The file data security attribute information is maintained by SETATTR, and the file name security attribute information is maintained by SETNAMELABEL.

3.7.5.2.3. MultiLevel Diversion Directory Procedure

The following descriptions are used to define the new procedure to support diversion directories.

Definitions used to identify the MLD request operations:

```
#define CREATE    1
#define REMOVE    2
#define ISMLD     3
```

Arguments for the MLD procedure:

mldargs

```
struct mldargs {
        fhandle  file;
        u_long   op;
 };
```

Response from the remote access procedure:

mldres

```
union mldres switch ( stat status ) {
   case NFS_OK:
      struct {
         bool_t ismld;   /* status: TRUE or FALSE  */
         fattr  attributes;  /* standard file attributes */
      }  mldok;
   default:
      void;

};
```

Procedure definition for maintaining diversion directories:

mldres
NFSPROC_MLD(mldargs) = 20

Description:

Support the creation and removal of diversion directories, and the ability to determine if a given directory is a diversion directory. The CREATE operation requests that a diversion directory be created, the REMOVE operation requests that a diversion directory be destroyed, and the ISMLD operation requests that the diversion status of the file be returned. If **status** is NFS_OK:

if the **mldargs.op** was ISMLD, then **mldok.ismld** will be set to **TRUE** if the file is a diversion directory, and set to **FALSE** otherwise

if the **mldargs.op** was not ISMLD, then mldok.ismld has no meaning

attributes will contain the complete set of file attributes

Otherwise:

the NFSERR error number returned identifies the error condition

Implementation:

The MLD procedure provides the means for creating, removing, and checking for the existence of a diversion directory.

MultiLevel Directory implementations which make use of file name attributes shall return **status** of NFS_OK in response to CREATE, REMOVE, and ISMLD requests, since all directories are MultiLevel Directories in such an environment and thus no explicit action is required.

3.7.5.2.4. TNFS Service Routines

The TNFS protocol definition is defined below as a set of procedures, arguments, and results. All modified data structure definitions are included in this specification. Most NFS V2 protocol data definitions remain unchanged, and are documented in the NFS V2 protocol specification. The complete set of TNFS protocol procedures are defined below. The ACCESS, SETNAMELABEL, and MLD procedures are new, but the other procedures are the same as those defined in the NFS V2 specification. The GETATTR, SETATTR, LOOKUP, READLINK, READ, WRITE, CREATE, MKDIR, READDIR, ACCESS, SETNAMELABEL, and MLD procedures for the TNFS protocol, however, include the extended file attribute structure *fattr* in the response message.

```
program TNFS_PROGRAM {
   version TNFS_VERSION {
      void        NFSPROC_NULL (void) = 0;
      attrstat    NFSPROC_GETATTR (fhandle) = 1;
      attrstat    NFSPROC_SETATTR (sattrargs) = 2;
      diropres    NFSPROC_LOOKUP (diropargs) = 4;
      readlinkres NFSPROC_READLINK (fhandle) = 5;
      readres     NFSPROC_READ (readargs) = 6;
      attrstat    NFSPROC_WRITE (writeargs) = 8;
      diropres    NFSPROC_CREATE (createargs) = 9;
      stat        NFSPROC_REMOVE (diropargs) = 10;
      stat        NFSPROC_RENAME (renameargs) = 11;
      stat        NFSPROC_LINK (linkargs) = 12;
      stat        NFSPROC_SYMLINK (symlinkargs) = 13;
      diropres    NFSPROC_MKDIR (createargs) = 14;
      stat        NFSPROC_RMDIR (diropargs) = 15;
      readdirres  NFSPROC_READDIR (readdirargs) = 16;
      statfsres   NFSPROC_STATFS (fhandle) = 17;
      accessres   NFSPROC_ACCESS (accessargs) = 18;
      diropres    NFSPROC_SETNAMELABEL (setlabargs) = 19;
      mldres      NFSPROC_MLD (mldargs) = 20;
   } = 1;     /* Trusted NFS Version 1  */
} = 390086;    /* Trusted NFS Program Number */
```

3.7.6. Using TNFS

With the TNFS protocol procedures described above, listing and modifying remote extended file attributes is now supported. The definition of a new application programming interface (API) to support the display of a file's security attributes will permit either a new file list command (e.g., lsacl, lsmac) or a modification to the existing *ls (2)* command to display the security attribute information associated with a remote file. Likewise, the definition of a new API for setting a file's security attributes will permit new change security attribute commands to be developed (e.g., chacl, chmac).

The file open enhancement discussed previously may now be supported. The open API will be translated into a GETATTR operation for the current directory, a LOOKUP operation for the file to be opened, and an ACCESS operation which returns a boolean value indicating whether the access requested would be permitted, along with the complete set of the file's attributes. Thus, the TNFS client can determine whether the application requesting to *open* the remote file will be able to access it based on the open request type and the application's security credentials. As described earlier, a server may choose to associate a set of privileges with the remote subject which are different from the privilege set associated with the subject on the client system. The ACCESS procedure call returns the server's assessment of the subject's access capabilities.

The information label adjustment policy is supported, since the AUTH_MLS credential contains the subject's information label, and the TNFS reply message contains an extended file attribute structure which includes the file object's information label. Note that the subject's information label may require adjustment as a result of reading a remote file (READ), reading a remote directory (READDIR), or reading a remote symbolic link (READLINK). A remote file's (object) information label may be adjusted as a result of SETATTR, WRITE, CREATE, RENAME, LINK, SYMLINK, and MKDIR TNFS procedure calls.

File names may now be protected by MLS policy with the introduction of file name security attributes, and the SETNAMELABEL procedure.

Finally, MultiLevel Directories are accommodated.

3.7.7. TNFS Access Control Policy

The access control policy recommended by this proposal may be stated as follows:

- a client system shall *always* apply the access control policy to a local request for access to a local resource,
- a server system shall *always* apply the access control policy to a local request for access to a local resource,
- a server system shall *always* apply the access control policy to a remote access request for a local resource, and
- a client system *may (temporarily)* apply the access control policy to a locally cached remote resource, iff:
  - client security attribute caching support is included in the implementation, and
  - a client security attribute caching policy is enabled by the host security officer

This TNFS access control policy ensures that no access will be made without the application of appropriate access control.

3.7.8. TNFS Auditing Policy

The auditing policy recommended by this proposal is stated as follows. When the security auditing function is enabled:

- **an implementation shall:**
  (1) audit all local requests for local file access:
    - a client system shall *always* audit a local request for access to a local resource,
    - a server system shall *always* audit a local request for access to a local resource

  (2) provide the **capability** to audit all remote file access requests:
    - the client shall support the capability to audit local requests for access to remote resources on a server, and
    - the server shall support the capability to audit remote requests for access to local resources on the server[8]

[8] This option may require the auditing of the specific TNFS protocol procedure calls, since the protocol procedures

(3) enable client system auditing of local requests for access to remote files by default

Thus, when the security auditing function is enabled:

- o all local requests for access to local files are audited,
- o client system requests for access to remote files are audited[9]
- o the capability to audit remote file access by both client and server is provided:
  - * client system auditing may be enabled to audit local requests for access to remote resources; client system auditing is enabled by default,
  - * server system auditing may be enabled to audit remote requests for access to local resources
- o enabling of the remote file access auditing capability shall be supported by a system management operation

This TNFS policy ensures that each TNFS host shall audit local requests for local file access, each TNFS client system shall audit requests for remote file access (by default), and both TNFS clients and servers shall have the cability to enable auditing of remote file access activity. In a given network environment, it may be desirable to optionally disable auditing of remote access on either the client or the server to avoid duplication.

3.7.9. The Extended Attribute Cache

NFS caching strategies are implementation specific, and are not part of the NFS protocol. Caching is not required to support TNFS interoperability. This specification will therefore not include specific details on the issue of attribute caching. However, since the caching mechanisms are included in the NFS reference source code releases, and since attribute caching is critical for achieving NFS performance goals, several suggestions are included in this section.

In most NFS client implementations, remote file attributes are cached on the client, improving performance and reducing network traffic. The attribute cache is updated frequently, as most NFS procedures return file attributes along with other requested information.

A client side cache for the extended security file attributes should also be considered for similar reasons. Since all of the file's security attributes are returned with each TNFS file access request, an extended security attribute cache can now be maintained on the client.

Extending the attribute validation procedure to include validating the security file attributes permits the complete set of file attributes to be checked and refreshed if they are no longer valid. If the file's cached attributes are not valid, a **GETATTR** procedure call can be made. The TNFS reply to this procedure now includes the complete set of file attribute information, permitting all of the file's cached attributes to be refreshed. Cached attribute entries shall be aged and eventually flushed unless refreshed. If client caching is enabled, then per process cached attribute entries shall be maintained.

Note again that an attribute caching policy is not part of the protocol, and is an implementation technique used to improve performance. During the window of time that the cache entry is valid, the client system applies the MLS access control policies on behalf of the server. It is recommended that if an implementation supports the use of client side attribute caching, it shall also support a mechanism for disabling the attribute cache. Specific implementation details are provided in [4].

4. Related Requirements and Expectations

This specification addresses extensions the NFS V2 protocol which accommodate network file access in a trusted, MLS network environment. Expectations for the environment for which this specification is applicable include:

---

are not translated into actual ''system calls'' in many server implementations.

[9] This is the default policy; site specific auditing policies are established by the site security officer.

- o the TNFS network environment is a trusted environment:
  - > TNFS authentication and message integrity support shall not be required
  - > use of TNFS in an untrusted environment (e.g., commercial network environment) is not addressed by this specification
- o other, related RPC services are required to support the execution of NFS; these services shall support the AUTH_MLS credential flavor, but may also support alternative policies which make use of other authentication flavors:
  - > the token management service is required to translate security attributes between expanded and tokenized formats [5],
  - > the mount service is required to support NFS mount requests,
  - > the lock manager and status monitor services are required to support NFS file and file region locking
- o client side mounts shall be restricted to the server's exported mount points:
  - > client requests to mount a subdirectory which resides below the export point in the server's exported directory shall be denied,
  - > without this restriction, client access to server files mounted below the server's export point bypass the authorization checks which would otherwise have been made using the access modes of the file components located higher in the server's exported tree[10]
- o most file access will take place between MLS modified clients and servers, but some TNFS systems will continue to interoperate with NFS V2 systems through the use of an appropriate policy; for example, a filter or gateway could be placed between a MLS system and an unmodified system to insert or delete appropriate security attribute information on behalf of the unmodified system

  note that client system auditing information will not be supplied for remote file access initiated from an unmodified NFS client; enabling server system auditing should be considered by the security officer to support these configurations
- o a TNFS client should not send any security extended NFS procedure calls to a server which does not support this service; a TNFS client should also refrain from sending extraneous security attribute information to a TNFS server that does not support those attributes
- o additional TCB information[11] is maintained by each MLS system to support trusted interoperability [10]; for example, each MLS host may:
  - > maintain a list of the hosts which it will communicate with,
  - > maintain the set of security attributes which it expects to use in the exchange of data with a given host, and
  - > maintain the specific translation scheme or schemes which will be used in translating tokens with a given host [5]
- o the security information defined within the AUTH_MLS credential and file attribute structures provides for the transfer of security attributes required to support MLS access policies without requiring the underlying network layer to provide security attribute information:
  - > if security attributes are provided by both the RPC layer and the underlying network layer, then the security attribute information provided by the RPC layer shall be applied to the file data transferred within the RPC message

[10] Note that appropriate use of symbolic links on the client will result in a client file name space similar to one previously constructed by mounting subdirectories of exported server file trees.

[11] Note that this information is needed by all trusted network applications, and is not limited to NFS file access.

> transferring security attributes within the RPC layer provides for the support of a policy where data may be transferred with a security classification which is different from the security classification of the network layer; for instance, file data with a given security classification might first be encrypted and then transferred through a network with a lower security classification.

> support for the transfer of MAC sensitivity labels for the Internet Protocol Suite has been addressed by the CIPSO [11], and IPSO [12] documents

5. Conclusion

This document describes the set of extensions which support network file access in a network environment consisting of MLS systems using the proposed TNFS protocol extensions. Unmodified NFS clients and servers are supported using the de facto standard NFS V2 protocol.

With the previously defined extensions, the MLS network file access requirements are met. The extended structure definitions support the DAC and MAC attributes required for modifying or displaying the security attribute information. The enhanced file open operation and the information label adjustment policies are also supported.

Thus, a small set of extensions to the NFS V2 environment permits MLS access control policies to be supported. Agreement on these changes will permit the current base of NFS clients and servers to be accommodated in the secure environment with no changes, and for TNFS modified systems to interoperate using MLS policies.

6. Acknowledgements

I would like to acknowledge the members of the ITEF/TSIG NFS Subcommittee, who were instrumental in evolving the MLS extended NFS Protocol Specification from the original proposal. Many comments were also made during the review of the later drafts which greatly improved the specification's readability. Contributing IETF TNFS working group members include Jeff Edelheit, Fran Fadden, Jonathon Fraser, Ali Gohshan, Carl Smith, Mark Saake, Dave Summers, and Charlie Watt. I'd also like to acknowledge the contributions of the original members of the TSIG Trusted NFS working group; in addition to the above, these members included Morgan Clark, Tricia Jordan, Will Lees, Scott Norton, and Mike Shipley.

The specification was also reviewed by numerous persons outside of the subcommittee. I would like to acknowledge these persons as well, as a number of their comments are also reflected in the final version.

7. Author's Address

Fred Glover
Digital Equipment Corporation
110 Spit Brook Road ZK03-3/U14
Nashua, New Hampshire 03062-2698

Phone: 603-881-0388

EMail: fglover@zk3.dec.com

8. References

[1] Sun Microsystems, Inc., "Network Filesystem Specification", RFC-1094, DDN Network Information Center, SRI International, Menlo Park, CA.

[2] National Computer Security Center, United States Department of Defense, "Trusted Computer Systems Evaluation Criteria" National Computer Security Center, Ft. George G. Meade, MD., 1985, DoD 5200.28-STD

[3] Defense Intelligence Agency, United States Department of Defense, "Compartmented Mode Workstation Evaluation Criteria", Defense Intelligence Agency, Washington, D.C., DIA document number

DDS-2600-6243-91

[4] Trusted Systems Interoperability Group, "The MLS NFS Implementor's Guide", TSIG Document

[5] Trusted Systems Interoperability Group, "The MLS Token Translation Specification", TSIG Document

[6] Sun Microsystems, Inc., "Remote Procedure Call Specification", RFC-1057, DDN Network Information Center, SRI International, Menlo Park, CA.

[7] Sun Microsystems, Inc., "External Data Representation Specification", RFC-1014, DDN Network Information Center, SRI International, Menlo Park, CA.

[8] Clark, D. D. and David R. Wilson, "A Comparison of Commercial and Military Computer Security Policies", Proceedings of the 1987 IEEE Symposium on Security and Privacy, IEEE Computer Society Press, Washington, DC.

[9] Biba, K. J., "Integrity Considerations for Secure Computer Systems", TR-76-372, Electronic Systems Division, Air Force Systems Command, U.S. Department of the Air Force, Hanscomb AFB, MA., April 1977

[10] Trusted Systems Interoperability Group, "Trusted Administration Specification", TSIG Document

[11] Trusted Systems Interoperability Group, "Commercial IP Security Option", TSIG Document

[12] "The IP Security Option", RFC-1108, DDN Network Information Center, SRI International, Menlo Park, CA.

[13] Postel, J., "User Datagram Protocol", RFC-768, DDN Network Information Center, SRI International, Menlo Park, CA.