Transarc Corporation **C. Everhart (Transarc)**
Request For Comments: 90.0
February 1996

SECURITY ENHANCEMENTS FOR DCE DFS

1. INTRODUCTION

This document reviews the authentication that DCE DFS employs in its RPC communications. It proposes changes and extensions to the DFS authentication model, not only to plug authentication holes but also to allow DFS administrators to affect the levels of authentication that DFS uses.

2. AUTHENTICATION USAGE IN DFS

2.1. High-Level Review of Existing Usage

In DCE 1.1 and prior releases, DFS has used a mix of authenticated and unauthenticated DCE RPCs. Most server-to-server communication has been authenticated at the `rpc_c_protect_level_pkt_integ` level, except for communication between peer `repserver` processes, which have been unauthenticated. RPC communication initiated by administrative commands (`fts`, `bos`, `bak`) to user-space servers has always been authenticated at the `rpc_c_protect_level_pkt_integ` level or above, except for communication to `repservers`, which again has been unauthenticated. RPC communication from the file exporter to the token-revocation interfaces of the token clients (the DFS Cache Manager, the `fts` and `dfsexport` commands, and the `repserver` process) has been unauthenticated. Lastly, communication initiated by the DFS Cache Manager has been mixed: RPC calls to the user-space `flserver` and `repserver` processes have been unauthenticated, and calls to the file exporter have been authenticated only when they are made on behalf of correctly-authenticated users running on the Cache Manager machine, and then only at the `rpc_c_protect_level_pkt` level.

2.2. Discussion

Several aspects of the existing authentication usage are inadequate in the face of a variety of attacks. Unauthenticated communication precludes all kinds of defenses against possible attacks; it should be used only as a last resort, if at all. However, DFS in the DCE 1.1 and earlier releases uses unauthenticated RPCs in several contexts where authentication is available and where performance requirements are not overwhelming. In addition, in some contexts, it makes sense for administrators to choose the relatively high level of security offered by the `rpc_c_protect_level_pkt_privacy` authentication level.

The `rpc_c_protect_level_pkt` authentication level could be the target of a data-substitution attack by a network-bridge-level attacker, as noted in the article by Eric Brewer et al. ([Brew95]). The DCE `rpc_c_protect_level_pkt_integ` level should be adequate to prevent such data substitution, but the DFS Cache Manager is using a lesser level of protection, `rpc_c_protect_level_pkt`, for performance reasons. Ideally, the administrators for the DFS Cache Manager's communication would be able to control the protection level being used, allowing them to choose a trade-off point between performance and security.

Other data-substitution attacks are possible in the DFS context, such as altering the data returned by the `flserver` to a DFS cache manager. Since such data traffic is relatively low-bandwidth, it costs little to use the higher protection levels available with DCE RPC in all cases.

3. PROPOSED CHANGES

The goal of this work is to close the authentication holes in a backward-compatible fashion. This work has several components:

(a) Alter the clients making RPCs to the `repserver` (the `fts` command, peer `repserver` processes) to use authenticated RPCs.

(b) Alter the file exporter to make authenticated token revocation RPCs when the exporter's token client passes in a usable principal name.

(c) Alter the token-obtaining clients of the file exporter to pass a usable server principal name to the file exporter, so that token revocation calls may be authenticated. The principal name will be for the server in the token-revocation protocol, which is the client in the file-exporter protocol.

(d) Alter the DFS Cache Manager's RPCs to the `flserver` and the `repserver` to be authenticated.

(e) Alter the DFS Cache Manager's RPCs to the file exporter to have their authentication controlled by the administrators for the cache manager and for the file exporter.

This last component of the proposed work may itself be broken down into several sub-tasks, since administrators will wish for a variety of fine controls on the behavior of their systems. One central concept behind the administrative controls is that we distinguish same-cell (intra-cell) communication from inter-cell communication, so that a DFS Cache Manager will use one set of guidelines for its intra-cell communication and another set for its inter-cell communication. The tasks that implement the administrative controls include the following:

(a) Definition of an initial (preferred) selection of DCE RPC authentication level and an authentication lower-bound level for the intra-cell and inter-cell file exporter communications carried out by the CM.

(b) Addition of four new `dfsd` options by which these levels may be set when a CM is initialized.

(c) Addition of new `cm getprotectlevels` and `cm setprotectlevels` subcommands to read and alter these levels in the running CM when executed by the machine administrator, as well as two new `pioctl()` sub-calls to obtain and set these levels.

(d) Addition of new distinguished DFS error codes that may be used to communicate violations of authentication policy bounds from a file exporter to a CM, along with CM modifications to react to these error codes and track the exporter's limits.

(e) Definition of DCE RPC authentication upper-bound and lower-bound levels that a file exporter may use in its DCE RPC service, one pair for intra-cell access and one pair for inter-cell access.

(f) Addition of four new `fxd` options by which these levels may be set when the file exporter is initialized.

(g) Addition of advisory DCE RPC authentication upper-bound and lower-bound levels that may be used when communicating with a particular fileset, a new `fts setprotectlevels` subcommand to set these levels in the FLDB entry for a fileset, and consumption of four bytes of spares in both the `vldbentry` passed across the RPC and in the `struct vlentry` saved in the Ubik database itself.

3.1. Changes to Repserver Clients

The changes in this area are straightforward. Both the `fts` command and the `repserver` itself are altered to establish authentication on binding handles in use for calls to `repservers`. The `repserver` itself, when making these peer-to-peer calls, always has available the authentication for the host's `dfs-server` principal. The `fts` command will use the authentication of the calling user, if any, and will obey the `-noauth` and `-localauth` command options as is done for communication with the `ftserver` as well.

There is no backward-compatibility problem, in that the old `repserver` is able to accept authenticated calls, and the new `repserver` is not at the moment being changed to require authentication in its incoming RPC calls.

3.2. Authenticated Token Revocations

It is straightforward to authenticate token-revocation RPC calls as the local machine identity (the host's `self` identity). The only difficulty is knowing the remote principal to which the calls should be authenticated. Fortunately, the `AFS_SetContext()` RPC call already has room for the principal name: the third parameter to the RPC is a pointer to an `afsNetData` structure that has a `principalName` field that was reserved for this purpose. Thus, the change here is to accept a value for the principal name (for the token-revocation RPC server, thus for the caller of the `AFS_SetContext()` RPC itself) and to call `rpc_binding_set_auth_info()` to establish authentication on the token-revocation binding.

Old clients of the `AFS_SetContext()` RPC all clear the entire `afsNetData` structure before using it, so the file exporter may readily distinguish whether a principal name has been filled in. It may thus reliably determine whether it should or should not attempt to authenticate the token-revocation binding.

3.3. Passing Principal Names to the File Exporter

Several processes in DFS need to obtain tokens in their ordinary operation, and they must be altered to pass their principal name to file exporters in order to allow those exporters to make authenticated token-revocation calls. These processes include the DFS CM itself, the `repserver`, the `fts` command, and the `dfsexport` command. These processes will select a candidate principal name, will call `rpc_server_register_auth_info()` with that information to register the proper information with the runtime, and will pass that principal name to the DFS file exporter for its use in authenticating the outgoing token-revocation RPCs.

The DFS CM can always use the principal name associated with the machine identity (the host's `self`); the `dfsd` program passes this principal name to the DFS CM at initialization time as the fourth parameter to the `CMOP_START_TKN` start-up system call. The `dfsexport` command, since it must run as local root, may use the same identity. The `repserver` can use the `dfs-server` principal. The `fts` command may or may not have access to a key table entry, so it may or may not be able to make use of authenticated token-revocation RPCs. If `fts` is being run with its `-localauth` option, it will use the `dfs-server` principal. Otherwise, if run as local root, it will use the `self` machine principal. Failing both of these, `fts` will not request any authentication for its token revocations.

There is no backward-compatibility problem with this new use of the `afsNetData.principalName` field, since old-style file exporters will completely ignore any value placed in this field.

3.4. Authenticating DFS CM Calls to the Flserver and the Repserver

RPCs from the DFS CM to these servers are currently being made without authentication. These calls may simply be made as the machine's `self` identity, using the `rpc_c_protect_level_pkt_integ` authentication level.

Both the `flserver` and `repserver` are able to accept both authenticated and unauthenticated calls without modification, so there is no problem with backward compatibility.

3.5. Authenticating DFS CM Calls to the File Exporter

In DCE 1.1 and earlier releases, RPCs from the DFS CM are made with the authentication of the caller. This allows for simple access checks at the file exporter, since all the DCE PAC/EPAC machinery identifies the calling application to the file exporter for interpreting ACLs and other access information. Thus, the distinction between unauthenticated and authenticated RPCs, made to the file exporter, is important: the file exporter will interpret the authentication state of the RPC as the authentication state of the caller of the file system. It would be inappropriate for the DFS CM to start making RPCs from unauthenticated clients as, say, RPCs authenticated with the machine `self` principal, since the file exporter would assign different privileges to such calls than it would to truly unauthenticated calls.

The variety of authorization services provided by DCE RPC offers an excellent work-around. The file exporter has always treated as unauthenticated all incoming calls that are authorized with anything other than the DCE PAC service (`rpc_c_authz_dce`). Thus, RPCs to the file exporter for which the authorization service is the client principal name (`rpc_c_authz_name`) will be interpreted as unauthenticated. The work-around is therefore that, when the DFS CM might ordinarily wish to make an unauthenticated call to a file exporter, it should instead use `rpc_c_authz_name` authorization for that call with authentication as the machine identity. The range of DCE authentication services is thus available for assuring data integrity even for those calls that the DFS file exporter will view as unauthenticated.

Removing these unauthenticated calls is only a partial repair; the performance/security tradeoff still requires attention. DFS in the DCE 1.1 and earlier releases uses `rpc_c_protect_level_pkt` authentication, which does not defend against the data-substitution attack cited in the article by Eric Brewer et al. [Brew95]. However, simply increasing the authentication level to `rpc_c_protect_level_pkt_integ` would require all data transfers to incur an overhead of (very roughly) one CPU second per megabyte of data for each of the client and the server, and this is viewed as excessive in many environments. Thus, we define a collection of administrative interfaces by which the authentication level may be selected by the administrators of the client and the server ends of the communication. Generally, the server policy imposes hard upper and lower bounds on the authentication levels at which it will operate, and the client policy selects an initial desired value for the authorization level as well as a lower bound for the authorization level, below which it will refuse to operate. Furthermore, these authorization policy values are duplicated: one set of values is used for intra-cell communication (where the client and the server are in the same DCE cell), and another set of values is used for inter-cell communication.

3.5.1. Initial and lower-bound authentication policy values for the CM

The DFS CM maintains both an initial value for DCE RPC authentication level and a lower bound for that level. The DFS CM also maintains shadow copies of other bounds imposed by policies on the file exporter, and it uses these bounds, as well as its own lower bound, to guide its selection of authentication level.

The DFS CM maintains two independent pairs of authentication values, so that at any given point, it will have an initial level and a lower bound for use with intra-cell RPCs and a separate initial level and lower bound for use with inter-cell RPCs.

Since the choice of DCE RPC authentication level is completely driven by the client, and since these authentication features have been in all prior releases of DCE, new clients will have no trouble establishing new authentication levels with old servers.

3.5.2. New dfsd options to initialize authentication bounds

The `dfsd` command accepts four new options by which the DFS CM's authentication policy values may be initialized, as follows:

(a) `-initiallocalprotectlevel` *`level`* specifies the initial value for the intra-cell authentication level.

(b) `-minlocalprotectlevel` *`level`* specifies the lower bound for the intra-cell authentication level.

(c) `-initialremoteprotectlevel` *`level`* specifies the initial value for the inter-cell authentication level.

(d) `-minremoteprotectlevel` *`level`* specifies the lower bound for the inter-cell authentication level.

The *`level`* value for this command may be specified as any of: (1) the DCE identifiers `rpc_c_protect_level_`*XXX*, or (2) `rpc_c_authn_level_`*XXX*, as given in the manual page for `rpc_binding_set_auth_info()`; (3) the *XXX* suffixes themselves; or (4) the integers to which they evaluate (0 through 6).

The `dfsd` program passes these policy values to the DFS CM as additional values in the `cm_cacheparams` structure. There is no backward-compatibility problem, primarily because mutually-compatible versions of the `dfsd` program and the DFS CM itself are presumed to be installed together, and secondarily because the new values were added to the end of the `cm_cacheparams` structure, so that at least a new `dfsd` and an old CM will work together.

3.5.3. New cm subcommands to read and alter authentication bounds

The `cm` command will provide two new subcommands, `cm getprotectlevels` and `cm setprotectlevels`, for examining and altering the authentication policy values in the running DFS CM. These policy values may be examined by anyone on the machine, but altering the values is restricted to processes running as the machine root.

The new `cm getprotectlevels` subcommand accepts no options (other than `-help`). The new `cm setprotectlevels` subcommand accepts the same set of four options that were added to the `dfsd` command, above:

```
% cm help getprotectlevels
cm getprotectlevels: get protection levels
Usage: cm getprotectlevels [-help]
```

```
% cm help setprotectlevels
cm setprotectlevels: set protection levels
Usage: cm setprotectlevels \
       [-initiallocalprotectlevel <level>] \
       [-minlocalprotectlevel <level>] \
       [-initialremoteprotectlevel <level>] \
       [-minremoteprotectlevel <level>] [-help]
```

By default (in the reference port), the CM's protection policy values are initialized as follows:

```
% cm getprotectlevels
Initial protection level in the local cell: \
        rpc_c_protect_level_pkt_integ
Minimum protection level in the local cell: \
        rpc_c_protect_level_none
Initial protection level in non-local cells: \
        rpc_c_protect_level_pkt_integ
Minimum protection level in non-local cells: \
        rpc_c_protect_level_pkt
```

Other default protection policy values may be chosen by a vendor without loss of interoperability. For example, in DCE 1.1 and prior releases, DFS operated approximately as if the initial protection level for all cells were `rpc_c_protect_level_pkt` and the minimum protection level for all cells were `rpc_c_protect_level_none`. Vendors and administrators must consider the tradeoff between security and performance in altering these default protection policy values.

To support these new subcommands, two new `pioctl()` sub-calls will be added: `VIOC_GETPROTBNDS`, to read the policy values, and `VIOC_SETPROTBNDS`, to change them.

3.5.4. New DFS error codes for authentication level feedback to the CM

Six new error codes are defined to indicate to the DFS CM that the CM's chosen authentication level exceeds a policy bound value. The error code values are defined in pairs, one to indicate that the chosen level is too high (exceeds the policy's upper bound) and the other to indicate that the chosen level is too low (exceeds the policy's lower bound). Three pairs are allocated to allow independent policies to be described: an exporter-wide policy, a per-fileset policy, and any other policy:

(a) `FSHS_ERR_AUTHNLEVEL_S_TOO_HIGH` indicates that the authentication level is too high for this file exporter.

(b) `FSHS_ERR_AUTHNLEVEL_S_TOO_LOW` indicates that the authentication level is too low for this file exporter.

(c) `FSHS_ERR_AUTHNLEVEL_F_TOO_HIGH` indicates that the authentication level is too high for this fileset.

(d) `FSHS_ERR_AUTHNLEVEL_F_TOO_LOW` indicates that the authentication level is too low for this fileset.

(e) `FSHS_ERR_AUTHNLEVEL_G_TOO_HIGH` indicates that the authentication level is too high for some other constraint.

(f) `FSHS_ERR_AUTHNLEVEL_G_TOO_LOW` indicates that the authentication level is too low for some other constraint.

The client is expected to react to these error codes either by adjusting its authentication level and retrying, or by giving up if its own limits are exceeded. The proposed DFS CM, however, will not be able to react gracefully to all of these error codes. It maintains *shadow copies* of the policy bounds within which it must operate, on both a per-exporter basis and a per-fileset basis. These bounds are initialized to the widest possible range ([`rpc_c_protect_level_none` ... `rpc_c_protect_level_pkt_privacy`]). When the CM receives one of these error codes, it knows that one of its shadow bounds is too generous, so it narrows the appropriate bound by one level and selects a new authentication level at which the RPC may be retried. The CM will react to the exporter-wide and per-fileset error codes in this way, since it knows the object to which the policy applies. However, for the third pair of error codes, it has no understanding of what internal limit should be adjusted, so if the DFS CM receives one of these two error codes, it will fail the RPC call.

This protocol for maintaining shadow copies of the authentication policy bounds is efficient in practice, even if each RPC adjusts a shadow bound by only one level, since there are so few separate levels defined and the shadow bounds are cached.

In the proposed work, only the exporter-wide calls are actually generated; the others are reserved for the future.

These codes are, in a sense, backward-compatible with existing DFS deployments, but those existing DFS deployments will not be able to handle them gracefully. That is, if a DFS CM from DCE 1.1 or earlier releases receives one of these error codes, it will not know how to respond, so it will fail the RPC call. This is appropriate behavior, in that the new DFS file exporter will have returned one of these error codes to reflect a policy that the CM's authentication choice must not be honored; since an old-style DFS CM will be unable to alter its authentication choice to meet the policy requirement, it must continue to fail. However, this less-than-graceful handling of the compatibility issue suggests that it is a serious matter for the file exporter's administrator to impose policies about authentication levels on the file exporter, since DCE 1.1-style DFS CMs will be unable to respond well.

3.5.5. Authentication bounds for DFS file exporters

For any of several reasons, the administrator for a file exporter might wish to bound the authentication support that the exporter may offer. For example, the administrator may wish to impose a lower bound in order to ensure that its data remains secure in transport, or an upper bound to limit the security overhead incurred by the exporter machine itself. This proposal includes the specification of such policy bounds for DFS file exporters, again duplicated as one bounds pair for intra-cell use and another bounds pair for inter-cell use. When a file exporter detects that an RPC that it is servicing is not within the applicable bounds, it responds to the RPC with the appropriate distinguished error code, either `FSHS_ERR_AUTHN_LEVEL_S_TOO_HIGH` or `FSHS_ERR_AUTHN_LEVEL_S_TOO_LOW`, as defined above.

3.5.6. New fxd options to initialize authentication bounds

The mechanism by which the administrator indicates policy bounds to the file exporter is a set of four new options to the `fxd` command, the command that initializes the file exporter:

(a) `−minlocalprotectlevel` *`level`* sets the lower bound for the intra-cell authentication level for this file exporter.

(b) `−maxlocalprotectlevel` *`level`* sets the upper bound for the intra-cell authentication level for this file exporter.

(c) `−minremoteprotectlevel` *`level`* sets the lower bound for the inter-cell authentication level for this file exporter.

(d) `−maxremoteprotectlevel` *`level`* sets the upper bound for the inter-cell authentication level for this file exporter.

The values for *`level`* may be specified as in the `dfsd` command or the new `cm` subcommands, as described earlier. The values are passed to the file exporter via a pointer passed as the fourth argument to the `PXOP_INITHOST` system call.

Compatible versions of the `fxd` program and the file exporter are expected to be installed together, so there should be no issue of backward compatibility.

3.5.7. New per-fileset advisory authentication bounds

This proposal includes the specification of per-fileset advisory authentication policy bounds, stored in the FLDB along with the data of the fileset, but not checked by the file exporter itself. These bounds are to be used by clients of the file exporter when selecting authentication levels for RPCs that operate on particular filesets, and they may in the future be validated by the file exporter itself. Within this work, though, they may be used by administrators wishing to push cooperating DFS CMs into using a particularly high or particularly low authentication value for accesses to a given fileset. These bounds are duplicated to permit distinguishing intra-cell accesses from inter-cell accesses.

While these bounds are at the moment purely advisory, they may in the future be enforced by the file exporter. The administrator is therefore cautioned against imposing excessively restrictive bounds.

These advisory bounds are specified by a new subcommand for the `fts` program: `fts setprotectlevels`. This subcommand identifies a fileset and alters any or all of its advisory authentication bounds. The advisory authentication bounds may be examined by any of the existing `fts` subcommands that print FLDB entries, such as `fts lsfldb`, `fts lsft`, and the like:

```
% fts help setprotectlevels
fts setprotectlevels: set range of permissible protection \
                      levels
Usage: fts setprotectlevels -fileset {<name> | <ID>} \
       [-minlocalprotectlevel <level>] \
       [-maxlocalprotectlevel <level>] \
       [-minremoteprotectlevel <level>] \
       [-maxremoteprotectlevel <level>] \
       [-cell <cellname>] [{-noauth | -localauth}] \
       [-verbose] [-help]
```

Execution of this subcommand will require permission to alter the FLDB entry for the given fileset, implying either membership in the cell-wide `admin.fl` list or membership in the group that owns the FLDB server entry for all servers on which copies of the fileset reside. If this command is later enhanced to allow the file exporter to enforce these limits, it will also require membership in the `admin.ft` list for the file exporter on which one or more copies of the fileset reside.

These advisory bounds are communicated to the `flserver` by storing them in the first four bytes of the `charSpares` array in both the `vldbentry` and `compactvldbentry` structures. The `flserver` stores them in the FLDB itself by storing them in the first four bytes of the `spares3` array in the `vlentry` structure that is stored in the Ubik database itself.

The proposed DFS CM will maintain copies of these bounds in its in-memory structure and will use these bounds to adjust its initial choice for authentication levels on a per-fileset basis. However, the DFS CM will maintain independent per-fileset shadow copies of any fileset-based authentication bounds, so that it may react correctly to any future file exporters that enforce per-fileset authentication policies.

The current per-fileset authentication bounds are only advisory. It would be preferable for the file exporter to be able to enforce them. Carrying this out would involve a good bit of additional work in modifying the `fts setprotectlevels` subcommand to set new status fields in one or more copies of the indicated fileset, defining extensions to the fileset dump format to represent these new bounds, enhancing the fileset dump and restoration operations to capture and reinstate these bounds, and causing the file exporter to verify the policy bounds. This may be done in the future based on the existing values in the FLDB, so administrators should beware of setting unduly restrictive per-fileset authentication bounds in the knowledge that they are at present not enforced by the file exporter.

Since these per-fileset authentication bounds are represented using fields that are at present spares initialized to zero, there are few issues of backward compatibility. One interesting issue is that existing versions of the `flserver` will not know about them. If the `fts setprotectlevels` subcommand is issued in a cell where none of the `flserver` processes has been extended as in this proposal, the command will appear to have succeeded but will have had no effect. If a cell has mixed versions of its `flserver` processes, the subcommand will have an effect only if the RPC to change the fileset's FLDB entry was directed at an updated `flserver` processes. Furthermore, even if the FLDB had been updated via an updated `flserver` process, if the FLDB information for a fileset is obtained by a call to a non-updated `flserver` process, the authentication bounds will not be present. This last issue affects not only the user-space `fts` and other commands, but also the DFS CM itself, which learns of these authentication bounds from calls made to an `flserver` process.

4. CONTINUING ISSUES AND PROBLEMS

(a) There is no way to determine whether an `rpc_c_authz_name` RPC caller is from the local cell or another cell. The proposed work-around is for the file exporter to treat all clients as from a non-local cell until a `rpc_c_authz_dce` request arrives that may then have the identity of its cell checked.

(b) This work assumes that there is a trust relationship between the cell of the DFS Cache Manager and the cell of the file exporter, since it expects to be able to fall back to authentication as the `self` identity. As of the current releases of DCE, the `dfsbind` helper process cannot make successful contact with untrusted cells, which therefore implies that the DFS CM itself cannot contact such cells. Since even the local machine identity cannot be used to establish any authentication with such cells, the DFS CM would need to allow completely unauthenticated non-local access in order to successfully communicate with the servers in such cells.

(c) Bill Sommerfeld (HP) has suggested that the DFS CM limit its behaviors on data that had been fetched over less-secure channels. In particular, he proposed that files fetched without authentication should never be used as device special files, as set-uid/set-gid executables, or in fact as executables at all. While such a feature is desirable, our sense is that it would require further design and elaboration to work smoothly, including an enhanced cache model in the DFS CM in which every cached datum would be tagged with the authentication level that had been used in fetching it, and in which a cached datum could potentially be re-fetched with a higher authentication level before using it. In addition, there would have to be administrative controls on this behavior, and DFS already provides some of these controls in the `cm setsetuid` and `cm setdevok` functions.

(d) Token-obtaining clients other than the DFS CM (the `repserver` process and the `fts` and `dfsexport` commands) use authenticated `rpc_c_protect_level_pkt_integ` whenever any DCE authentication is available to them. However, they do not know how to react to the new error codes that indicate a violation of authentication policy bounds, nor do they know how to choose other authentication levels. Thus, if an administrator imposes a policy such that the `rpc_c_protect_level_pkt_integ` authentication level is not allowed, they may be causing these other programs to fail.

(e) It would be possible for the DFS CM to react to the third pair of new policy-violation error codes by retaining a third set of shadow policy bounds in a transient structure used only for the duration of a given RPC retry sequence. This has not been done, chiefly on efficiency grounds, since the adjustment of authentication levels and retrying would have to be carried out on each RPC.

(f) While causing the per-fileset advisory authentication bounds to be enforced by the file exporter itself is a sizable amount of work, it is also desirable.

(g) Future work should include providing the means for administrators to control what minimum authentication levels are required for various DFS services that are now honored regardless of authentication state, e.g., the RPC service provided by the `repserver` process and some of the service provided by the `flserver` process. In addition, it should be an option to use and possibly require `rpc_c_protect_level_pkt_privacy` for some of these DFS services.

5. ACKNOWLEDGEMENTS

Thanks are due Dan Nydick (Transarc), Ted Anderson (Transarc), Bill Sommerfeld (HP) and Mike Burati (HP) for their involvement in resolving the initial design problems, in particular discovering a reasonable method of allowing authenticated RPCs to be made to a DFS file exporter without forcing the exporter to use that authentication in deciding file access questions. Rajesh Agarwalla (Transarc), Rich Salz (OSF), and Carl Burnett (IBM) helped further broaden and refine the scope of the problem and its solutions. The author particularly thanks Carl Burnett for contributing a draft of part of the code for the DFS Cache Manager. Dan Nydick earned still greater thanks both by carrying out extensive testing and by providing valuable suggestions for this document.

REFERENCES

[Brew95] ‘‘Basic Flaws in Internet Security and Commerce’’, Eric Brewer, Paul Gauthier, Ian Goldberg, and David Wagner, in *http://http.cs.berkeley.edu/~gauthier/endpoint-security.html*, 10 October 1995.

Also posted to cypherpunks@toad.com mailing list, 10 October 1995.

AUTHOR'S ADDRESS

Craig Everhart
Transarc Corporation
The Gulf Tower
707 Grant Street
Pittsburgh, PA 15219-1909
USA

Internet email: Craig_Everhart@transarc.com
Telephone: +1-412-338-4400