The following paper was originally published in the
Proceedings of the 2nd USENIX Windows NT Symposium
Seattle, Washington, August 3–4, 1998

Merging NT and UNIX Filesystem Permissions

Dave Hitz, Bridget Allison, Andrea Borr, Rob Hawley, and Mark Muhlestein
Network Appliance

For more information about USENIX Association contact:

1. Phone: 510 528-8649
2. FAX: 510 548-5738
3. Email: office@usenix.org
4. WWW URL: http://www.usenix.org/

Merging NT and UNIX Filesystem Permissions

Dave Hitz (hitz@netapp.com)
Bridget Allison (bridget@netapp.com)
Andrea Borr (aborr@netapp.com)
Rob Hawley (hawleyr@netapp.com)
Mark Muhlestein (mmm@netapp.com)

Network Appliance (www.netapp.com)

Abstract

Sharing network data between NT and UNIX systems is becoming increasingly important as NT moves into areas previously serviced entirely by UNIX. One difficulty in sharing data is that the two filesystem security models are quite different. NT file servers use access control lists (ACLs) that allow permissions to be specified for an arbitrary number of users and groups, while UNIX NFS servers use traditional UNIX permissions that provide control only for owner, group, and other. This paper describes an integrated security model in which a single filesystem can contain both files with NT-style ACLs and files with UNIX-style permissions. For native file service requests (NT requests to NT-style files and NFS requests to UNIX-style files) the security model exactly matches an NT or UNIX fileserver. For non-native requests, heuristics allow a reasonable level of access without compromising the security guarantees of the native model.

1 Introduction

Network Appliance file servers support the native file service protocols for both NT and UNIX (CIFS for NT and NFS for UNIX), but until now, the filer's underlying security model has been based on UNIX. This paper describes a new version of NetApp's system software that supports UNIX permissions as well as NT access control lists (ACLs).

See [Hitz95], [Watson96] and [Hitz94] for details on the NetApp filer and the WAFL filesystem that it uses. For the purposes of this paper, a brief summary will suffice.

NetApp filers are dedicated devices, or "appliances," that perform just one function. Just as a Cisco router is a dedicated device optimized entirely for routing, a NetApp filer is optimized entirely for file service. We believe that an appliance that performs just one function can be faster, simpler, and more reliable than a general-purpose computer performing that same function.

NetApp's WAFL filesystem is designed to accommodate multiple filesystem protocols. When accessed via NT, for instance, WAFL does case-insensitive name lookups, but for NFS it does case-sensitive lookups. The on-disk inode structure for each file includes both UNIX metadata, such as the UNIX-permissions, as well as NT metadata, such as the hidden and archive bits. Similarly, the on-disk directory format includes both a long file name and a DOS-style eight-dot-three file name.

Filers can be administered by either NT or UNIX system administrators. NT administrators can use familiar tools such as Server Manager and User Manager. For UNIX administrators, there is a command line interface with UNIX-like commands such as **ifconfig**, **nfsstat**, and **ping**. There is also a browser-based GUI, written in Java.

2 Design Overview

The new integrated security system was designed to meet three goals:

(1) Make NT/Win95 Users Happy.

Support an NT-centric security model based on ACLs. To an NT or Win95 client, this security model should behave exactly like an NTFS filesystem on an NT file server.

(2) Make UNIX Users Happy.

Support a UNIX-centric security model based on UNIX permissions. To an NFS client, this security model should behave exactly like a UNIX NFS server. (This is the only security model that NetApp filers have historically supported.)

(3) Let NT and UNIX Users Share Data.

Provide reasonable heuristics so that NT/Win95 users can access UNIX-style files, and UNIX users can access NT-style files.

To meet these goals, NetApp allows administrators to designate specific sections of the filesystem as an

NTFS-qtree or a UNIX-qtree. (A qtree is simply a designated subtree within the filesystem.) NT and UNIX users can safely access both types, but the NTFS-qtree seems more natural for NT users, and the UNIX-qtree for UNIX users.

Native requests – NT networking to an NTFS-qtree or NFS to a UNIX-qtree – work exactly as expected. UNIX-qtrees are modeled after Solaris, and NTFS-qtrees are modeled after NT. Non-native requests use heuristics designed to operate as intuitively as possible while still maintaining security.

Non-native NT requests are handled by mapping the NT user to an equivalent UNIX user, and then validating against the standard UNIX permissions. In the simplest case, John Smith might have the account “john” on both NT and UNIX. When John accesses a UNIX-qtree from NT, the filer looks up “john” in /etc/passwd (or over NIS), and uses the specified UID and GID for all access validation. A user-mapping file handles the case where John's NT account is “john”, but his UNIX account is “jsmith”. NT users with no UNIX account may be mapped to “nobody”, to a specified UNIX account, or they may simply be denied access.

NFS requests to NTFS-qtrees are validated using special UNIX permissions that are set whenever an ACL is updated. The UNIX permissions are guaranteed to be at least as restrictive as the ACL, which means that users can never circumvent ACL-based security by coming in through NFS. On the other hand, since UNIX permissions are less rich than NT ACLs, a multiprotocol user may be unable to access some files over NFS even though they are accessible via NT. In practice, NFS access to NTFS-qtrees works well for the owner, and for access granted to the NT “everyone” account, but not for other cases. (These restrictions will be removed in a future release, as described in section 7.)

The filer also supports a Mixed-qtree in which the security style is determined on a file-by-file basis. Files created by NT users get NT ACLs, and files created by UNIX users get UNIX permissions. A file's security style may be changed from one style to another by NFS “set attribute” or NT “set ACL” requests, assuming – of course – that the requestor has the appropriate permissions. This style is ideal for users who actively use both NT and UNIX and want access to both styles of security.

Several features allow special control over privileged users. The filer supports the NT “administrators” local group, which lists the NT accounts that have administrator privileges. The NT User Manager interface can be used to manage the “administrators” local group over the network. The user-mapping file can be used to map the NT “administrator” account into UNIX “root”, or to a non-privileged UNIX account such as “nobody”. Privileges for UNIX “root” are controlled using the /etc/exports “root=“ flag. Requests from “root” are mapped to “nobody” unless the “root=“ flag on an export explicitly allows root privileges.

3 Background

This section briefly describes UNIX and NT filesystem security, since many people are familiar with one or the other, but not both.

3.1 UNIX Filesystem Security

UNIX uses user IDs (UIDs) to identify users, and group IDs (GIDs) to identify groups. The permission bits themselves control read access (r), write access (w), and execute access (x). The full set of UNIX permission information stored with each file consists of:

- UID of the owner
- GID of the owner
- User perm bits (controls rwx for owner)
- Group perm bits (controls rwx for the group)
- Other perm bits (controls rwx for anyone else)

When performing validation, UNIX determines whether the request is from the file's owner, someone in the file's group, or anyone else, and then uses the appropriate permission bits.

See [McKusick84] or [Bach86] for more details.

3.2 NT Filesystem Security

NT uses security IDs (SIDs) to identify both users and groups. The NT permissions for each file consist of:

- SID of the owner
- SID of the owner's group
- ACL (Access Control List) for the file

The ACL contains one or more access control entries (ACEs). Each ACE contains a SID, indicating the user or group to which the ACE applies, and a set of permission bits. NT permission bits include the three UNIX bits – read, write, and execute – as well as “change permissions” (P), “take ownership” (O), “delete” (D), and others. An ACE can either grant or deny the specified permissions. One ACE might grant read and write permission to the engineering group, but another ACE might specifically deny write permission to John Smith. Even if John is in the engineering group, he will be denied write access.

NFS and NT also differ in how they authenticate users. NFS is a connectionless protocol, and each NFS request includes the UID and GIDs of the user making the request. The UNIX client determines the UIDs and GIDs when the user first logs in, by looking at the files /etc/passwd and /etc/groups. NT networking is session based, so the identity of the user can be determined just once, when the session is first set up. At session connect time, the client sends the user's login name and encrypted password (actually the challenge and the client's response) to the file server, and the server determines the session's user SID and group SIDs. Servers commonly forward the name and password to an NT domain controller (DC) and let the DC perform authentication.

See [Reichel93] for more details.

4 Philosophy

4.1 Surprise and Insecurity

Given that NT and UNIX have fundamentally different models of filesystem security, it is impossible to design an integrated model that performs exactly as expected for all users. NT ACLs provide a richer security model than UNIX permissions. Many NT ACLs cannot be accurately reflected to a UNIX user. Yet, when a UNIX user lists a directory, NFS must return something for the UNIX permissions.

We conclude that any integrated filesystem security model must present users with some combination of surprise and/or insecurity. If we validate UNIX requests directly against the NT ACL, then the UNIX user may be surprised to see behavior that's different than what the faked-up UNIX permissions would seem to imply. But if we validate UNIX requests against faked-up UNIX permissions, then this may result in insecurity if the UNIX permissions grant more access than the NT ACL, or surprise if the UNIX permissions deny access where the NT ACL would have granted it.

When a filesystem accepts a request to set security (NT or UNIX) on a file, it has – in essence – made a promise to the user. Violating this promise is little different than losing data that a user thought was safely written. Hence, NetApp's implementation allows no insecurity, and it minimizes surprise as much as possible given this constraint.

Some users may dislike surprise more than they dislike insecurity. In a mixed NT and UNIX development environment, making the development tools work is probably the most important goal. Sacrificing security may be acceptable. (At NetApp, the software developers all have privileged access anyway.) Perhaps someday we will add options to control the balance between surprise and insecurity, but for the first release, erring towards safety seemed best.

In examining NetApp's integrated security model, it is important to remember that no perfect solution is possible. We must be willing to make trade-offs between various types of surprise, and – for sites willing to allow it – perhaps even between surprise and insecurity.

4.2 Which to Map: Users or Permissions?

As described in the Design Overview, NetApp handles non-native NT requests by mapping the NT user into an equivalent UNIX user, and validating requests directly against the UNIX permissions. We call this *user mapping*.

On the other hand, NetApp handles non-native NFS requests by using faked-up UNIX permissions that are set whenever an NT ACL is updated. We call this *permission mapping*.

We believe that user mapping reflects the intended security more accurately than permission mapping. The security models of NT and UNIX are so different that permission mapping can never be completely accurate. However, the two operating systems have very similar definitions of a user, so user mapping is straightforward.

We use permission mapping for non-native NFS requests simply because it is easier and cheaper. Permissions need be mapped only when an ACL is set or updated, which is rare. User mapping would need to be performed for every single NFS request. Unlike NFS, which is stateless, NT's CIFS protocol is session based, so user mapping need be done only once, when the session is established, rather than for each separate request.

Although we didn't implement it in our first ACL release, we now believe that with appropriate caching, it should be possible to do UNIX to NT user mapping efficiently. Section 7 describes our plans.

4.3 Issues for Non-Native Security

As described above, a native request is an NT request to an NTFS-qtree, or an NFS request to a UNIX-qtree. A non-native request is the reverse: NT to UNIX-qtree or NFS to NTFS-qtree.

This section examines the issues that are important for non-native requests, and it provides the outline for sections 5 and 6 below, which discuss how non-native NT and non-native NFS requests are handled.

The primary function of any filesystem security model is to validate requests – to accept them or deny them based on the authenticated user and the permissions for the file. Thus, validating non-native requests is one important topic, and is covered in 5.1 and 6.1.

In addition, there are several file system actions that require special attention. In particular, we must ask:

- How are requests to display permissions handled?
- How are requests to set permissions handled?
- How are permissions set for newly created files?

These are covered in 5.2 and 6.2.

5 Handling Non-Native NT Requests

This section describes how NetApp filers handle NT requests to UNIX-style files. Remember, files with UNIX permissions occur both in UNIX-qtrees, where all files are UNIX-style, and in Mixed-qtrees, which have both UNIX-style and NT-style files.

Section 5.1 discusses how non-native NFS requests are validated, and section 5.2 discusses non-native handling for displaying permissions, setting permissions, and creating new files.

5.1 Validation

NetApp filers validate NT requests to UNIX-style files by generating a mapped UID (and GIDs) for each NT networking session, and then using the UID (and GIDs) to check against the UNIX permissions.

Suppose that the NT user “john” connects to a filer. Here are the steps that the filer takes to determine the mapped UID and GIDs for “john”.

(1) The filer sends a request to the NT domain controller (DC), to authenticate “john”, and to find the NT SID for “john”.

(2) The filer looks in the user-mapping file to determine whether the NT account “john” maps into a different account name under UNIX. In this case, let's assume that “john” maps into the UNIX account “jsmith”.

(3) The filer looks up “jsmith” in /etc/passwd (possibly via NIS) to determine the UNIX UID and primary GID for John.

(4) The filer uses /etc/groups (possibly via NIS) to determine the UNIX GIDs for John.

These steps provide each NT networking session with a full set of UNIX authentication information, which allows the filer to easily validate most requests against the UNIX permissions.

Some NT operations don't map well to UNIX operations, so they must be handled specially:

- **Set ACL**

  The NT “set ACL” operation (similar to UNIX chmod) is always denied in UNIX-qtrees. In Mixed-qtrees, a “set ACL” operation is only allowed by the owner – that is the mapped UID for the NT session must match the file's UID. This operation converts the file from UNIX-style permissions to NT-style permissions.

  Only the owner can set an ACL because in UNIX only the owner is allowed to set attributes. (Remember, we are discussing requests to UNIX-style files.)

- **Take Ownership**

  The NT “take ownership” operation (similar to UNIX chown) is always denied in UNIX-qtrees. In Mixed-qtrees, only the file's owner can “take ownership” of a UNIX-style file. Like the “set ACL” request, this converts the file to NT-style permissions.

5.2 Request Processing

This section considers non-native NT requests in light of the three questions listed above, in Section 4.3, Issues for Non-Native Security:

- How are requests to display permissions handled?
- How are requests to set permissions handled?
- How are permissions set for newly created files?

5.2.1 Displaying Permissions

For non-native NT requests to display permissions, WAFL dynamically builds an ACL designed to represent the UNIX permission as well as possible.

On might hope to build an NT ACL that perfectly represents the UNIX permission like this:

- Owner – map the file's UID into an NT SID
- Group – map the file's GID into an NT SID
- ACE for owner SID – based on UNIX user perms
- ACE for group SID – based on UNIX group perms
- ACE for special NT *everyone* SID– based on UNIX other perms

Unfortunately, we currently have no way to map UIDs or GIDs into SIDs, so this approach isn’t possible. Instead, we construct an ACL using only well known NT SIDs and the SID for the NT networking session itself. These are sufficient to let us construct an ACL that, while not perfect, does provide useful information.

Each faked-up ACL contains two ACEs (access control entries):

- ACE for NT “everyone” SID – based on the UNIX other permissions.
- ACE for the SID of the NT networking session – based on whichever UNIX permission is appropriate. If the mapped UID for the session is the file's owner, the ACE is based on the UNIX owner perms. If the group matches, then it's based on the group perms. Otherwise it's based on the other perms.

Note that this faked-up ACL always contains an entry for the user making the request, so users can always determine their own access rights.

If the NT session owns the file, then in the faked-up ACL the session's SID is shown as the owner. If not, then the well known NT SID “CREATOR_OWNER” is shown as the owner.

5.2.2 Setting Permissions

In UNIX-qtrees, NT requests to set permissions are always denied.

Outside of UNIX-qtrees, non-native requests to set permissions are allowed only by the file's owner. If allowed, the specified ACL takes effect just as it would have if the file had already been an NT-style file. After the “set ACL” request is processed, the file becomes an NT-style file.

5.2.3 Setting Permissions on Create

Unlike UNIX, which passes the permissions for a new file as part of the create request, NT expects permissions to be inherited from the parent directory.

The filer handles NT create requests in UNIX-qtrees as follows:

- The file's owner is set to the mapped UID for the NT networking session.
- The file's group is set to the mapped GID for the NT session, or inherited from the parent directory if the directory’s SGID bit is set.
- The UNIX permission bits are inherited from the parent directory, except that SUID and SGID bits are cleared for non-directory creates.

In Mixed-qtrees, the newly created file inherits NT ACLs if the parent is an NT-style directory, but it inherits UNIX permissions, as described above, if the parent is a UNIX-style directory.

6 Handling Non-Native NFS Requests

This section describes how NetApp filers handle NFS requests to NT-style files.

Section 6.1 discusses how non-native NFS requests are validated, and section 6.2 discusses non-native handling for displaying permissions, setting permissions, and creating new files.

6.1 Validation

Whenever an NT ACL is set or changed, WAFL calculates a corresponding set of UNIX permissions. As a result, very little special processing is required to validate NFS requests to NT-style files. Simply doing the normal checks against the UNIX permissions usually does the right thing. The rest of this section describes how the UNIX permissions are constructed from the ACL, and explains a few special exceptions.

Converting an NT ACL into UNIX permissions is surprisingly tricky. This section gives a brief overview, but an observant user may encounter slight differences in the actual implementation.

- The file's UID is set to the mapped UID for the NT session.

  (Remember that the faked-up UNIX permissions are generated right when the ACL is set, so it makes sense to set the owner of a newly created file to the mapped UID for the NT session.)
- The file's GID is set to the mapped GID for the NT session.
- The UNIX user perm is set based on the access rights that the ACL grants to the NT session creating the file.
- The UNIX other perm is set based on the access rights granted to the NT “everyone” account. (If the ACL contains any denies, then the denied permissions are subtracted from the other perms.)
- The UNIX group perm is set equal to the other perm.

This design avoids security holes by ensuring that the UNIX permission is always at least as restrictive as the NT ACL. Unfortunately, UNIX permissions cannot represent the full richness of the NT security model. As a result, a file that a user can reach via NT may not be accessible via NFS.

Because NT supports some specific permissions that UNIX lacks, it is not possible to rely entirely on the UNIX permissions to validate some NFS requests:

- **REMOVE/RMDIR**

  Only the owner of a file is allowed to delete it. This is necessary to avoid violating the NT "delete child" permission.

- **CREATE**

  Only the owner of a directory can create anything in it. This is required in order for NT ACL inheritance to work properly. (This is explained more fully below, in section 6.2.3.)

Both of these restrictions will be removed by the future enhancements described in section 7.

6.2 Request Processing

This section considers non-native NFS requests in light of the three questions listed above, in Section 4.3, Issues for Non-Native Security:

- How are requests to display permissions handled?
- How are requests to set permissions handled?
- How are permissions set for newly created files?

6.2.1 Displaying Permissions

As described above, in section 6.1, every file with an NT ACL also has a set of UNIX permissions stored with it. To handle an NFS "get attributes" request, the filer simply returns those stored permissions.

6.2.2 Setting Permissions

In NTFS-qtrees, NFS requests to set permissions are always denied.

In Mixed-qtrees, only a file's owner is allowed to set permissions. When UNIX permissions are set on a file, the NT ACL is deleted – the file changes from NT-style to UNIX-style.

Note that "set attribute" requests that update non-security information such as access time or modify time are allowed even in NTFS-qtrees, and they do not delete the NT ACL.

6.2.3 Setting Permissions on Create

NFS creates in NTFS-qtrees are only allowed by the directory's owner. This is because an NT SID is required to handle NT ACL inheritance. An NFS request has no SID, but for a create request from a directory's owner, WAFL can use the owning SID from the directory's ACL and handle ACL inheritance according to normal NT rules.

In Mixed-qtrees, NFS create requests are handled according to normal UNIX rules.

7 UNIX to NT User Mapping

The first ACL release handles non-native NFS requests using permission mapping rather than user mapping, because of the cost and complexity of doing user mapping on every single NFS request.

However, we believe that with appropriate caching, the cost of UNIX to NT user mapping can be reduced to an acceptable level. User mapping requires the following steps:

- When an NFS request arrives, it contains a UID. The filer uses /etc/passwd (or NIS) to convert the UID into a UNIX username.
- The filer converts the UNIX username into an NT username using a mapping file. (If no mapping is specified, the filer uses the UNIX username.)
- The filer contacts the NT domain controller (DC) to determine the SID for the NT username. If there is no account for the name, the filer uses a default SID (set to "guest" by default).
- The filer contacts the DC to get the SIDs of all groups to which the user belongs.

With these mapping rules, the filer has a full set of NT authentication information, which allows it to validate NFS requests based on the NT ACL.

With NFS, there is no concept of a session, so the mapping must be done for each request. The steps above are too time consuming to perform on a per-request basis, so WAFL must cache the mappings. NT networking sessions may last for days or weeks, so it should be safe to cache UID-to-SID mappings for at least a few hours.

8 Other Issues

8.1 FAT versus NTFS

NT servers support both FAT filesystems and NTFS filesystems. FAT is the traditional DOS filesystem – it has no file-level security at all. The NTFS filesystem was designed for NT and supports NT ACLs.

Since NTFS-qtrees and Mixed-qtrees both support ACLs, they must be advertised to NT networking clients as "NTFS". (If they were advertised as "FAT", clients would assume that they had no ACLs, and would disable the interfaces for controlling ACLs.)

It is less obvious how to advertise UNIX-qtrees. One can make a case either way, as these two conflicting arguments show:

- Advertise as “FAT”

  UNIX-qtrees don't support ACLs, so advertising them as FAT sends a clear message to clients not to use ACLs. Advertising as NTFS would be confusing, since no ACLs are really present and any request to set ACLs will fail.

- Advertise as “NTFS”

  UNIX-qtrees support file level security, and advertising them as NTFS allows the filer to display the UNIX permissions using faked-up ACLs. Advertising as FAT would be confusing, because it would seem to imply that no file-level security is present.

In the end, we decided to advertise UNIX-qtrees as FAT, because this seems least likely to confuse Windows programs that absolutely must have ACLs.

Still, there are several situations in which it is useful to construct a fake ACL for an NT-style file, as described above in 5.2.1:

(1) In Mixed-qtrees

  Mixed-qtrees contain both UNIX-style and NT-style files. To support ACLs they must be advertised as “NTFS”, yet not all files in them contain ACLs.

(2) In NTFS-qtrees that originated as UNIX or Mixed-qtrees

  A qtree's security style can be changed at any time, so a qtree that began as UNIX-qtree may later be converted to NTFS. In this case, it will clearly be advertised as “NTFS”, but it may contain files without ACLs.

(3) In UNIX-qtrees accessed via an NTFS or Mixed Share.

  The root of a filesystem may have NTFS or Mixed security, but it may contain a UNIX-qtree. In this case, the C$ (or root) share will be advertised as “NTFS”, but a user can go down into the UNIX-qtree and then try to display an ACL.

8.2 Migration

For sites using old filers, migration to the NTFS security model is an important issue.

System administrators can update the security model for any qtree (including the root of the filesystem), using the **qtree** command. The syntax is:

qtree security *qtree* [**unix**|**ntfs**|**mixed**]

When a UNIX-qtree is converted to an NTFS-qtree, shares are advertised as “NTFS” instead of “FAT”. The files themselves are not converted to NT-style files, so they behave as described in section 5, Handling Non-Native NT Requests. Of course, the owner of a file can set an ACL, converting the file to NT-style. Also, NetApp will ship a Windows utility to run through a tree and set a real ACL on each file based on its UNIX permissions. This is useful since the faked-up ACL doesn’t show the exact permissions for a file.

When an NTFS-qtree is converted to a UNIX-qtree, shares are advertised as “FAT” instead of “NTFS”, and any ACLs in the qtree are simply ignored. ACLs are not actually deleted – however – so if the qtree is converted back to NTFS, the ACLs will still be present. The best way to delete the ACLs is to write a script that runs as root and chowns each file to its existing owner. (Remember that doing a chown or chmod deletes the ACL on a file.)

8.3 Group Mapping

With user mapping, it shouldn’t be necessary to map between NT and UNIX groups. When an NT user is mapped into a UNIX user, the filer also identifies the UNIX groups for that user, so access based on group rights will work correctly.

Unfortunately, this approach doesn't work for a user that isn't successfully mapped. Consider an NT user named “nt-john” who is a member of the group “nt-engineering”. If “nt-john” successfully maps to the UNIX account “unix-john”, then he'll get all group membership associated with “unix-john”, presumably including “unix-engineering”. On the other hand, if “nt-john” doesn't map to “unix-john”, then he'll simply become UNIX “nobody”, with no special group rights at all.

Thus, it might seem useful to explicitly map “nt-engineering” to “unix-engineering” so that group level access would be permitted even if user mapping fails.

On the other hand, maintaining a group-mapping file seems at least as hard as maintaining a user-mapping file. If NT and UNIX administration are sufficiently coordinated to map groups, why not just map users instead? It seems simpler to support just one kind of mapping rather than two.

8.4 Share Level ACLs

Share level ACLs are now based on NT SIDs, and they can be edited over the network using the NT Server Manager.

For backward compatibility, share level ACLs based on UNIX user names will continue to function, although they cannot be controlled via Server Manager.

8.5 POSIX ACLs

Although POSIX ACLs are not currently a requirement, we wanted to ensure that our design for NT ACLs did not preclude support for POSIX ACLs later. In fact, our integrated security model could easily be enhanced to allow UNIX-style files to have either POSIX ACLs or traditional UNIX permissions. Since there are subtle semantic differences between NT ACLs and POSIX ACLs, we would maintain the distinction between NT-style files and UNIX-style files, and we would continue to use user mapping to handle non-native requests.

POSIX ACLs would allow – but not require – some additional enhancements. For instance, a faked-up POSIX ACL could reflect the NT ACL more accurately than faked up UNIX permissions can.

9 Implementation

Earlier sections describe how WAFL uses NT ACLs when processing requests. This section focuses on two additional implementation issues. Section 9.1 describes how WAFL stores the ACL on disk, and section 9.2 describes the NT administrative protocols that the filer must support in order to correctly handle NT ACLs.

9.1 Storing NT ACLs

UNIX permissions are easy to store, because they have a small, fixed size. NT ACLs are more difficult to store, because they have a variable size, depending on the number of ACEs (Access Control Entries) they contain, and they can get quite large. NT currently restricts ACLs to 64KB, but that limit is arbitrary and could easily grow in the future.

WAFL's on-disk format uses 128-byte inodes to describe files, much like the Berkeley Fast Filesystem [Hitz94, McKusick84]. WAFL stores UNIX permissions in the inode itself, but NT ACLs obviously don't fit. Instead, files with NT ACLs have a pointer to a second inode, called a xinode (extended inode), that contains the ACL data. In essence, the ACL is being stored as a special hidden file.

To reduce storage overhead, WAFL shares xinodes whenever possible. If two files have exactly the same ACL, then WAFL points both files at the same xinode. The link count in the xinode tracks the number of references, just as it does for a regular file, and the xinode is deleted only when its link count drops to zero.

This technique produces incredible storage savings because large numbers of files have the same ACL. This makes sense if you consider how ACLs are set. At create time, files inherit the ACL from their parent directory, which means that the ACL will match an already existing one. And when ACLs are set manually, they are commonly applied to an entire subtree at once, so – again – a large number of files share the same ACL.

9.2 NT Administrative Protocols

To handle NT ACLs correctly, the NetApp filer must support a surprising number of NT administrative protocols. To understand why, consider what happens when an NT user pops up the ACL editor on a remote file.

First, the ACL editor contacts the filer server and requests the ACL. In order to display the ACL, the ACL editor must convert the SIDs in the ACL into human readable user names. One might expect the editor to contact the NT Domain Control (DC) directly to perform this conversion, but it does not. Instead it sends conversion requests to the file server. At first this seems surprising, but it makes sense when you consider that the client may be in a different NT domain than the file server it is talking to. In addition, an NT file server may define local users that are not known to the domain controller.

Editing an ACL generates even more requests. When creating a new ACL entry, the ACL editor must display a list of all possible users and groups, and – again – instead of contacting the DC directly, it requests this information from the file server.

Thus, to support ACLs, the NetApp filer must support a wide variety of NT administrative protocols, both as a server, in order to receive the appropriate requests, but also as a client so that it can forward requests on to the DC. In addition, in order to convince clients to talk with it, the filer must advertise itself as a full-fledged NT server, which requires it to speak even more NT administrative protocols.

The end result is that the NetApp filer supports many of the administrative interfaces that NT administrators expect in an NT file server. The filer is visible in the network neighborhood, and it can be managed using Server Manager and User Manager.

10 Bibliography

[Bach86] Maurice J. Bach. *The Design of the UNIX Operating System.* Prentice-Hall, 1986

[Hitz94] Dave Hitz, James Lau, and Michael Malcolm, “File System Design for an NFS File Server Appliance.” Winter USENIX Conference Proceedings, USENIX Association, Berkeley, CA, January, 1994.

[Hitz95] Dave Hitz. “An NFS File Server Appliance (TR-3001).” Network Appliance, Mountain View, California, March 1995.

[McKusick84] Marshall K. McKusick. “A Fast File System for UNIX.” ACM Transactions on Computer Systems 2(3): 181-97, August 1984.

[Reichel93] Rob Reichel. “Inside Windows NT Security.” Windows/DOS Developer’s Journal, April 1993.

[Watson96] Andy Watson. “Multiprotocol Data Access: NFS, CIFS, and HTTP (TR-3014).” Network Appliance, Mountain View, California, December 1996.