Understanding NFSv4 ACL’s

John Hixson
john@ixsystems.com
iXsystems, Inc.

1 Introduction

Traditional UNIX permissions are very limited in the security they can provide. UNIX permissions can only be set for the owner, group and everyone else (other). NFSv4 access control lists can give a lot more flexibility in the security and access that can be provided. In the NFSv4 security model, owner, group, specific users, specific groups, everyone, etc can be configured. Complete control over the type of access you want is possible. This paper assumes you are running FreeBSD or FreeNAS. The object of this paper is to demonstrate how NFSv4 access control lists work by example with some explanation. It is assumed the reader is familiar with traditional UNIX permissions.

2 Anatomy of an ACE

Access Control Lists (ACL’s) consist of one or more Access Control Entries (ACE’s). Each ACE is made up of four or five fields delimited by colons. The first field is the ACL tag. If the ACL tag is a user or group, then the second field has to be an ACL qualifier which specifies the user or group. If the ACL tag is not a user or group, then the second field is the access permissions that describe the ACE access. The field after the access permissions is the inheritance flags that describe if and how the ACE will be inherited. That last field is the type of ACE access that describes if this entry is to allow or deny the specified access. In the following sections I will provide the name of the field type, a brief description of it, and a breakdown of it with comments.

1. Principal

As previously mentioned, the principal consists of one or two parts: an ACL tag and an optional ACL qualifier. If the ACL tag is “u”, “user”, “g” or “group”, then a qualifier must be provided. The “u” and “user” tag require a user for the qualifier, while the “g” or “group” tag require a group for the qualifier.

- ACL tag
  - owner@
    - Access permissions apply to owner of the file
  - group@
    - Access permissions apply to the group owner of the file
  - everyone@
    - Access permissions apply to everyone (including the owner and group)

  - u or user
    - Access permissions apply to user specified in the qualifier
  - g or group
    - Access permissions apply to the group specified in the qualifer

- ACL qualifier
  - username

- groupname

2. Access Permissions

Access permissions describe the entry access. The first three permissions are the same as the traditional UNIX read, write and execute permissions. The rest are more granular.

- r read_data
  - Read access on a file
- w write_data
  - Write access on a file
- x execute
  - Execute access on a file
  - Search access on a directory
- p append_data
  - Append access on a file
- D delete_child
  - Permission to delete a file or directory within a directory
- d delete
  - Permission to delete a file
- a read_attributes
  - Read attribute (stat) access on a file or directory
- A write_attributes
  - Write attribute (stat) access on a file or directory
- R read_xattr
  - Read extended attributes on a file or directory (Not implemented)
- W write_xattr
  - Write extended attributes on a file or directory (Not implemented)
- c read_acl
  - Read ACL access on a file or directory
- C write_acl
  - Write ACL access on a file or directory
- o write_owner
  - Permission to change file or directory owner or group
- s synchronize
  - Not implemented

3. Inheritance Flags

Inheritance flags describe the way that the ACE will be inherited (or not inherited).

- f file_inherit
  - Only inherit the ACL from the parent directory to the directory's files
- d dir_inherit
  - Only inherit the ACL from the parent directory to the directory's subdirectories
- i inherit_only
  - Inherit the ACL from the parent directory but only applies to newly created files and/or subdirectories and not the directory itself. file_inherit and/or dir_inherit need to be set.
- n no_propogate
  - Only inherit the ACL to the fist level file or subdirectories. file_inherit and/or dir_inherit need to be set.

- I inherited
    - Indicates an inherited ACE

4. Type

The ACE type describes if this ACE allows or denies the access permissions and inheritance flags.

- allow
    - Allow the ACE
- deny
    - Deny the ACE

3 Access permission examples

ACL's are best demonstrated using examples. Here I will demonstrate how they work and explain the individual parts. These examples assume that the aclmode and acltype on the dataset are set to "passthrough" or "restricted". I will explain the differences later.

Directory creation with different umask values:

```
# Set the umask prior to creating each directory to demonstrate the affect on the ACL

john@x1 /usr/home/john$ umask 022
john@x1 /usr/home/john$ mkdir example022
john@x1 /usr/home/john$ umask 002
john@x1 /usr/home/john$ mkdir example002
john@x1 /usr/home/john$ umask 000
john@x1 /usr/home/john$ mkdir example000

# The UNIX permissions are as expected (755)
john@x1 /usr/home/john$ ls -ld example022
drwxr-xr-x  2 john  john  2 Jan 13 13:54 example022/

# Now, look at the ACL. The UNIX permissions are reflected in it as well as all the other
# access permissions. owner@ has rwx, group@ has rx and everyone@ has rx.
john@x1 /usr/home/john$ getfacl -q example022
            owner@:rwxp--aARWcCos:-------:allow
            group@:r-x---a-R-c--s:-------:allow
         everyone@:r-x---a-R-c--s:-------:allow

# The UNIX permissions are as expected (775)
john@x1 /usr/home/john$ ls -ld example002
drwxrwxr-x  2 john  john  2 Jan 13 13:54 example002/

# With the changed umask value, the owner@ still has rwx, but group@ has rwx as well.
john@x1 /usr/home/john$ getfacl -q example002
            owner@:rwxp--aARWcCos:-------:allow
            group@:rwxp--a-R-c--s:-------:allow
         everyone@:r-x---a-R-c--s:-------:allow

# The UNIX permissions are as expected (777)
```

```
john@x1 /usr/home/john$ ls -ld example000
drwxrwxrwx  2 john  john  2 Jan 13 13:54 example000/

# With the 000 umask, owner@, group@ and everyone@ has rwx
john@x1 /usr/home/john$ getfacl -q example000
            owner@:rwxp--aARWcCos:-------:allow
            group@:rwxp--a-R-c--s:-------:allow
         everyone@:rwxp—a-R-c--s:-------:allow
```

File creation with different umask values would look the same so I won't show it here. So, what do all of those access permissions even mean? Let's see what they do! I will assume the reader understands standard UNIX permissions therefore I will not cover those here. Let's start with "p" (append_data). On FreeBSD, this flag by itself is ignored for files. In order to append to a file, the "w" (write) permission must be set. However, for directories, this permission must be set in order to create subdirectories. Here is an example:

```
# The "p" permission is not set, however "w" is.
john@x1 /usr/home/john/example002$ getfacl -q .
            owner@:rwx---aARWcCos:-------:allow
            group@:r-----a-R-c--s:-------:allow
         everyone@:r-----a-R-c--s:-------:allow

# Try to create a directory
john@x1 /usr/home/john/example002$ mkdir foo
mkdir: foo: Permission denied

# Add the append_data permission
john@x1 /usr/home/john/example002$ setfacl -m owner@:rwxpaARWcCos::allow .

# See the "p" exists
john@x1 /usr/home/john/example002$ getfacl -q .
            owner@:rwxp--aARWcCos:-------:allow
            group@:r-----a-R-c--s:-------:allow
         everyone@:r-----a-R-c--s:-------:allow

# Try to create directory again
john@x1 /usr/home/john/example002$ mkdir foo
john@x1 /usr/home/john/example002$ ls
foo/

# Success!
```

Next is the "D" (delete_child) permission. This permission controls the ability to delete subdirectories.

```
# Using the same example directory, set a deny ACE for delete_child
john@x1 /usr/home/john/example002$ setfacl -a 3 'everyone@:D::deny' .
john@x1 /usr/home/john/example002$ getfacl -q .
            owner@:rwxp--aARWcCos:-------:allow
            group@:r-----a-R-c--s:-------:allow
         everyone@:r-----a-R-c--s:-------:allow
         everyone@:----D---------:-------:deny
```

```
# See that “foo” still exists
john@x1 /usr/home/john/example002$ ls
foo/

# Try and delete it
john@x1 /usr/home/john/example002$ rmdir foo
rmdir: foo: Operation not permitted

# Remove delete_child ACE entry
john@x1 /usr/home/john/example002$ setfacl -x 3 .
john@x1 /usr/home/john/example002$ getfacl -q .
           owner@:rwxp--aARWcCos:-------:allow
           group@:r-----a-R-c--s:-------:allow
        everyone@:r-----a-R-c--s:-------:allow

# Now try and delete subdirectory again
john@x1 /usr/home/john/example002$ rmdir foo
john@x1 /usr/home/john/example002$

# Success!
```

Now the “d” (delete) permission. This controls the ability to delete a file.

```
# Create an empty file and view it’s ACL
john@x1 /usr/home/john/example002$ touch foo.txt
john@x1 /usr/home/john/example002$ getfacl -q foo.txt
           owner@:rw-p--aARWcCos:-------:allow
           group@:rw-p--a-R-c--s:-------:allow
        everyone@:rw-p—a-R-c--s:-------:allow

# Remove write ability (This allows delete)
john@x1 /usr/home/john/example002$ chmod 400 foo.txt
john@x1 /usr/home/john/example002$ getfacl -q foo.txt
           owner@:r-----aARWcCos:-------:allow
           group@:------a-R-c--s:-------:allow
        everyone@:------a-R-c--s:-------:allow

# Create deny ACE for delete and view ACL
john@x1 /usr/home/john/example002$ setfacl -a 3 everyone@:d::deny foo.txt
john@x1 /usr/home/john/example002$ getfacl -q foo.txt
           owner@:r-----aARWcCos:-------:allow
           group@:------a-R-c--s:-------:allow
        everyone@:------a-R-c--s:-------:allow
        everyone@:-----d--------:-------:deny

# Try and delete the file
john@x1 /usr/home/john/example002$ rm foo.txt
override r--------  john/john uarch for foo.txt?

# Remove deny ACE for delete
john@x1 /usr/home/john/example002$ setfacl -x 3 foo.txt
```

```
# Try and delete the file again
john@x1 /usr/home/john/example002$ rm foo.txt
john@x1 /usr/home/john/example002$

# Success!
```

Now for “a” (read_attributes). This controls the ability to read file attributes, for instance, information in the stat structure.

```
# View ACL on directory
[root@x1] ~# getfacl -q example/
    owner@:rwxp--aARWcCos:-------:allow
    group@:r-x-----R-c--s:-------:allow
 everyone@:r-x-----R-c--s:-------:allow

# Create deny ACE for read_attributes and view ACL (as root)
[root@x1] ~# setfacl -a 3 everyone@:a::deny example
[root@x1] ~# getfacl -q example
    owner@:rwxp--aARWcCos:-------:allow
    group@:r-x-----R-c--s:-------:allow
 everyone@:r-x-----R-c--s:-------:allow
 everyone@:------a-------:-------:deny

# Try and read attributes as non-root user
john@x1 /root$ getfacl -q example
getfacl: example: stat() failed: Permission denied

# Remove deny ACE and view ACL
[root@x1] ~# setfacl -x 3 example
[root@x1] ~# getfacl -q example
    owner@:rwxp--aARWcCos:-------:allow
    group@:r-x-----R-c--s:-------:allow
 everyone@:r-x-----R-c--s:-------:allow

# Try and read attributes again as non-root user
john@x1 /root$ getfacl -q example
    owner@:rwxp--aARWcCos:-------:allow
    group@:r-x-----R-c--s:-------:allow
 everyone@:r-x-----R-c--s:-------:allow

# Success!
```

The “W” (write_attributes) permission controls access to the same attributes as the read_attributes permission. The easiest attributes to modify are the various timestamps set on a file. We won’t cover this here. The “R” (read_xattr) and “W” (write_xattr) permissions are not used by FreeBSD so they will not be covered as well.

The next permission is “c” (read_acl). This controls the ability to read the ACL on a file or directory.

```
# View ACL that has read_acl permission allowed as non-root user
```

```
john@x1 /root$ getfacl -q example
          owner@:rwxp--aARWcCos:-------:allow
          group@:r-x-----R----s:-------:allow
       everyone@:r-x-----R-c--s:-------:allow

# Remove read_acl permission as root and view ACL
[root@x1] ~# setfacl -m everyone@:rxRs::allow example
[root@x1] ~# getfacl -q example
          owner@:rwxp--aARWcCos:-------:allow
          group@:r-x-----R----s:-------:allow
       everyone@:r-x-----R----s:-------:allow

# Try and read ACL as non-root user
john@x1 /root$ getfacl -q example
getfacl: example: Permission denied

# Add read_acl permission back and view ACL
[root@x1] ~# setfacl -m everyone@:rxRcs::allow example
[root@x1] ~# getfacl -q example
          owner@:rwxp--aARWcCos:-------:allow
          group@:r-x-----R----s:-------:allow
       everyone@:r-x-----R-c--s:-------:allow

# Try and read ACL as non-root user again
john@x1 /root$ getfacl -q example
          owner@:rwxp--aARWcCos:-------:allow
          group@:r-x-----R----s:-------:allow
       everyone@:r-x-----R-c--s:-------:allow

# Success!
```

“C” (write_owner) controls the ability to write or modify a file or directory ACL.

```
# View ACL
[root@x1] ~# getfacl -q example
          owner@:rwxp--aARWcCos:-------:allow
          group@:r-x-----R----s:-------:allow
       everyone@:r-x-----R-c--s:-------:allow

# Try and change permissions with chmod as non-root user
john@x1 /root$ chmod 644 example
chmod: example: Operation not permitted

# Update ACE to allow ACL to be written
[root@x1] ~# setfacl -m everyone@:rxRCacs::allow example
[root@x1] ~# getfacl -q example
          owner@:rw-p--aARWcCos:-------:allow
          group@:r-----a-R-c--s:-------:allow
       everyone@:r-x---a-R-cC-s:-------:allow

# Try and change permissions with chmod again as non-root user
```

```
john@x1 /root$ chmod 644 example
john@x1 /root$

# Success!
```

The last of the access permissions to cover is “o” (write_owner). “s” synchronize is not implemented on FreeBSD so we will skip it. The write_owner permissions controls the ability to change user and group ownership on a file or directory.

```
# View ACL
[root@x1] ~# getfacl -q example
           owner@:rw-p--aARWcCos:-------:allow
           group@:r-----a-R-c--s:-------:allow
        everyone@:r-x---a-R-cC-s:-------:allow

# Try and change ownership to non-root user john
john@x1 /root$ chown john example
chown: example: Operation not permitted

# Update ACE to allow write_owner
[root@x1] ~# setfacl -m everyone@:rxaRcCos::allow example
[root@x1] ~# getfacl -q example
           owner@:rw-p--aARWcCos:-------:allow
           group@:r-----a-R-c--s:-------:allow
        everyone@:r-x---a-R-cCos:-------:allow

# Try and change ownership to non-root user john as john again
john@x1 /root$ chown john example
john@x1 /root$

# Success!
```

4 Inheritance flag examples

Inheritance flags determine how an ACL is inherited to files and directories. Inheritance flags may only be set on directories. The single exception to this rule is the “I” flag which indicates that the file or directory inherited the ACL.

Let’s first look at “f” (file_inherit):

```
# Example directory will full_set without file_inherit set
john@x1 /usr/home/john/example$ getfacl -q .
           owner@:rwxpDdaARWcCos:-------:allow
           group@:rwxpDdaARWcCos:-------:allow
        everyone@:rwxpDdaARWcCos:-------:allow

# Create a file and view its ACL
john@x1 /usr/home/john/example$ touch foo.txt
john@x1 /usr/home/john/example$ getfacl -q foo.txt
           owner@:rw-p--aARWcCos:-------:allow
           group@:rw-p--a-R-c--s:-------:allow
```

```
       everyone@:rw-p—a-R-c--s:-------:allow

# Set file_inherit and view ACL
john@x1 /usr/home/john/example$ setfacl -m
owner@:full_set:f:allow,group@:full_set:f:allow,everyone@:full_set:f:allow .
john@x1 /usr/home/john/example$ getfacl -q .
          owner@:rwxpDdaARWcCos:f------:allow
          group@:rwxpDdaARWcCos:f------:allow
       everyone@:rwxpDdaARWcCos:f------:allow

# Create another file and view its ACL
john@x1 /usr/home/john/example$ getfacl -q bar.txt
          owner@:rwxpDdaARWcCos:------I:allow
          group@:rwxpDdaARWcCos:------I:allow
       everyone@:rwxpDdaARWcCos:------I:allow

# We get the inherited ACE's. Success!
```

“d” (dir_inherit) is identical to file_inherit only it is for subdirectories, so we will skip demonstrating it. Next up is the “i” (inherit_only) flags. This flag is a bit tricky to understand. If you set it on an ACE, you are basically removing the ACE from the ACL. Newly created files and/or directories (depending on if you set file_inherit and/or dir_inherit) will inherit the ACE’s that have it set, but the ACE does not reflect on the directory itself that has the entry. Here is an example:

```
# View ACL
john@x1 /usr/home/john/example$ getfacl -q .
          owner@:rwxpDdaARWcCos:-------:allow
          group@:rwxpDdaARWcCos:-------:allow
       everyone@:rwxpDdaARWcCos:-------:allow

# Set inherit_only for group@ and everyone@ and view ACL
john@x1 /usr/home/john/example$ setfacl -m group@:full_set:fi:allow,everyone@:full_set:fi:allow .
john@x1 /usr/home/john/example$ getfacl -q .
          owner@:rwxpDdaARWcCos:-------:allow
          group@:rwxpDdaARWcCos:f-i----:allow
       everyone@:rwxpDdaARWcCos:f-i----:allow

# Create a file and view its ACL
john@x1 /usr/home/john/example$ touch foo.txt
john@x1 /usr/home/john/example$ getfacl -q foo.txt
          owner@:rw-p--aARWcCos:-------:allow
          group@:rwxpDdaARWcCos:------I:allow
       everyone@:rwxpDdaARWcCos:------I:allow

# While it appears similar to file_inherit, the example directory itself really only has an entry for
owner@
# Success!
```

Last, but not least is the “n” (no_propagate) flag. This flag controls the ability for first level files and/or directories to inherit ACE’s but not for subdirectories or files beneath the first level subdirectory to inherit ACE’s. Here is an example of how this works:

```
# View ACL with both file_inherit and dir_inherit set
john@x1 /usr/home/john/example$ getfacl -q .
          owner@:rwxpDdaARWcCos:fd-----:allow
          group@:rwxpDdaARWcCos:fd-----:allow
       everyone@:rwxpDdaARWcCos:fd-----:allow

# Create a sub directory and view its ACL
john@x1 /usr/home/john/example$ mkdir sub1
john@x1 /usr/home/john/example$ getfacl -q sub1
          owner@:rwxpDdaARWcCos:fd----I:allow
          group@:rwxpDdaARWcCos:fd----I:allow
       everyone@:rwxpDdaARWcCos:fd----I:allow

# Now create a directory in the subdirectory and view its ACL
john@x1 /usr/home/john/example$ mkdir sub1/foo1
john@x1 /usr/home/john/example$ getfacl -q sub1/foo1
          owner@:rwxpDdaARWcCos:fd----I:allow
          group@:rwxpDdaARWcCos:fd----I:allow
       everyone@:rwxpDdaARWcCos:fd----I:allow

# Just as you'd expect when dir_inherit is set
# Now, we will set the no_propagate flag and see how it changes things
john@x1 /usr/home/john/example$ setfacl -m
'owner@:full_set:fdn:allow,group@:full_set:fdn:allow,everyone@:full_set:fdn:allow' .
john@x1 /usr/home/john/example$ getfacl -q .
          owner@:rwxpDdaARWcCos:fd-n---:allow
          group@:rwxpDdaARWcCos:fd-n---:allow
       everyone@:rwxpDdaARWcCos:fd-n---:allow

# Create a subdirectory and view its ACL
john@x1 /usr/home/john/example$ mkdir sub2
john@x1 /usr/home/john/example$ getfacl -q sub2
          owner@:rwxpDdaARWcCos:------I:allow
          group@:rwxpDdaARWcCos:------I:allow
       everyone@:rwxpDdaARWcCos:------I:allow

# The newly created subdirectory inherits ACL as expted
# Now create a subdirectory beneath this and view its ACL
john@x1 /usr/home/john/example$ mkdir sub2/foo1
john@x1 /usr/home/john/example$ getfacl -q sub2/foo1
          owner@:rwxp--aARWcCos:-------:allow
          group@:rwxp--a-R-c--s:-------:allow
       everyone@:rwxp—a-R-c--s:-------:allow

# The newly create directory beneath the subdirectory does not inherit the ACL. Success!
```

5 Conclusion

In summary, ACL's provide more finer grained access for system administrators. I have shown how every access permission and inheritance flag works. With this knowledge, system administrators and power users can define security models that fit their needs in any given environment.