FreeBSD Access Control Lists

Daniel Harris

August 14, 2003

1 Introduction

Unix permissions are flexible and can solve almost any access control problem, but what about the ones they can't? Do you really want to make a group every time you want to share a file with another user? Perhaps you don't have root, and you can't create a group at will. Sometimes the limitations can cause security problems; it would be nice to be able to make a directory available to a web server or other user without making the files world-readable or world-writable. Root-owned configuration files often need to be edited by those without root privileges; instead of using programs like sudo[1] or calife[2] and risking shell escapes in editors, it would be better just to allow certain non-owners to edit these files.

Access Control Lists (ACLs) solve these problems. They allow more flexibility than the standard Unix user/group/other set of permissions. ACLs have been available in commercial UNIXes such as IRIX[3] and Solaris[4] (and in Windows NT[5]) for years. Now, thanks to the TrustedBSD[6] project's work, ACLs are available[7] in FreeBSD[8] 5.0-RELEASE and beyond. Much of the information below applies, at least in part, to ACL implementations on other platforms; however, you will want to look at specific documentation to avoid being tripped up by differences in syntax. There shouldn't be many, as FreeBSD attempts to conform to the latest `POSIX.1e` draft.

[1] http://sudo.stikman.com/
[2] http://mutt.frmug.org/calife/
[3] http://www.sgi.com/software/irix6.5/
[4] http://www.sun.com/solaris/
[5] http://www.microsoft.com/ntserver/
[6] http://www.trustedbsd.org/
[7] http://www.freebsd.org/doc/en_US.ISO8859-1/books/handbook/fs-acl.html
[8] http://www.freebsd.org/

2 Enabling ACLs

ACLs are enabled by an option in the file system superblock, which is edited by the `tunefs` command.

2.1 Setting the superblock option

The tunefs[9] command can be used only on a read-only or unmounted file system. This means that you normally must boot into single-user mode before running `/sbin/tunefs -a enable /fs`, where `/fs` represents the file system mount point (`/`, `/usr`, etc.. If you do not have access to the console of the machine (a colocated machine, for example), you can add the tunefs lines to the beginning of `/etc/rc` to enable ACLs at the next boot.

If you use the UFS2 file system, you are done. ACLs require only options `UFS_ACL`, which is built into the default `GENERIC` kernel. Reboot and enjoy. If you use UFS1, though, don't reboot yet.

2.2 Additional configuration for UFS1

Things are more difficult if you, like most FreeBSD 5.0 users, are using UFS1. (FreeBSD 5.1 and later come with UFS2 as the default file system.) ACLs are built on top of extended attributes, which are not native to UFS1. To enable extended attributes, you must add

```
options UFS_EXTATTR
options UFS_EXTATTR_AUTOSTART
```

to your kernel configuration and compile and install the new kernel. Don't reboot yet; you still need to initialize the extended attributes on each file system.

For example, to initialize attributes on the `/var` file system:

```
% mkdir -p /var/.attribute/system
% cd /var/.attribute/system
% extattrctl initattr -p /var 388 posix1e.acl_access
% extattrctl initattr -p /var 388 posix1e.acl_default
```

Just replace `/var` with the mount point of the desired file system. After initializing the attributes, reboot and extended attributes should be enabled.

[9] http://www.freebsd.org/cgi/man.cgi?query=tunefs&apropos=0&sektion=0&manpath=FreeBSD+5.1-RELEASE&format=html

3 Using ACLs

You've rebooted and you've enabled ACLs. Now what?

3.1 Looking at ACLs

Looking at ACLs is simple. Files with ACLs will be designated with a `+` in the long listing provided by `ls -l`:

```
$ getfacl acl-test
#file:acl-test
#owner:1000
#group:1000
user::rw-
user:nobody:rw-
group::r--
group:wheel:rw-
mask::rw-
other::r--
```

The `user::`, `group::`, and `other::` fields should all be familiar. They are nothing but the ACL representations of the standard UNIX permissions system. The `nobody` and `wheel` lines, however, are new. These specify permissions for specific users and groups (in this case `nobody` and `wheel`) in addition to the normal set of permissions.

3.2 Adding and Subtracting ACLs

The setfacl[10] command adds, changes, and deletes ACLs. It has many options, but you need to know only a few of them to start manipulating ACLs.

First, a word on syntax. ACLs are specified just as they're printed by `getfacl`. Let's remove and reconstruct the ACL for `acl-test`:

```
$ setfacl -b acl-test
$ setfacl -m user:nobody:rw-,group:wheel:rw- acl-test
```

[10] http://www.freebsd.org/cgi/man.cgi?query=setfacl&apropos=0&sektion=0&manpath=FreeBSD+5.0-RELEASE&format=html

The `-b` option removes all of the ACL, except for the standard user, group, and other lines. The `-m` option modifies the ACL with the specified entry (or comma-separated entries). Entries may also be abbreviated: the code above could have been shortened to `u:nobody:rw-,g:wheel:rw-`. You can even use `setfacl` to modify traditional permissions; setting a `user::rw-` ACL entry is equivalent to running `chmod u=rw` on a file.

Removing ACLs is almost identical: `setfacl -x u:nobody:rw-,g:wheel:rw-` removes that ACL. You can also specify ACLs in files. The `-M` and `-X` options perform the functions of their lowercase relatives, reading the entries from a file. Consider the `acl-test` file again:

```
$ cat test-acl-list
u:nobody:rw-
# this is a comment
g:wheel:rw-
$ setfacl -X test-acl-list acl-test
$ getfacl acl-test
#file:acl-test
#owner:1000
#group:1000
user::rw-
group::r--
mask::r--
other::r--
```

3.2.1 ACLs and other Unix tools

Unfortunately, most Unix tools do not yet support ACLs. For example, `tar` won't back up or restore ACLs, and NFS in FreeBSD ignores them too. Neither `tar`'s file format nor NFS's protocol has any place for ACLs. However, whole-file system UFS1 backups made with `tar` or `dump` will back up the `.attribute` directories, and FreeBSD's `dump` has been modified to understand UFS2 (including ACLs). The `archivers/star` port supports ACLs. You can even exchange Linux and FreeBSD archives created with `star` and preserve extended attributes (including ACLs).

3.2.2 Using ACLs with Samba and Windows

If you compile Samba[11] with ACL support, you can edit ACLs on files shared by Samba with the native Windows ACL tools. Simply compile (or recompile) Samba with ACL support. Using the FreeBSD ports system, you can specify the `WITH_ACL_SUPPORT` make flag using the `net/samba` port's configuration dialog (Figure 1).

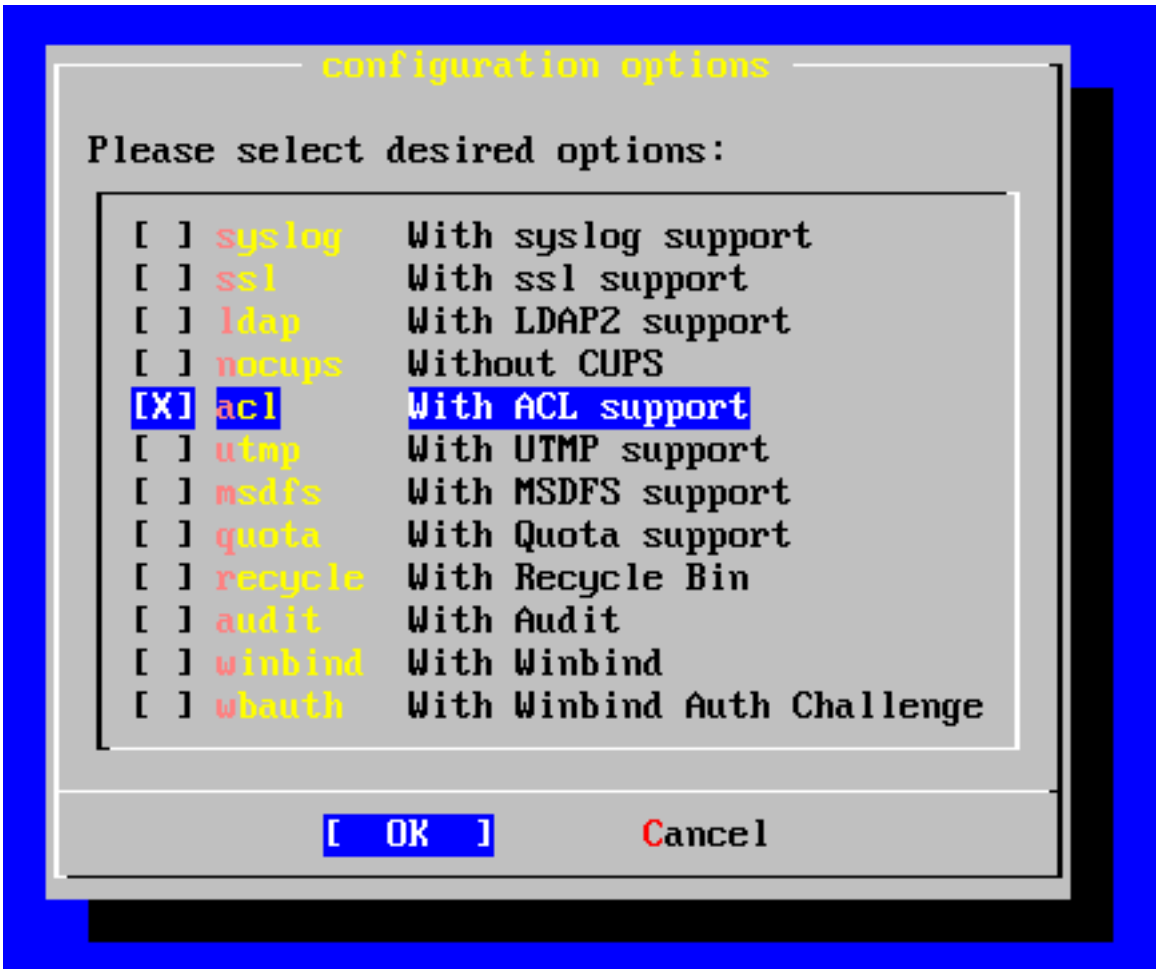

Figure 1: Samba port configuration dialog, with ACL support enabled

Once you have Samba up and running, browse to a share on an ACL-enabled

[11] http://www.samba.org/

file system. Right-click any file and select `Properties`. Go to the `Security` tab, and you can see and change the ACL as though it were on a Windows server (Figure 2).

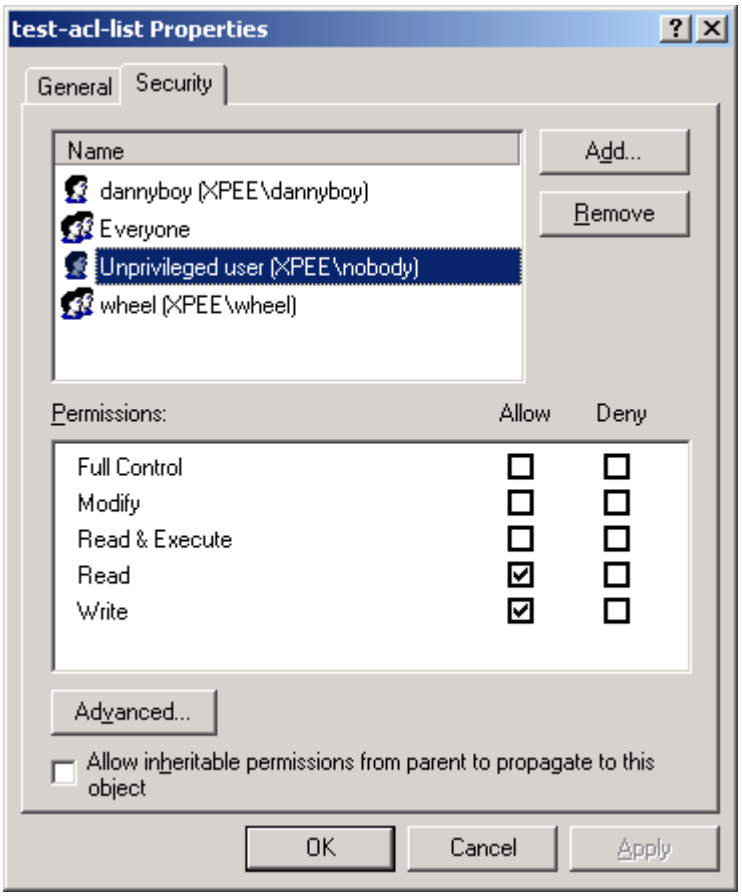

Figure 2: Windows 2000 client manipulating ACLs on FreeBSD by means of Samba

If you've been reluctant to move from a Windows server to Samba because of lack of ACLs, you can start seriously thinking about deploying Samba and FreeBSD on your file servers.

3.3 Multiplication—Default ACLs

Let's consider a more advanced example. You want to make your `cool_widgets` directory accessible to Bob, your partner in coolness, but not to the world. Just add an ACL entry. When you add files to this directory, though, they won't automatically pick up the directory's ACL. You need to set a default ACL on the directory. Any files created in the directory will inherit the default ACL.

Passing the `-d` option to either `getfacl` or `setfacl` will make it operate on the default ACL of a directory instead of on the directory itself.

```
$ mkdir cool_widgets
$ chmod o-rwx cool_widgets
$ ls -l
...
drwxr-x---  2 rob  rob   512 Apr 19 21:21 cool_widgets
...
$ getfacl -d cool_widgets
#file:cool_widgets
#owner:1000
#group:1000
```

Pretty boring, isn't it? Let's try to add a default ACL:

```
$ setfacl -d -m u:bob:rw- cool_widgets
setfacl: acl_calc_mask() failed: Invalid argument
setfacl: failed to set ACL mask on cool_widgets
```

Oops. Default ACLs don't work quite like regular ACLs do. You cannot set specific entries on a default ACL until you add the generic `user::`, `group::`, and `other::` entries.

```
$ setfacl -d -m u::rw-,g::r--,o::---,u:bob:rw- cool_widgets
$ setfacl -m u:bob:r-x cool_widgets
```

Note the non-default `r-x` entry for `bob` on the directory: the default ACL affects files that will be created inside the directory but not the directory itself. An ACL entry `u:bob:rw-` will now be added to any file created in `cool_widgets`.

Now you have a `cool_widgets` directory whose files can be read and written by both rob and bob, without the use of a group. If you later decide to get rid of the default ACL, the `-k` option to `setfacl` works for default ACLs just as the `-b` option does for file ACLs.

4 Conclusion

ACLs take care of access control problems that are overly complicated or impossible to solve with the normal Unix permissions system. By avoiding the creation of groups and overuse of root privileges, ACLs can keep administrators saner and servers more secure.

5 Author

Daniel Harris[12] is a student and occasional consultant in West Virginia. He is interested in computer networking, documentation, and security; he also enjoys writing, armchair politics, and amateur radio.

[12] http://people.freebsd.org/~dannyboy/