The journal of Apple technology.

Volume Number: 19 (2003)
Issue Number: 11
Column Tag: Programming

Section 7

File System Security

by Rich Morin

How (not) to make OS X as secure as MS Windows...

Mac OS X inherits most of its notions of file system security from BSD. Each file system node (file, directory, ...) has sets of permission (i.e., mode) bits for its owner, its group, and everyone else. The node's owner is restricted by the first set of mode bits. Other members of the node's group are restricted by the next set. Everyone else is restricted by the final set.

Let's look at some of the top-level permissions on a Mac OS X (10.2.8) system, to see how this plays out in practice:

```
% ls -dl /bin /sbin
drwxr-xr-x  35 root  wheel ... /bin
drwxr-xr-x  60 root  wheel ... /sbin
```

The first set of mode bits (rwx) allows these directories' owner (root) full access permissions. S/he can read (e.g., look up entries), write (e.g., add or remove entries), and execute (e.g., access entries) in the directory.

The following sets (r-x, r-x) restrict other users from writing, but allow read and execute access. Note, by the way, that this does not prevent someone from writing into an existing file in one of these directories, if the permissions of the file allow this.

In summary, nobody but root is able to write (e.g., create files) in any of these directories. So, "normal" users (and the programs they may accidentally or unsuspectingly) aren't able to add, remove, or rename programs.

This is very much what we'd expect in a well-designed, BSD-based system. Allowing user errors, programming mistakes, or malware to modify the system's executable code is (as myriad Microsoft-specific viruses demonstrate) a serious design error.

Unfortunately, Apple doesn't follow BSD's example everywhere; some Mac OS X system directories are all too vulnerable to the aforementioned threats:

```
% ls -dl /App*s /Developer /Library
drwxrwxr-x 59 root admin ... /Applications
drwxrwxr-x 14 root admin ... /Developer
drwxrwxr-x 40 root admin ... /Library
```

In an effort to support "ease of use", Apple's engineers have made some critical directories far more open than they would be on a conventional Unix system. As a result, most users (and any programs they may run) can add, delete,

or replace any node in these directories.

In a fine example of the "Law of Unintended Consequences", several plausible decisions work together to produce this undesirable result. Here's how it goes:

- The first account created on a new system has "admin" privileges, by default, and few users bother to set up a separate administrative account. So, most users have admin privileges.
- Any user who has admin privileges is put into the admin group.
- The admin group has write permission for all three of these directories, so any member of the group can add, delete, or replace any node in these directories.
- Any program run by a user has, by default, the same permissions as the user.

Here's a simple (and safe :-) experiment you can try. Note that the system prevents you from modifying /bin, but allows you to modify /Applications:

```
% groups
admin
% touch /Applications /bin
touch: /bin: Permission denied
```

In most cases, the system asks the user for authorization before taking any unusual or suspect action. Consider the password that sudo(8) requires and the authorization dialogs that come up on occasion (e.g., when installing software).

In this case, however, no warning is given. Any user with admin privileges is quite free to drag folders in and out of /Applications; no authorization dialog will come up. Apple is quite aware of this situation; in fact, their documentation suggests a possible workaround:

> Only admins can install software in the Applications folder. You may find that you want to set up a user account that doesn't have admin privileges and use that for day-to-day tasks. That way you won't absent-mindedly delete an application by accident.
>
> - http://www.apple.com/macosx/learning

I strongly suspect, however, that most machine owners will never see this advice. Even if they do, they may decide to ignore it. Logging in and out of accounts is a time-consuming hassle. Panther's "fast user switching" will improve this situation, but it will still break the user's concentration.

So, most users will run as admin, expecting the Finder (and other apps) to ask them before doing anything odd. Unfortunately, the Finder won't even be called into play if a rogue application is bent on rewriting parts of the file system (e.g., installing virus code).

In summary, Apple has opened up a major security hole that is not present in Mac OS X's forebears (Unix, FreeBSD, ...). Expecting application programmers (or worse, users!) to compensate for insecure directory permissions is simply bad design. The underlying system needs to be secure; exceptions can then be made on a carefully-controlled basis.

In this case, this means that the permissions need to be fixed. Administrative actions can then be performed using "privilege elevation", under the control of authorization dialogs, etc. Administrative users are quite used to being asked for this sort of authorization, so ease of use isn't being compromised.

Note: Because the root directory allows write permission to members of the admin group, you might think that it opens up a similar security hole. However, its permissions (drwxrwxr-t) include the use of the "sticky" bit (as indicated by a "t" in the last position). This allows admin users add items to the root directory, but prevents them from removing or renaming anything that they don't own. See sticky(8) for more details.

Third-party Apps

Third-party developers have some excuse for being unfamiliar with permissions issues (classic Mac OS wasn't real big on security :-), but by now, they should have learned the basics. So, it's disturbing to find many vendors leaving their application packages wide-open to writing by any user (or program) on the system.. Try:

```
% cd /Applications
% ls -ld * | grep rwxrwxrwx
drwxrwxrwx  ...  Alarm Clock S.E..app
drwxrwxrwx  ...  AutoSync.app
drwxrwxrwx  ...  Classic Toggler folder
drwxrwxrwx  ...  Cocoa Browser.app
drwxrwxrwx  ...  GraphicConverter US
drwxrwxrwx  ...  Multiple Launcher X.app
drwxrwxrwx  ...  OmniDictionary.app
drwxrwxrwx  ...  OmniGraffle.app
drwxrwxrwx  ...  PTHPasteboard
drwxrwxrwx  ...  RBrowser.app
drwxrwxrwx  ...  ShuX.app
drwxrwxrwx  ...  SliMP3 Server.app
drwxrwxrwx  ...  Text Wielder
drwxrwxrwx  ...  Tri-BACKUP Folder
drwxrwxrwx  ...  VLC.app
-rwxrwxrwx  ...  iCab
```

Let's say that Susie downloads a nifty-looking program and runs it (e.g., from her Downloads directory). Gee, it didn't do anything. That's no fun; let's try something else... Meanwhile, the "nifty-looking program" has infected any vulnerable third-party apps. When Susie's mom runs one of these (using an admin account), the infection can spread to the rest of the system.

Can anything be done?

Some folks at Apple are very concerned by these (and other) security holes, but they (clearly) aren't in control of Apple's overall security policy. As a result, OSX is ripe for the kind of bad publicity that MS Windows has recently received.

As a developer, you have a responsibility to set appropriate permissions on your package directories. Do that, and your app won't be part of the problem. Then, file bug reports with Apple, asking them to tighten up their own security holes and give developers automated feedback and assistance in closing holes in third-party applications.

Several possibilities spring to mind. If Interface Builder can draw helpful blue lines to indicate that a widget is too near the edge of a window, why can't Xcode (or whatever tool is used for package creation) tell the developer when a package's permissions are "too near the edge"?

For that matter, why can't the installation software check for this sort of thing? And, while Disk Utility is looking for weird permissions and ownerships, why can't it look for wide-open package directories? Like that...

More generally, when you talk to Apple, tell them that you don't want security concerns to be completely overridden by "ease of use" considerations. Both are critical; a proper solution won't ignore either one.

Rich Morin has been using computers since 1970, Unix since 1983, and Mac-based Unix since 1986 (when he helped Apple create A/UX 1.0). When he isn't writing this column, Rich runs Prime Time Freeware (www.ptf.com), a publisher of books and CD-ROMs for the Free and Open Source software community. Feel free to write to Rich at rdm@ptf.com.