

FOCUS on Linux: Filesystem Security - ext2 extended attributes

Michael Shaffer 2000-11-20

Introduction

If asked to name the top five security features of the Linux kernel, most administrators would probably not mention ext2 filesystem attributes. Although the definitions for most of the useful ext2 filesystem flags appeared in the kernel source at least as early as the 1.1 development series, this humble feature often takes a back seat to more exotic and recently-introduced tools for preserving and assuring system integrity such as [LIDS](#), [Tripwire](#), and others.

What are ext2 attributes?

The Linux ext2 filesystem has supported extra flags or 'attributes' for files and directories in all Linux kernels since the 1.1 development series. The ext2 filesystem driver in the 2.2 and 2.4 kernel series permits setting and querying the following attributes:

A	Atime	The system should not update the atime or 'access time' of this file.
S	Sync	The system should flush all changes to this file to the physical disk synchronously when an application performs a write to it.
a	Append Only	The system should only allow opening of this file for appending and should not allow any process to overwrite or truncate it. In the case of a directory, processes may create or modify files in the directory but not delete them.
i	Immutable	The system should disallow all changes to this file. In the case of a directory, processes may modify files that already exist in the directory but may neither create nor delete files.
d	No Dump	The dump(8) program should ignore this file when performing filesystem backups.
c	Compress	The system should transparently compress this file. A read from the file should return uncompressed data, and a write to the file should result in data compression before the

data reaches the physical disk.

s	Secure Del.	When the system deletes this file it should overwrite all of its data blocks on disk with zero bytes.
u	Undelete	When an application requests deletion of this file, the system should preserve its data blocks in such a way as to allow 'un-deletion' of the file at a later time.

While the filesystem will accept and preserve flags indicating each of these attributes, the kernel and various applications may or may not honor them in any meaningful way depending on their individual versions. The table below shows a cross reference of the flags with each kernel version and displays whether each kernel version will:

- * allow setting the flag and honor its value
- i allow setting the flag but ignore its value
- ignore the flag altogether

	1.0	1.2	2.0	2.2	2.4
A	-	-	*	*	*
S	*	*	*	*	*
a	-	*	*	*	*
i	-	*	*	*	*
d	-	*	*	*	*
c	i	i	i	i	i
s	*	*	i	i	i
u	i	i	i	i	i

Although earlier kernels honored the 'secure deletion' flag, during the development of the 1.3 series the developers dropped the implementation of this property since it seemed to provide at best only a trivial amount of additional security and at worst a false sense of real security to users unfamiliar with the inherent problems of any 'secure deletion' scheme.

The 'A' or 'atime' flag may provide some performance and power savings when used on certain files since it relieves the system of the obligation to update the access time field of those files

each time they are read. The 'S' or 'synchronous' attribute may provide a somewhat higher level of data integrity at the expense of performance since it forces the system to flush all changes to a file to disk immediately and causes calls to `write(2)` to block until the flush completes.

This discussion will focus mainly on the 'a' or 'append only' flag and the 'i' or 'immutable' flag since they present the most obvious benefits with respect to enhancing filesystem security and integrity. The various open source BSD systems such as FreeBSD and OpenBSD have supported similar flags in their UFS or FFS implementations for some time, and their record of usefulness on these other platforms was likely no small part of the inspiration for the Linux implementation.

What commands are used to set and display ext2 attributes?

The standard `ls(1)` command will not display the extended attributes of a file under any circumstances. The ext2 filesystem utilities package, however, contains two utilities designed specifically for setting and querying these attributes, namely `chattr(1)` and `lsattr(1)`. Since ext2 serves as the standard workhorse filesystem for Linux, nearly all distributions will have the 'e2fsprogs' package installed by default. If, for some reason, a Linux system does not include these utilities, then they may be easily installed by downloading and compiling the e2fsprogs source from:

The Ext2fs Home Page

<http://e2fsprogs.sourceforge.net/ext2.html>

The `lsattr(1)` command supports only a few options, most notably:

```
-a      list all files and directories
-d      when listing directories, do not list their contents
```

which behave similarly to the identical options available with the `ls(1)` command. I usually create a shell command alias to make 'la' equal to 'lsattr -a' as the latter becomes tedious if used often.

The `chattr(1)` command, may be invoked in one of three ways:

```

chattr +Si test.txt    --    to add the sync and immutable flags on test.txt

chattr -ai test.txt   --    to unset or remove both the append-only and
                             immutable flags from test.txt

chattr =aiA test.txt  --    to set the attributes for test.txt to
                             contain only the a, i, and A flags

```

Finally, both commands accept a `-R` option which will cause them to recursively list the contents of directories and their children when encountered.

How are ext2 attributes different from filesystem permissions?

Most UNIX administrators are familiar with the standard UNIX style filesystem permissions and ownerships displayed by the `ls(1)` command:

```

[root@typhoid shaffer]# ls -al test*
-rw-rw-r--    1 shaffer  users          0 Nov 17 17:02 test.conf
-rw-rw-r--    1 shaffer  users          0 Nov 17 17:02 test.log
-rw-rw-r--    1 shaffer  users          0 Nov 16 19:41 test.txt

```

This output from the `ls(1)` command indicates that all of the test files are owned by user `shaffer` and group `users` with read and write permission for each. In addition, the file permissions also allow global or 'other' read access. The output of the `lsattr` command, however:

```

[root@typhoid shaffer]# lsattr -a test*
---i----- test.conf
----a----- test.log
----- test.txt

```

reveals that test.log has the append-only flag set and test.conf has the immutable flag set. If a user logs in as root or a process is running as root, then their filesystem accesses are not subject to any restrictions imposed by standard permissions and will always succeed. This behavior has been carried over to most UNIX-like systems from the earliest incarnations and is the root (pun intended) of many successful local and remote security exploits. Ext2 attributes, on the other hand, are checked and honored by various system calls such as `sys_open()` and `sys_truncate()` without regard to uid or any other influence. The mere presence of the immutable flag on an inode will cause all system calls involving file modification to fail regardless of any other circumstance. By providing these attributes and some closely related kernel capabilities (see below), Linux allows an administrator to easily and effectively curtail the absolute power traditionally afforded to processes with uid 0. Because the a and i attributes are enforced without regard to permissions or privilege level, they can serve as an effective underlayment of defense against the exploitation of any privileged processes which may contain undisclosed or unpatched vulnerabilities. It should be noted, however, that although these attributes are an effective means of protecting and ensuring system integrity at the filesystem level, they will not suffice as a sole means of defense since they can do nothing to prevent unauthorized disclosure of data or denial of service style attacks.

Finally, filesystem attributes by themselves are only a half-measure of protection for critical files. Although the a and i attributes will prevent even root owned processes from altering files, under normal circumstances the super user still has the ability to simply remove the flags and then proceed without hindrance. If this were the only factor involved, then it would simply add a slight bit of complexity to most existing exploits to ensure that the attributes were removed before proceeding.

In kernel versions prior to the 2.1 series, the 'securelevel' feature was used to 'close the circle' so to speak since a securelevel greater than zero would disallow any changes in the a and i attributes for all files. For these earlier kernels, the securelevel is controlled through the `sysctl` variable 'kernel.securelevel', and if this variable is set to one or higher early in the boot process then the system will not allow changes to immutable and append-only files at all unless booted into single-user mode.

In kernel versions greater than 2.0, the new and more flexible 'kernel capabilities' system allows the system to be configured into a similar restricted mode which will completely protect immutable and append-only files. The `lcap(8)` tool allows querying and adjustment of the kernel capability bounding set, and a configuration which will rely on the extended attributes of the

ext2 filesystem should include at least the following lcap calls early in its startup scripts:

```
lcap CAP_LINUX_IMMUTABLE
lcap CAP_SYS_RAWIO
```

The first invocation removes the capability for even root owned processes to change the a and i flags, and the second removes raw access to block devices such as disks to prevent modification of immutable files through direct writes to the underlying disk partitions housing them. The CAP_SYS_RAWIO capability, should **always** be removed early in the system startup process if any serious use of kernel capabilities is intended since failing to remove it allows a root process to circumvent or alter the capability bounding set by directly modifying kernel memory through the /dev/kmem device. If invoked with no arguments, the lcap(8) utility will list the complete kernel capability bounding set and highlight those which are currently available. For the interested reader, a link to an article with a much more thorough discussion of kernel capabilities is provided below.

Once lcap has removed a kernel capability, no facility exists for restoring the capability short of rebooting the system. This 'trapdoor' property of the capability bounding set serves to ensure that (on a properly configured system) the capability limits themselves may not be circumvented without gaining physical access to the system and booting it in single user mode.

More information on lcap(8) as well as the source code for this tool may be found at:

```
LCAP - Linux Kernel Capabilities Bounding Set Editor
http://pwl.netcom.com/~spoon/lcap/
```

Things that one should usually 'chattr'

When configuring systems which will be directly exposed to the Internet or other hostile environments and which must host shell accounts or services such as HTTP and FTP, I usually include the following commands in my routine once I have installed and configured all necessary software and user accounts:

```
chattr -R +i /bin /boot /etc /lib /sbin
chattr -R +i /usr/bin /usr/include /usr/lib /usr/sbin
chattr +a /var/log/messages /var/log/secure (...)
```

If accounts are rarely added, changed, or deleted, then marking /home itself immutable (but not individual home directories) should not cause many problems either. In many cases, the entire /usr tree may be safely made immutable. In fact, in addition to running 'chattr -R +i /usr', I will also usually place /usr on its own partition and mount it read-only using the 'ro' option in /etc/fstab. Adding the append-only flag to the system log files should serve to greatly inhibit the ability of any intruder to conceal their presence and their actions should they successfully penetrate the system.

Of course, a system secured in this way will require some modifications in typical administrative procedures. Let's look at some of the different aspects to take into account.

Installing and upgrading software

Software installation or upgrade will typically require removal of the immutable and append only flags on areas where the installer needs to add and remove files. On rpm based systems, the command:

```
rpm -qpl newpackage.rpm
```

will display all the files that a new package contains (and hence must be able to create or replace). Most packages will require the rpm command to write to one or more of:

```
/bin
/sbin
/usr/bin
/usr/sbin
/usr/man
/lib
```

```
/etc
```

Note that in order to allow replacement of the file `/usr/sbin/someprogram`, for example, it will usually be necessary to remove the immutable flags from both the file itself and the `/usr/sbin` directory which contains it.

Managing user and group accounts

All of the following:

```
/etc  
/etc/.pwd.lock  
/etc/passwd  
/etc/passwd-  
/etc/shadow  
/etc/shadow-  
/etc/group  
/etc/group-  
/etc/gshadow  
/etc/gshadow-
```

must remain writable and deleteable since the `passwd(1)`, `chsh(1)`, `chfn(1)`, `vipw(8)`, `vigr(8)`, and `useradd(8)` commands all process these files by creating a temporary copy in `/etc`, modifying the copy, deleting the original file, and then renaming the copy.

Things that one should **never** `chattr`

```
/
```

This will silently break a lot of things (most notably `syslog`) with no apparent clues. Since the problem that this creates affects `syslog` first, it will also prevent the logging of most errors from other applications, making it a real head-scratcher. When I first started experimenting with `ext2` attributes, I lost the better part of an afternoon troubleshooting a system on which I had

naively executed 'chattr +i /'.

/dev

Syslog needs to remove and re-create the /dev/log socket when it starts up, and setting either the immutable or append only flags on /dev will prevent this. Unless the system startup scripts execute syslogd with the '-p' option to specify an alternative socket such as /var/run/syslog.sock, then setting the immutable or append only flags on /dev will cause problems. Even when the '-p' option is used, syslog clients will still expect to find a socket at /dev/log, so a soft link should be created to point /dev/log to the location of the real socket. Finally, even this change will not suffice on systems that run lpd since the entire lpr package contains /dev/printer as a hard coded path for the lpd service socket, and lpd attempts to remove and recreate its socket at startup as well. Changing this path requires recompiling the entire lpr package from source.

/tmp

Setting the append only or immutable attributes on this directory would obviously break nearly anything that uses any sort of temporary file.

/var

Generally the append only and immutable flags should be applied sparsely to only those files in /var which can clearly accept them without ill effect. For instance, using chattr +a on most of the files in /var/log will not cause problems for anything except logrotate, with some exceptions. For instance, the /var/log/sendmail.st file must remain writable and deletable since sendmail does not simply append statistics to this file but instead periodically truncates and overwrites it.

Relevant Links

[Linux Kernel Capabilities](#)

Jeremy Rauch

[Ext2fs Home Page](#)

Ext2fs Project

[LCAP](#)

Spoon

[Kernel Source Cross Reference](#)

Arne Georg Gleditsch and Per Kristian Gjermshus

[FreeBSD](#)

FreeBSD Project

[OpenBSD](#)

OpenBSD Project

[Privacy Statement](#)

Copyright 2006, SecurityFocus