

# Introduction to Linux Capabilities and ACL's

Jeremy Rauch 2000-08-28

## I. Introduction

Unix systems have always utilized a security system that gives normal users a minimal amount of privilege, while creating a single account, known as the 'root' account, that has full privileges. The root account is used to administer the machine, install software, create new users, and run certain services. Many common activities that require root privileges are often run as the root user, via the concept of `setuid`.

This dependence upon a single account for performing all actions requiring privilege has proven to be somewhat dangerous. Programs often need root privileges for a single activity, such as binding to a privileged port, or opening a file only root can access. Vulnerabilities are often found that could perhaps be eliminated if these programs didn't run as root.

In version 2.1 of the Linux kernel, work was started to add what are known as `capabilities`. The goal of this work was to eliminate the dependence on the root account for certain actions. As of the 2.2 kernels, this code is beginning to become useful. While problems still exist with it, it is a step in the right direction.

Another interesting project being worked on, also to compensate for some short comings in the file access control realm, is the Linux ACL project. These extend the `ext2fs` to allow for a finer degree of access control than is normally allowed on a Unix filesystem.

We'll briefly discuss each of these features, talk about where they may be useful, and give short examples on using them and their associated utilities. Some portions of this article will require installing kernel patches, and patching and replacing system utilities. Make sure you take extreme care when doing so, don't attempt to do so on production or other critical systems.

## II. Linux Capabilities

As was briefly touched upon in the introduction, the Linux capabilities system was designed to help remove the problems associated with the need for root privileges. Before we go any further, we need to touch upon why certain privileges are reserved for root, and why certain programs need to run as root.

According to Unix security conventions, there are certain privileges reserved only for the root account. Privileged actions include things like binding a process to a privileged port, loading kernel modules, mounting and unmounting file systems, and a variety of other system activities. root is also the only account capable of adding/altering account information, installing system utilities. The privileges of root are extensive.

So what kinds of things would an average user need root access for? A simple example in the `ping` program. Ping needs to be able to open a raw socket in order to send packets to the network. This is an operation that only root may perform, so the `ping` program is installed `setuid root`. This allows any user to use the program, and still be able to open a raw socket. Unfortunately, this means the program has the full privileges of root. Should a flaw be discovered in the program, it could be possible for the attacker to perform any activity root can perform.

Capabilities work by breaking the actions normally reserved for root down in to smaller portions. Originally broken down in a POSIX specification, Linux has implemented 7 of the capabilities outlined in the Posix document 1003.1e, and another 20 or so Linux specific ones. Some of the more useful ones are described in the table below.

Capability Name	Meaning
<code>CAP_CHOWN</code>	Allow for the changing of file ownership
<code>CAP_DAC_OVERRIDE</code>	Override all DAC access restrictions
<code>CAP_DAC_READ_SEARCH</code>	Override all DAC restrictions regarding read and search
<code>CAP_KILL</code>	Allow the sending of signals to processes belonging to others
<code>CAP_SETGID</code>	Allow changing of the GID
<code>CAP_SETUID</code>	Allow changing of the UID
<code>CAP_SETPCAP</code>	Allow the transferring and removal of current set to any PID

<b>CAP_LINUX_IMMUTABLE</b>	<b>Allow the modification of immutable and append-only files</b>
<b>CAP_NET_BIND_SERVICE</b>	<b>Allow binding to ports below 1024</b>
<b>CAP_NET_RAW</b>	<b>Allow use of raw sockets</b>

There are a number of other capabilities implemented; if you're interested in looking at them, and you have a 2.2.x kernel installed, you should be able to view them in the `/usr/include/linux/capability` file.

For the most part, capabilities are most useful to programmers. The use of capabilities is only beginning to trickle in to userland applications; most system utilities do not shed their root privileges. In addition, the mechanism to set capabilities on binaries on the filesystem does not exist. We should expect to actually see capabilities become useful in the 2.4 kernels. There are, however, possible uses of capabilities for system administrators.

In Linux kernels 2.2.11 and later, the concept of capability bounding sets exists. This is a table of the capabilities allowed on the system. By removing capabilities from the set, it is possible to eliminate capabilities outright. Once a capability has been removed, it can not be added again. The `/proc/sys/kernel/cap-bound` proc file allows the controlling of this set. The `lcap` program provides a clean, simple interface for removing capabilities. It is available at <http://home.netcom.com/~spoon/lcap/>. One of the nicer things which can now be performed is the disabling of module loading.

```
[root@hamachi lcap-0.0.3]# ./lcap
Current capabilities: 0xFFFFFFFF
0) *CAP_CHOWN                1)
*CAP_DAC_OVERRIDE
2) *CAP_DAC_READ_SEARCH     3)
*CAP_FOWNER
4) *CAP_FSETID              5)
*CAP_KILL
6) *CAP_SETGID              7)
*CAP_SETUID
8) *CAP_SETPCAP             9)
*CAP_LINUX_IMMUTABLE
10) *CAP_NET_BIND_SERVICE  11)
```

```

*CAP_NET_BROADCAST
    12) *CAP_NET_ADMIN                13)
*CAP_NET_RAW
    14) *CAP_IPC_LOCK                15)
*CAP_IPC_OWNER
    16) *CAP_SYS_MODULE              17)
*CAP_SYS_RAWIO
    18) *CAP_SYS_CHROOT              19)
*CAP_SYS_PTRACE
    20) *CAP_SYS_PACCT              21)
*CAP_SYS_ADMIN
    22) *CAP_SYS_BOOT                23)
*CAP_SYS_NICE
    24) *CAP_SYS_RESOURCE            25)
*CAP_SYS_TIME
    26) *CAP_SYS_TTY_CONFIG
* = Capabilities currently allowed
[root@hamachi lcap-0.0.3]# ./lcap CAP_SYS_MODULE
[root@hamachi lcap-0.0.3]# ./lcap
Current capabilities: 0xFFFFEFFFF
    0) *CAP_CHOWN                    1)
*CAP_DAC_OVERRIDE
    2) *CAP_DAC_READ_SEARCH          3)
*CAP_FOWNER
    4) *CAP_FSETID                   5)
*CAP_KILL
    6) *CAP_SETGID                   7)
*CAP_SETUID
    8) *CAP_SETPCAP                   9)
*CAP_LINUX_IMMUTABLE
    10) *CAP_NET_BIND_SERVICE        11)
*CAP_NET_BROADCAST
    12) *CAP_NET_ADMIN                13)
*CAP_NET_RAW
    14) *CAP_IPC_LOCK                15)
*CAP_IPC_OWNER
    16) CAP_SYS_MODULE                17)
*CAP_SYS_RAWIO

```

	18)	*CAP_SYS_CHROOT	19)
*CAP_SYS_PTRACE			
	20)	*CAP_SYS_PACCT	21)
*CAP_SYS_ADMIN			
	22)	*CAP_SYS_BOOT	23)
*CAP_SYS_NICE			
	24)	*CAP_SYS_RESOURCE	25)
*CAP_SYS_TIME			
	26)	*CAP_SYS_TTY_CONFIG	
		* = Capabilities currently allowed	

As capabilities begin to enjoy more widespread use, vulnerabilities that lead to complete root access should become far less common.

### III. Posix ACL's

Another interesting, although seemingly less mainstream kernel project than capabilities, is the work being done to introduce Posix file access control list (ACL) support in Linux. Those familiar with commercial Unix OS's may have used, or at least seen Posix ACL's in action.

ACL's present a way to add finer file-level access control. Whereas default Unix permissions allow for permissions to be associated with a single owner, group and the rest of the world, ACL's allow permissions to be set for multiple groups or individuals. Rather than rehash, it's worthwhile to read an article on the use of ACL's under Solaris. It is available [here](#). It covers basic use of the getfacl and setfacl utilities, both of which are implemented with an almost identical interface under Linux.

ACL's are not included by default in the Linux kernel, nor are the associated changes that need to be made to certain system utilities to support ACL's correctly. The userland ACL utilities, kernel patches, and patches to e2fsck and GNU fileutils are all available at <http://acl.bestbits.at/download.html>, which is the home site of the project. The e2fsprogs tarfile is available at <ftp://download.sourceforge.net/pub/sourceforge/e2fsprogs/>, and fileutils at <ftp://ftp.gnu.org/pub/gnu/fileutils>. Directions for installing the patches and building the userland utilities can be found at <http://acl.bestbits.at/>.

The ACL implementation is still very much a work in progress, and is not for the faint

of heart; there is a decent chance you will have compilation problems. However, ACL's can go a long way in reducing the need to give people root access. Instead of having to rely on an application like `sudo` to allow limited root access, ACL's can instead be used to allow specific users to run a setuid application, without needing to create large quantities of groups.

## IV. Conclusion

A number of the projects being developed in the Linux kernel will go very far towards improving security on Linux machines. By introducing methods for reducing the dependence upon root for specific privileged actions, the possible exposure from having large quantities of programs running as root will be eliminated. If applications can only conduct the actions they are required to, the likelihood of a vulnerability allowing a user to conduct arbitrary actions as root will be much smaller. Using finer grained access control mechanisms, like those presented by ACL's will even further reduce this exposure.

### Relevant Links

[Capabilities FAQ](#)

*kernel.org*

[lcap](#)

*spoon@ix.netcom.com*

[Focus On Sun: ACL's](#)

*Jeremy Rauch*

[Linux ACL's](#)

*Andreas Grunbacher*

[Privacy Statement](#)

Copyright 2006, SecurityFocus