A life of privilege

Taking Root

Granting root access, even temporarily, is rife with danger. Capabilities could help. By Kurt Seifried

One topic that keeps coming up in security discussions of Windows vs. Linux is the whole "Administrator vs. root" argument. With Windows, you pretty much *have* to log in with administrative privileges. If you log in to Windows as a normal user, you'll have a heck of a time using a lot of programs because as a non-administrative user, installing software is usually out of the question.

Contrast that to Linux, where the use of a system with normal privileges is generally easy (notwithstanding things like printer setup, changing the time, joining wireless networks, and so on, which might require additional privileges), and the root account is usually allowed to lie fallow, only to be used when significant system maintenance or changes are needed. However, in an effort to increase security, you need to be careful how you apply administrative privileges, or you'll end up with some program leveraging elevated privileges to gain complete control of the system.

Capabilities

Traditionally, Linux had two levels of privilege – root and everything else – which lacks granularity to say the least. Early attempts at solving this included `setuid` and `setgid` binaries, which could run with elevated privileges; tools like `sudo`, which allows you to run specific commands with elevated privileges; and ultimately things like PAM, which allow execution or privileged binaries on the basis of whether you were logged in locally or over the network, for example. However, these solutions had their own share of problems: A vulnerability like a buffer overflow in a program, now running as root, meant that the local user code would run arbitrary code as root.

In an effort to address this weakness, the Linux kernel added "capabilities" that were meant to break up the privileged operations needed on a modern Linux system into more discrete pieces. Currently, 36 capabilities exist. Some common ones are listed in Table 1.

CAP_NET_BIND_SERVICE

The `CAP_NET_BIND_SERVICE` capability allows you to bind to privileged ports (between 1 and 1024) as a non-root user. The advantage of this is that virtually all the interesting network services (FTP, WWW, email, etc.) need to start as root so they can bind to their privileged port; then, they drop privileges. With `CAP_NET_BIND_SERVICE`, you could dispense with this use of root entirely.

CAP_NET_RAW

Speaking of network access and root, another common need for elevated privileges is network sniffing. Programs such as Wireshark, probably the most commonly used GUI network sniffing tool, are infamous for security flaws because of the complexity of writing protocol dissectors that can parse and display network protocol information correctly. To date, around 152 CVE identifiers have been assigned to various security issues in Wireshark [1]. At this point, pretty much everyone recommends not running Wireshark as root. With the `CAP_NET_RAW` capability assigned, you can indeed entirely avoid the need for root privileges with Wireshark, as well as with other network sniffing programs (e.g., tcpdump).

CAP_NET_ADMIN

Of late, it seems Linus Torvalds has been getting somewhat fed up with the state of desktop Linux, and I can't say that I blame him. To quote:

I don't think I can talk about "security" people without cursing, so you might want to avert your eyes now. … I first spent weeks arguing on a bugzilla that the security policy of requiring the root password for changing the timezone and adding a new wireless network was moronic and wrong. Linus Torvalds [2]

Speaking as one of the security people, all I can say is "I'm sorry" (more than one vendor does this). I suspect what happens is that most of these operations (changing a time zone, adding a wireless

KURT SEIFRIED

Kurt Seifried is an Information Security Consultant specializing in Linux and networks since 1996. He often wonders how it is that technology works on a large scale but often fails on a small scale.

TABLE 1: Common Capabilities

Capability	Function
CAP_CHOWN	Make arbitrary changes to file UIDs and GIDs.
CAP_DAC_OVERRIDE	Bypass file read, write, and execute permission checks.
CAP_DAC_READ_SEARCH	Bypass file and directory read and execute permission checks.
CAP_KILL	Bypass permission checks for sending signals.
CAP_SETFCAP	Set file capabilities.
CAP_SETUID	Make arbitrary manipulations of process UIDs.
CAP_SYS_ADMIN	Perform a range of system administration operations and perform operations on trusted and security Extended Attributes.
CAP_SYS_MODULE	Load and unload kernel modules.
CAP_SYS_NICE	Raise process nice value.
CAP_SYS_TIME	Set system clock.
CAP_SYSLOG	Perform privileged `syslog(2)` operations.

network, etc.) generally require touching files in `/etc/`, so to be on the safe (and less than user-friendly) side, vendors typically require root for these operations. Obviously, a privileged group could be created and accounts added to it at creation time, but you would still need to run network-changing programs (e.g., `ifconfig`, `ip`, `route`, etc.) as root. With the `CAP_NET_ADMIN` privilege, you could allow tools like NetworkManager to run without the need for root-level access.

CAP_NET_BROADCAST

Although this one is not used much right now, I hope it will be in future. Like most people, I've only ever seen multicast used once, in a research environment. Generally speaking, the idea of multicast (send one packet to many systems spread across various networks at the same time, like a TV broadcast or an operating system update) is an attractive idea, it just never really took off. One hopes IPv6 would change this; of course, I could be wrong (this also depends on widespread deployment of IPv6 to end nodes). With `CAP_NET_BROADCAST`, a program could send and receive broadcasts and multicasts without requiring special privileges. And, with TV moving to the Internet, this is one attractive option for media delivery, especially for consumer networks in which the ISP wants to deliver your content – basically all of them.

The Downside of Capabilities

When writing software that contains a privileged operation as a feature, which capability do you require in order to use it? The documentation for capabilities contains examples for each capability but is by no means complete. The source code file [3] is the ultimate reference and contains a complete list of all capabilities, along with comments detailing what the capabilities are being used for; however, it isn't completely up to date. Additionally, you don't get a whole lot of guidance on which capability to use, so unless it's pretty obvious (e.g., you want to set the system time or fiddle with the networking or firewall settings), it isn't always clear which capability to require. This has led to the general issue of "when in doubt, use `CAP_SYS_ADMIN`," and has resulted in almost half of all capabilities requiring `CAP_SYS_ADMIN`.

Capability Helpers

To add insult to injury, it was then discovered that users could abuse capabilities in other ways. For example, a user with the `CAP_NET_ADMIN` capability could load modules to add network support – such as a network driver, which makes sense, right? Unfortunately, the name of the module being loaded wasn't checked properly, and a user could load arbitrary modules with a command such as:

```
ifconfig xfs
```

Like most security issues, this was dealt with pretty quickly through the addition of limits on the usermode helpers (limiting their capabilities).

Is the Cure Worse Than the Disease?

As I mentioned before, the `CAP_SYS_ADMIN` capability is widely used – some would say widely abused. Certain capabilities potentially can be abused to gain other capabilities. A holy grail for any attacker is loading a custom kernel module. Once this is accomplished, the attacker can modify the kernel and completely control the system. Some would argue that Linux capabilities incur a false sense of security. The thinking goes that if they are used properly, you can largely dispense with the root account, making it a lot harder to compromise the system. A walk-through of these types of attacks is available at GRSecurity [4].

On the other hand, you're coming from a world in which getting root privilege pretty much meant instant and complete system compromise; root could modify the kernel and system in any number of ways, generally with few or no restrictions. Like many new security technologies, things are a little rough around the edges and will likely remain so for a while, but over the long term, I think Linux capabilities will pay off.

The Long Term

What improvements are feasible? If you aren't a kernel developer there isn't a whole lot you can do. Probably the best thing is to be aware of is the overuse of `CAP_SYS_ADMIN`. Also, be careful which user accounts you allow to access programs that have this privilege. More coordination with the Linux kernel could restrict capabilities, as was the original intent.

With luck, the goal – and this is a long shot, I admit – would be to dispense with the root account entirely and make it hard, if not impossible, to gain root-level privileges in the future. This objective is especially important in light of the rapid development of virtualization and cloud computing, to which we are literally selling (in some cases, giving away) root access to the bad guys. ■■■

INFO

[1] Wireshark security issues: *http://cve.mitre.org/cgi-bin/cvekey.cgi?keyword=wireshark*

[2] Linus Torvalds rant: *https://plus.google.com/102150693225130002912/posts/1vyfmNCYpi5*

[3] Capability.h: *http://lxr.linux.no/linux+v3.4.2/include/linux/capability.h*

[4] False Boundaries and Arbitrary Code Execution: *http://forums.grsecurity.net/viewtopic.php?f=7&t=2522*