Search ...

Linux Audit

The Linux security blog about Auditing, Hardening, and Compliance

Home | Linux Security | Lynis | About

2014-11-30 (last updated at July 25th, 2018)
Michael Boelen
Hardening, Linux, System Administration
Leave a comment

# Linux capabilities 101

Security of Linux systems and applications can be greatly improved by using hardening measures. One of these measures is called **Linux capabilities**. Capabilities are supported by the kernel for some while now. Using capabilities we can strengthen applications and containers. Unfortunately, this powerful tool is still underutilized. Time to change that! This article helps to understand and apply them.

Table of Contents

**About Linux Audit**

This blog is part of our mission: *help individuals and companies, to scan and secure their systems*. We simply love Linux security, system hardening, and questions regarding compliance.

Besides the blog, we have our security auditing tool Lynis. Open source, GPL, and free to use.

Lynis project page

For those with enterprise needs, or want to audit multiple systems,

## What are Linux capabilities?

Normally the root user (or any ID with UID of 0) gets a special treatment when running processes. The kernel and applications are usually programmed to skip the restriction of some activities when seeing this user ID. In other words, this user is allowed to do (almost) anything.

Linux capabilities provide a subset of the available root privileges to a process. This effectively breaks up root privileges into smaller and distinctive units. Each of these units can then be independently be granted to processes. This way the full set of privileges is reduced and decreasing the risks of exploitation.

## Why capabilities?

To better understand how Linux capabilities work, let's have a look first at the problem it tries to solve.

Let's assume we are running a process as a normal user. This means we are non-privileged. We can only access data that owned by us, our group, or which is marked for access by all users. At some point in time, our process needs a little bit more permissions to fulfill its duties, like opening a network socket. The problem is that normal users can not open a socket, as this requires root permissions.

### Option 1: Giving everyone root permissions

One of the solutions is to allow some permissions (by default) to all users. There is a serious flaw in this approach. Allowing this kind of permissions, for all users, would open up the system for a flood of system abuse. The reason is that every small opportunity is being used for good, but also for bad. Giving away too many privileges by default will result in unauthorized changes of data, backdoors and circumventing access controls, just to name a few.

### Option 2: Using a fine-grained set of privileges

For example, a web server normally runs at port 80. To start listening on one of the lower ports (<1024), you need root permissions. This web server daemon needs to be able to listen to port 80. However, it does not need access to kernel modules as that would be a serious threat to the integrity of the system!. Instead of giving this daemon all root permissions, we can set a capability on the related binary, like CAP_NET_BIND_SERVICE. With this specific capability, it can open up port 80. Much better!

## Replacing setuid with capabilities

Assigning the setuid bit to binaries is a common way to give programs root permissions. Linux capabilities is a great alternative to reduce the usage of setuid.

> **Insight**: *Capabilities break up root privileges in smaller units, so root access is no longer needed. Most of the binaries that have a setuid flag, can be changed to use capabilities instead.*

See the related article Hardening Linux binaries by removing the setuid bit and apply this to your system.

# Available capabilities

## Support by the Linux kernel

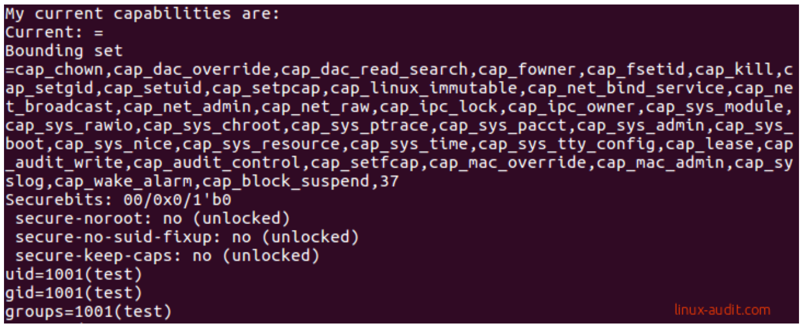
Linux capabilities are defined in a header file with the non-surprising name **capability.h**. The number of capabilities supported by recent Linux versions is close to 40. To see the highest capability number for your kernel, use the data from the /proc file system.

```
cat /proc/sys/kernel/cap_last_cap
```

The full list of available Linux capabilities for the active kernel can be displayed using the **capsh** command.

```
capsh --print
```

The same number from the **cap_last_cap** file might be also displayed at the end of a capability set.



```
My current capabilities are:
Current: =
Bounding set
=cap_chown,cap_dac_override,cap_dac_read_search,cap_fowner,cap_fsetid,cap_kill,c
ap_setgid,cap_setuid,cap_setpcap,cap_linux_immutable,cap_net_bind_service,cap_ne
t_broadcast,cap_net_admin,cap_net_raw,cap_ipc_lock,cap_ipc_owner,cap_sys_module,
cap_sys_rawio,cap_sys_chroot,cap_sys_ptrace,cap_sys_pacct,cap_sys_admin,cap_sys_
boot,cap_sys_nice,cap_sys_resource,cap_sys_time,cap_sys_tty_config,cap_lease,cap
_audit_write,cap_audit_control,cap_setfcap,cap_mac_override,cap_mac_admin,cap_sy
slog,cap_wake_alarm,cap_block_suspend,37
Securebits: 00/0x0/1'b0
 secure-noroot: no (unlocked)
 secure-no-suid-fixup: no (unlocked)
 secure-keep-caps: no (unlocked)
uid=1001(test)
gid=1001(test)
groups=1001(test)
```

The capsh command shows the available Linux capabilities

Normal users typically don't have any capabilities assigned. This is also what we have seen in the screenshot. That is why the *current* list is empty. Now that changes if you switch to your root user.

## Current capabilities

To see the capabilities for a particular process, use the **status** file in the /proc directory. As it provides more details, let's limit it only to the information related to Linux capabilities.

```
cat /proc/1234/status | grep Cap
```

This command should return 5 lines on most systems.

- CapInh = Inherited capabilities
- CapPrm – Permitted capabilities
- CapEff = Effective capabilities
- CapBnd = Bounding set
- CapAmb = Ambient capabilities set

An explanation about these specific types will follow. Let's start with some example output that you may get on your system.

```
CapInh: 0000000000000000
CapPrm: 0000003fffffffff
CapEff: 0000003fffffffff
CapBnd: 0000003fffffffff
CapAmb: 0000000000000000
```

These hexadecimal numbers don't make sense. Using the capsh utility we can decode them into the capabilities name.

```
capsh --decode=0000003fffffffff
```

Although that works, there is another and easier way. To see the capabilities of a running process, simply use the **getpcaps** tool followed by its process ID (PID). You can also provide a list of process IDs.

```
getpcaps 1234
```

The *getpcaps* tool uses the **capget()** system call to query the available capabilities for a particular thread. This system call only needs to provide the PID to obtain more information.

```
capget({_LINUX_CAPABILITY_VERSION_3, 1234},
{CAP_CHOWN|CAP_DAC_OVERRIDE|CAP_DAC_READ_
SEARCH|CAP_FOWNER|CAP_FSETID|CAP_KILL|CAP_SE
TGID|CAP_SETUID|CAP_SETPCAP|CAP_LINUX_IMMUT
ABLE|CAP_NET_BIND_SERVICE|CAP_NET_BROADCAST
|CAP_NET_ADMIN|CAP_NET_RAW|CAP_IPC_LOCK|CAP
_IPC_OWNER|CAP_SYS_MODULE|CAP_SYS_RAWIO|CA
P_SYS_CHROOT|CAP_SYS_PTRACE|CAP_SYS_PACCT|C
AP_SYS_ADMIN|CAP_SYS_BOOT|CAP_SYS_NICE|CAP_
SYS_RESOURCE|CAP_SYS_TIME|CAP_SYS_TTY_CONFI
G|CAP_MKNOD|CAP_LEASE|CAP_AUDIT_WRITE|CAP_
AUDIT_CONTROL|CAP_SETFCAP|CAP_MAC_OVERRID
E|CAP_MAC_ADMIN|CAP_SYSLOG|CAP_WAKE_ALAR
M|CAP_BLOCK_SUSPEND|CAP_AUDIT_READ,
CAP_CHOWN|CAP_DAC_OVERRIDE|CAP_DAC_READ_
SEARCH|CAP_FOWNER|CAP_FSETID|CAP_KILL|CAP_SE
TGID|CAP_SETUID|CAP_SETPCAP|CAP_LINUX_IMMUT
ABLE|CAP_NET_BIND_SERVICE|CAP_NET_BROADCAST
|CAP_NET_ADMIN|CAP_NET_RAW|CAP_IPC_LOCK|CAP
_IPC_OWNER|CAP_SYS_MODULE|CAP_SYS_RAWIO|CA
P_SYS_CHROOT|CAP_SYS_PTRACE|CAP_SYS_PACCT|C
AP_SYS_ADMIN|CAP_SYS_BOOT|CAP_SYS_NICE|CAP_
SYS_RESOURCE|CAP_SYS_TIME|CAP_SYS_TTY_CONFI
G|CAP_MKNOD|CAP_LEASE|CAP_AUDIT_WRITE|CAP_
AUDIT_CONTROL|CAP_SETFCAP|CAP_MAC_OVERRID
E|CAP_MAC_ADMIN|CAP_SYSLOG|CAP_WAKE_ALAR
M|CAP_BLOCK_SUSPEND|CAP_AUDIT_READ, 0}) = 0
```

In this output, we see also **version 3** of the capabilities. This was added since Linux version 2.6.26.

It is also interesting to see the capabilities of a set of processes that have a relationship.

```
getpcaps $(pgrep nginx)
```

If you run this on a system with nginx, you will see something special. The PID of the master process has capabilities, while the child processes or workers have none. This is because only the master requires the special permissions, like listening to a network port. The child processes then can do the work, like answering HTTP requests.

### Capability bounding set

The *capability bounding set* defines the upper level of available capabilities. During the time a process runs, no capabilities can be added to this list. Only the capabilities in the bounding set can be added to the inheritable set, which uses the capset() system call. If a capability is dropped from the boundary set, that process or its children can no longer have access to it.

## Capabilities overview

Let's have a look at some of the available capabilities and what they do.

### CAP_CHOWN

If you ever changed the owner of a file, you will be familiar with the *chown* command. This capability provides the privilege to do this. It allows changing both the owner as the group. Good to know is that this only applies when **_POSIX_CHOWN_RESTRICTED** is active, which is true on most Linux systems. By using the getconf command we can validate this.

```
getconf -a | grep _POSIX_CHOWN_RESTRICTED
```

## Limiting the capabilities for processes

You can test what happens when a particular capability is dropped by using the capsh utility. This is a way to see what capabilities a particular program may need to function correctly. The *capsh* command can run a particular process and restrict the set of available capabilities.

Run the same command with one single ping to our local system.

```
 capsh --print -- -c "/bin/ping -c 1
localhost"
```

## Dropping capabilities with capsh

If we drop the CAP_NET_RAW capabilities for *ping*, then the ping utility should no longer work.

```
 capsh --drop=cap_net_raw --print -- -c
"/bin/ping -c 1 localhost"
```

Besides the output of *capsh* itself, the *ping* command itself should also raise an error.

```
ping: icmp open socket: Operation not permitted
```

The error clearly shows that the ping command is not allowed to open an ICMP socket. Now we know for sure that this works as expected.

## Binaries with setuid bit

Capabilities are a great way to replace binaries with the setuid bit set. This special bit gives users full root permissions under the context of that process. As you can imagine, if the program contains a flaw, the non-privileged user can “break out” and become the equivalent of the root user.

Still many Linux distributions use the setuid on several binaries, while capabilities can replace the bit.

## Conclusion

Capabilities are a great way to split up root permissions and hand out some permissions to non-privileged users. Unfortunately, still many binaries have the setuid bit set, while they should be replaced with capabilities instead.

**Related article**: Linux Capabilities: Hardening Linux binaries by removing setuid

101 capabilities linux setuid tutorial

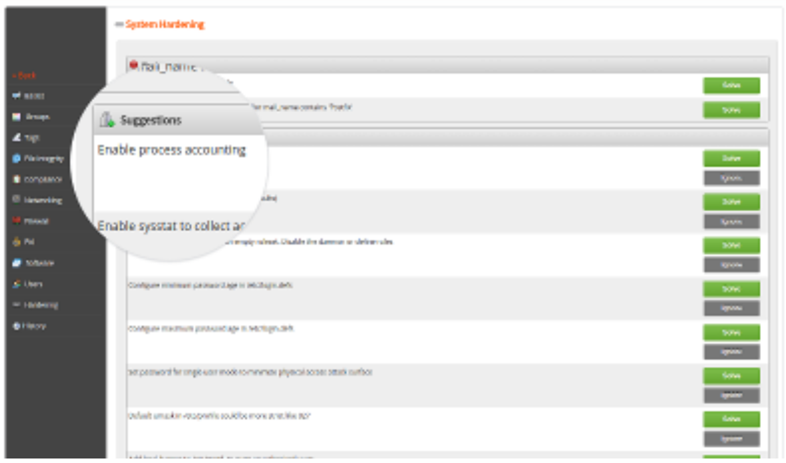
## One more thing...

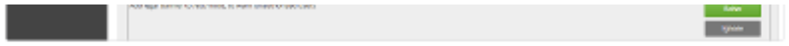
### Keep learning

So you are interested in Linux security? Join the **Linux Security Expert** training program, a practical and lab-based training ground. For those who want to become (or stay) a Linux security expert.

See training package

---





« Protect against ptrace of processes: kernel.yama.ptrace_scope

Alternative for netstat: ss tool »

**Continue reading**

Livepatch: Linux kernel updates without rebooting

How to secure a Linux system

The state of Linux security in 2017

Linux security myths

**Leave a Reply**

Your email address will not be published. Required fields are marked *

**Comment**

**Name ***

**Email ***

**Website**

**Post Comment**

**Linux and UNIX security automation**

Lynis is a free and open source security scanner. It helps with testing the defenses of your Linux, macOS, and Unix systems.

**Recent Posts**

- Major release: Lynis 3.x
- The 101 of ELF files on Linux: Understanding and Analysis
- How to promote your open source project
- OpenSSH

**Contact**

This blog is part of our mission to share valuable tips about Linux security. We are reachable via @linuxaudit

**Company details**

Typical use-cases for this software include system hardening, vulnerability scanning, and checking compliance with security standards (PCI-DSS, ISO27001, etc).

Download

security and hardening

- Livepatch: Linux kernel updates without rebooting
- How to secure a Linux system

CISOfy
De Klok 28,
5251 DN,
Vlijmen, The Netherlands
+31-20-2260055

Website:
https://cisofy.com

A Linux security blog about system auditing, server hardening, and compliance.