Linux Capabilities and Namespaces

Capabilities

Michael Kerrisk, man7.org © 2025

August 2025

mtk@man7.org

Outline

Rev: # caf166f4161b

Outline

Rationale for capabilities

- Traditional UNIX privilege model divides users into two groups:
  - Normal users, subject to privilege checking based on UID and GIDs
  - Effective UID 0 (superuser) bypasses many of those checks
- Coarse granularity is a problem:
  - E.g., to give a process power to change system time, we must also give it power to bypass file permission checks
    - ⇒ No limit on possible damage if program is compromised

man7.org

[TLPI §39.1]

Rationale for capabilities

- Capabilities divide power of superuser into small pieces
  - 41 capabilities, as at Linux 6.16
  - Traditional superuser == process that has full set of capabilities
- Goal: replace set-UID-*root* programs with programs that have capabilities
  - Compromise in set-UID-*root* binary ⇒ very dangerous
  - Compromise in binary with file capabilities ⇒ less dangerous
- Capabilities are not specified by POSIX
  - A 1990s standardization effort was ultimately abandoned
  - Some other implementations have something similar
    - E.g., Solaris, FreeBSD

man7.org

A selection of Linux capabilities

Capability	Permits process to
CAP_CHOWN	Make arbitrary changes to file UIDs and GIDs
CAP_DAC_OVERRIDE	Bypass file RWX permission checks
CAP_DAC_READ_SEARCH	Bypass file R and directory X permission checks
CAP_IPC_LOCK	Lock memory
CAP_FOWNER	*chmod()*, *utime()*, set ACLs on arbitrary files
CAP_KILL	Send signals to arbitrary processes
CAP_NET_ADMIN	Various network-related operations
CAP_SETFCAP	Set file capabilities
CAP_SETGID	Make arbitrary changes to process's (own) GIDs
CAP_SETPCAP	Make changes to process's (own) capabilities
CAP_SETUID	Make arbitrary changes to process's (own) UIDs
CAP_SYS_ADMIN	Perform a wide range of system admin tasks
CAP_SYS_BOOT	Reboot the system
CAP_SYS_NICE	Change process priority and scheduling policy
CAP_SYS_MODULE	Load and unload kernel modules
CAP_SYS_RESOURCE	Raise process resource limits, override some limits
CAP_SYS_TIME	Modify the system clock

More details: *capabilities(7)* manual page and TLPI §39.2

man7.org

Supporting capabilities

- To support implementation of capabilities, the kernel must:
  1. **Check process capabilities** for each privileged operation
     - Cf. traditional check: is process's effective UID 0?
  2. Provide **system calls** allowing a process to modify its capabilities
     - So process can *raise* (add) and *lower* (remove) capabilities
     - (Capabilities analog of *set*id()* calls)
  3. Support **attaching capabilities to executable files**
     - When file is executed, process gains attached capabilities
     - (Capabilities analog of set-UID-*root* program)
- Implemented as follows:
  - Support for first two pieces available since Linux 2.2 (1999)
  - Support for file capabilities added in Linux 2.6.24 (2008)
    - (Delay due to design concerns rather than technical reasons)

[TLPI §39.4]

Outline

Process and file capabilities

- Processes and (binary) files can each have capabilities
- **Process capabilities** define power of process to do privileged operations
  - Traditional superuser == process that has **all** capabilities
- **File capabilities** are a mechanism to give a process capabilities when it execs the file
  - Stored in `security.capability` extended attribute
    - (File metadata; `getfattr -m - <file>`)

man7.org

[TLPI §39.3]

Process and file capability sets

- Capability set: bit mask representing a group of capabilities
- Each **process**† has 3‡ capability sets:
  - Permitted
  - Effective
  - Inheritable

†In truth, capabilities are a per-thread attribute
‡In truth, there are more capability sets

- An **executable file** may have 3 associated capability sets:
  - Permitted
  - Effective
  - Inheritable
- ⚠ Inheritable capabilities are little used; can mostly ignore

man7.org

Viewing process capabilities

- `/proc/PID/status` fields (hexadecimal bit masks):

```
$ cat /proc/4091/status
...
CapInh: 0000000000000000
CapPrm: 0000000000200020
CapEff: 0000000000000000
```

  - See `<sys/capability.h>` for capability bit numbers
    - Here: `CAP_KILL` (bit 5), `CAP_SYS_ADMIN` (bit 21)
- *getpcaps(1)* (part of *libcap* package):

```
$ getpcaps 4091
Capabilities for `4091': = cap_kill,cap_sys_admin+p
```

  - More readable notation, but a little tricky to interpret
  - Here, single '=' means all sets are empty
- *capsh(1)* can be used to decode hex masks:

```
$ capsh --decode=200020
0x0000000000200020=cap_kill,cap_sys_admin
```

Modifying process capabilities

- A process can modify its capability sets by:
  - **Raising** a capability (adding it to set)
    - Synonyms: add, enable
  - **Lowering** a capability (removing it from set)
    - Synonyms: **drop**, **clear**, remove, disable
  - Mostly, we'll defer discussion of the APIs until later
- There are various rules about changes a process can make to its capability sets

Outline

Process permitted and effective capabilities

- *Permitted* : capabilities that process *may* employ
  - "Upper bound" on effective capability set
  - Once dropped from permitted set, a capability can't be reacquired
    - (But see discussion of *execve()* later)
  - Can't drop while capability is also in effective set
- *Effective* : capabilities that are currently in effect for process
  - I.e., capabilities that are examined when checking if a process can perform a privileged operation
  - Capabilities can be dropped from effective set and reacquired
    - Operate with least privilege....
    - Reacquisition possible only if capability is in permitted set

[TLPI §39.3.3]

File permitted and effective capabilities

- *Permitted* : a set of capabilities that may be added to process's permitted set during *exec()*
- *Effective* : ⚠ a **single bit** that determines state of process's new effective set after *exec()* :
  - If set, all capabilities in process's new permitted set are also enabled in effective set
    - Useful for so-called *capabilities-dumb* applications
  - If not set, process's new effective set is empty
- File capabilities allow implementation of capabilities analog of set-UID-*root* program
  - Notable difference: setting effective bit off allows a program to start in **unprivileged** state
    - Set-UID/set-GID programs always start in **privileged** state

[TLPI §39.3.4]

Outline

Setting and viewing file capabilities from the shell

- *setcap(8)* sets capabilities on files
  - Requires privilege (`CAP_SETFCAP` – "set file capabilities")
  - E.g., to set `CAP_SYS_TIME` as a permitted and effective capability on an executable file:

```
$ cp /bin/date mydate
$ sudo setcap "cap_sys_time=pe" mydate
```

- *getcap(8)* displays capabilities associated with a file

```
$ getcap mydate
mydate = cap_sys_time+ep
```

- *filecap(8)* searches for files that have capabilities:

```
$ filecap                              # Report files in $PATH
$ sudo filecap -a 2> /dev/null         # Check all files on system
                          # "2>" to discard "not supported" messages
```

  - *filecap* is part of the *libcap-ng-utils* package

[TLPI §39.3.6]

cap/demo_file_caps.c

```
int main(int argc, char *argv[]) {
    cap_t caps = cap_get_proc();       /* Fetch process capabilities */
    char *str = cap_to_text(caps, NULL);
    printf("Capabilities: %s\n", str);
    ...
    if (argc > 1) {
        fd = open(argv[1], O_RDONLY);
        if (fd >= 0)
            printf("Successfully opened %s\n", argv[1]);
        else
            printf("Open failed: %s\n", strerror(errno));
    }
    exit(EXIT_SUCCESS);
}
```

- Display process capabilities
- Report result of opening file named in *argv[1]* (if present)

cap/demo_file_caps.c

```
$ id -u
1000
$ cc -o demo_file_caps demo_file_caps.c -lcap
$ ./demo_file_caps /etc/shadow
Capabilities: =
Open failed: Permission denied
$ ls -l /etc/shadow
----------. 1 root root 1974 Mar 15 08:09 /etc/shadow
```

- All steps in demos are done from unprivileged user ID 1000
- Binary has no capabilities ⇒ process gains no capabilities
  - "=" in the output means "all capability sets empty"
- *open()* of `/etc/shadow` fails
  - Because `/etc/shadow` is readable only by privileged process
  - Process needs `CAP_DAC_READ_SEARCH` capability

cap/demo_file_caps.c

```
$ sudo setcap cap_dac_read_search=p demo_file_caps
$ ./demo_file_caps /etc/shadow
Capabilities: = cap_dac_read_search+p
Open failed: Permission denied
```

- Binary confers permitted capability to process, but capability is not effective
- Process gains capability in permitted set
- *open()* of `/etc/shadow` fails
  - Because `CAP_DAC_READ_SEARCH` is not in *effective* set

cap/demo_file_caps.c

```
$ sudo setcap cap_dac_read_search=pe demo_file_caps
$ ./demo_file_caps /etc/shadow
Capabilities: = cap_dac_read_search+ep
Successfully opened /etc/shadow
```

- Binary confers permitted capability and has effective bit on
- Process gains capability in permitted and effective sets
- *open()* of `/etc/shadow` succeeds

Outline

Notes for online practical sessions

- Small groups in **breakout rooms**
  - Write a note into Slack if you have a preferred group
- **We will go faster, if groups collaborate** on solving the exercise(s)
  - You can **share a screen** in your room
- I will circulate regularly between rooms to answer questions
- Zoom has an "**Ask for help**" button...
- **Keep an eye on the `#general` Slack channel**
  - Perhaps with further info about exercise;
  - Or a note that the exercise merges into a break
- When your room has finished, write a message in the Slack channel: **"***** Room X has finished *****"**
  - Then I have an idea of how many people have finished

man7.org

Shared screen etiquette

- It may help your colleagues if you **use a larger than normal font!**
  - In many environments (e.g., *xterm*, *VS Code*), we can adjust the font size with `Control`+`Shift`+"`+`" and `Control`+"`-`"
  - Or (e.g., *emacs*) hold down `Control` key and use mouse wheel
- **Long shell prompts** make reading your shell session difficult
  - Use `PS1='$ '` or `PS1='# '`
- **Low contrast** color themes are difficult to read; change this if you can
- Turn on **line numbering** in your editor
  - In *vim* use: `:set number`
  - In *emacs* use: `M-x display-line-numbers-mode <RETURN>`
    - `M-x` means `Left-Alt`+`x`
- For collaborative editing, **relative line-numbering is evil**....
  - In *vim* use: `:set nornu`
  - In *emacs*, the following should suffice:

```
M-: (display-line-numbers-mode) <RETURN>
M-: (setq display-line-numbers 'absolute) <RETURN>
```

    - `M-:` means `Left-Alt`+`Shift`+`:`

man7.org

Using *tmate* in in-person practical sessions

In order to share an X-term session with others, do the following:

- Enter the command *tmate* in an X-term, and you'll see the following:

```
$ tmate
...
Connecting to ssh.tmate.io...
Note: clear your terminal before sharing readonly access
web session read only: ...
ssh session read only: ssh S0mErAnD0m5Tr1Ng@lon1.tmate.io
web session: ...
ssh session: ssh S0mEoTheRrAnD0m5Tr1Ng@lon1.tmate.io
```

- Share last "ssh" string with colleague(s) via Slack or another channel
  - Or: "ssh session read only" string gives others read-only access
- Your colleagues should paste that string into an X-term...
- Now, you are sharing an X-term session in which anyone can type
  - Any "mate" can cut the connection to the session with the 3-character sequence <ENTER> ~ .
- To see above message again: `tmate show-messages`

man7.org

Exercises

1. Compile and run the `cap/demo_file_caps` program, without adding any capabilities to the file, and verify that when you run the binary, the process has no capabilities:

```
$ cc -o demo_file_caps demo_file_caps.c -lcap
$ ./demo_file_caps
```

   - The string "=" means all capability sets empty.

2. Now make the binary set-UID-*root*:

```
$ sudo chown root demo_file_caps  # Change owner to root
$ sudo chmod u+s demo_file_caps   # Turn on set-UID bit
$ ls -l demo_file_caps            # Verify
-rwsr-xr-x. 1 root mtk 8624 Oct  1 13:19 demo_file_caps
```

3. Run the binary and verify that the process gains all capabilities. (The string "=ep" means "all capabilities in the permitted + effective sets".)
   - If the process does not gain all capabilities, check whether the filesystem is mounted with the `nosuid` option (`findmnt -T <dir>`). If it is, either remount the filesystem without that option or do the exercise on a filesystem that is not mounted with `nosuid` (typically, `/tmp` should work).
4. Take the existing set-UID-*root* binary, add a permitted capability to it, and set the effective capability bit:

```
$ sudo setcap cap_dac_read_search=pe demo_file_caps
```

man7.org

Exercises

5. When you now run the binary, what capabilities does the process have?

```
$ ./demo_file_caps
```

6. Suppose you assign empty capability sets to the binary. When you execute the binary, what capabilities does the process then have?

```
$ sudo setcap = demo_file_caps
$ getcap demo_file_caps
$ ./demo_file_caps
```

7. Use the following command to remove capabilities from the binary and verify that when executed, the binary once more grants all capabilities to the process:

```
$ sudo setcap -r demo_file_caps
$ getcap demo_file_caps
$ ./demo_file_caps
```

8. Use the following command to find the binaries on your system that have capabilities attached:

```
$ sudo filecap -a 2> /dev/null
```

Write the name of your distribution, and paste the list of binaries into the Slack channel.

Outline

Capabilities-dumb and capabilities-aware applications

- **Capabilities-dumb** application:
  - (Typically) an existing set-UID-*root* binary whose code we can't change
    - Thus, binary does not know to use capabilities APIs (Binary simply uses traditional *set*uid()* APIs)
  - But want to make legacy binary less dangerous than set-UID-*root*
- Converse is **capabilities-aware** application
  - Program that was written/modified to use capabilities APIs
  - Set binary up with file effective capability bit **off**
  - Program "knows" it must use capabilities APIs to enable effective capabilities

Adding capabilities to a capabilities-dumb application

To convert existing set-UID-*root* binary to use file capabilities:

- Setup:
  - Binary remains set-UID-*root*
  - Enable a subset of file permitted capabilities + **set effective bit on**
    - I.e., **capabilities-dumb == binary with effective bit on**
  - (Note: code of binary isn't changed)
- Operation:
  - When binary is executed, process gets (just the) specified subset of capabilities in its permitted and effective sets
    - IOW: file-capabilities override effect of set-UID-*root* bit, which would normally confer **all** capabilities to process
  - Process UID changes between zero and nonzero automatically raise/lower process's capabilities
    - (Covered in more detail later)

How do I work out what capabilities a program needs?

Some possibilities to discover what capabilities are needed by an arbitrary program:

- System call manual pages (section 2) are a good start
  - Look for capability requirements documented in DESCRIPTION or ERRORS
- Run the program (without capabilities) under *strace(1)*:
  - System call failures due to lack of capabilities normally return `EPERM` in *errno*
    - ⚠ But not all `EPERM` errors are due to lack of capabilities
  - If program displays an error message that seems to relate to capabilities, look in trace output for nearby `EPERM` errors
    - You may want to use the *–v* option so that *strace* doesn't abbreviate strings
- In extreme cases, you may need to read kernel source

man7.org

Outline

Textual representation of capabilities

- Both *setcap(8)* and *getcap(8)* work with **textual representations** of capabilities
  - Syntax described in *cap_from_text(3)* manual page
- String read left to right, containing space-separated clauses
  - (The capability sets are initially considered to be empty)
- Clause: *caps-list operator flags* [*operator flags*] ...
  - *caps-list* is comma-separated list of capability names, or *all*
  - *operator* is `+`, `-`, or `=`
  - *flags* is zero or more of *p* (permitted), *e* (effective), or *i* (inheritable)
  - Clause can contain multiple [*operator flags*] parts:
    - E.g., `"cap_sys_time+p-i"` (is same as `"cap_sys_time+p cap_sys_time-i"`)

man7.org

Textual representation of capabilities

Operators:

- `+` operator: raise capabilities in sets specified by *flags*
- `-` operator: lower capabilities in sets specified by *flags*
- `=` operator:
  - Raise capabilities in sets specified by *flags*; lower those capabilities in remaining sets
    - So, `"CAP_KILL=p"` is same as `"CAP_KILL+p-ie"`
  - *caps-list* can be omitted; defaults to *all*
  - *flags* can be omitted ⇒ clear capabilities from all sets
    ⇒ Thus: `"="` means clear all capabilities in all sets
- What does `"=p cap_kill,cap_sys_admin+e"` mean?
  - All capabilities in permitted set, plus `CAP_KILL` and `CAP_SYS_ADMIN` in effective set

man7.org

Outline

Exercises

1. What capability bits are enabled by each of the following text-form capability specifications?
   - `"=p"`
   - `"="`
   - `"cap_setuid=p cap_sys_time+pie"`
   - `"=p cap_kill-p"`
   - `"cap_kill=p = cap_sys_admin+pe"`
   - `"cap_chown=i cap_kill=pe cap_setfcap,cap_chown=p"`
2. The program `cap/cap_text.c` takes a single command-line argument, which is a text-form capability string. It converts that string to an in-memory representation and then iterates through the set of all capabilities, printing out the state of each capability within the permitted, effective, and inheritable sets. It thus provides a method of verifying your interpretation of text-form capability strings. Try supplying each of the above strings as an argument to the program (**remember to enclose the entire string in quotes!**) and check the results against your answers to the previous exercise.

Exercises

3. The *pscap* command (part of *libcap-ng*) displays a list of the processes on the system that have permitted, effective, or inheritable capabilities. In addition to showing the PPID, PID, UID, command, and capabilities for each of the displayed processes, output lines may be annotated with one of the following characters:
   - `+`: the process has a nonempty capability bounding set
   - `@`: the process has a nonempty ambient capability set (later)
   - `*`: the process is in a child user namespace (later)

   Use the *pscap* command to display the processes that have capabilities on your system. (By default, PID 1 (*init*) is excluded from the list; use the *–a* option to include PID 1, if you wish.)

This page intentionally blank