Linux Capabilities and Namespaces

Capabilities

Michael Kerrisk, man7.org © 2025

January 2025

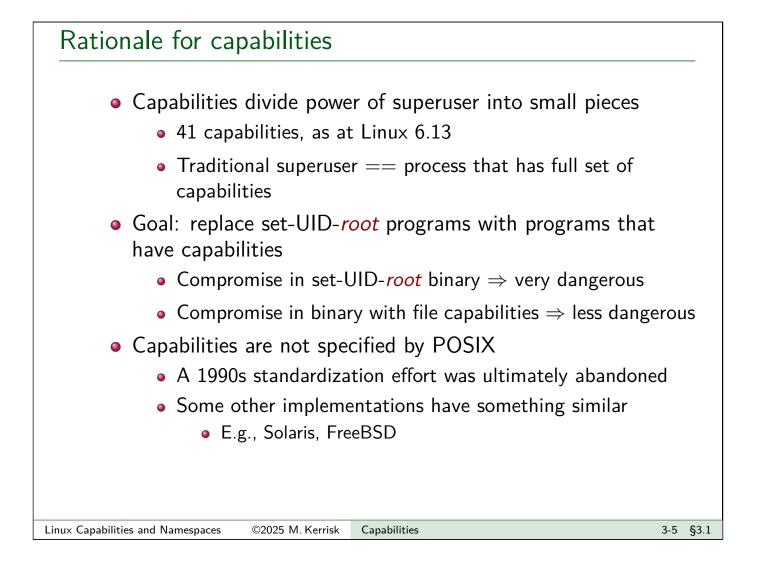
mtk@man7.org

Outline Rev: #80	17fc39ab521
3 Capabilities	3-1
3.1 Overview	3-3
3.2 Process and file capabilities	3-8
3.3 Permitted and effective capabilities	3-14
3.4 Setting and viewing file capabilities	3-18
3.5 Exercises	3-25
3.6 Capabilities-dumb and capabilities-aware applications	3-31
3.7 Text-form capabilities	3-35
3.8 Exercises	3-38

3 Capabilities	3-1
3.1 Overview	3-3
3.2 Process and file capabilities	3-8
3.3 Permitted and effective capabilities	3-14
3.4 Setting and viewing file capabilities	3-18
3.5 Exercises	3-25
3.6 Capabilities-dumb and capabilities-aware applications	3-31
3.7 Text-form capabilities	3-35
3.8 Exercises	3-38

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
 - $\bullet \ \Rightarrow$ No limit on possible damage if program is compromised

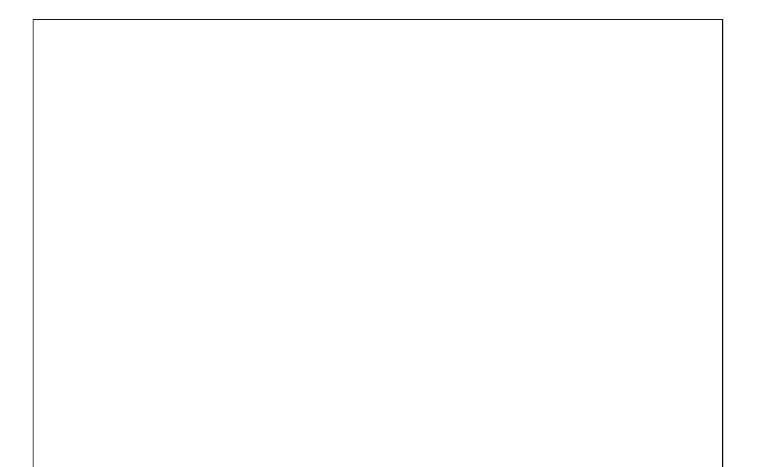


A selection of Linux capabilities

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	<pre>chmod(), utime(), set ACLs on arbitrary files</pre>
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

Supporting capabilities	
 To support implementation of capabilities, the kernel must: Check process capabilities for each privileged operation Cf. traditional check: is process's effective UID 0? 	
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) 	
 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]	
Linux Capabilities and Namespaces ©2025 M. Kerrisk Capabilities 3-7 §3.	.1

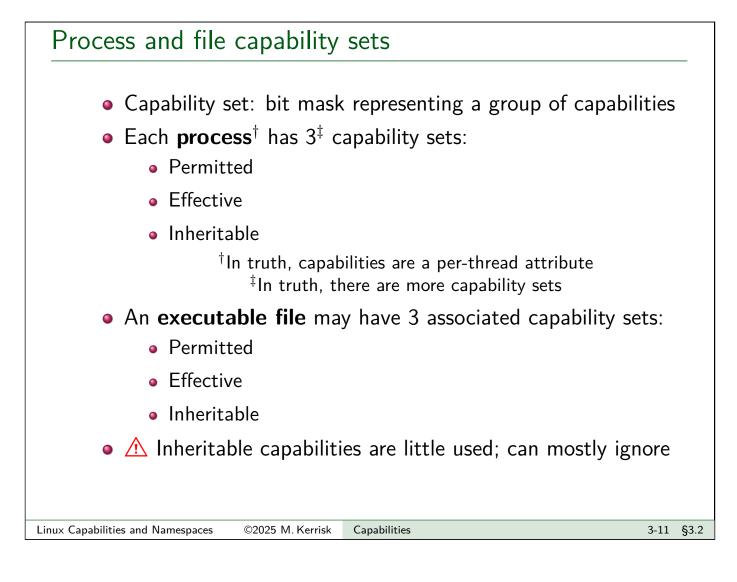


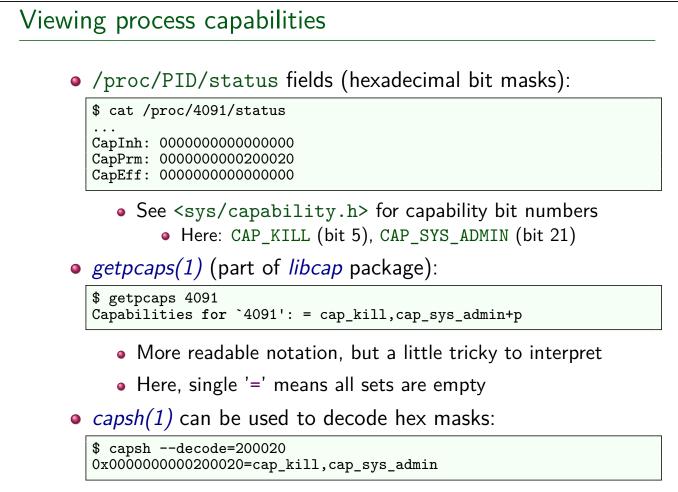
3 Capabilities	3-1
3.1 Overview	3-3
3.2 Process and file capabilities	3-8
3.3 Permitted and effective capabilities	3-14
3.4 Setting and viewing file capabilities	3-18
3.5 Exercises	3-25
3.6 Capabilities-dumb and capabilities-aware applications	3-31
3.7 Text-form capabilities	3-35
3.8 Exercises	3-38

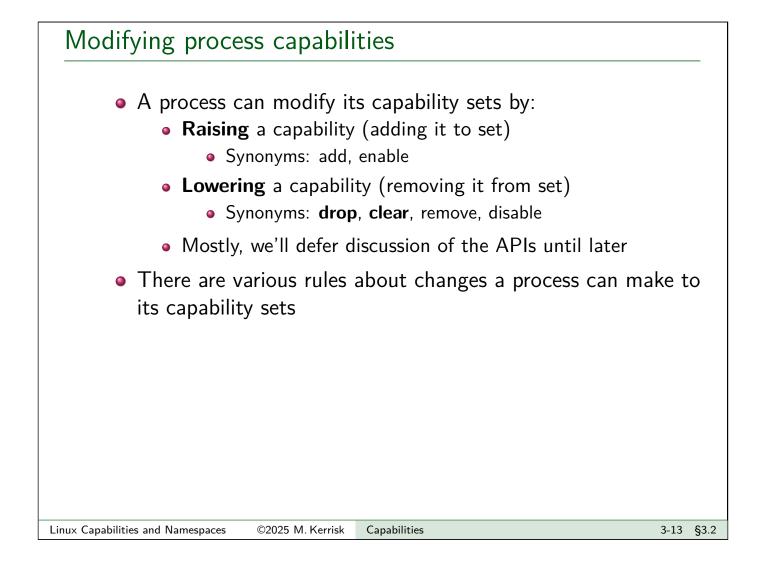
Process and file capabilities

- 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>)

[TLPI §39.3]







3 Capabilities	3-1
	-
3.1 Overview	3-3
3.2 Process and file capabilities	3-8
3.3 Permitted and effective capabilities	3-14
3.4 Setting and viewing file capabilities	3-18
3.5 Exercises	3-25
3.6 Capabilities-dumb and capabilities-aware applications	3-31
3.7 Text-form capabilities	3-35
3.8 Exercises	3-38

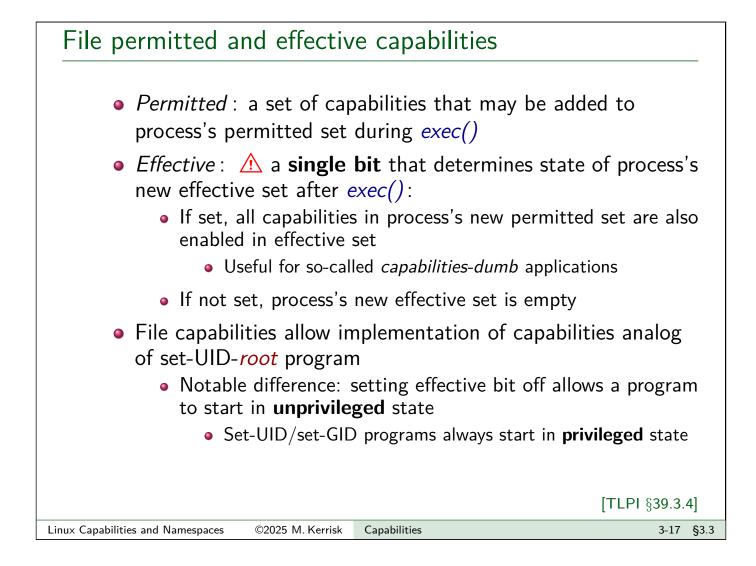
Process permitted and effective capabilities

• *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

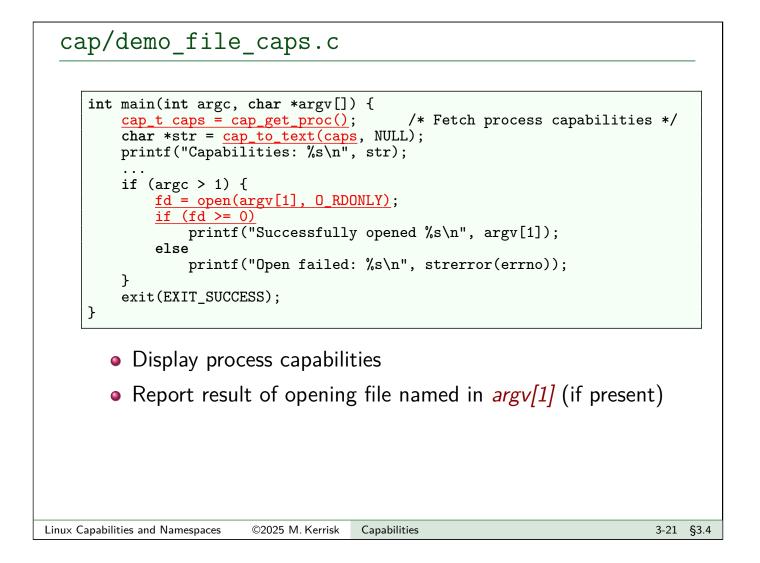
3-16 §3.3

[TLPI §39.3.3]



3 Capabilities	3-1
3.1 Overview	3-3
3.2 Process and file capabilities	3-8
3.3 Permitted and effective capabilities	3-14
3.4 Setting and viewing file capabilities	3-18
3.5 Exercises	3-25
3.6 Capabilities-dumb and capabilities-aware applications	3-31
3.7 Text-form capabilities	3-35
3.8 Exercises	3-38

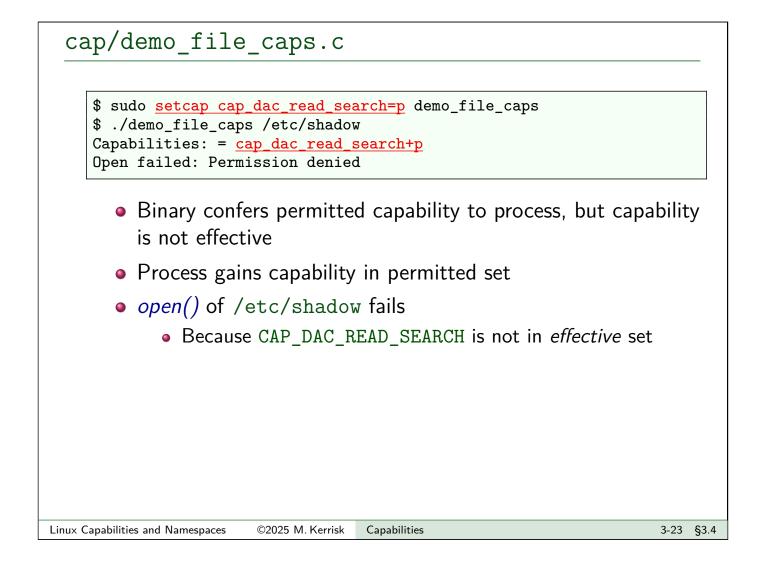
Setting and viewing file capabilities from the shell setcap(8) sets capabilities on files • Requires privilege (CAP_SETFCAP) • E.g., to set CAP_SYS_TIME as a permitted and effective capability on an executable file: \$ cp /bin/date mydate \$ sudo setcap "cap_sys_time=pe" mydate • getcap(8) displays capabilities 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

```
$ 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 \Rightarrow process gains no capabilities
- 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
```

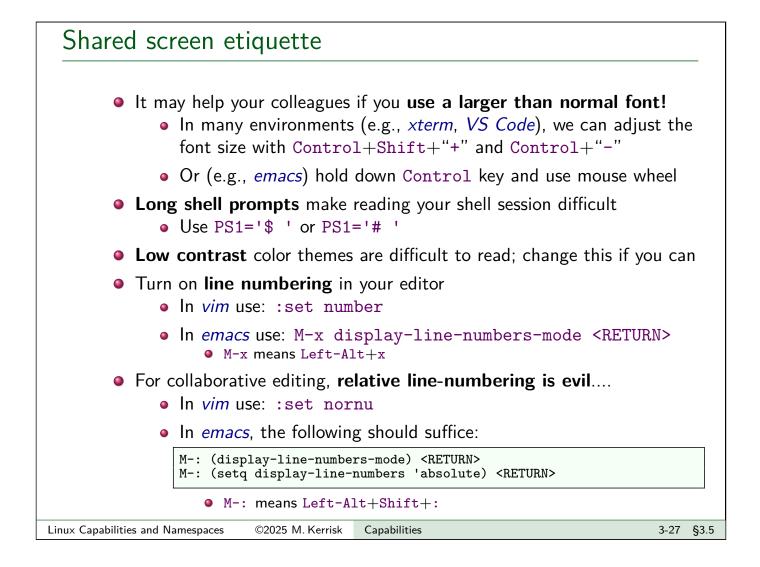
```
$ sudo setcap cap_dac_read_search=pe demo_file_caps
$ ./demo_file_caps /etc/shadow
Capabilities: = cap_dac_read_search+ep
Successfully opened /etc/shadow
```

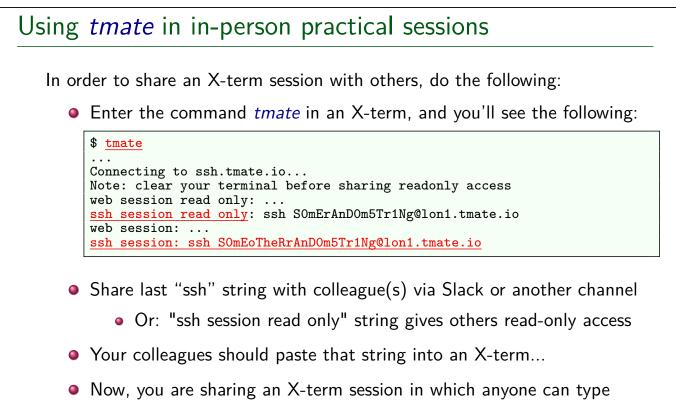
- Binary confers permitted capability and has effective bit on
- Process gains capability in permitted and effective sets
- open() of /etc/shadow succeeds

3 Capabilities	3-1
3.1 Overview	3-3
3.2 Process and file capabilities	3-8
3.3 Permitted and effective capabilities	3-14
3.4 Setting and viewing file capabilities	3-18
3.5 Exercises	3-25
3.6 Capabilities-dumb and capabilities-aware applications	3-31
3.7 Text-form capabilities	3-35
3.8 Exercises	3-38

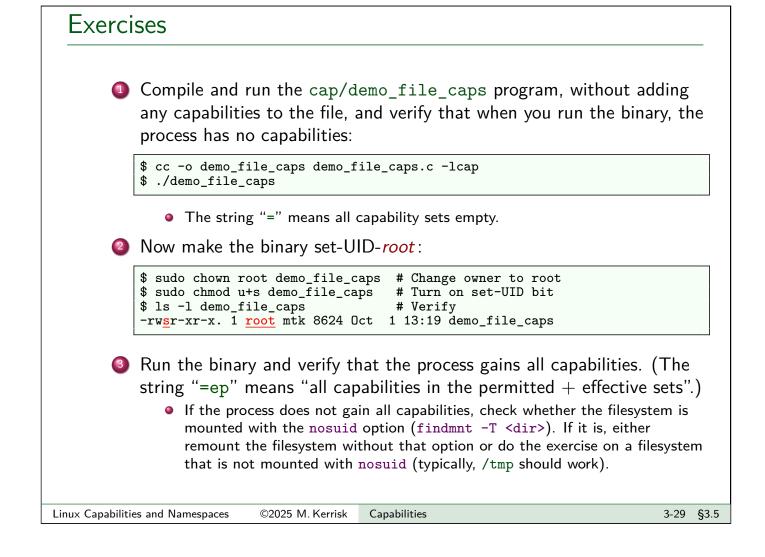
Notes for online practical sessions

- 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





- Any "mate" can cut the connection to the session with the 3-character sequence <ENTER> \sim .
- To see above message again: tmate show-messages



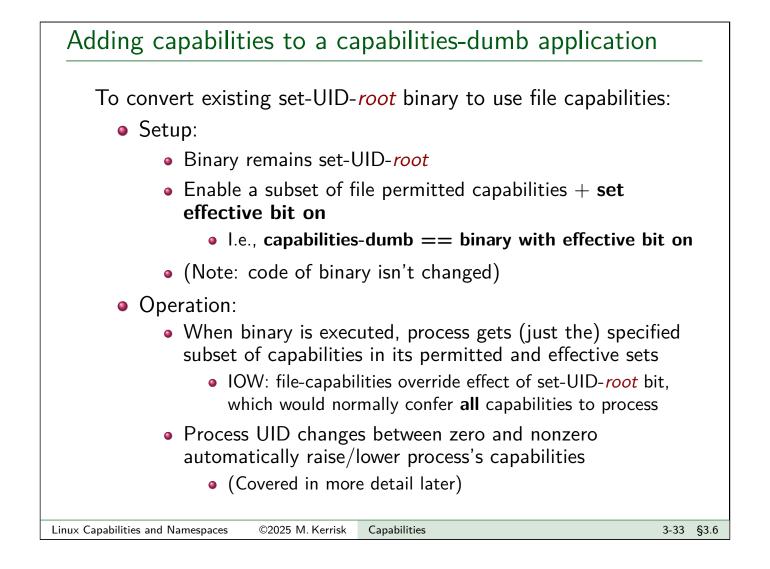
4	Take the existing set-UID- <i>root</i> binary, add a permitted capability to it, and set the effective capability bit:
	<pre>\$ sudo setcap cap_dac_read_search=pe demo_file_caps</pre>
5	When you now run the binary, what capabilities does the process have
6	Suppose you assign empty capability sets to the binary. When you execute the binary, what capabilities does the process then have?
	<pre>\$ sudo setcap = demo_file_caps</pre>
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:
	<pre>\$ sudo setcap -r demo_file_caps</pre>

3 Capabilities	3-1
3.1 Overview	3-3
3.2 Process and file capabilities	3-8
3.3 Permitted and effective capabilities	3-14
3.4 Setting and viewing file capabilities	3-18
3.5 Exercises	3-25
3.6 Capabilities-dumb and capabilities-aware applications	3-31
3.7 Text-form capabilities	3-35
3.8 Exercises	3-38

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



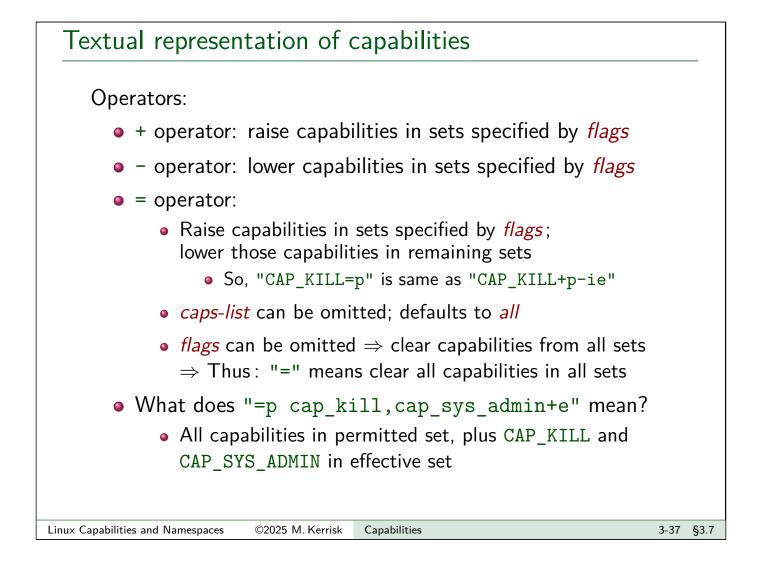
How do I work out what capabilities a program needs? Some possibilities to discover what capabilities are needed by an arbitrary program: System call manual pages (section 2) are a good start Look for capability requirements documented in DESCRIPTION or ERRORS Run the program (without capabilities) under *strace(1)*: System call failures due to lack of capabilities normally return EPERM in *errno*M But not all EPERM errors are due to lack of capabilities

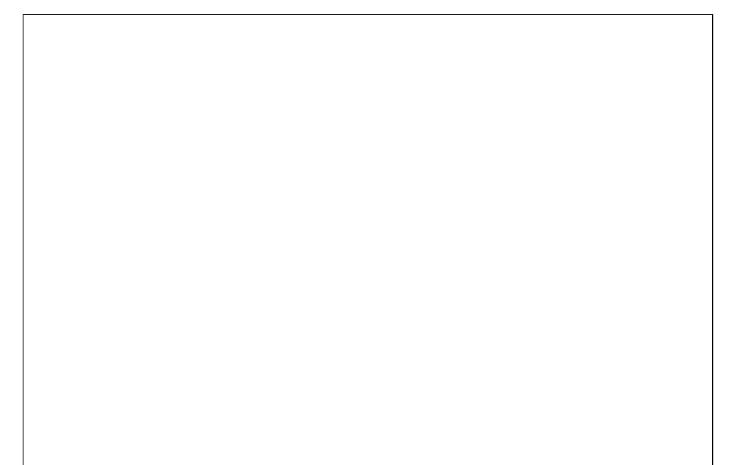
- If program displays an error message that seems to relate to capabilities, look in trace output for nearby EPERM errors
 - You may want to use the -v option so that strace doesn't abbreviate strings
- In extreme cases, you may need to read kernel source

3 Capabilities	3-1
3.1 Overview	3-3
3.2 Process and file capabilities	3-8
3.3 Permitted and effective capabilities	3-14
3.4 Setting and viewing file capabilities	3-18
3.5 Exercises	3-25
3.6 Capabilities-dumb and capabilities-aware applications	3-31
3.7 Text-form capabilities	3-35
3.8 Exercises	3-38

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")





3 Capabilities	3-1
3.1 Overview	3-3
3.2 Process and file capabilities	3-8
3.3 Permitted and effective capabilities	3-14
3.4 Setting and viewing file capabilities	3-18
3.5 Exercises	3-25
3.6 Capabilities-dumb and capabilities-aware applications	3-31
3.7 Text-form capabilities	3-35
3.8 Exercises	3-38

Exercises

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"

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.