Securing & Hardening container hosts

Conf42 - DevSecOps 2021

2nd December 2021

```
$ gcloud container clusters create
```

Container OS

OS designed to run container workloads

COREOS

K3OS

FLAT CAR

PHOTON™

Sayan

Mathieu

FLAT
CAR

CONF42

Alice

Bob

“Rad! This looks quite interesting. After this, we surely need to look into our infrastructure to understand how it looks from the security point of view.

What other improvements can be done to the stack?”

"Oh! Yes. Surely there are.

You know about audit? Let's talk about **audit.**

Linux **audit** framework

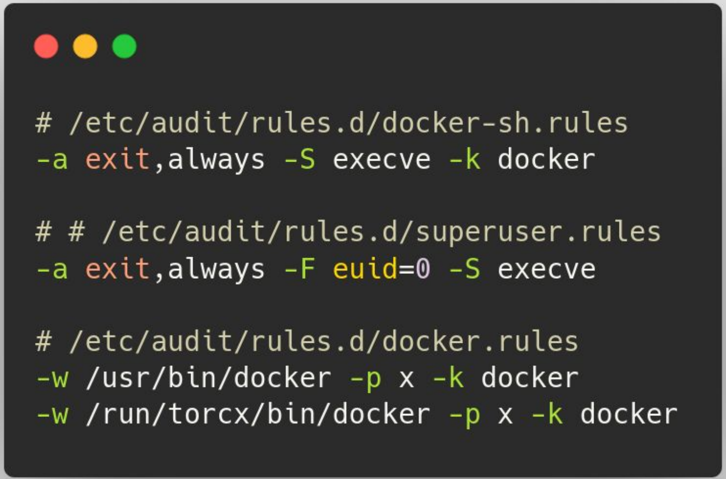
- Collect metadata around a security-related, and create an audit trail
- CAPP-compliant (Controlled Access Protection Profiles) auditing system
- Built to report the exploits

CONF42

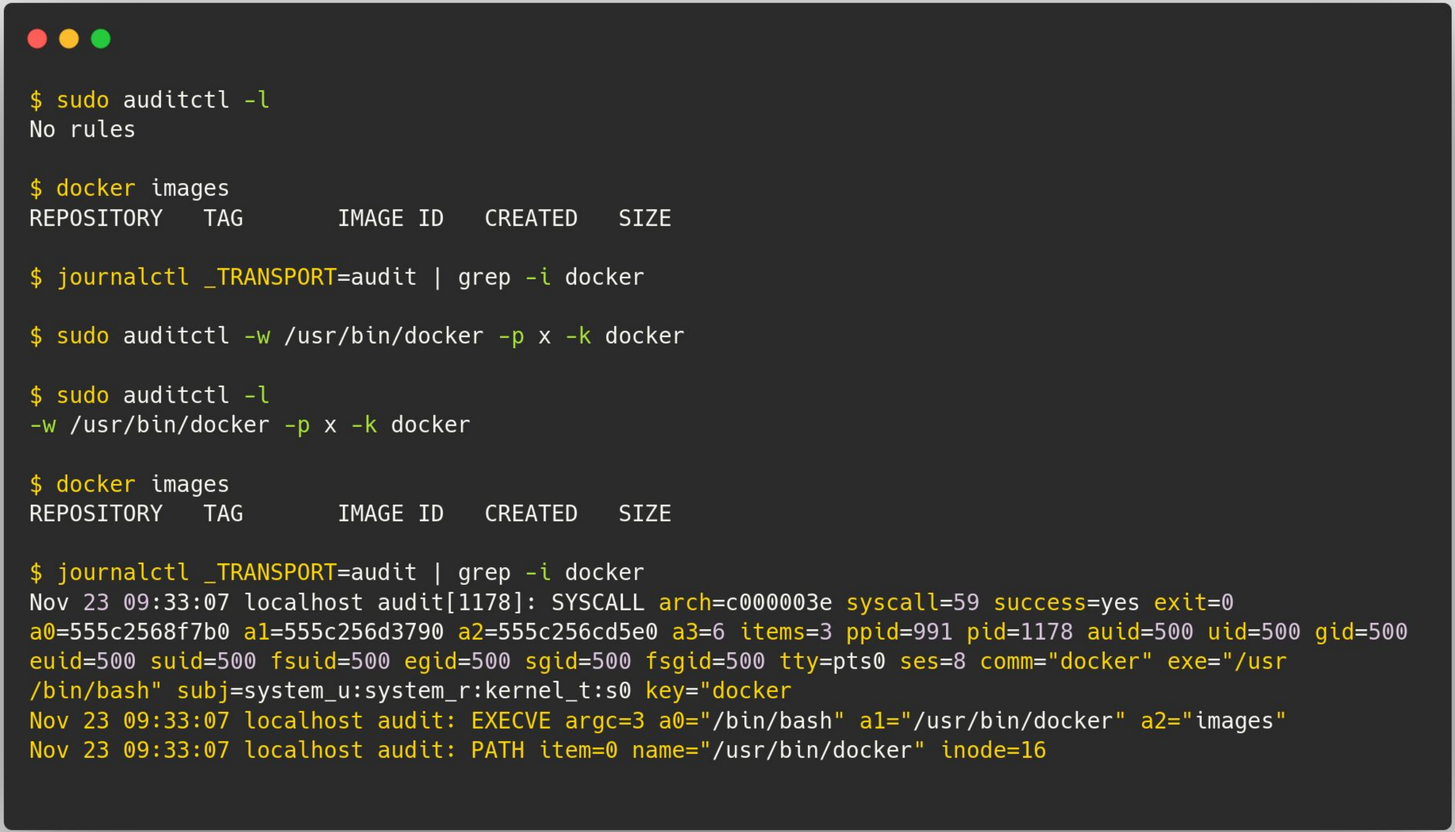
auditctl, audit.rules & the daemon

- /etc/audit.rules holds the rules
- auditctl to interact with daemon
- Rule is a combination of flag

-a exit,always -S execve -k docker



```
# /etc/audit/rules.d/docker-sh.rules
-a exit,always -S execve -k docker

# # /etc/audit/rules.d/superuser.rules
-a exit,always -F euid=0 -S execve

# /etc/audit/rules.d/docker.rules
-w /usr/bin/docker -p x -k docker
-w /run/torcx/bin/docker -p x -k docker
```



```
$ sudo auditctl -l
No rules

$ docker images
REPOSITORY   TAG       IMAGE ID   CREATED   SIZE

$ journalctl _TRANSPORT=audit | grep -i docker

$ sudo auditctl -w /usr/bin/docker -p x -k docker

$ sudo auditctl -l
-w /usr/bin/docker -p x -k docker

$ docker images
REPOSITORY   TAG       IMAGE ID   CREATED   SIZE

$ journalctl _TRANSPORT=audit | grep -i docker
Nov 23 09:33:07 localhost audit[1178]: SYSCALL arch=c000003e syscall=59 success=yes exit=0
a0=555c2568f7b0 a1=555c256d3790 a2=555c256cd5e0 a3=6 items=3 ppid=991 pid=1178 auid=500 uid=500 gid=500
euid=500 suid=500 fsuid=500 egid=500 sgid=500 fsgid=500 tty=pts0 ses=8 comm="docker" exe="/usr
/bin/bash" subj=system_u:system_r:kernel_t:s0 key="docker
Nov 23 09:33:07 localhost audit: EXECVE argc=3 a0="/bin/bash" a1="/usr/bin/docker" a2="images"
Nov 23 09:33:07 localhost audit: PATH item=0 name="/usr/bin/docker" inode=16
```

“Wow, that's nice - We're not blind on the system now.

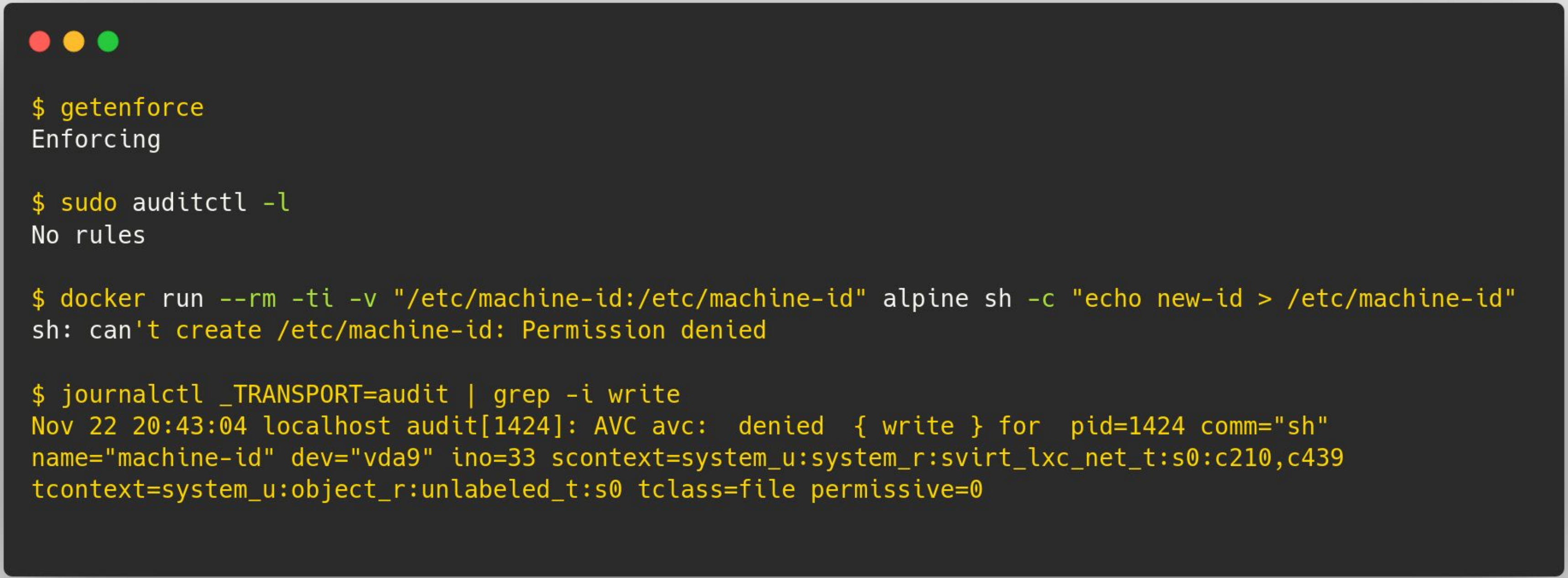
But what can of stuff can we show in the audit ?"

CONF42

“As mentioned, audit is powerful when associated with other tools!
Do you know **SELinux**?"

SELinux

- SELinux defines access controls for the applications, processes, and files on a system. *Red Hat definition*
- Sysadmin granular control
- TL; DR : avoid to deactivate SELinux



```
$ getenforce
Enforcing

$ sudo auditctl -l
No rules

$ docker run --rm -ti -v "/etc/machine-id:/etc/machine-id" alpine sh -c "echo new-id > /etc/machine-id"
sh: can't create /etc/machine-id: Permission denied

$ journalctl _TRANSPORT=audit | grep -i write
Nov 22 20:43:04 localhost audit[1424]: AVC avc:  denied  { write } for  pid=1424 comm="sh"
name="machine-id" dev="vda9" ino=33 scontext=system_u:system_r:svirt_lxc_net_t:s0:c210,c439
tcontext=system_u:object_r:unlabeled_t:s0 tclass=file permissive=0
```

“SELinux is cool - we can log but also prevent action on the system.

We're wondering if we can go deeper in the control ?"

“Yes, SELinux is powerful to define general policies applied on the whole system.

But if you are talking of control, then **capabilities** is your friend."

Capabilities

Linux capabilities provide a fine-grain control of the superuser privileges.

Implementation

Traditional

- *privileged process*
  (run by root or superuser)
  can bypass all kernel permission checks

- *unprivileged processes*
  (run by others)
  require full permission checking

Implementation

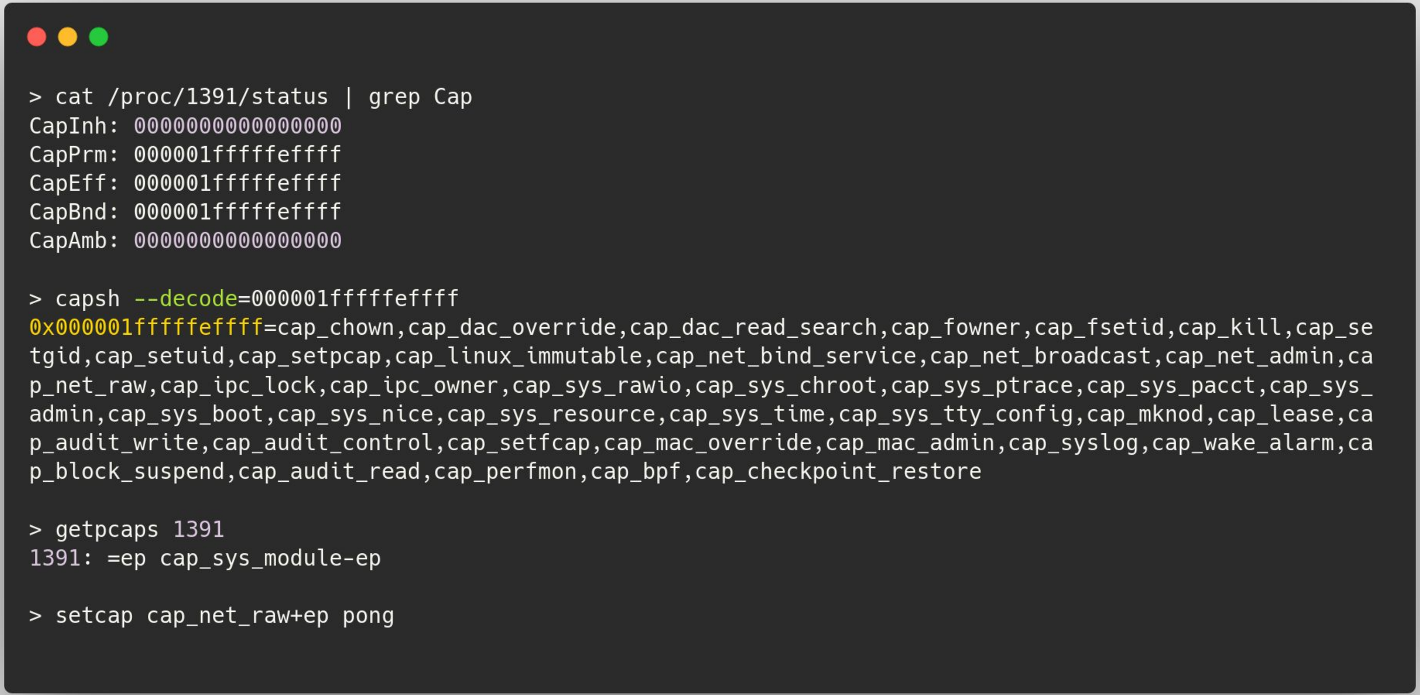
Starting kernel 2.2, Linux divides the privileges traditionally associated with superuser into distinct units, known as **capabilities**, which can be **independently** enabled and disabled.

CONF42

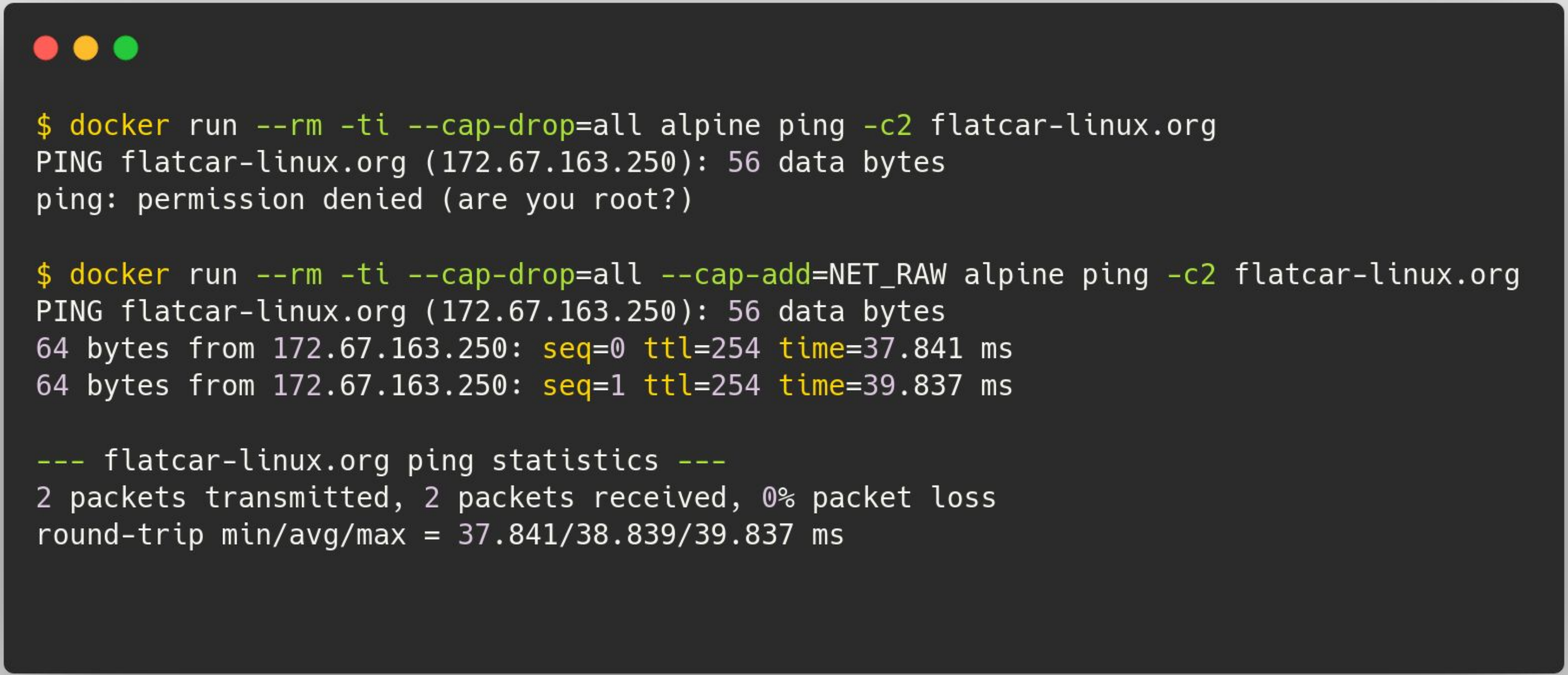
Capabilities Set

Capabilities are per-thread attribute

- Permitted
- Inheritable
- Effective
- Bounding
- Ambient



```
> cat /proc/1391/status | grep Cap
CapInh: 0000000000000000
CapPrm: 000001ffffffeffff
CapEff: 000001ffffffeffff
CapBnd: 000001ffffffeffff
CapAmb: 0000000000000000

> capsh --decode=000001fffffeffff
0x000001fffffeffff=cap_chown,cap_dac_override,cap_dac_read_search,cap_fowner,cap_fsetid,cap_kill,cap_se
tgid,cap_setuid,cap_setpcap,cap_linux_immutable,cap_net_bind_service,cap_net_broadcast,cap_net_admin,ca
p_net_raw,cap_ipc_lock,cap_ipc_owner,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_mknod,cap_lease,ca
p_audit_write,cap_audit_control,cap_setfcap,cap_mac_override,cap_mac_admin,cap_syslog,cap_wake_alarm,ca
p_block_suspend,cap_audit_read,cap_perfmon,cap_bpf,cap_checkpoint_restore

> getpcaps 1391
1391: =ep cap_sys_module-ep

> setcap cap_net_raw+ep pong
```



```
$ docker run --rm -ti --cap-drop=all alpine ping -c2 flatcar-linux.org
PING flatcar-linux.org (172.67.163.250): 56 data bytes
ping: permission denied (are you root?)

$ docker run --rm -ti --cap-drop=all --cap-add=NET_RAW alpine ping -c2 flatcar-linux.org
PING flatcar-linux.org (172.67.163.250): 56 data bytes
64 bytes from 172.67.163.250: seq=0 ttl=254 time=37.841 ms
64 bytes from 172.67.163.250: seq=1 ttl=254 time=39.837 ms

--- flatcar-linux.org ping statistics ---
2 packets transmitted, 2 packets received, 0% packet loss
round-trip min/avg/max = 37.841/38.839/39.837 ms
```

seccomp

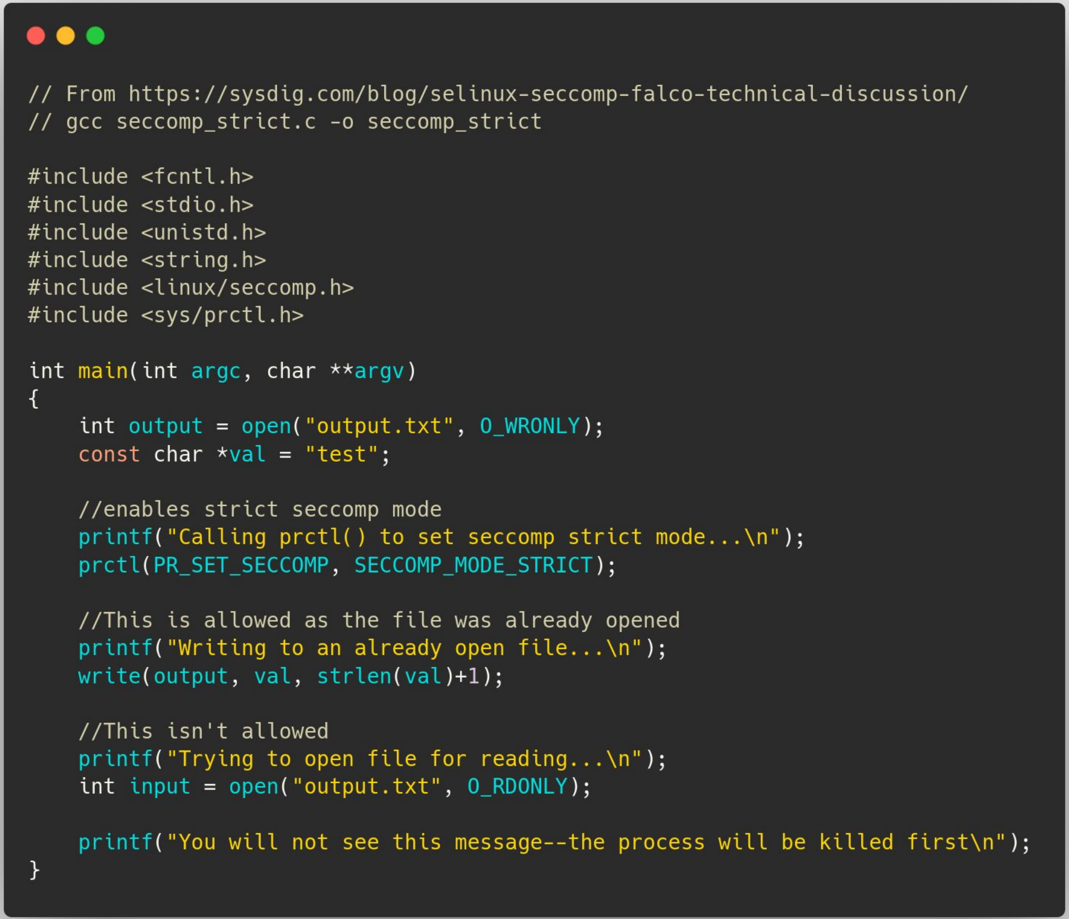
seccomp stands for secure computing mode. Introduced in Linux 2.6.12, it a simple sandboxing tool to filter system calls.

seccomp modes

- *seccomp strict mode*
  seccomp mode is enabled via the **prctl(2)** system call using the PR_SET_SECCOMP

  Only read(), write(), exit() and sigreturn() syscalls are allowed to already open file-descriptors, all other process are killed using SIGKILL.

CONF42



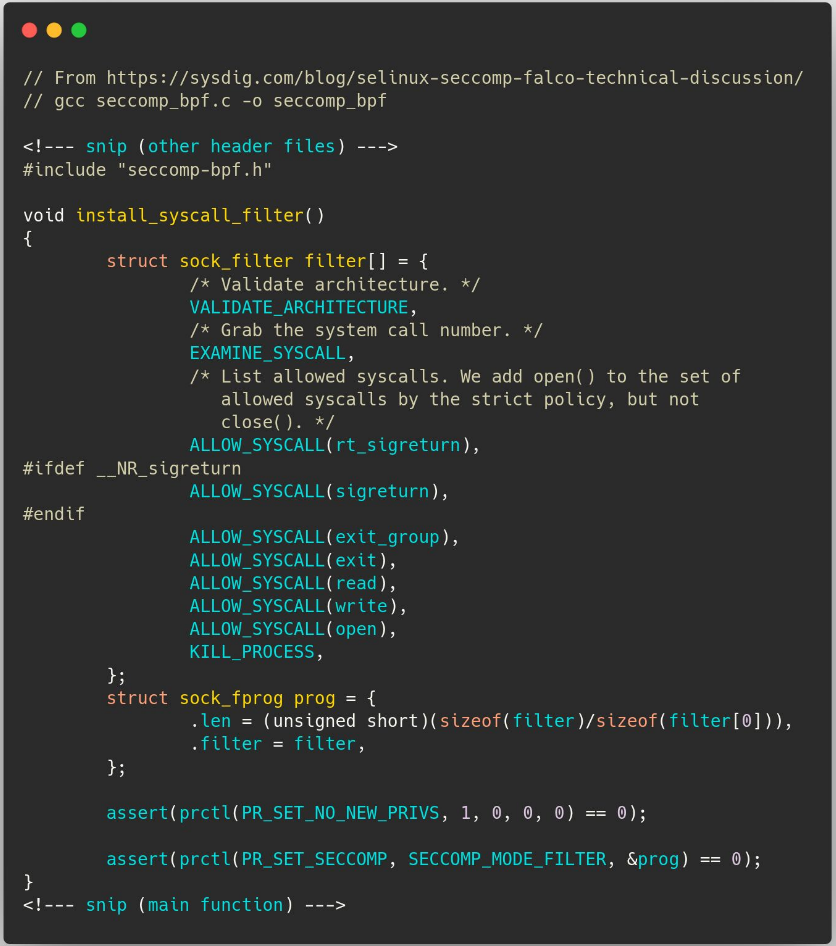
```c
// From https://sysdig.com/blog/selinux-seccomp-falco-technical-discussion/
// gcc seccomp_strict.c -o seccomp_strict

#include <fcntl.h>
#include <stdio.h>
#include <unistd.h>
#include <string.h>
#include <linux/seccomp.h>
#include <sys/prctl.h>

int main(int argc, char **argv)
{
    int output = open("output.txt", O_WRONLY);
    const char *val = "test";

    //enables strict seccomp mode
    printf("Calling prctl() to set seccomp strict mode...\n");
    prctl(PR_SET_SECCOMP, SECCOMP_MODE_STRICT);

    //This is allowed as the file was already opened
    printf("Writing to an already open file...\n");
    write(output, val, strlen(val)+1);

    //This isn't allowed
    printf("Trying to open file for reading...\n");
    int input = open("output.txt", O_RDONLY);

    printf("You will not see this message--the process will be killed first\n");
}
```

seccomp modes

- *seccomp-bpf*
  seccomp-bpf is an recent extension, allowing filtering the system calls using the Berkeley Packet Filter (BPF) program

  Extends to allow or deny any set of system calls, as well as system call arguments.

  Also, instead of simply killing using SIGKILL, seccomp-bpf supports 7 return values.

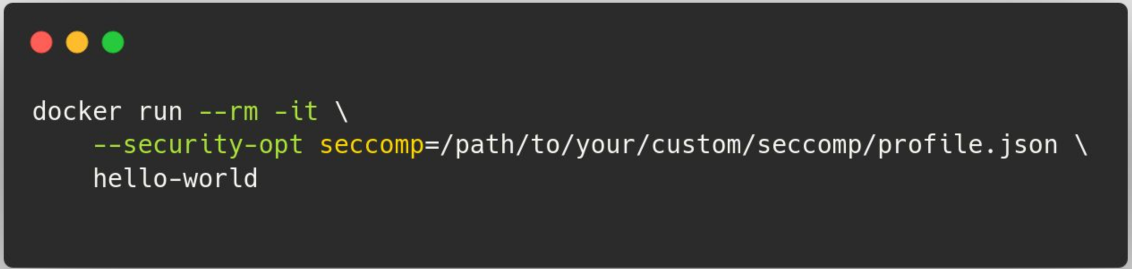


```c
// From https://sysdig.com/blog/selinux-seccomp-falco-technical-discussion/
// gcc seccomp_bpf.c -o seccomp_bpf

<!--- snip (other header files) --->
#include "seccomp-bpf.h"

void install_syscall_filter()
{
        struct sock_filter filter[] = {
                /* Validate architecture. */
                VALIDATE_ARCHITECTURE,
                /* Grab the system call number. */
                EXAMINE_SYSCALL,
                /* List allowed syscalls. We add open() to the set of
                   allowed syscalls by the strict policy, but not
                   close(). */
                ALLOW_SYSCALL(rt_sigreturn),
#ifdef __NR_sigreturn
                ALLOW_SYSCALL(sigreturn),
#endif
                ALLOW_SYSCALL(exit_group),
                ALLOW_SYSCALL(exit),
                ALLOW_SYSCALL(read),
                ALLOW_SYSCALL(write),
                ALLOW_SYSCALL(open),
                KILL_PROCESS,
        };
        struct sock_fprog prog = {
                .len = (unsigned short)(sizeof(filter)/sizeof(filter[0])),
                .filter = filter,
        };

        assert(prctl(PR_SET_NO_NEW_PRIVS, 1, 0, 0, 0) == 0);

        assert(prctl(PR_SET_SECCOMP, SECCOMP_MODE_FILTER, &prog) == 0);
}
<!--- snip (main function) --->
```

seccomp in Docker

seccomp-bpf is used to restrict system calls in Docker. Docker seccomp documentation documents the default list of block system calls.

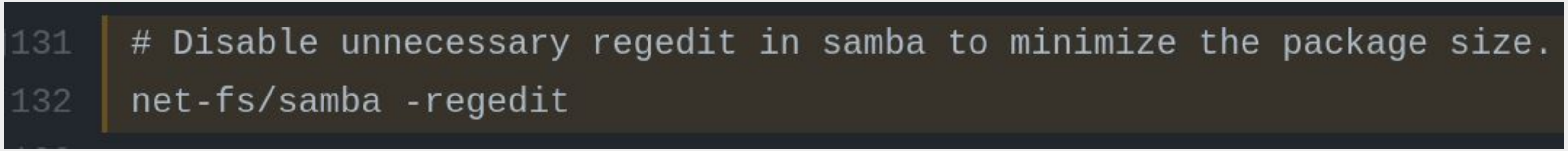
To run the container with a custom secomp policy, use the --security-opt option.



```
docker run --rm -it \
    --security-opt seccomp=/path/to/your/custom/seccomp/profile.json \
    hello-world
```

Flatcar - a container OS built around security

- No package manager
- /usr is read-only
- Control of package dependencies



```
# Disable unnecessary regedit in samba to minimize the package size.
net-fs/samba -regedit
```

"We have some work to do !"

“Yes, security is an incremental process, but a needed one."

CONF42

Sayan Chowdhury

hello@yudocaa.in

@yudocaa

@sayanchowdhury

Mathieu Tortuyaux

mathieu.tortuyaux@gmail.com

@tormath1

@tormath1