**Basic Information**

**Seccomp**, standing for Secure Computing mode, is a security feature of the **Linux kernel designed to filter system calls**. It restricts processes to a limited set of system calls (exit(), sigreturn(), read(), and write() for already-open file descriptors). If a process tries to call anything else, it gets terminated by the kernel using SIGKILL or SIGSYS. This mechanism doesn't virtualize resources but isolates the process from them.

There are two ways to activate seccomp: through the prctl(2) system call with PR\_SET\_SECCOMP, or for Linux kernels 3.17 and above, the seccomp(2) system call. The older method of enabling seccomp by writing to /proc/self/seccomp has been deprecated in favor of prctl().

An enhancement, **seccomp-bpf**, adds the capability to filter system calls with a customizable policy, using Berkeley Packet Filter (BPF) rules. This extension is leveraged by software such as OpenSSH, vsftpd, and the Chrome/Chromium browsers on Chrome OS and Linux for flexible and efficient syscall filtering, offering an alternative to the now unsupported systrace for Linux.

**Original/Strict Mode**

In this mode Seccomp **only allow the syscalls** exit(), sigreturn(), read() and write() to already-open file descriptors. If any other syscall is made, the process is killed using SIGKILL

seccomp\_strict.c

Copy

#include <fcntl.h>

#include <stdio.h>

#include <unistd.h>

#include <string.h>

#include <linux/seccomp.h>

#include <sys/prctl.h>

//From https://sysdig.com/blog/selinux-seccomp-falco-technical-discussion/

//gcc seccomp\_strict.c -o seccomp\_strict

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-bpf**

This mode allows **filtering of system calls using a configurable policy** implemented using Berkeley Packet Filter rules.

seccomp\_bpf.c

Copy

#include <seccomp.h>

#include <unistd.h>

#include <stdio.h>

#include <errno.h>

//https://security.stackexchange.com/questions/168452/how-is-sandboxing-implemented/175373

//gcc seccomp\_bpf.c -o seccomp\_bpf -lseccomp

void main(void) {

/\* initialize the libseccomp context \*/

scmp\_filter\_ctx ctx = seccomp\_init(SCMP\_ACT\_KILL);

/\* allow exiting \*/

printf("Adding rule : Allow exit\_group\n");

seccomp\_rule\_add(ctx, SCMP\_ACT\_ALLOW, SCMP\_SYS(exit\_group), 0);

/\* allow getting the current pid \*/

//printf("Adding rule : Allow getpid\n");

//seccomp\_rule\_add(ctx, SCMP\_ACT\_ALLOW, SCMP\_SYS(getpid), 0);

printf("Adding rule : Deny getpid\n");

seccomp\_rule\_add(ctx, SCMP\_ACT\_ERRNO(EBADF), SCMP\_SYS(getpid), 0);

/\* allow changing data segment size, as required by glibc \*/

printf("Adding rule : Allow brk\n");

seccomp\_rule\_add(ctx, SCMP\_ACT\_ALLOW, SCMP\_SYS(brk), 0);

/\* allow writing up to 512 bytes to fd 1 \*/

printf("Adding rule : Allow write upto 512 bytes to FD 1\n");

seccomp\_rule\_add(ctx, SCMP\_ACT\_ALLOW, SCMP\_SYS(write), 2,

SCMP\_A0(SCMP\_CMP\_EQ, 1),

SCMP\_A2(SCMP\_CMP\_LE, 512));

/\* if writing to any other fd, return -EBADF \*/

printf("Adding rule : Deny write to any FD except 1 \n");

seccomp\_rule\_add(ctx, SCMP\_ACT\_ERRNO(EBADF), SCMP\_SYS(write), 1,

SCMP\_A0(SCMP\_CMP\_NE, 1));

/\* load and enforce the filters \*/

printf("Load rules and enforce \n");

seccomp\_load(ctx);

seccomp\_release(ctx);

//Get the getpid is denied, a weird number will be returned like

//this process is -9

printf("this process is %d\n", getpid());

}

**Seccomp in Docker**

**Seccomp-bpf** is supported by **Docker** to restrict the **syscalls** from the containers effectively decreasing the surface area. You can find the **syscalls blocked** by **default** in <https://docs.docker.com/engine/security/seccomp/> and the **default seccomp profile** can be found here <https://github.com/moby/moby/blob/master/profiles/seccomp/default.json>. You can run a docker container with a **different seccomp** policy with:

Copy

docker run --rm \

-it \

--security-opt seccomp=/path/to/seccomp/profile.json \

hello-world

If you want for example to **forbid** a container of executing some **syscall** like uname you could download the default profile from <https://github.com/moby/moby/blob/master/profiles/seccomp/default.json> and just **remove the uname string from the list**. If you want to make sure that **some binary doesn't work inside a a docker container** you could use strace to list the syscalls the binary is using and then forbid them. In the following example the **syscalls** of uname are discovered:

Copy

docker run -it --security-opt seccomp=default.json modified-ubuntu strace uname

If you are using **Docker just to launch an application**, you can **profile** it with **strace** and **just allow the syscalls** it needs

**Example Seccomp policy**

[Example from here](https://sreeninet.wordpress.com/2016/03/06/docker-security-part-2docker-engine/)

To illustrate Seccomp feature, let’s create a Seccomp profile disabling “chmod” system call as below.

Copy

{

"defaultAction": "SCMP\_ACT\_ALLOW",

"syscalls": [

{

"name": "chmod",

"action": "SCMP\_ACT\_ERRNO"

}

]

}

In the above profile, we have set default action to “allow” and created a black list to disable “chmod”. To be more secure, we can set default action to drop and create a white list to selectively enable system calls. Following output shows the “chmod” call returning error because its disabled in the seccomp profile

Copy

$ docker run --rm -it --security-opt seccomp:/home/smakam14/seccomp/profile.json busybox chmod 400 /etc/hosts

chmod: /etc/hosts: Operation not permitted

Following output shows the “docker inspect” displaying the profile:

Copy

"SecurityOpt": [

"seccomp:{\"defaultAction\":\"SCMP\_ACT\_ALLOW\",\"syscalls\":[{\"name\":\"chmod\",\"action\":\"SCMP\_ACT\_ERRNO\"}]}"

],

**Deactivate it in Docker**

Launch a container with the flag: **--security-opt seccomp=unconfined**

As of Kubernetes 1.19, **seccomp is enabled by default for all Pods**. However, the default seccomp profile applied to the Pods is the "**RuntimeDefault**" profile, which is **provided by the container runtime** (e.g., Docker, containerd). The "RuntimeDefault" profile allows most system calls while blocking a few that are considered dangerous or not generally required by containers.