Overview and Recent Developments: seccomp and small LSMs

Linux Security Summit EU
October 26, 2018
Edinburgh, Scotland

Kees (“Case”) Cook
keescook@chromium.org
@kees_cook

https://outflux.net/slides/2018/lss-eu/seccomp.pdf

Agenda

- Background
  - what's a "small" LSM?
- LoadPin
  - purpose, recent development, and demo
- Yama
  - purpose, recent development, and demo
- seccomp
  - why is seccomp not an LSM?
  - purpose, matching, filters
  - recent development
  - demo

I. Lietuvių Susivažiavimas Vilniuje.

Lapkričio 21—22 (Gruodžio 4—5) d. 1905.

PROGRAMA:

1. *Lietuvystės pradžia ir dabartis.*
2. *Lietuviai, latviai ir įvairios tautos Lietuvoje gyvenančios.*
3. *Caro manifestas nuo Spalinio 17 (30) dienos.*
4. *Lietuvos autonomija ir prikergimas Suvalkų gubernijos prie autonomiškosios Lietuvos.*
5. *Rinkimai į Viešpatystės seimą.*
6. *Lietuvos sodiečiai po rusų valdžia.*
7. *Lietuvos mokyklos.*
8. *Apie įvairius mokesčius.*
9. *Apie žemiečių įstaigas.*
10. *Apie išeivystę (emigraciją).*
11. *Apie kareiviavimą.*

Gali buti įnešami ir kiloki klausimai.

Organizacijos Komitetas.

Background

- Regular LSMs (SELinux, AppArmor, Smack, TOMOYO) have a comprehensive policy language covering a full Mandatory Access Control system

- “small” LSMs have a very narrow (or fixed) policy

LoadPin: Overview

- Built with `CONFIG_SECURITY_LOADPIN=y`
- If you think `CONFIG_MODULE_SIG_FORCE=y` is redundant in your environment, LoadPin is for you!
  - Chrome OS uses `dm-verity` to provide a cryptographically verified read-only root filesystem
  - There is no need to sign modules – they just have to come only from the root filesystem
- Also protects other files the kernel reads:
  - kexec images, firmware, security policy, certs, etc

LoadPin: Recent developments

- Thankfully pretty stable (maybe only Chrome OS uses it?)
- v4.20
  - human readable device name in initialization output
  - boot param name changing from “enabled” to “enforce”

LoadPin: Demo

```
# df -h /tmp /trusted
Filesystem      Size  Used Avail Use% Mounted on
/dev/root        24G   17G  6.4G  72% /
/dev/loop0      488M  956K  452M   1% /trusted
# insmod /trusted/test_module.ko
[..] LoadPin: loop0 (7:0): writable
[..] LoadPin: enforcement can be disabled.
[..] LoadPin: kernel-module pinned obj="/trusted/test_module.ko" pid=3208 cmdline="insmod /trusted/test_module.ko"
[..] test_module: Hello, world
# rmmod test_module
[..] test_module: Goodbye
# insmod /tmp/test_module.ko
insmod: ERROR: could not insert module /tmp/test_module.ko: Operation not permitted
[..] LoadPin: kernel-module denied obj="/tmp/test_module.ko" pid=3215 cmdline="insmod /tmp/test_module.ko"
# umount /trusted
[..] LoadPin: umount pinned fs: refusing further loads
```

Yama: Overview

NOT
ACTUAL LOGO

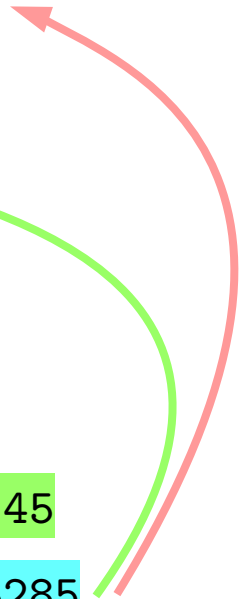
- Built with `CONFIG_SECURITY_YAMA=y`
- The first “stacked” LSM (sorry not sorry)
- Narrows scope of ptracing from “same uid” to “ancestor and explicit whitelist”
- Basic goal is to expand the time window needed to steal a user’s credentials after successfully breaking into a machine

Yama: Recent developments

- Also pretty stable (and many distros use it!)
  - (I just had to write this in a slide, didn't I? syzkaller just sent me an RCU bug it found in Yama's task walking...)
- I may have future work cut out for me
  - Jann Horn has started looking at bypasses …

Yama: Demo

```
systemd,1,root
  |-daemon,2214,otheruser
  |-bash,2223,kees
  |   `-secret,2230
  |   `-gdb,2235
  |       `-program,2236
  `-bash,3101,kees
      |-crashing-program,3145
      |   `-evil-attacker,3285
      `-crash-handler,3286
```



Before Yama:

"evil-attacker" wants to read memory of "daemon", but gets blocked by DAC (kees != otheruser):

ptrace(PTRACE_ATTACH, 2214)

→ EPERM

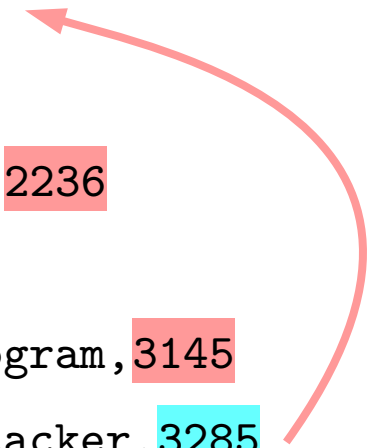
But it can read “secret” (same uid, but started in the past):

ptrace(PTRACE_ATTACH, 2230)

→ ok

Yama: Demo

```
systemd,1,root
  |-daemon,2214,otheruser
  |-bash,2223,kees
  |   |-secret,2230
  |   `-gdb,2235
  |       `-program,2236
  `-bash,3101,kees
      |-crashing-program,3145
      |   `-evil-attacker,3285
      `-crash-handler,3286
```



With Yama:

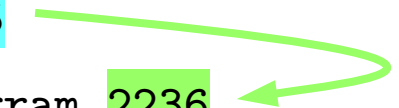
"evil-attacker" wants to read memory of "secret", but gets blocked (not in ancestry tree):

ptrace(PTRACE_ATTACH, 2230)

→ EPERM

Yama: Demo

```
systemd,1,root
  |-daemon,2214,otheruser
  |-bash,2223,kees
  |   |-secret,2230
  |   `-gdb,2235
  |       `-program,2236
  `-bash,3101,kees
      |-crashing-program,3145
      |   `-evil-attacker,3285
      `-crash-handler,3286
```



With Yama:

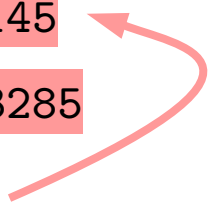
"gdb" wants to read memory of "program" and is fine (gdb is in ancestory tree):

ptrace(PTRACE_ATTACH, 2236)

→ ok

Yama: Demo

```
systemd,1,root
  |-daemon,2214,otheruser
  |-bash,2223,kees
  |   `-secret,2230
  |   `-gdb,2235
  |       `-program,2236
  `-bash,3101,kees
      |-crashing-program,3145
      |   `-evil-attacker,3285
      `-crash-handler,3286
```



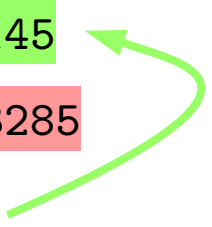
But ancestory tree checking can break crash handlers:

"crash-handler" wants to read "crashing-program", but gets blocked:

ptrace(PTRACE_ATTACH, 3145)
→ EPERM

Yama: Demo

```
systemd,1,root
  |-daemon,2214,otheruser
  |-bash,2223,kees
  |   `-secret,2230
  |   `-gdb,2235
  |       `-program,2236
  `-bash,3101,kees
      |-crashing-program,3145
      |   `-evil-attacker,3285
      `-crash-handler,3286
```



so "crashing-program" declares "crash-handler" can read its memory:

prctl(PR_SET_PTRACER, 3286)

now "crash-handler" can read memory of "crashing-program":

ptrace(PTRACE_ATTACH, 3145)

→ ok

seccomp: Overview

NOT
ACTUAL LOGO

- Not an LSM. More low-level: it filters system calls
- `no_new_privs` saves the day against setuid issues
- As seen in Chrome, Chrome OS, Android, Docker, systemd
  and Firefox, QEMU, OpenSSH, vsftpd, LXD, Tor, the list goes on…
- Easy to add seccomp to your code!
  - To quickly wrap a program, use `minijail -S policy.txt`
  - To use normal filtering, you want `libseccomp`
  - To do really special things, you'll need to learn BPF
    - actually a subset of classic BPF (not eBPF)

seccomp: Filter matching

- filter can match anything in the seccomp BPF “packet data”

```
struct seccomp_data {
  int   nr;                    /* System call number            */
  __u32 arch;                  /* AUDIT_ARCH_* <linux/audit.h> */
  __u64 instruction_pointer;   /* CPU instruction pointer       */
  __u64 args[6];               /* Up to 6 system call args      */
};
```

- Can not read process memory (e.g. filename arg contents) – would be racey

seccomp: Filter results

- `SECCOMP_RET_ALLOW`: continue with syscall
- `SECCOMP_RET_LOG`: emit audit record and continue
- `SECCOMP_RET_TRACE`: generate `PTRACE_EVENT_SECCOMP`
- `SECCOMP_RET_ERRNO`: skip syscall, return specified `errno`
- `SECCOMP_RET_TRAP`: deliver `SIGSYS` signal
- `SECCOMP_RET_KILL_THREAD`: kill the thread
- `SECCOMP_RET_KILL_PROCESS`: kill the process (thread group)

seccomp: Recent developments

- Tycho Andersen suffering endless discussion on how to add `SECCOMP_RET_USER_NOTIF`
  - Basically `SECCOMP_RET_TRACE` using a file descriptor instead of ptrace
- We know what it *won't* be, at least:
  - Not NLA structs
  - Not eBPF ("eBPF will never block userspace")

seccomp: Demo

```
$ ./minijail0 -S cat.policy /bin/cat cat.policy
$ echo $?
134 (SIGABRT)
$ tail -n2 /var/log/syslog
... cat: libminijail[42532]: prctl(seccomp_filter) failed: Permission denied
... minijail0: libminijail[42531]: child process 42532 received signal 6
$ errno --search denied
EACCES 13 Permission denied
$ man seccomp
...
   EACCES
          The caller did not have the CAP_SYS_ADMIN capability in its user
          namespace, or had not set no_new_privs before using SEC-
          COMP_SET_MODE_FILTER.
...
$ ./minijail0 -h | grep privs
  -n:           Set no_new_privs.
```

seccomp: Demo

```
$ ./minijail0 -nS cat.policy /bin/cat cat.policy
openat: 1
fstat: 1
mmap: 1
fadvise64: 1
read: 1
write: 1
munmap: 1
close: 1
exit_group: 1
$ head -n1 cat-einval.policy
openat: return EINVAL
$ ./minijail0 -nS cat-einval.policy /bin/cat cat.policy
/bin/cat: cat.policy: Invalid argument
```

seccomp: Demo

```
$ ./minijail0 -nS cat.policy /bin/ls cat.policy
$ echo $?
253
$ tail -n1 /var/log/syslog
... minijail0: libminijail[43853]: child process 43854 received signal 31
$ kill -l 31
SYS
$ cp cat.policy ls.policy

$ strace -f ./minijail0 -nS ls.policy /bin/ls ls.policy
...
[pid 41026] ioctl(1, TCGETS <unfinished ...>) = ?
[pid 41026] +++ killed by SIGSYS (core dumped) +++
...
$ echo "ioctl: arg0 == 1" >> ls.policy
```

Thoughts?

Kees (“Case”) Cook

keescook@chromium.org
keescook@google.com
kees@outflux.net

@kees_cook

https://outflux.net/slides/2018/lss-eu/seccomp.pdf