Avoiding Security Flaws

or, “How I learned to love attack surface reduction”

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

“Security Flaws” are just “Bugs”

- There isn’t a way for upstream to objectively identify severity for security flaws: reachability
  - remote exec in TCP? Not everyone uses network...
  - use-after-free in Foo-brand USB stick? Machine locked in a colo...
  - heap overflow in vendor ioctl()? SELinux blocks it...
  - root hole in BPF verifier? Not built with CONFIG_BPF_SYSCALL...
- Use the latest stable kernel!
  - All bugs are worth fixing; remember the CIA of security:
    - Confidentiality, Integrity, **Availability**
  - Maybe some bug’s security implications aren’t known for a year...

Find (or avoid) bugs: determinism

- Make “uninitialized memory” deterministically initialized...
  - Initialize all heap memory with:
    - init_on_alloc=1 (or CONFIG_INIT_ON_ALLOC_DEFAULT_ON=y)
  - Initialize all stack variables, depending on compiler:
    - GCC: CONFIG_GCC_PLUGIN_STRUCTLEAK_BYREF_ALL=y
    - Clang: CONFIG_INIT_STACK_ALL_ZERO=y
- Turn off randomization: nokaslr

Attack surface reduction: LSMs

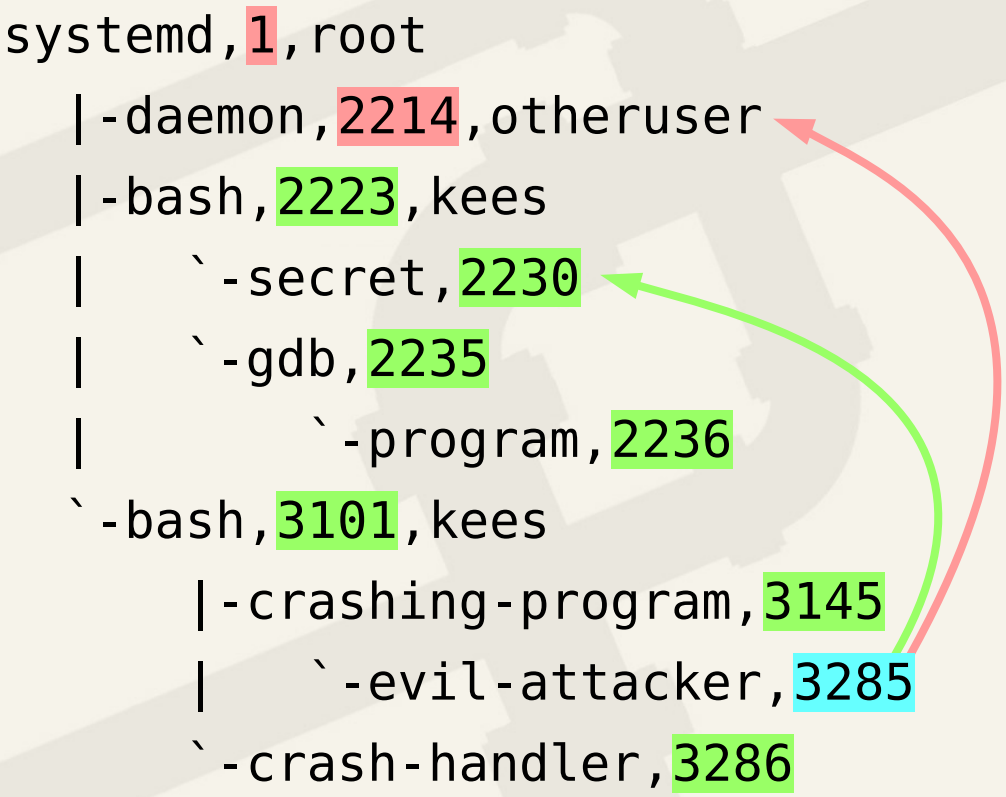
- DAC (users, permissions, capabilities, etc)
- MAC (SELinux, AppArmor, etc)
- Yama
  - avoid surprises from unexpected ptrace
- seccomp
  - avoid surprises from unexpected syscalls

Yama: Overview

MAYBE 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: Demo



Before Yama:

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

ptrace(PTRACE_ATTACH, 2214)

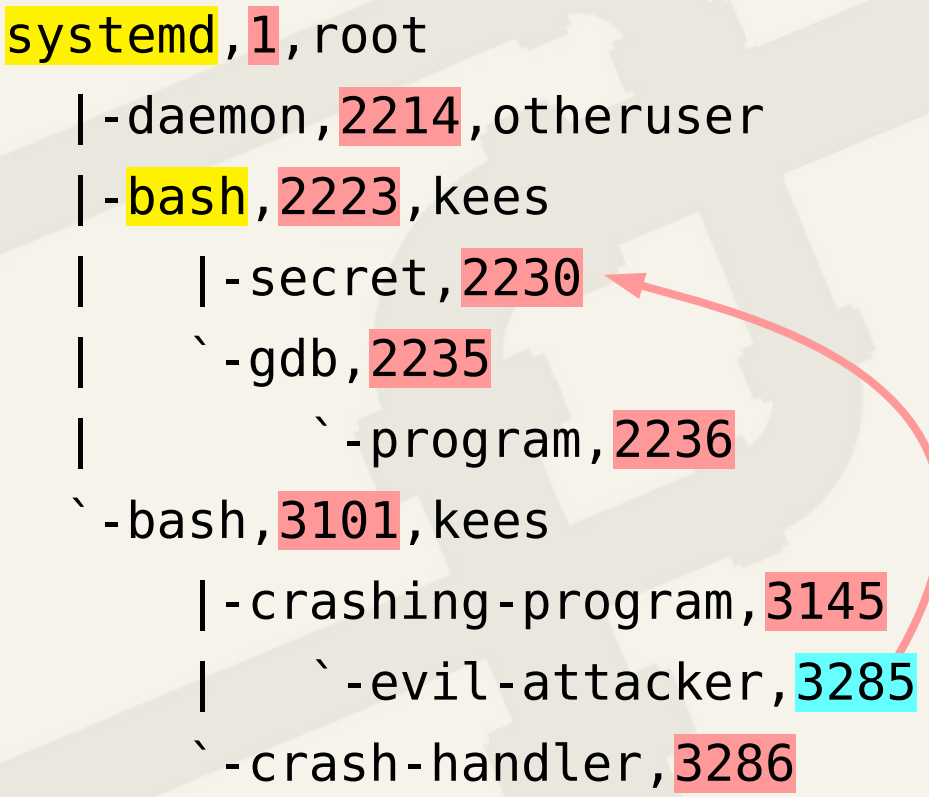
→ EPERM

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

ptrace(PTRACE_ATTACH, 2230)

→ ok

Yama: Demo



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:

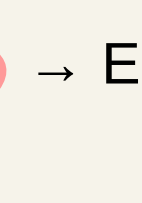
"gdb" wants to read memory of "program" and is fine (gdb is in ancestory tree):

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_USER_NOTIF`: use fd for event resolution
- `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: 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
```

Thank you; stay safe!

Thoughts? Questions?

Kees (“Case”) Cook

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

@kees_cook