Sandboxing Linux code to mitigate exploitation

(Or: How to ship a secure operating system that includes third-party code)

Work by Will Drewry, Elly Jones, Kees Cook, Chris Evans, Julien Tinnes, Markus Gutschke, and me:

Jorge Lucángeli Obes
jorgelo@{chromium.org,google.com}

Truth is...

Code will always have bugs.

Reasoning about the system

- What can an attacker do if they compromise X?
- We need to understand how programs interact with each other.

Reasoning about the system

- Minimize the number of ways programs can interact with or influence each other.
- Sandboxing

Sandboxing

Running a program in a restricted, controlled environment.

Sandboxing

- Capabilities
- setuid sandbox
- Seccomp filtering using BPF

User ids

- Easiest sandboxing primitive.
- Present on Unix from the beginning.

`root`

- On Linux, the `root` user is equivalent to kernel mode.
- Unless module loading has been disabled.

We don't want to run things as `root`.

Things want to run as root

- Problem: system programs expect root privileges.
- Realization: programs don’t need the full extent of root privileges.
- Solution: capabilities.

Linux capabilities

- Divides the privileges traditionally associated with superuser
- into distinct units, known as capabilities,
- which can be independently enabled and disabled.

Linux capabilities

- CAP_NET_ADMIN
- CAP_NET_BIND_SERVICE
- CAP_NET_RAW
- CAP_SYS_ADMIN
- CAP_SYS_BOOT
- CAP_SYS_CHROOT
- CAP_SYS_MODULE
- CAP_SYS_NICE

Setting capabilities

```
cap_value_t flag[1];
flag[0] = CAP_NET_ADMIN;
cap_set_flag(caps, CAP_EFFECTIVE, 1,
             flag, CAP_SET)
cap_set_flag(caps, CAP_PERMITTED, 1,
             flag, CAP_SET)
cap_set_flag(caps, CAP_INHERITABLE, 1,
             flag, CAP_SET)
```

Use capabilities

How do we get all Linux programs to use capabilities?

Answer: we don’t.

Enter Minijail

Helper/launcher program to set up **jails**.

Minijail

- User/group changes
- Linux capabilities
- PID/VFS namespacing
- chroot()'ing, bind-mounting
- `no_new_privs`
- Seccomp filtering

LD_PRELOAD

Every program launched by Minijail will be launched with:

```
LD_PRELOAD=libminijailpreload.so
```

__libc_start_main

```
libc_handle = dlopen("libc.so.6", RTLD_NOW);
sym = dlsym(libc_handle, "__libc_start_main");

real_libc_start_main = sym;
real_main = main;

return real_libc_start_main(
      fake_main, argc, ubp_av, init, fini,
      rtld_fini, stack_end);
```

fake_main

```
minijail_enter(j);

dlclose(libc_handle);

return real_main(argc, argv, envp);
```

Chrome

- Multi-process browser.
- Renderers take HTML, request resources and return a bitmap.
- Renderers run as the same user.

setuid sandbox

- First layer, semantic sandbox.
- Remove FS/net access.
- Uses a seutid-root binary.

setuid sandbox

```
chrome --fork-> chrome-sandbox
--clone(CLONE_NEWPID | CLONE_NEWNET)->
-> chroot helper --clone(CLONE_FS)->
-> zygote
```

setuid sandbox

1. zygote: chroot me!
2. helper:
   - `chroot(/proc/self/fdinfo)`
   - `_exit(0)`

What about the kernel?

- The kernel exposes a **huge** attack surface
- CVE-2012-0056 (Mempodipper)
- CVE-2013-2094 (PERF_EVENTS)

Filtering system calls

- Don’t allow every syscall.
- Many attempts through the years.

seccomp

- SECure COMPuting
- `read()`, `write()`, `sigreturn()`, `exit()`

We need more granularity

- How do we tell the kernel what to filter?
- Enter BPF.

Berkeley Packet Filter

- Assembler-like language to describe network packet filters.
- No loops, just a decision tree.
- Allows to examine parts of the packet.

Will’s brilliant idea

- Treat the system call number and the register set as a network packet.
- Express the system call policy as a filter.

Example filter

```
struct seccomp_data {
    int nr;
    __u32 arch;
    __u64 instruction_pointer;
    __u64 args[6];
};
```

Example filter

```
struct sock_filter filter[] = {
    BPF_STMT(BPF_LD+BPF_W+BPF_ABS,
        offsetof(struct seccomp_data, nr)),
    BPF_JUMP(BPF_JMP+BPF_JEQ+BPF_K, __NR_getpid, 0, 1),
    BPF_STMT(BPF_RET+BPF_K, SECCOMP_RET_KILL),
    BPF_STMT(BPF_RET+BPF_K, SECCOMP_RET_ALLOW),
};
```

Example filter

```
...

struct sock_fprog prog = {
    .len = (unsigned short)(sizeof(filter)/sizeof(filter[0])),
    .filter = filter,
};

int ret = prctl(PR_SET_NO_NEW_PRIVS, 1, 0, 0, 0);
ret = prctl(PR_SET_SECCOMP, SECCOMP_MODE_FILTER, &prog);
```

Return codes

- SECCOMP_RET_KILL
- SECCOMP_RET_TRAP
- SECCOMP_RET_ERRNO
- SECCOMP_RET_TRACE
- SECCOMP_RET_ALLOW

Using Seccomp filtering

- Minijail
- Chrome
- libseccomp

Seccomp filtering with Minijail

We designed a policy language based on ftrace.

Seccomp filtering with Minijail

```
# open: return ENOENT

open: return 1

read: 1

# socket: arg0 == PF_FILE

socket: arg0 == 1
```

Seccomp filtering in Chrome

- Separate implementation.
- Policies are compiled.
- Involved system calls (e.g. `open()`) use TRAP handler.

Sandboxing in Chrome OS

- Only two services running as root
- Seccomp filtering for all services accessing devices (USB, et c.)
- setuid sandbox + seccomp filtering for Chrome

Circling back

- Capabilities
- setuid sandbox
- Seccomp filtering with BPF

Links

Chromium OS security:

http://www.chromium.org/chromium-os/chromiumos-design-docs/system-hardening

Minijail:

https://chromium.googlesource.com/chromiumos%2Fplatform%2Fminijail

Chrome:

https://code.google.com/p/chromium/codesearch#chromium/src/sandbox/linux/seccomp-bpf/sandbox_bpf.h

Seccomp filtering:

http://outflux.net/teach-seccomp/

Questions?

jorgelo@{chromium.org,google.com}