Safer Seccomp: Dead Syscalls Elimination

Speakers: Tan Yuan & Fan Siqi, Lanzhou University
Contributors: Tan Yuan, Fan Siqi, Liu Xiao, Wu Zhangjin, Liu Xin

The Attack Surface Introduced by System Calls

- Kernel provides more than 400 system calls.
- Most programs use only small subset of available system calls.
- Attackers can exploit certain system calls to carry out attacks.
- So we have seccomp to restrict the syscalls that an application can use.

Seccomp

- System Call Filtering
  - Strict mode: only permitted system calls are read(), write(), _exit(), and sigreturn().
  - Filter mode: control which system calls are permitted to caller.
- Wildly Used in Production Environment

Seccomp is not 100% safe

- Software Implementation Mistake
- Configuration Mistake

CVE-2009-0835

- In the Linux kernel 2.6.28.7 and earlier on the x86_64 platform, when CONFIG_SECCOMP is enabled, does not properly handle (1) a 32-bit process making a 64-bit syscall or (2) a 64-bit process making a 32-bit syscall, which allows local users to bypass intended access restrictions via crafted syscalls that are misinterpreted as (a) stat or (b) chmod, a related issue to CVE-2009-0342 and CVE-2009-0343.
- CVSS 2.x Base Score: **3.6 LOW**

CVE-2019-2054

- In the seccomp implementation prior to kernel version 4.8, there is a possible seccomp bypass due to seccomp policies that allow the use of ptrace. This could lead to local escalation of privilege with no additional execution privileges needed. User interaction is not needed for exploitation
- CVSS 3.x Base Score: **7.8 HIGH**


Is there a more secure solution?

How about a policy that can never be bypassed?

- Remove the code related to prohibited system calls from the kernel image.

Dead Syscalls Elimination(DSE) Usage

- Configure used syscalls

```
init/Kconfig:
CONFIG_SYSCALLS_USED=“write exit reboot”
CONFIG_TRIM_UNUSED_SYSCALLS=[y|n]
```

Linux Kernel Refactoring

- Allow remove unused syscalls automatically
  - Tell kernel what we used and let linker remove the others
  - Based on dead code data elimination (--gc-sections) to remove unused functions

Linux Kernel Refactoring

- Syscalls shrink support
  - Use sed to comment out unused system calls in syscall_table.c

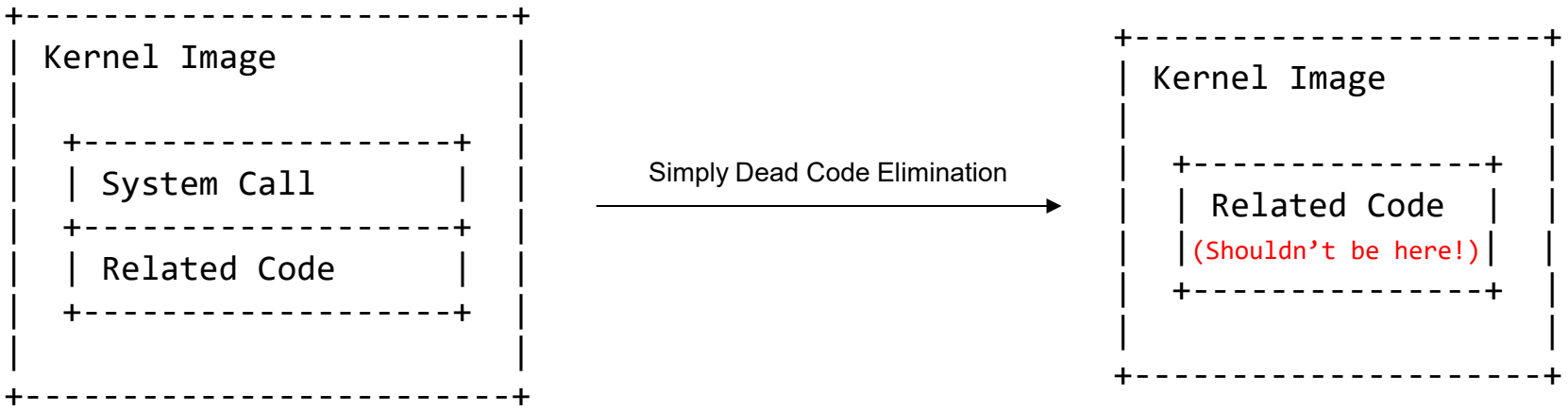
```
sed arch/riscv/kernel/{compat_,}syscall_table.c:

{compat_,}sys_call_table = {[0 ... __NR_syscalls - 1] = sys_ni_syscall,
                                …
                                /* [92] = __riscv_sys_personality, */
                                [93] = sys_exit,
                                /* [95] = __riscv_compat_sys_waitid, */
                                …}
```

Dead Code Elimination

```c
int A(){
    return 0;
}
int B(){
    return 0;
}
int main()
{
    A();
    return 0;
}
```

- The .text.A section will be kept as it is used in the main() function.
- The .text.B section will be removed.
- Every function and variable has its own dependencies. If a section is not part of the dependency tree, it will be deleted.

Challenge


Kernel Image
System Call
Related Code
Simply Dead Code Elimination
Kernel Image
Related Code
(Shouldn’t be here!)

Ownership Reversal Issue

arch/riscv/include/asm/uaccess.h

sys_sendfile() -> get_user() -> __ASM_EXTABLE_RAW(insn, fixup, type, data)

fs/read_write.c *arch/riscv/include/asm/asm-extable.h*

What it should be:

The sys_sendfile() call get_user(), which then calls __ASM_EXTABLE_RAW().
Thus, the existence of __ASM_EXTABLE_RAW() depends on the existence of sys_sendfile().
If get_user() exists, all variables and functions used in get_user(), such as __ASM_EXTABLE_RAW, will not be garbage collected.

So, we can say that get_user() owns __ASM_EXTABLE_RAW().

Ownership Reversal Issue

arch/riscv/include/asm/uaccess.h

sys_sendfile() -> get_user() -> __ASM_EXTABLE_RAW(insn, fixup, type, data)

fs/read_write.c *arch/riscv/include/asm/asm-extable.h*

```
#define __ASM_EXTABLE_RAW(insn, fixup, type, data)  \
    .pushsection    __ex_table, "a";          \
    .balign     4;              \
    .long       ((insn) - .);           \
    .long       ((fixup) - .);          \
    .short      (type);             \
    .short      (data);             \
    .popsection;
```

.pushsection directive will temporarily interrupt the creation of the current ELF section, and then insert a new ELF section.

The linker is unable to obtain dependency information, so it will default to garbage collecting the section created by .pushsection. The __extable will be removed.

To prevent this, developer use KEEP(*(__ex_table)) in the linker script to ensure that __extable is preserved.

Ownership Reversal Issue

arch/riscv/include/asm/uaccess.h

sys_sendfile() -> get_user() -> __ASM_EXTABLE_RAW(insn, fixup, type, data)

fs/read_write.c *arch/riscv/include/asm/asm-extable.h*

```
#define __get_user_asm(insn, x, ptr, err)                   \
do {                                                        \
    __typeof__(x) __x;                                      \
    __asm__ __volatile__ (                                  \
        "1:\n"                                              \
        "   " insn " %1, %2\n"                              \
        "2:\n"                                              \
        _ASM_EXTABLE_UACCESS_ERR_ZERO(1b, 2b, %0, %1)       \
        : "+r" (err), "=&r" (__x)                           \
        : "m" (*(ptr)));                                    \
    (x) = __x;                                              \
} while (0)
```

```
#define __ASM_EXTABLE_RAW(insn, fixup, type, data)  \
    .pushsection    __ex_table, "a";                \
    .balign     4;                                  \
    .long       ((insn) - .);                       \
    .long       ((fixup) - .);                      \
    .short      (type);                             \
    .short      (data);                             \
    .popsection;
```

What it actually is:

__ex_table is forcibly retained by KEEP(), so all the variables and functions it uses will also be kept.

__ex_table uses variables that belong to get_user(), so get_user() and sys_sendfile() depend on __ex_table, which in turn __ex_table owns get_user().

Ownership Reversal Issue

arch/riscv/include/asm/uaccess.h

sys_sendfile() -> get_user() -> __ASM_EXTABLE_RAW(insn, fixup, type, data)

fs/read_write.c *arch/riscv/include/asm/asm-extable.h*

```
#define __get_user_asm(insn, x, ptr, err)                   \
do {                                                        \
    __typeof__(x) __x;                                      \
    __asm__ __volatile__ (                                  \
        "1:\n"                                              \
        "   " insn " %1, %2\n"                              \
        "2:\n"                                              \
        _ASM_EXTABLE_UACCESS_ERR_ZERO(1b, 2b, %0, %1)       \
        : "+r" (err), "=&r" (__x)                           \
        : "m" (*(ptr)));                                    \
    (x) = __x;                                              \
} while (0)
```

```
#define __ASM_EXTABLE_RAW(insn, fixup, type, data)  \
    .pushsection    __ex_table, "a";                \
    .balign     4;                                  \
    .long       ((insn) - .);                       \
    .long       ((fixup) - .);                      \
    .short      (type);                             \
    .short      (data);                             \
    .popsection;
```

- Some entries of the __ex_table may never be used, but they are still forcibly retained.
- If an unused __ex_table entry is kept, it can lead to the incorrect retention of the functions it owns, such as get_user().

Solution

```
#define __ASM_EXTABLE_PUSH_SECTION                \
    __LABEL_NAME(.L__ex_table) ":"                \
    ".pushsection " __SECTION_NAME(__ex_table) ", \"ao\","
__LABEL_NAME(.L__ex_table) "\n"
```

- Ensure that the sections generated by .pushsection have the correct dependency relationships.
    - O flag: section references a symbol defined in another section in the same file. So the dependencies can be established.
- Remove the keep() directive from the kernel linker script.

Weakness of Dead Syscalls Elimination

- If the workload changes, the kernel needs to be recompiled.
- All applications share the same policy, which is more suitable for single-application deployments.

Friendly to Embedded Devices

- The running applications are relatively unchanging.
- No more overhead introduced by seccomp.
- Reduce the kernel image size by 7%.

Takeaway

- Dead Syscalls Elimination

Keep the necessary syscalls and remove the code of unnecessary syscalls.

- No More KEEP()

Ensure that the sections generated by .pushsection also have the correct dependency relationships.

All places in the kernel code that require KEEP() can be rewritten this way.

[PATCH v1 0/7] DCE/DSE: Add Dead Syscalls Elimination support, part1

[PATCH v1 00/14] DCE/DSE: Add Dead Syscalls Elimination support, part2