# HOOKING 輕鬆做 SYSCALL 輕鬆學

劉哲豪
Dylandy

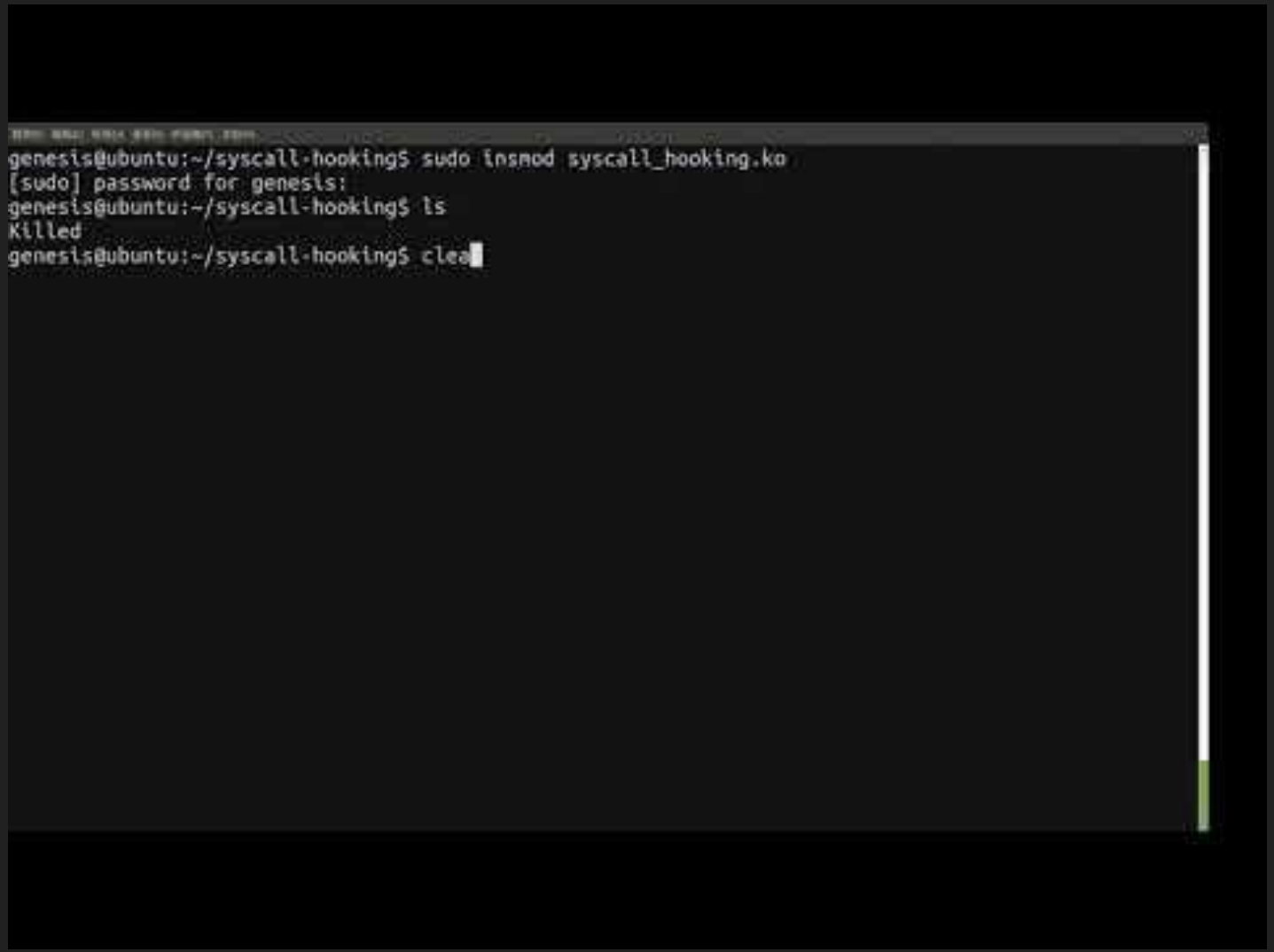
# WHO AM I

- 劉哲豪 (貌似沒綽號?)
  - NCU ADL PhD(準備退回碩班= =)
  - Linux Foundation Open Source Summit Japan 2019 Speaker
    - SMACK-based Application Whitelisting on AGL
  - 其他好像沒什麼好講的了 ...
- Dylandy
  - Ruby 廚
  - Web developer 小弱弱
  - 最近莫名其妙熱衷人類學

為保護當事人肖像權
以下圖片皆經過後製處理

# 第一次 HOOKING SYSTEM CALL 就上手

所以 HOOKING SYSTEM CALL 可以幹麻勒


genesis@ubuntu:~/syscall-hooking$ sudo insmod syscall_hooking.ko
[sudo] password for genesis:
genesis@ubuntu:~/syscall-hooking$ ls
Killed
genesis@ubuntu:~/syscall-hooking$ clea

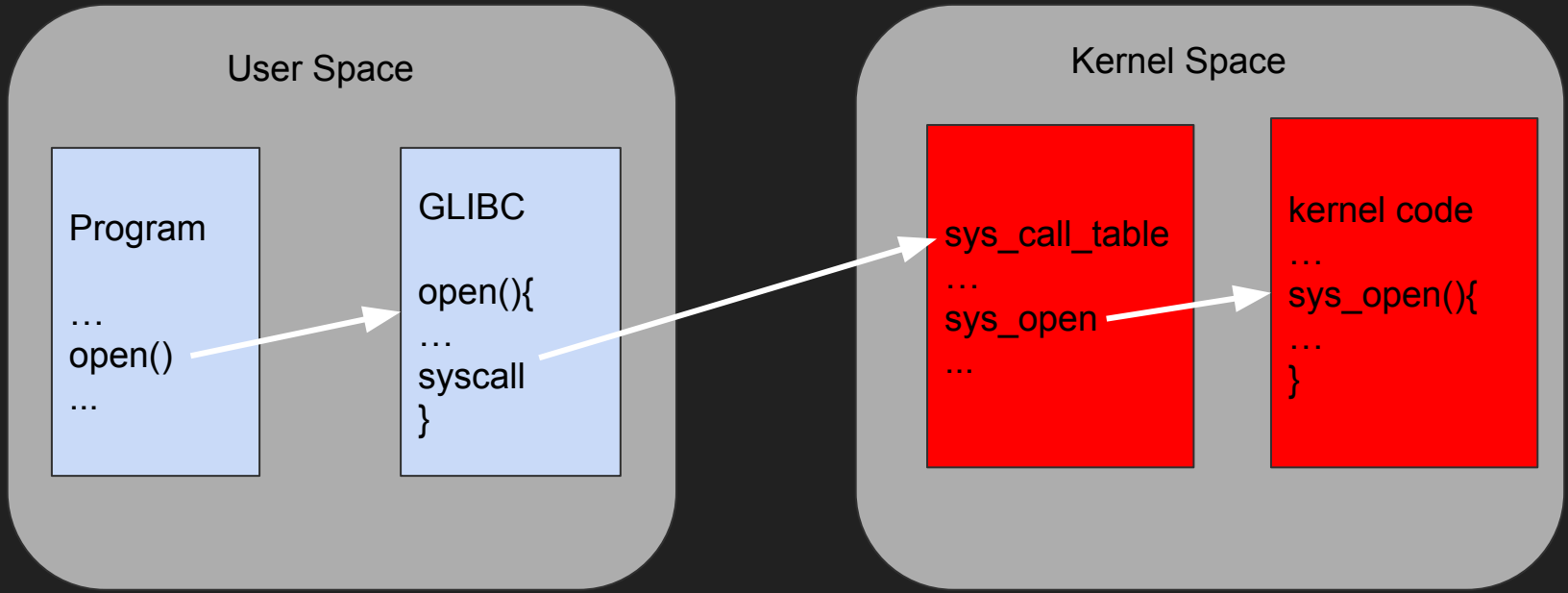
# SYSTEM CALL 是什麼？能吃嗎？


研究所
2019
(105~107年試題)
試題大補帖
計算機概論
[redacted]
考場致勝
2大特色
Characteristic
1.網羅名師新題，步驟深入淺出，讓考生快速掌握新題要領。
2.相關試題一次讀有，全面縮減考生彙整試題的時間。
[redacted]

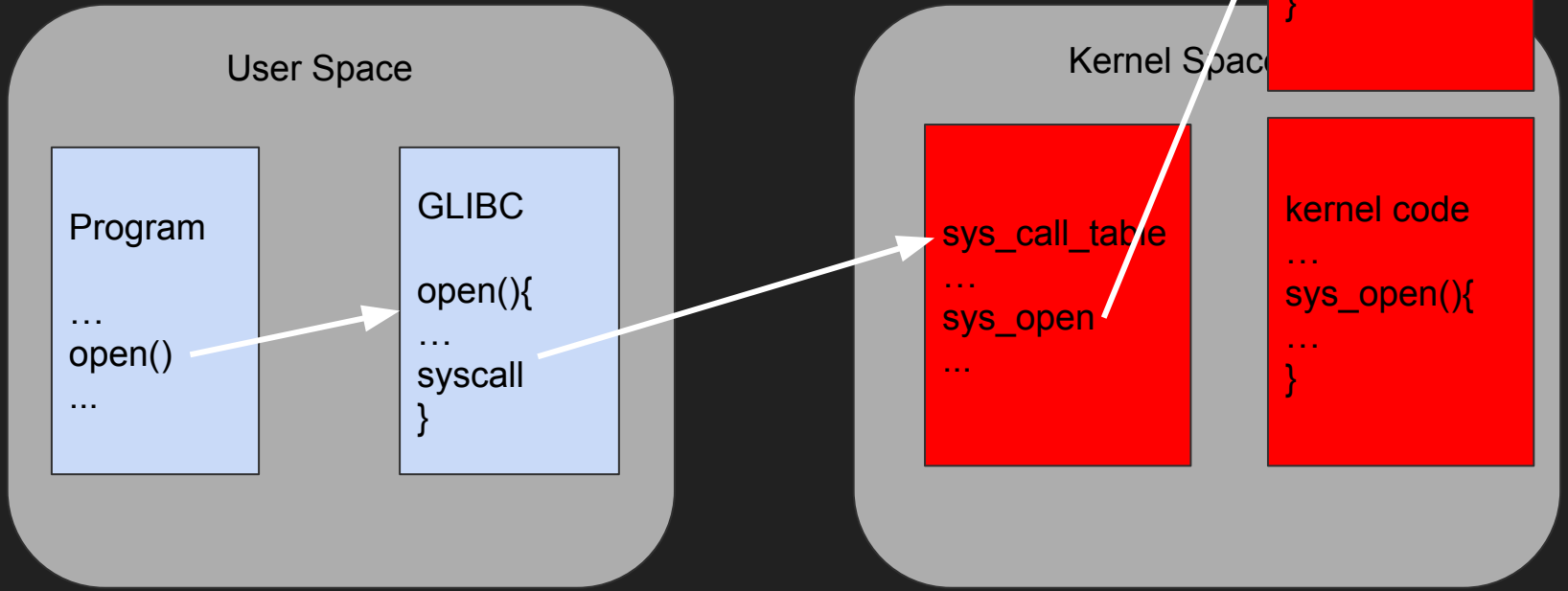
| %rax | System call | %rdi | %rsi | %rdx | %r10 | %r8 | %r9 |
|---|---|---|---|---|---|---|---|
| 0 | sys_read | unsigned int fd | char *buf | size_t count | | | |
| 1 | sys_write | unsigned int fd | const char *buf | size_t count | | | |
| 2 | sys_open | const char *filename | int flags | int mode | | | |
| 3 | sys_close | unsigned int fd | | | | | |
| 4 | sys_stat | const char *filename | struct stat *statbuf | | | | |
| 5 | sys_fstat | unsigned int fd | struct stat *statbuf | | | | |
| 6 | sys_lstat | fconst char *filename | struct stat *statbuf | | | | |
| 7 | sys_poll | struct poll_fd *ufds | unsigned int nfds | long timeout_msecs | | | |
| 8 | sys_lseek | unsigned int fd | off_t offset | unsigned int origin | | | |
| 9 | sys_mmap | unsigned long addr | unsigned long len | unsigned long prot | unsigned long flags | unsigned long fd | unsigned long off |

http://blog.rchapman.org/posts/Linux_System_Call_Table_for_x86_64/

# SYSCALL FLOW



# SYSCALL FLOW



ㅋㅋ

# 寫個 KERNEL MODULE

# 找到辣個 SYSTEM CALL TABLE

- 佛系通靈法
- 腳踏實地法(苦幹實幹法)
- 偷看路標法
- 直接導航法
- 融合超強法

# 腳踏實地法(苦幹實幹法)

1. 從 PAGE_OFFSET 開始找
2. 每 8 bytes 作比對
3. 檢查第 __NR_close 項是否等於 sys_close 的 address

# Why sys_close ?

```
EXPORT_SYMBOL(sys_close);
```

# 小問題

- 碰巧遇到一樣的值，但卻不是 syscall table
- e.g. 全台都有中山路

# 偷看路標法

從 MSR_LSTAR 中取出 entry_SYSCALL_64 的位置

```
void syscall_init(void)
{
        /*
         * LSTAR and STAR live in a bit strange symbiosis.
         * They both write to the same internal register. STAR allows to
         * set CS/DS but only a 32bit target. LSTAR sets the 64bit rip.
         */
        wrmsrl(MSR_STAR,  ((u64)__USER32_CS)<<48  | ((u64)__KERNEL_CS)<<32);
        wrmsrl(MSR_LSTAR, (unsigned long)entry_SYSCALL_64);
```

https://elixir.bootlin.com/linux/v4.4/source/arch/x86/kernel/cpu/common.c#L1189

# 偷看路標法

從 entry_SYSCALL_64 往下追

```
entry_SYSCALL_64_fastpath:
#if __SYSCALL_MASK == ~0
        cmpq    $__NR_syscall_max, %rax
#else
        andl    $__SYSCALL_MASK, %eax
        cmpl    $__NR_syscall_max, %eax
#endif
        ja      1f
        movq    %r10, %rcx
        call    *sys_call_table(, %rax, 8)
        movq    %rax, RAX(%rsp)
```

https://elixir.bootlin.com/linux/v4.4/source/arch/x86/entry/entry_64.S

# 偷看路標法

```
00000000000000ac <entry_SYSCALL_64_fastpath>:
      ac:   25 ff ff ff bf          and    eax,0xbfffffff
      b1:   3d 21 02 00 00          cmp    eax,0x221
      b6:   77 0f                   ja     c7 <entry_SYSCALL_64_fastpath+0x1b>
      b8:   4c 89 d1                mov    rcx,r10
      bb:   ff 14 c5 00 00 00 00    call   QWORD PTR [rax*8+0x0]
      c2:   48 89 44 24 50          mov    QWORD PTR [rsp+0x50],rax
      c7:   50                      push   rax
```

```
entry_SYSCALL_64_fastpath:
#if __SYSCALL_MASK == ~0
        cmpq    $__NR_syscall_max, %rax
#else
        andl    $__SYSCALL_MASK, %eax
        cmpl    $__NR_syscall_max, %eax
#endif
        ja      1f
        movq    %r10, %rcx
        call    *sys_call_table(, %rax, 8)
        movq    %rax, RAX(%rsp)
```

# 偷看路標法

比對 machine code

```
code[0] == 0xFF && code[1] == 0x14 && code[2] == 0xC5 && code[7] == 0x48
&& code[8] == 0x89 && code[9] == 0x44 && code[10] == 0x24
```

```
00000000000000ac <entry_SYSCALL_64_fastpath>:
      ac:   25 ff ff ff bf          and    eax,0xbfffffff
      b1:   3d 21 02 00 00          cmp    eax,0x221
      b6:   77 0f                   ja     c7 <entry_SYSCALL_64_fastpath+0x1b>
      b8:   4c 89 d1                mov    rcx,r10
      bb:   ff 14 c5 00 00 00 00    call   QWORD PTR [rax*8+0x0]
      c2:   48 89 44 24 50          mov    QWORD PTR [rsp+0x50],rax
      c7:   50                      push   rax
```

# 偷看路標法

sys_call_table 的位置就在 code[3~6]

P.S. 要記得signed extended to 8 bytes

```
00000000000000ac <entry_SYSCALL_64_fastpath>:
      ac:   25 ff ff ff bf          and    eax,0xbfffffff
      b1:   3d 21 02 00 00          cmp    eax,0x221
      b6:   77 0f                   ja     c7 <entry_SYSCALL_64_fastpath+0x1b>
      b8:   4c 89 d1                mov    rcx,r10
      bb:   ff 14 c5 00 00 00 00    call   QWORD PTR [rax*8+0x0]
      c2:   48 89 44 24 50          mov    QWORD PTR [rsp+0x50],rax
      c7:   50                      push   rax
```

# 小問題

- Spectre CVE
- patch with retpoline

```
*/
#ifdef CONFIG_RETPOLINE
        movq    sys_call_table(, %rax, 8), %rax
        call    __x86_indirect_thunk_rax
#else
        call    *sys_call_table(, %rax, 8)
#endif
```

https://elixir.bootlin.com/linux/v4.4.189/source/arch/x86/entry/entry_64.S

# 小問題 - 解

- Spectre CVE
- patch with retpoline

```
*/
#ifdef CONFIG_RETPOLINE
        movq    sys_call_table(, %rax, 8), %rax
        call    __x86_indirect_thunk_rax
#else
        call    *sys_call_table(, %rax, 8)
#endif
```

```
48 8b 04 c5 00 00 00    mov    rax,QWORD PTR [rax*8+0x0]
00
e8 00 00 00 00          call   1a1 <entry_SYSCALL_64_fastpath+0x24>
```

# 小問題

- Meltdown CVE
- KPTI patch (trampoline)

```
void syscall_init(void)
{
        extern char _entry_trampoline[];
        extern char entry_SYSCALL_64_trampoline[];

        int cpu = smp_processor_id();
        unsigned long SYSCALL64_entry_trampoline =
                (unsigned long)get_cpu_entry_area(cpu)->entry_trampoline +
                (entry_SYSCALL_64_trampoline - _entry_trampoline);

        wrmsr(MSR_STAR, 0, (__USER32_CS << 16) | __KERNEL_CS);
        if (static_cpu_has(X86_FEATURE_PTI))
                wrmsrl(MSR_LSTAR, SYSCALL64_entry_trampoline);
        else
                wrmsrl(MSR_LSTAR, (unsigned long)entry_SYSCALL_64);
```

https://elixir.bootlin.com/linux/v4.14.138/source/arch/x86/kernel/cpu/common.c#L1491

# 小問題

- Meltdown CVE
- KPTI patch (trampoline)

```
GLOBAL(entry_SYSCALL_64_after_hwframe)
        pushq   %rax                                    /* pt_regs->orig_ax */

        PUSH_AND_CLEAR_REGS rax=$-ENOSYS

        TRACE_IRQS_OFF

        /* IRQs are off. */
        movq    %rsp, %rdi
        call    do_syscall_64           /* returns with IRQs disabled */
```

https://elixir.bootlin.com/linux/v4.14.138/source/arch/x86/kernel/cpu/common.c#L1491

# 小問題

- Meltdown CVE
- KPTI patch (trampoline)

```
__visible void do_syscall_64(struct pt_regs *regs)
{
        struct thread_info *ti = current_thread_info();
        unsigned long nr = regs->orig_ax;

        enter_from_user_mode();
        local_irq_enable();

        if (READ_ONCE(ti->flags) & _TIF_WORK_SYSCALL_ENTRY)
                nr = syscall_trace_enter(regs);

        /*
         * NB: Native and x32 syscalls are dispatched from the same
         * table.  The only functional difference is the x32 bit in
         * regs->orig_ax, which changes the behavior of some syscalls.
         */
        if (likely((nr & __SYSCALL_MASK) < NR_syscalls)) {
                nr = array_index_nospec(nr & __SYSCALL_MASK, NR_syscalls);
                regs->ax = sys_call_table[nr](
                        regs->di, regs->si, regs->dx,
                        regs->r10, regs->r8, regs->r9);
        }

        syscall_return_slowpath(regs);
}
```

https://elixir.bootlin.com/linux/v4.15.18/source/arch/x86/entry/common.c#L269

# 直接導航法

- /proc/kallsyms
- /boot/System.map
- kallsyms_lookup_name(“sys_call_table”)

# 直接導航法

```
genesis@genesis:~$ sudo cat /proc/kallsyms | grep sys_call_table
ffffffff81a00200 R sys_call_table
```

```
genesis@genesis:~$ sudo cat /boot/System.map-4.4.0-138-generic | grep sys_call_table
ffffffff81a00200 R sys_call_table
```

# 小問題

- /proc/kallsyms
  - 需要 /proc/sys/kernel/kptr_restrict 設定為 0 or 1
- /boot/System.map
  - 如果有KASLR的話，會跟實際mapping的記憶體位置不一樣

# 大問題

- CONFIG_KALLSYMS_ALL=n

# 融合超強法

- 直接導航 X 腳踏實地
- 利用 sys_call_table 被存放在 rodata 段的特性
  - 透過 kallsyms_lookup_name 找到 _etext
  - 對齊 HPAGE_SIZE 後就是 rodata 的起始位置
- 之後比照腳踏實地法

# 融合超強法

```
ffffffff916031d1 T _etext
ffffffff916031d1 T __indirect_thunk_end
ffffffff916031d4 R __start_notes
ffffffff916033a8 R __stop_notes
ffffffff916033b0 R __start___ex_table
ffffffff91609818 R __stop___ex_table
ffffffff91800000 r __func__.57807
ffffffff91800000 R __start_rodata
ffffffff91800020 r __func__.57782
ffffffff91800033 r __func__.4836
ffffffff91800040 r __func__.4805
ffffffff91800048 r __func__.4795
ffffffff91800050 r __param_str_initcall_debug
ffffffff91800060 R linux_proc_banner
ffffffff918000c0 R linux_banner
ffffffff91800180 r __func__.39705
ffffffff918001a0 R sys_call_table
```

寫個~~做壞事的~~FUNCTION

只是想放這張

# write protection

- modify PTE
- disable WP in CR0

# 總結一下

1. 寫個 kernel module
2. 找到 `sys_call_table`
3. 寫個 function
4. 把你要 hook 的 syscall 跟你寫的 function 互換

# Q&A