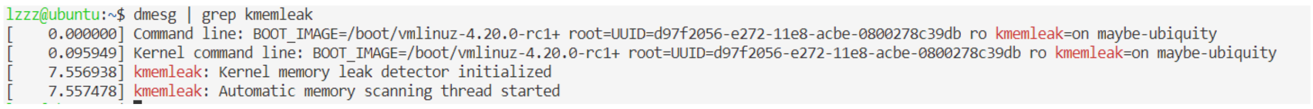
Lab08

Kmemleak

- make menuconfig修改编译配置文件 Kernel hacking --> Memory debugging --> Kernel memory leak detector & (4096) Maximum kmemleak early log entries & Simple test for the kernel memory leak detector
- 开启kmemleak detector，增大early log entries，并开启Simple test

重新编译替换内核，在grub_cmdline中加入kmemleak=on

```
dmesg | grep kmemleak
```



```
lzzz@ubuntu:~$ dmesg | grep kmemleak
[    0.000000] Command line: BOOT_IMAGE=/boot/vmlinuz-4.20.0-rc1+ root=UUID=d97f2056-e272-11e8-acbe-0800278c39db ro kmemleak=on maybe-ubiquity
[    0.095949] Kernel command line: BOOT_IMAGE=/boot/vmlinuz-4.20.0-rc1+ root=UUID=d97f2056-e272-11e8-acbe-0800278c39db ro kmemleak=on maybe-ubiquity
[    7.556938] kmemleak: Kernel memory leak detector initialized
[    7.557478] kmemleak: Automatic memory scanning thread started
```

kmemleak开启成功

找到内核中的kmemleak-test.c，它是一个模块，初始化时行为是动态申请一个链表，删除模块时只释放表头的空间。

```
sudo insmod kmemleak-test.ko
sudo rmmod kmemleak_test
echo scan >/sys/kernel/debug/kmemleak
cat /sys/kernel/debug/kmemleak
```

结果：

```
unreferenced object 0xffff9e6b4defcc00 (size 1024):
  comm "insmod", pid 10236, jiffies 4295092481 (age 492.700s)
  hex dump (first 32 bytes):
    32 9e 97 85 4f 93 69 fd 00 00 00 00 00 00 00 00  2...O.i.........
    00 00 00 00 01 00 00 00 03 70 02 00 ff ff ff ff  .........p......
  backtrace:
    [<000000008b3b9247>] kmem_cache_alloc_trace+0x12d/0x1f0
    [<00000000ed2b4e0c>] 0xffffffffc08bf0b0
    [<0000000035d0b195>] do_one_initcall+0x4a/0x1c9
    [<00000000d3474bae>] do_init_module+0x5f/0x206
    [<00000000865c57eb>] load_module+0x2581/0x2b80
    [<00000000b65e7956>] __do_sys_finit_module+0xfc/0x120
    [<00000000ec83923d>] __x64_sys_finit_module+0x1a/0x20
    [<0000000040658f74>] do_syscall_64+0x5a/0x120
    [<000000008bf51691>] entry_SYSCALL_64_after_hwframe+0x44/0xa9
    [<00000000bdfbe9ba>] 0xffffffffffffffff
unreferenced object 0xffff9e6af8d21e00 (size 512):
  comm "insmod", pid 10236, jiffies 4295092481 (age 492.700s)
  hex dump (first 32 bytes):
    00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
    00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
  backtrace:
    [<000000008b3b9247>] kmem_cache_alloc_trace+0x12d/0x1f0
    [<00000000dc53163f>] 0xffffffffc08bf1e6
    [<0000000035d0b195>] do_one_initcall+0x4a/0x1c9
    [<00000000d3474bae>] do_init_module+0x5f/0x206
    [<00000000865c57eb>] load_module+0x2581/0x2b80
    [<00000000b65e7956>] __do_sys_finit_module+0xfc/0x120
    [<00000000ec83923d>] __x64_sys_finit_module+0x1a/0x20
    [<0000000040658f74>] do_syscall_64+0x5a/0x120
    [<000000008bf51691>] entry_SYSCALL_64_after_hwframe+0x44/0xa9
    [<00000000bdfbe9ba>] 0xffffffffffffffff
unreferenced object 0xffff9e6af1509800 (size 512):
  comm "insmod", pid 10236, jiffies 4295092481 (age 492.700s)
  hex dump (first 32 bytes):
    00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
    00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
  backtrace:
    [<000000008b3b9247>] kmem_cache_alloc_trace+0x12d/0x1f0
    [<00000000dc53163f>] 0xffffffffc08bf1e6
    [<0000000035d0b195>] do_one_initcall+0x4a/0x1c9
    [<00000000d3474bae>] do_init_module+0x5f/0x206
    [<00000000865c57eb>] load_module+0x2581/0x2b80
    [<00000000b65e7956>] __do_sys_finit_module+0xfc/0x120
    [<00000000ec83923d>] __x64_sys_finit_module+0x1a/0x20
    [<0000000040658f74>] do_syscall_64+0x5a/0x120
    [<000000008bf51691>] entry_SYSCALL_64_after_hwframe+0x44/0xa9
    [<00000000bdfbe9ba>] 0xffffffffffffffff
```

Hung_task

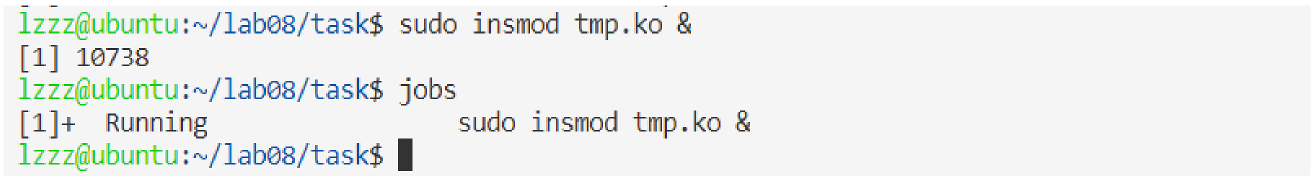
- make menuconfig修改编译配置文件 Kernel hacking --> Debug Lockups and Hangs --> Detect Hung Tasks & (20) Default timeout
- 开启hang_task检测，将Hung状态持续时间检测缩短为20s

重新编译替换内核

编写导致死锁的内核模块：

```
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/init.h>
#include <linux/mutex.h>

DEFINE_MUTEX(dlock);

static int __init dlock_init(void)
{
	mutex_lock(&dlock);
	mutex_lock(&dlock);
	return 0;
}

static void __exit dlock_exit(void)
```

```c
{
        return;
}

module_init(dlock_init);
module_exit(dlock_exit);
MODULE_LICENSE("GPL");
```



```
lzzz@ubuntu:~/lab08/task$ sudo insmod tmp.ko &
[1] 10738
lzzz@ubuntu:~/lab08/task$ jobs
[1]+  Running                 sudo insmod tmp.ko &
lzzz@ubuntu:~/lab08/task$ 
```

```
dmesg
```

```
[ 4281.473159] INFO: task insmod:10574 blocked for more than 20 seconds.
[ 4281.473218]       Tainted: G         C OE     4.20.0-rc1+ #2
[ 4281.474318] "echo 0 > /proc/sys/kernel/hung_task_timeout_secs" disables this message.
[ 4281.475403] insmod          D    0 10574  10573 0x80000000
[ 4281.475407] Call Trace:
[ 4281.475419]  __schedule+0x2b7/0x880
[ 4281.475427]  schedule+0x2c/0x80
[ 4281.475429]  schedule_preempt_disabled+0xe/0x10
[ 4281.475431]  __mutex_lock.isra.8+0x183/0x4d0
[ 4281.475436]  ? 0xffffffffc08ba000
[ 4281.475439]  __mutex_lock_slowpath+0x13/0x20
[ 4281.475441]  ? __mutex_lock_slowpath+0x13/0x20
[ 4281.475443]  mutex_lock+0x2f/0x40
[ 4281.475448]  dlock_init+0x21/0x1000 [tmp]
[ 4281.475453]  do_one_initcall+0x4a/0x1c9
[ 4281.475457]  ? kmem_cache_alloc_trace+0x186/0x1f0
[ 4281.475461]  ? do_init_module+0x27/0x206
[ 4281.475464]  do_init_module+0x5f/0x206
[ 4281.475467]  load_module+0x2581/0x2b80
[ 4281.475472]  __do_sys_finit_module+0xfc/0x120
[ 4281.475474]  ? __do_sys_finit_module+0xfc/0x120
[ 4281.475478]  __x64_sys_finit_module+0x1a/0x20
[ 4281.475480]  do_syscall_64+0x5a/0x120
[ 4281.475484]  entry_SYSCALL_64_after_hwframe+0x44/0xa9
[ 4281.475486] RIP: 0033:0x7f32b74f7839
[ 4281.475496] Code: Bad RIP value.
[ 4281.475498] RSP: 002b:00007ffcd6e68ee8 EFLAGS: 00000246 ORIG_RAX: 0000000000000139
[ 4281.475500] RAX: ffffffffffffffda RBX: 000055dee8920790 RCX: 00007f32b74f7839
[ 4281.475502] RDX: 0000000000000000 RSI: 000055dee7ca6d2e RDI: 0000000000000003
[ 4281.475503] RBP: 000055dee7ca6d2e R08: 0000000000000000 R09: 00007f32b77ca000
[ 4281.475504] R10: 0000000000000003 R11: 0000000000000246 R12: 0000000000000000
[ 4281.475505] R13: 000055dee8920760 R14: 0000000000000000 R15: 0000000000000000
[ 4331.636372] kmemleak: 1 new suspected memory leaks (see /sys/kernel/debug/kmemleak)
[ 4633.635680] (missing .modinfo name field): No module found in object
```

可以看到hung_task监测到了死锁的任务，并打印出Trace Back调试信息。

Hide task

查找网上的的资料，kernel 2.x可以通过修改进程pid为0来隐藏进程，编写代码测试之后，内核直接崩溃。kernel 4.15用这种方法行不通。

ps获取进程信息的方式是读取/proc/<pid>目录，有一种奇技淫巧--将空目录mount到/proc/<pid>下，这样ps就对信息不可见。

回到题目，编写一个kthread并隐藏自己（ps无法查看）。

查看ps的调用过程：

```
strace ps
:
stat("/proc/11880", {...})
openat(...)
read(...)
close(...)
```

ps通过系统调用stat读取文件元信息，成功后调用openat打开文件然后读文件信息。

通拓截获系统调用stat，如果要读取的是需要隐藏的进程的信息，就返回-1表现出读取stat失败的假象，ps就无法继续open并read进程信息，也就无法显示该进程的信息。

编写模块，在init时启动线程，并在线程中截获系统调用，模块退出时将系统调用改回。

```
static int __init thread_init(void)
{

        kthread_run(test, NULL, "test_thread");
        // pid = st->pid;
        // printk("spying on <pid> %ld\n", pid);
        return 0;
}

static void __exit thread_exit(void)
{
        cr0 = read_cr0();
        write_cr0(cr0 & ~CR0_WP);
        sys_call_table[__NR_stat] = original_call;
        write_cr0(cr0);

        return;
}

module_init(thread_init);
module_exit(thread_exit);
MODULE_LICENSE("GPL");
```

线程内容：

先劫持系统调用stat，用自己的系统调用替换，将pid写入buf中。sleep 10s，然后修改buf的值，sleep 20s

```
int test(void *input)
{
        int i;
        // struct task_struct *st;
        cr0 = read_cr0();
        write_cr0(cr0 & ~CR0_WP);
        original_call = sys_call_table[__NR_stat];
        sys_call_table[__NR_stat] = my_sys_stat;
        write_cr0(cr0);

        itoa(current->pid, buf + 6);
        printk("U can't see me(<pid> %d) now!\n", current->pid);
        printk("but 10s later you will see me\n");
        for (i = 0; i < 100; ++i)
                msleep(100);
        buf[6] = 0;
        printk("I'll exit after 20s.\n");
        msleep(20000);
        return 0;
}
```

其中itoa为自己编写的函数。将pid写入buf

替换的stat系统调用:

判断buf内容与调用stat的第一个参数是否一致，如果一致则返回-1.

```
asmlinkage int my_sys_stat(const char *s, struct stat *st)
{
        int ret;
        ret = original_call(s, st);
        if (strcmp(s, buf) == 0) {
                printk("s = %s\n", s);
                printk("buf = %s\n", buf);
                return -1;
        }
        return ret;
}
```

编译模块后加载:

```
sudo insmod tmp.ko
dmesg
```

```
[  816.491829] U can't see me(<pid> 1363) now!
[  816.491831] but 10s later you will see me
```

```
ps -el
```

```
# 最后几行内容为:
5 S  1000  1276  1131  0  80   0 - 26996 -      ?        00:00:00       sshd
0 S  1000  1277  1276  0  80   0 -  5398 wait   pts/0    00:00:00       bash
1 I     0  1300     2  0  80   0 -     0 -      ?        00:00:00       kworker/u6:1
0 R  1000  1366  1277  0  80   0 -  6904 -      pts/0    00:00:00       ps
# 没有test_thread的信息
```

```
dmesg
```

```
[  816.491829] U can't see me(<pid> 1363) now!
[  816.491831] but 10s later you will see me
[  823.185646] s = /proc/1363
[  823.185648] buf = /proc/1363
```

dmesg中可以看到已经截获了stat的调用并对1363号进程隐藏

10s之后，buf值被修改，不再隐藏/proc/<pid>

```
dmesg
```

```
[  816.491829] U can't see me(<pid> 1363) now!
[  816.491831] but 10s later you will see me
[  823.185646] s = /proc/1363
[  823.185648] buf = /proc/1363
[  827.789631] I'll exit after 20s.
```

```
ps -el
```

```
# 最后几行的信息
0 S  1000  1277  1276  0  80   0 -  5398 wait   pts/0    00:00:00       bash
1 I     0  1300     2  0  80   0 -     0 -      ?        00:00:00       kworker/u6:1
1 D     0  1363     2  0  80   0 -     0 -      ?        00:00:00       test_thread
0 R  1000  1368  1277  0  80   0 -  6904 -      pts/0    00:00:00       ps
# 可以看到test_thread不再被隐藏
```

附件

```
.
├── kmemleak
|   ├── kmemleak-test.c      # 内核中的kmemleak-test代码
|   ├── Makefile             # 编译内核模块
|   ├── res.txt              # cat /sys/kernel/debug/kmemleak 结果
├── task
|   ├── Makefile
|   ├── tmp.c                # 内核死锁模块
└── thread
    ├── a                    # 隐藏进程时ps -el打印信息
    ├── b                    # 不隐藏时ps -el打印信息
    ├── Makefile
    ├── tmp.c                # 创建隐藏自己的内核线程的内核模块
```