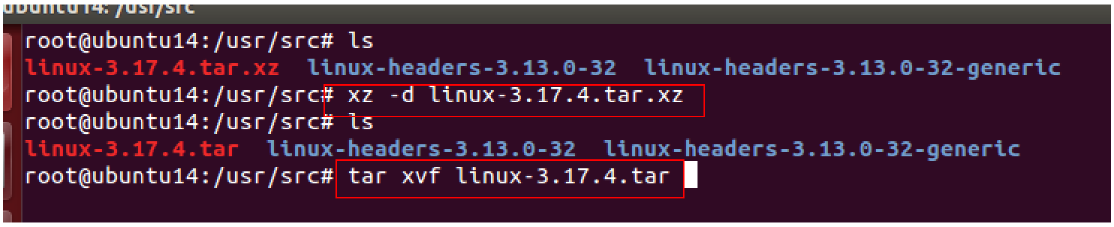
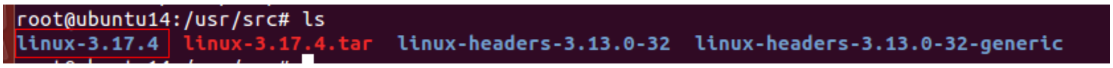
第十章实训 4 实验主要过程

1、　从网上下载一份内核 3.17.4：https://www.kernel.org/ 然后拷到/usr/src 目录下(为了后面操作的方便，建议切换到 root 用户)，然后执行：xz –d linux-3.17.4.tar.xz 得到 linux-3.17.4.tar，接着执行 tar xvf linux-3.17.4.tar



```
root@ubuntu14:/usr/src# ls
linux-3.17.4.tar.xz  linux-headers-3.13.0-32  linux-headers-3.13.0-32-generic
root@ubuntu14:/usr/src# xz -d linux-3.17.4.tar.xz
root@ubuntu14:/usr/src# ls
linux-3.17.4.tar  linux-headers-3.13.0-32  linux-headers-3.13.0-32-generic
root@ubuntu14:/usr/src# tar xvf linux-3.17.4.tar
```

2、经过前面两步解压过程，可以得到内核源码文件

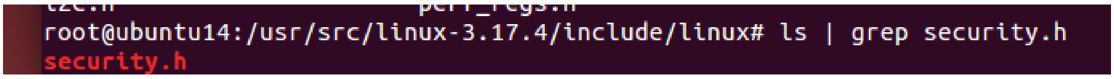


```
root@ubuntu14:/usr/src# ls
linux-3.17.4  linux-3.17.4.tar  linux-headers-3.13.0-32  linux-headers-3.13.0-32-generic
```

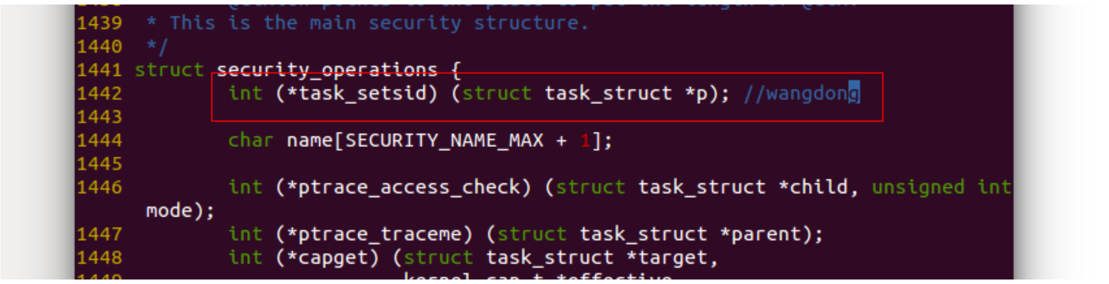
3、在 Include/linux/secuirty.h 下,

在结构体 struct security_operations 中添加:

int (*task_setsid) (struct task_struct *p);



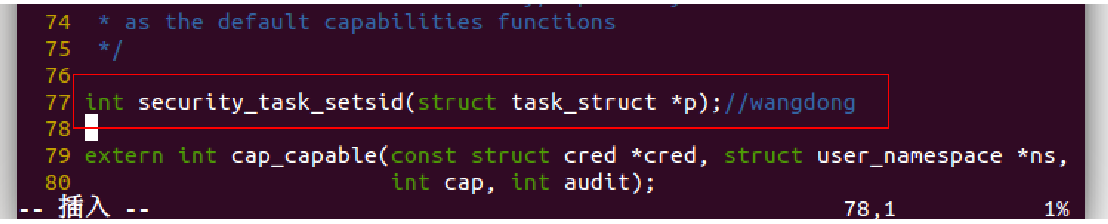
```
root@ubuntu14:/usr/src/linux-3.17.4/include/linux# ls | grep security.h
security.h
```



```
 * This is the main security structure.
 */
struct security_operations {
        int (*task_setsid) (struct task_struct *p); //wangdong

        char name[SECURITY_NAME_MAX + 1];

        int (*ptrace_access_check) (struct task_struct *child, unsigned int
 mode);
        int (*ptrace_traceme) (struct task_struct *parent);
        int (*capget) (struct task_struct *target,
```

4、另外在 Include/linux/secuirty.h 下添加函数声明:

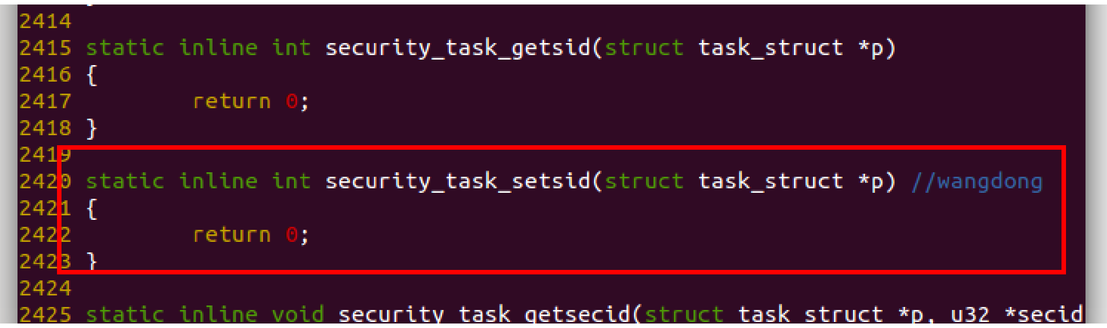
int security_task_setsid(struct task_struct *p);



```
 * as the default capabilities functions
 */

int security_task_setsid(struct task_struct *p);//wangdong

extern int cap_capable(const struct cred *cred, struct user_namespace *ns,
                       int cap, int audit);
-- 插入 --                                              78,1          1%
```

5、在 Include/linux/secuirty.h 增加函数定义:

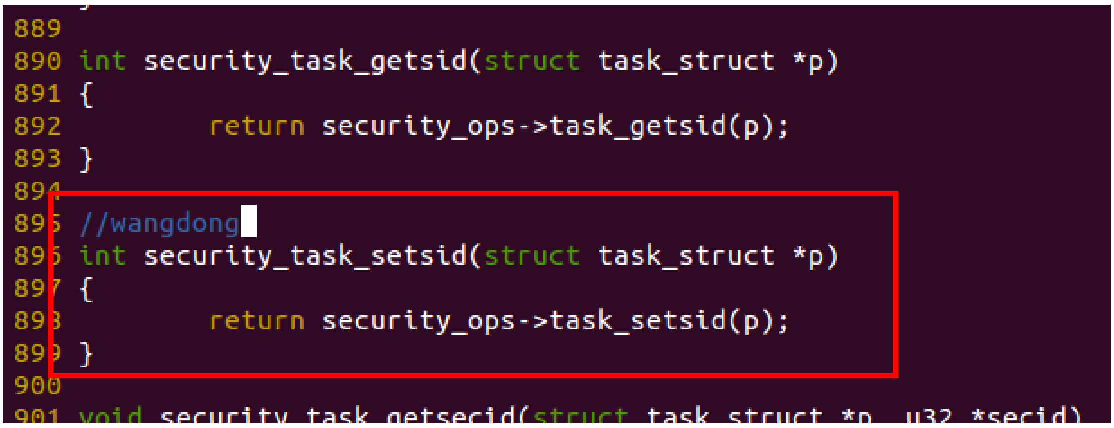
```
static inline int security_task_setsid(struct task_struct *p) //zxk

{

    return 0;

}
```



```

static inline int security_task_getsid(struct task_struct *p)
{
        return 0;
}

static inline int security_task_setsid(struct task_struct *p) //wangdong
{
        return 0;
}

static inline void security_task_getsecid(struct task_struct *p, u32 *secid
```

6、在 Security/security.c 定义函数:

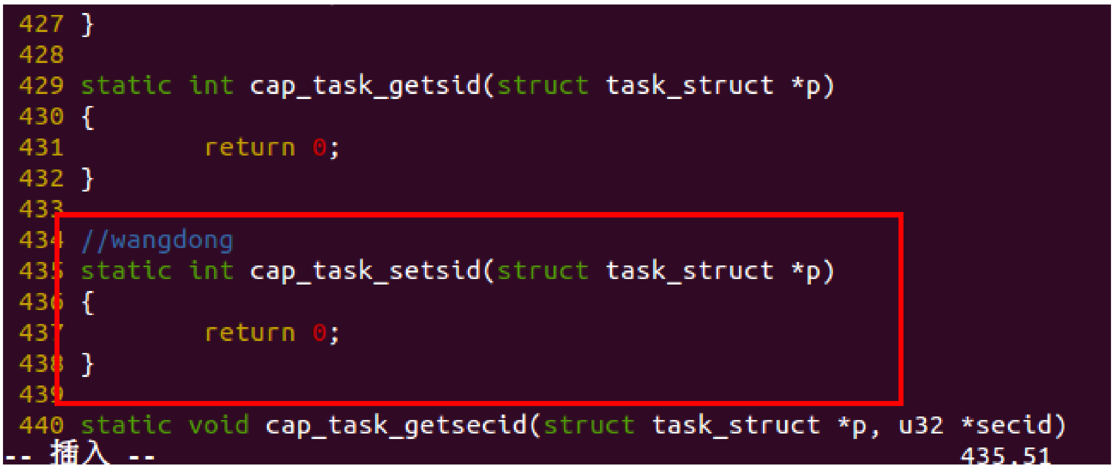
```
int security_task_setsid(struct task_struct *p)

{

    return security_ops->task_setsid(p);

}
```



```
}

int security_task_getsid(struct task_struct *p)
{
        return security_ops->task_getsid(p);
}

//wangdong
int security_task_setsid(struct task_struct *p)
{
        return security_ops->task_setsid(p);
}

void security_task_getsecid(struct task_struct *p, u32 *secid)
```

7、在 Security/capability.c 中定义函数

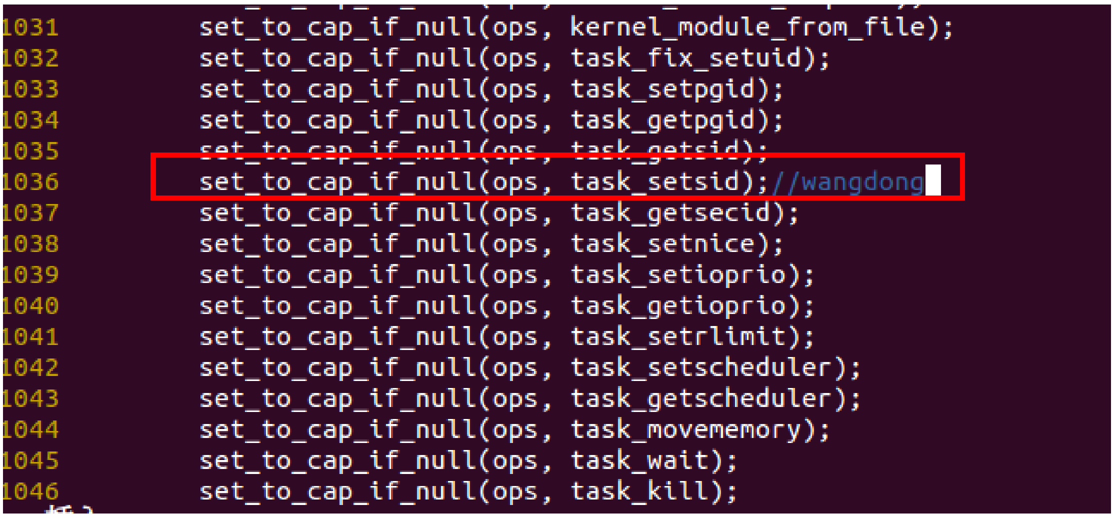
```
static int cap_task_setsid(struct task_struct *p)
{
    return 0;
}
```



```
}

static int cap_task_getsid(struct task_struct *p)
{
        return 0;
}

//wangdong
static int cap_task_setsid(struct task_struct *p)
{
        return 0;
}

static void cap_task_getsecid(struct task_struct *p, u32 *secid)
-- 插入 --                                                    435,51
```

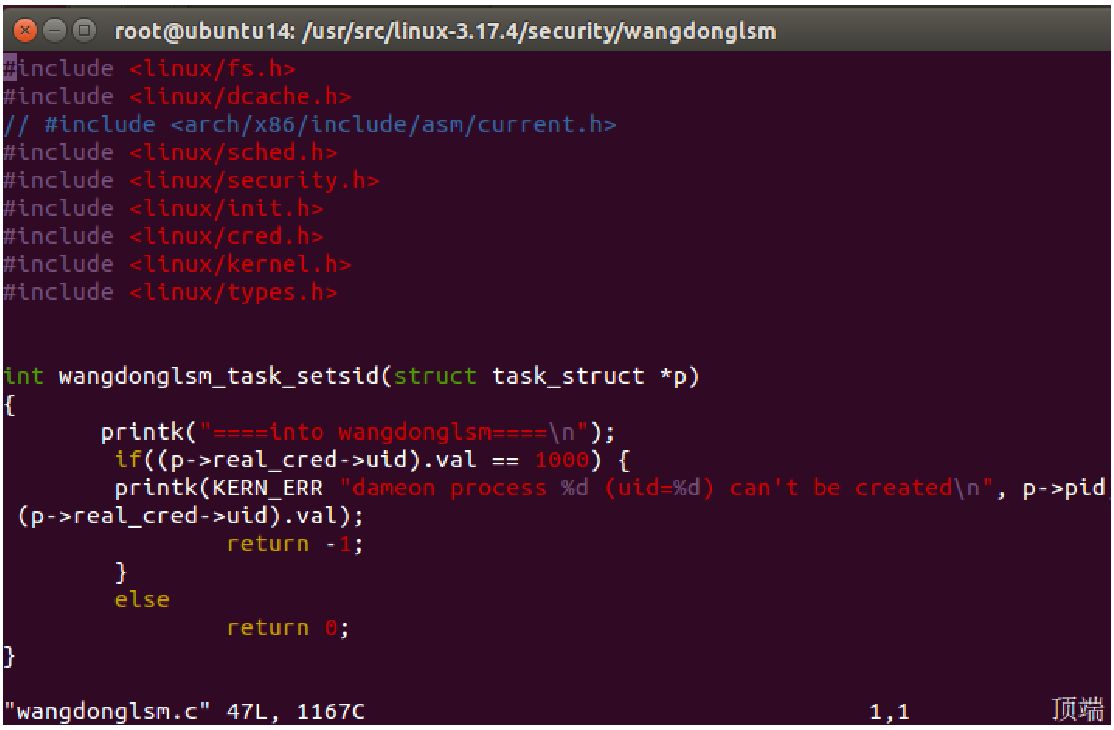
8、在 Security/capability.c 的 ecurity_fixup_ops 函数增加如下语句:

```
set_to_cap_if_null(ops, task_setsid);
```



```
        set_to_cap_if_null(ops, kernel_module_from_file);
        set_to_cap_if_null(ops, task_fix_setuid);
        set_to_cap_if_null(ops, task_setpgid);
        set_to_cap_if_null(ops, task_getpgid);
        set_to_cap_if_null(ops, task_getsid);
        set_to_cap_if_null(ops, task_setsid);//wangdong
        set_to_cap_if_null(ops, task_getsecid);
        set_to_cap_if_null(ops, task_setnice);
        set_to_cap_if_null(ops, task_setioprio);
        set_to_cap_if_null(ops, task_getioprio);
        set_to_cap_if_null(ops, task_setrlimit);
        set_to_cap_if_null(ops, task_setscheduler);
        set_to_cap_if_null(ops, task_getscheduler);
        set_to_cap_if_null(ops, task_movememory);
        set_to_cap_if_null(ops, task_wait);
        set_to_cap_if_null(ops, task_kill);
```

9、在新建的 wangdonglsm 目录下新建一个.c 文件，命名为 wangdonglsm.c，实现相应的功能



root@ubuntu14: /usr/src/linux-3.17.4/security/wangdonglsm

```
#include <linux/fs.h>
#include <linux/dcache.h>
// #include <arch/x86/include/asm/current.h>
#include <linux/sched.h>
#include <linux/security.h>
#include <linux/init.h>
#include <linux/cred.h>
#include <linux/kernel.h>
#include <linux/types.h>


int wangdonglsm_task_setsid(struct task_struct *p)
{
       printk("====into wangdonglsm====\n");
        if((p->real_cred->uid).val == 1000) {
        printk(KERN_ERR "dameon process %d (uid=%d) can't be created\n", p->pid
 (p->real_cred->uid).val);
                return -1;
        }
        else
                return 0;
}
"wangdonglsm.c" 47L, 1167C                                    1,1          顶端
```

参考代码：

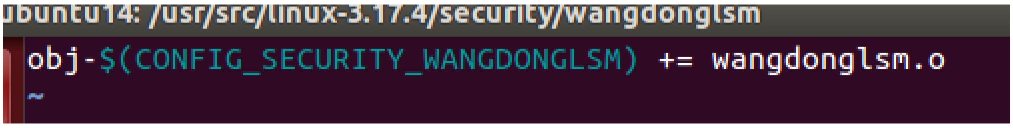
#include <linux/fs.h>

#include <linux/dcache.h>

#include <linux/sched.h>

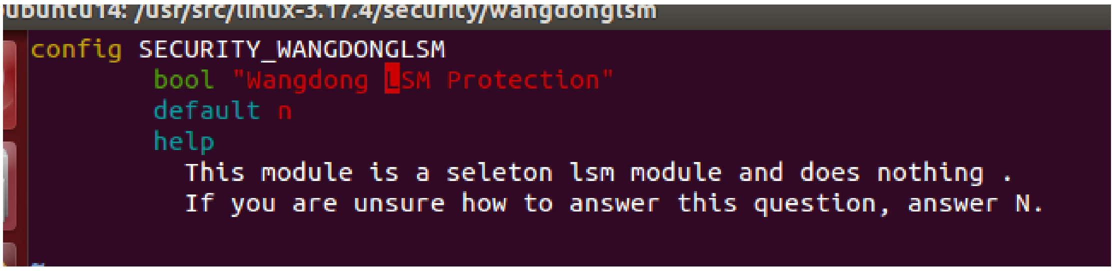
```
#include <linux/security.h>
#include <linux/init.h>
#include <linux/kernel.h>
#include <linux/types.h>


static int wangdonglsm_task_setsid(struct task_struct *p)
{
    int task_pid = p->pid;
    unsigned int task_uid    = (p->real_cred->uid).val;
    if(task_uid == 1000){
        printk("damenon task %d (uid=%d) shouldn't be created\n",task_pid, task_uid);
        return -1;
    }
    return 0;
}

static struct security_operations wangdonglsm_ops = {
     .name =  "wangdonglsm",
     .task_setsid = wangdonglsm_task_setsid,
};

static __init int wangdonglsm_init(void)
{
     if (register_security(&wangdonglsm_ops))
     {
          panic(KERN_INFO "Failed to register wangdonglsm module\n");
     }
     printk(KERN_ALERT "wangdonglsm started");
     return 0;
}

security_initcall(wangdonglsm_init);
```

10、在写完 wangdonglsm.c 文件之后，还是在 wangdonglsm 目录下编写 Makefile 文件



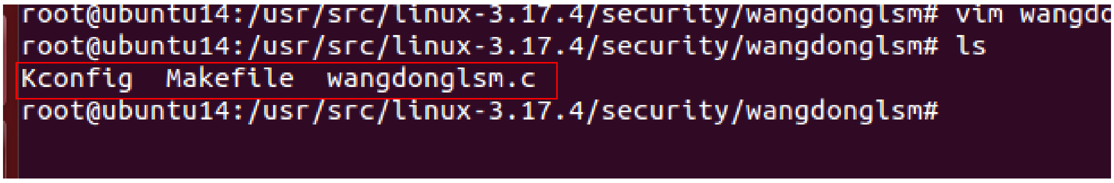
```
obj-$(CONFIG_SECURITY_WANGDONGLSM) += wangdonglsm.o
~
```

11、继续在 wangdonglsm 目录下编写 Kconfig 文件



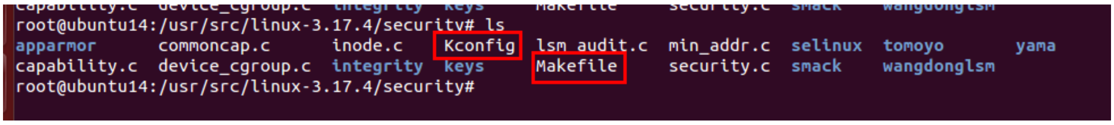
```
ubuntu14: /usr/src/linux-3.17.4/security/wangdonglsm
config SECURITY_WANGDONGLSM
        bool "Wangdong LSM Protection"
        default n
        help
          This module is a seleton lsm module and does nothing .
          If you are unsure how to answer this question, answer N.
```

12、最后我们在 wangdonglsm 这个目录下面保存 3 个文件，即 wangdonglsm.c、Makefile、Kconfig，如下图所示



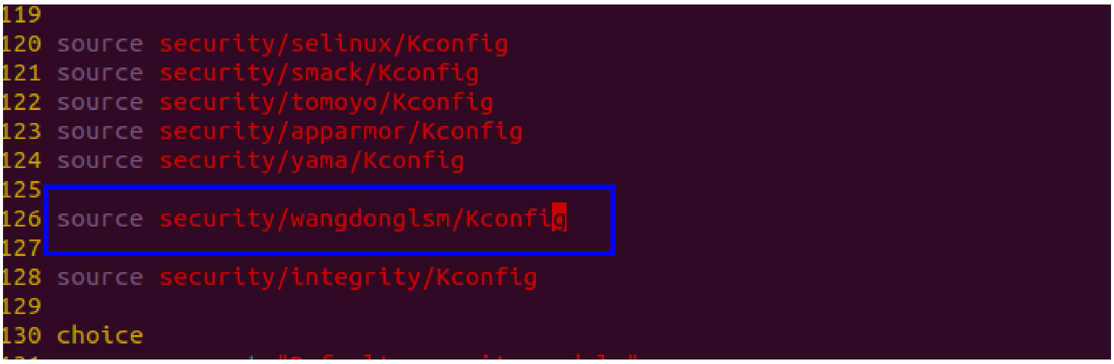
```
root@ubuntu14:/usr/src/linux-3.17.4/security/wangdonglsm# vim wangdo
root@ubuntu14:/usr/src/linux-3.17.4/security/wangdonglsm# ls
Kconfig  Makefile  wangdonglsm.c
root@ubuntu14:/usr/src/linux-3.17.4/security/wangdonglsm#
```

13、在写完之前的三个文件之后，我们返回到 /linux-3.17.4/security 目录下，找到这个目录下的 Kconfig 和 Makefile 文件，将我们之前在 wangdonglsm 目录下编写的 Kconfig 和 Makefile 文件的相关信息在对应的这两个文件中进行登记。



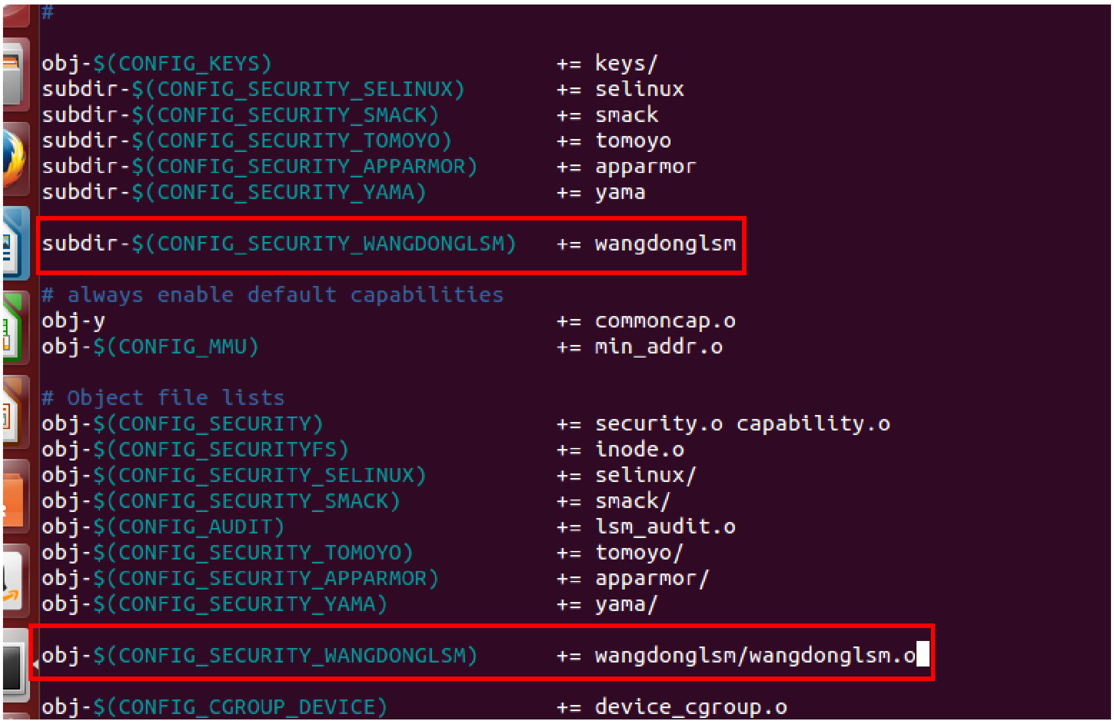
```
capability.c  device_cgroup.c  integrity  keys      Makefile      security.c  smack    wangdonglsm
root@ubuntu14:/usr/src/linux-3.17.4/security# ls
apparmor      commoncap.c      inode.c    Kconfig   lsm_audit.c   min_addr.c  selinux  tomoyo       yama
capability.c  device_cgroup.c  integrity  keys      Makefile      security.c  smack    wangdonglsm
root@ubuntu14:/usr/src/linux-3.17.4/security#
```

14、用 vim 打开 Kconfig 文件，添加下面一行



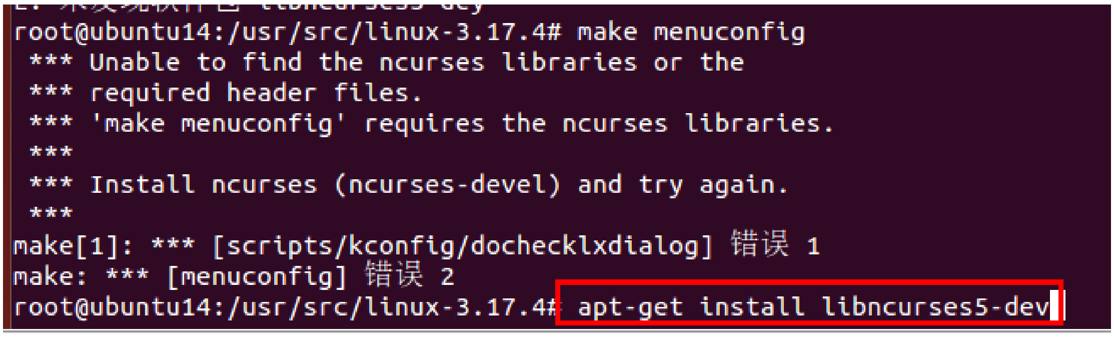
```
source security/selinux/Kconfig
source security/smack/Kconfig
source security/tomoyo/Kconfig
source security/apparmor/Kconfig
source security/yama/Kconfig

source security/wangdonglsm/Kconfig

source security/integrity/Kconfig

choice
```

15、用 vim 打开 Makefile 文件，添加下面两行



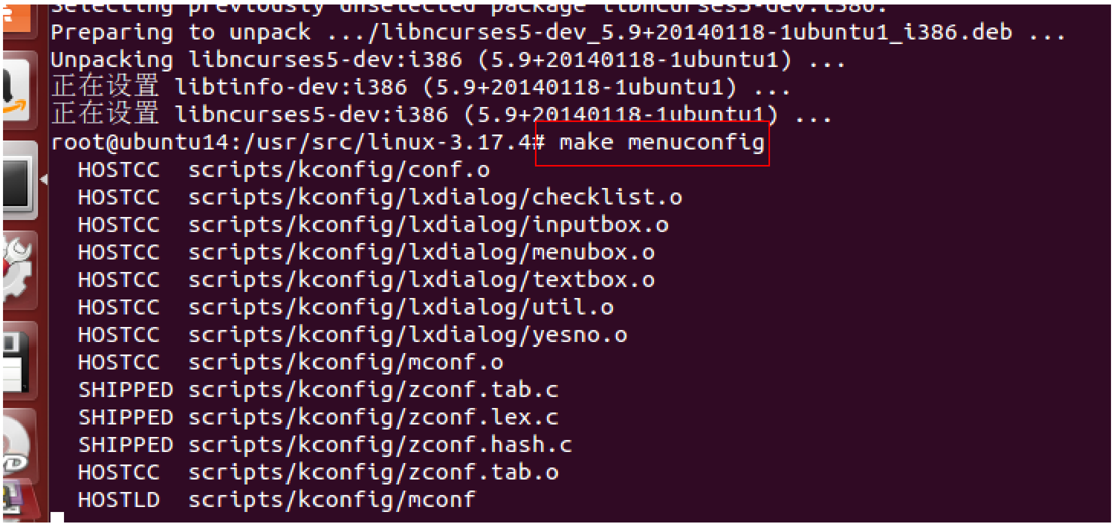
```
#

obj-$(CONFIG_KEYS)                      += keys/
subdir-$(CONFIG_SECURITY_SELINUX)       += selinux
subdir-$(CONFIG_SECURITY_SMACK)         += smack
subdir-$(CONFIG_SECURITY_TOMOYO)        += tomoyo
subdir-$(CONFIG_SECURITY_APPARMOR)      += apparmor
subdir-$(CONFIG_SECURITY_YAMA)          += yama

subdir-$(CONFIG_SECURITY_WANGDONGLSM)   += wangdonglsm

# always enable default capabilities
obj-y                                   += commoncap.o
obj-$(CONFIG_MMU)                       += min_addr.o

# Object file lists
obj-$(CONFIG_SECURITY)                  += security.o capability.o
obj-$(CONFIG_SECURITYFS)                += inode.o
obj-$(CONFIG_SECURITY_SELINUX)          += selinux/
obj-$(CONFIG_SECURITY_SMACK)            += smack/
obj-$(CONFIG_AUDIT)                     += lsm_audit.o
obj-$(CONFIG_SECURITY_TOMOYO)           += tomoyo/
obj-$(CONFIG_SECURITY_APPARMOR)         += apparmor/
obj-$(CONFIG_SECURITY_YAMA)             += yama/

obj-$(CONFIG_SECURITY_WANGDONGLSM)      += wangdonglsm/wangdonglsm.o

obj-$(CONFIG_CGROUP_DEVICE)             += device_cgroup.o
```

16、到这一步，我们自己编写的内核安全模块的准备工作算是完成了，下面就可以开始编译内核了，首先执行 make menuconfig 命令，如果出现如图所示错误，那么先按照 ncurse，即执行 apt-get install libncurses5-dev



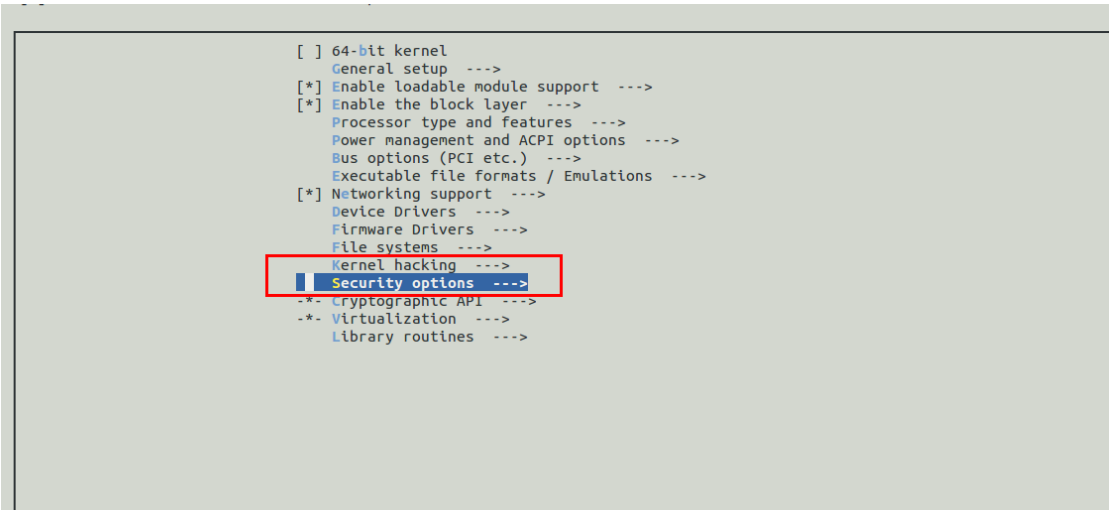
```
E: 未发现软件包 libncurses5-dev
root@ubuntu14:/usr/src/linux-3.17.4# make menuconfig
 *** Unable to find the ncurses libraries or the
 *** required header files.
 *** 'make menuconfig' requires the ncurses libraries.
 ***
 *** Install ncurses (ncurses-devel) and try again.
 ***
make[1]: *** [scripts/kconfig/dochecklxdialog] 错误 1
make: *** [menuconfig] 错误 2
root@ubuntu14:/usr/src/linux-3.17.4# apt-get install libncurses5-dev
```

17、安装完 ncurse 之后，再输入 make menuconfig



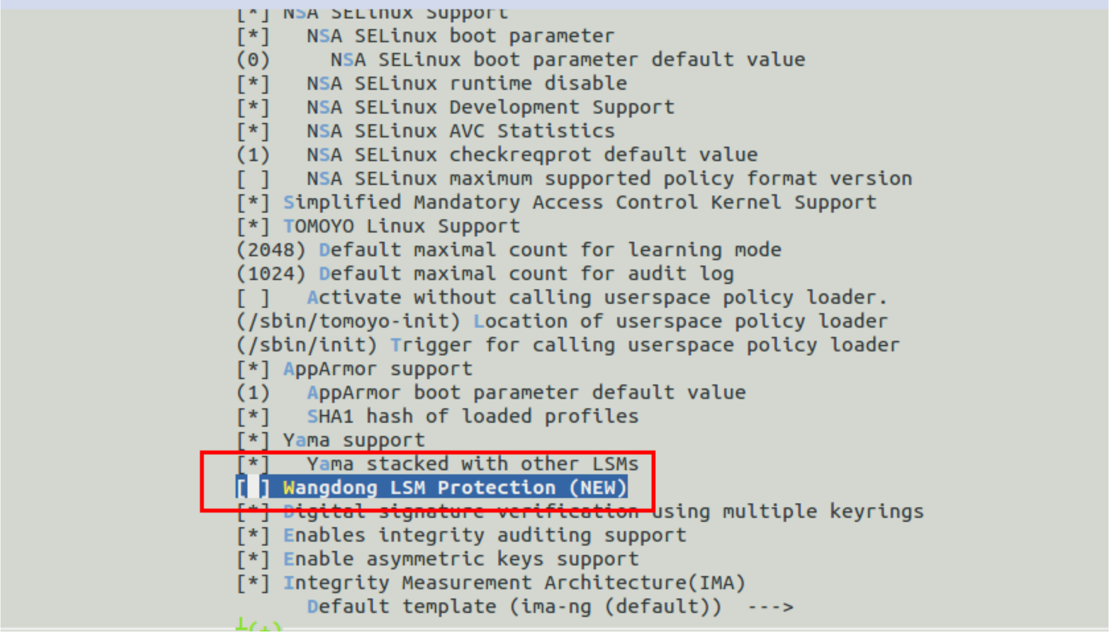
```
Selecting previously unselected package libncurses5-dev:i386.
Preparing to unpack .../libncurses5-dev_5.9+20140118-1ubuntu1_i386.deb ...
Unpacking libncurses5-dev:i386 (5.9+20140118-1ubuntu1) ...
正在设置 libtinfo-dev:i386 (5.9+20140118-1ubuntu1) ...
正在设置 libncurses5-dev:i386 (5.9+20140118-1ubuntu1) ...
root@ubuntu14:/usr/src/linux-3.17.4# make menuconfig
  HOSTCC  scripts/kconfig/conf.o
  HOSTCC  scripts/kconfig/lxdialog/checklist.o
  HOSTCC  scripts/kconfig/lxdialog/inputbox.o
  HOSTCC  scripts/kconfig/lxdialog/menubox.o
  HOSTCC  scripts/kconfig/lxdialog/textbox.o
  HOSTCC  scripts/kconfig/lxdialog/util.o
  HOSTCC  scripts/kconfig/lxdialog/yesno.o
  HOSTCC  scripts/kconfig/mconf.o
  SHIPPED scripts/kconfig/zconf.tab.c
  SHIPPED scripts/kconfig/zconf.lex.c
  SHIPPED scripts/kconfig/zconf.hash.c
  HOSTCC  scripts/kconfig/zconf.tab.o
  HOSTLD  scripts/kconfig/mconf
```

18、会弹出以下的图形界面，找到 Security options 选项，按回车键进入

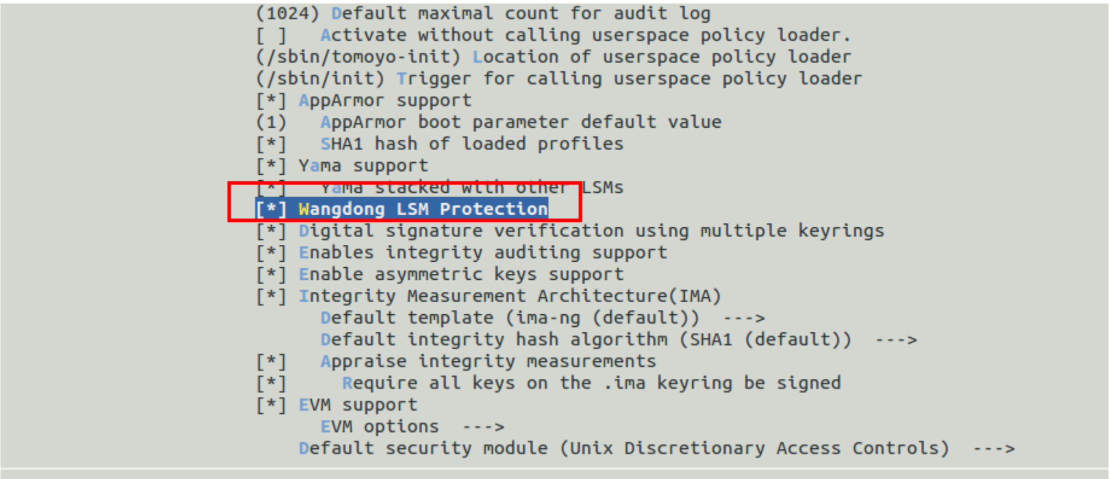


```
[ ] 64-bit kernel
    General setup  --->
[*] Enable loadable module support  --->
[*] Enable the block layer  --->
    Processor type and features  --->
    Power management and ACPI options  --->
    Bus options (PCI etc.)  --->
    Executable file formats / Emulations  --->
[*] Networking support  --->
    Device Drivers  --->
    Firmware Drivers  --->
    File systems  --->
    Kernel hacking  --->
    Security options  --->
-*- Cryptographic API  --->
-*- Virtualization  --->
    Library routines  --->
```

19、找到我们自己编写的内核安全模块，如下图所示的Wangdong LSM Protection，输入 y，然后会在这个选项前面加入一个*号，即表示我们要将这个模块编译进内核

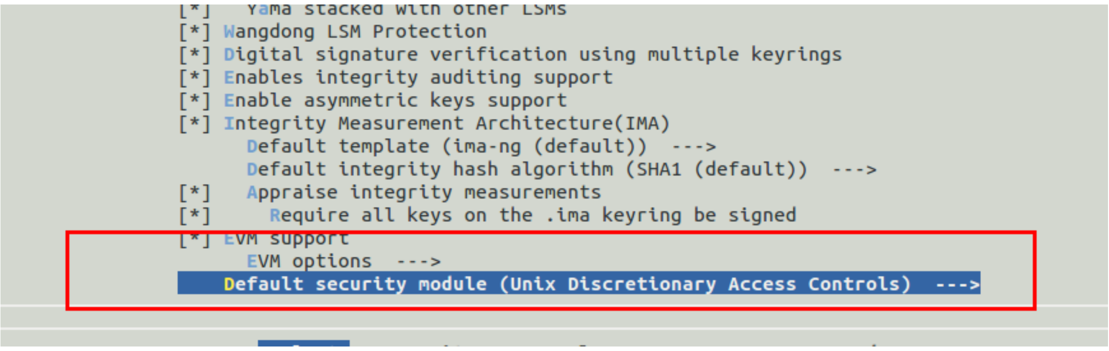


```
[*] NSA SELinux Support
[*]   NSA SELinux boot parameter
(0)     NSA SELinux boot parameter default value
[*]   NSA SELinux runtime disable
[*]   NSA SELinux Development Support
[*]   NSA SELinux AVC Statistics
(1)   NSA SELinux checkreqprot default value
[ ]   NSA SELinux maximum supported policy format version
[*] Simplified Mandatory Access Control Kernel Support
[*] TOMOYO Linux Support
(2048) Default maximal count for learning mode
(1024) Default maximal count for audit log
[ ]   Activate without calling userspace policy loader.
(/sbin/tomoyo-init) Location of userspace policy loader
(/sbin/init) Trigger for calling userspace policy loader
[*] AppArmor support
(1)   AppArmor boot parameter default value
[*]   SHA1 hash of loaded profiles
[*] Yama support
[*]   Yama stacked with other LSMs
[ ] Wangdong LSM Protection (NEW)
[*] Digital signature verification using multiple keyrings
[*] Enables integrity auditing support
[*] Enable asymmetric keys support
[*] Integrity Measurement Architecture(IMA)
      Default template (ima-ng (default))  --->
```

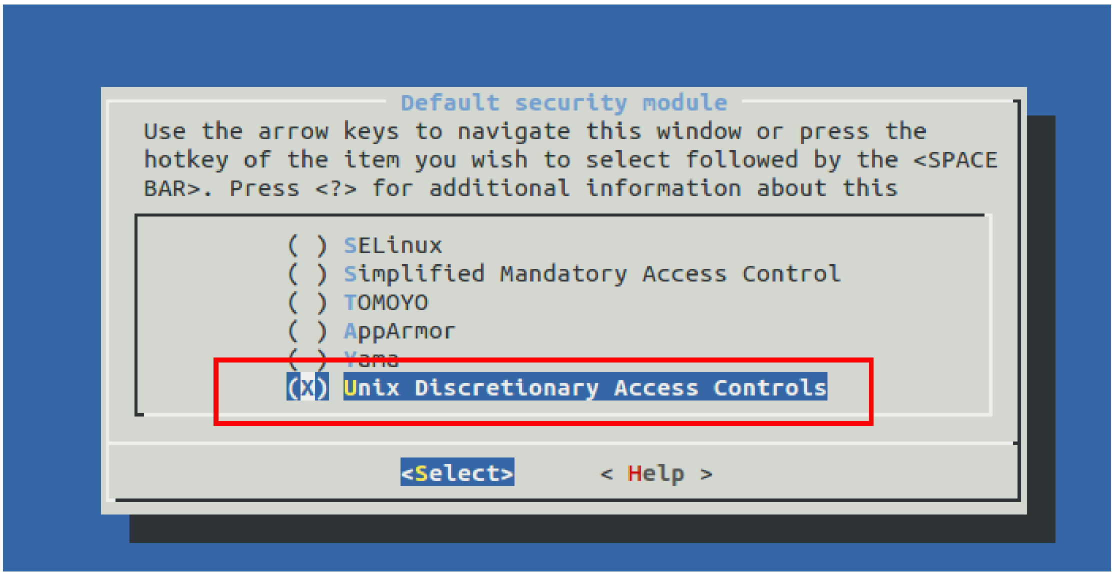


```
(1024) Default maximal count for audit log
[ ]   Activate without calling userspace policy loader.
(/sbin/tomoyo-init) Location of userspace policy loader
(/sbin/init) Trigger for calling userspace policy loader
[*] AppArmor support
(1)   AppArmor boot parameter default value
[*]   SHA1 hash of loaded profiles
[*] Yama support
[*]   Yama stacked with other LSMs
[*] Wangdong LSM Protection
[*] Digital signature verification using multiple keyrings
[*] Enables integrity auditing support
[*] Enable asymmetric keys support
[*] Integrity Measurement Architecture(IMA)
      Default template (ima-ng (default))  --->
      Default integrity hash algorithm (SHA1 (default))  --->
[*]   Appraise integrity measurements
[*]     Require all keys on the .ima keyring be signed
[*] EVM support
      EVM options  --->
    Default security module (Unix Discretionary Access Controls)  --->
```

20、这一步骤比较关键，选择 default security modules 选项，按回车进入，然后选择 Unix Discretionary Access Controls，即选择自主访问模式，否则可能无法正确将我们自己的内核安全模块编译进内核

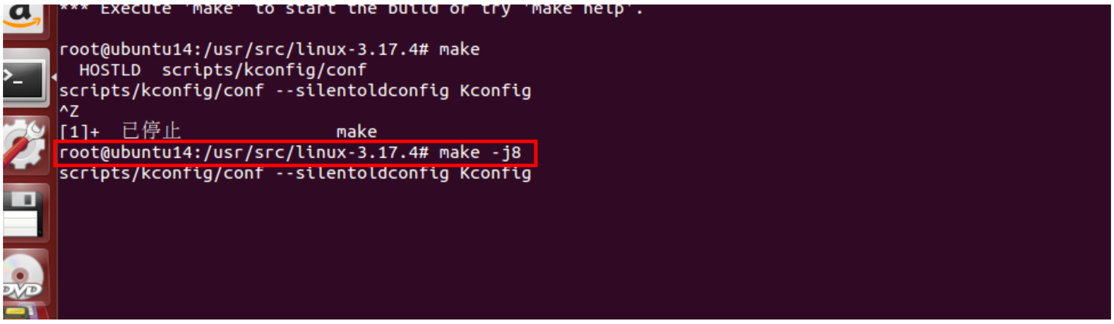


```
[*]   Yama stacked with other LSMs
[*] Wangdong LSM Protection
[*] Digital signature verification using multiple keyrings
[*] Enables integrity auditing support
[*] Enable asymmetric keys support
[*] Integrity Measurement Architecture(IMA)
      Default template (ima-ng (default))  --->
      Default integrity hash algorithm (SHA1 (default))  --->
[*]   Appraise integrity measurements
[*]     Require all keys on the .ima keyring be signed
[*] EVM support
      EVM options  --->
    Default security module (Unix Discretionary Access Controls)  --->
```



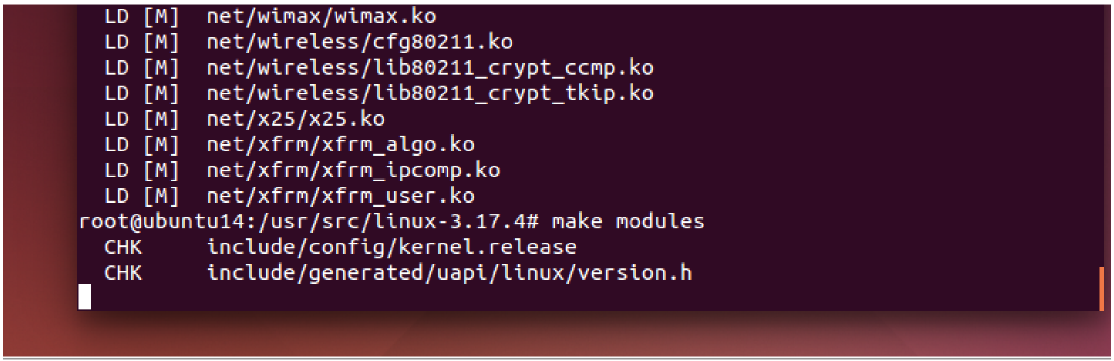
```
                     Default security module
   Use the arrow keys to navigate this window or press the
   hotkey of the item you wish to select followed by the <SPACE
   BAR>. Press <?> for additional information about this

               ( ) SELinux
               ( ) Simplified Mandatory Access Control
               ( ) TOMOYO
               ( ) AppArmor
               ( ) Yama
               (X) Unix Discretionary Access Controls

                    <Select>      < Help >
```

21、将之前做的选择保存下来，然后退出，之后就可以进入 make 阶段，为了加速编译过程，可以开多个线程进行，这里是开了 8 个线程。第一次编译的话这一阶段会很耗时



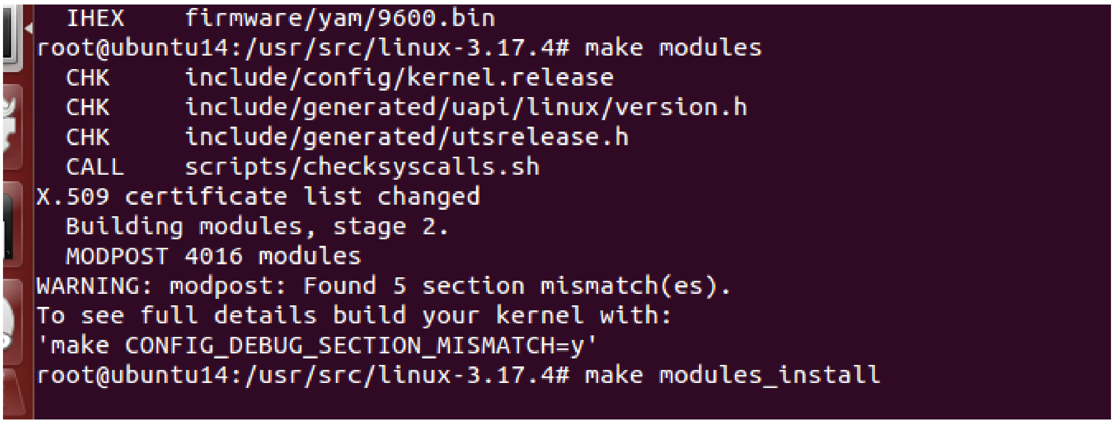
```
*** Execute 'make' to start the build or try 'make help'.

root@ubuntu14:/usr/src/linux-3.17.4# make
  HOSTLD  scripts/kconfig/conf
scripts/kconfig/conf --silentoldconfig Kconfig
^Z
[1]+  已停止               make
root@ubuntu14:/usr/src/linux-3.17.4# make -j8
scripts/kconfig/conf --silentoldconfig Kconfig
```

22、make 完成之后，执行 make　modules



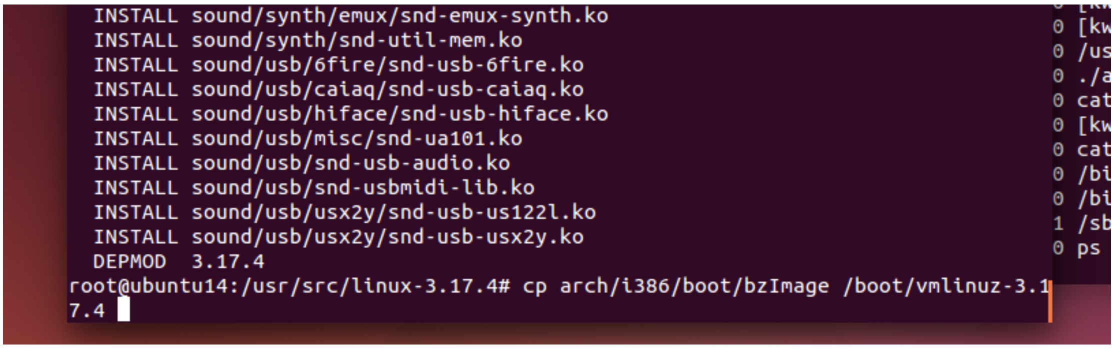
```
  LD [M]  net/wimax/wimax.ko
  LD [M]  net/wireless/cfg80211.ko
  LD [M]  net/wireless/lib80211_crypt_ccmp.ko
  LD [M]  net/wireless/lib80211_crypt_tkip.ko
  LD [M]  net/x25/x25.ko
  LD [M]  net/xfrm/xfrm_algo.ko
  LD [M]  net/xfrm/xfrm_ipcomp.ko
  LD [M]  net/xfrm/xfrm_user.ko
root@ubuntu14:/usr/src/linux-3.17.4# make modules
  CHK     include/config/kernel.release
  CHK     include/generated/uapi/linux/version.h
```

23、make　modules 之后执行 make　modules_install



```
  IHEX    firmware/yam/9600.bin
root@ubuntu14:/usr/src/linux-3.17.4# make modules
  CHK     include/config/kernel.release
  CHK     include/generated/uapi/linux/version.h
  CHK     include/generated/utsrelease.h
  CALL    scripts/checksyscalls.sh
X.509 certificate list changed
  Building modules, stage 2.
  MODPOST 4016 modules
WARNING: modpost: Found 5 section mismatch(es).
To see full details build your kernel with:
'make CONFIG_DEBUG_SECTION_MISMATCH=y'
root@ubuntu14:/usr/src/linux-3.17.4# make modules_install
```

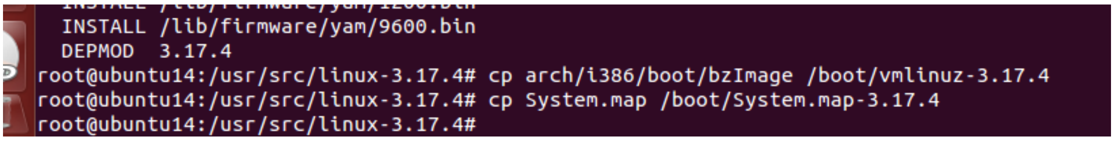
24、make modules_install 之后执行 cp arch/i386/boot/bzImage /boot/vmlinuz-3.17.4



```
  INSTALL sound/synth/emux/snd-emux-synth.ko
  INSTALL sound/synth/snd-util-mem.ko
  INSTALL sound/usb/6fire/snd-usb-6fire.ko
  INSTALL sound/usb/caiaq/snd-usb-caiaq.ko
  INSTALL sound/usb/hiface/snd-usb-hiface.ko
  INSTALL sound/usb/misc/snd-ua101.ko
  INSTALL sound/usb/snd-usb-audio.ko
  INSTALL sound/usb/snd-usbmidi-lib.ko
  INSTALL sound/usb/usx2y/snd-usb-us122l.ko
  INSTALL sound/usb/usx2y/snd-usb-usx2y.ko
  DEPMOD  3.17.4
root@ubuntu14:/usr/src/linux-3.17.4# cp arch/i386/boot/bzImage /boot/vmlinuz-3.1
7.4
```

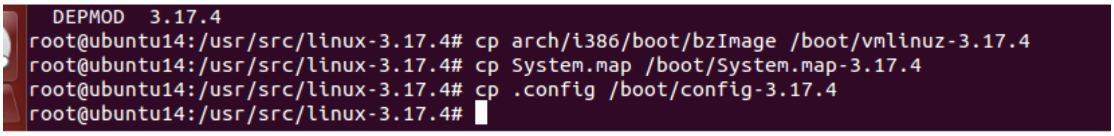
25、上一步执行完之后执行：

cp System.map /boot/System.map-3.17.4



```
  INSTALL /lib/firmware/yam/9600.bin
  DEPMOD  3.17.4
root@ubuntu14:/usr/src/linux-3.17.4# cp arch/i386/boot/bzImage /boot/vmlinuz-3.17.4
root@ubuntu14:/usr/src/linux-3.17.4# cp System.map /boot/System.map-3.17.4
root@ubuntu14:/usr/src/linux-3.17.4#
```

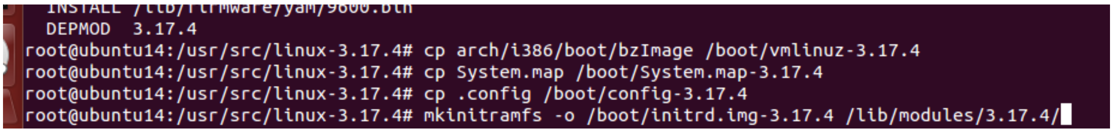
26、然后执行 cp .config /boot/config-3.17.4



```
  DEPMOD  3.17.4
root@ubuntu14:/usr/src/linux-3.17.4# cp arch/i386/boot/bzImage /boot/vmlinuz-3.17.4
root@ubuntu14:/usr/src/linux-3.17.4# cp System.map /boot/System.map-3.17.4
root@ubuntu14:/usr/src/linux-3.17.4# cp .config /boot/config-3.17.4
root@ubuntu14:/usr/src/linux-3.17.4#
```

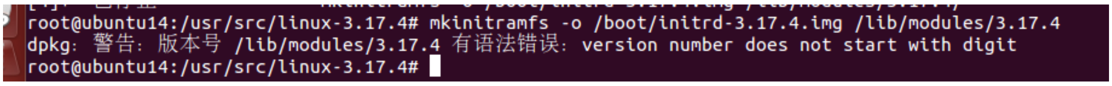
27、然后执行

mkinitramfs -o /boot/initrd.img-3.17.4 /lib/modules/3.17.4



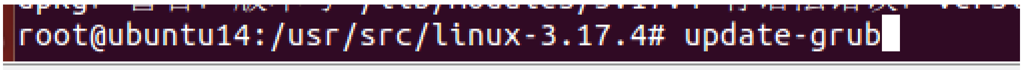
```
  INSTALL /lib/firmware/yam/9600.bin
  DEPMOD  3.17.4
root@ubuntu14:/usr/src/linux-3.17.4# cp arch/i386/boot/bzImage /boot/vmlinuz-3.17.4
root@ubuntu14:/usr/src/linux-3.17.4# cp System.map /boot/System.map-3.17.4
root@ubuntu14:/usr/src/linux-3.17.4# cp .config /boot/config-3.17.4
root@ubuntu14:/usr/src/linux-3.17.4# mkinitramfs -o /boot/initrd.img-3.17.4 /lib/modules/3.17.4/
```

28、执行这一命令可能会有一个警告，可忽略



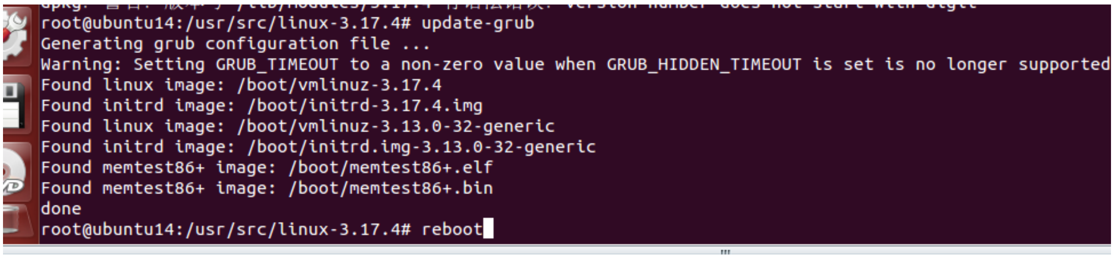
```
root@ubuntu14:/usr/src/linux-3.17.4# mkinitramfs -o /boot/initrd-3.17.4.img /lib/modules/3.17.4
dpkg: 警告: 版本号 /lib/modules/3.17.4 有语法错误: version number does not start with digit
root@ubuntu14:/usr/src/linux-3.17.4# 
```

29、最后更新 grub



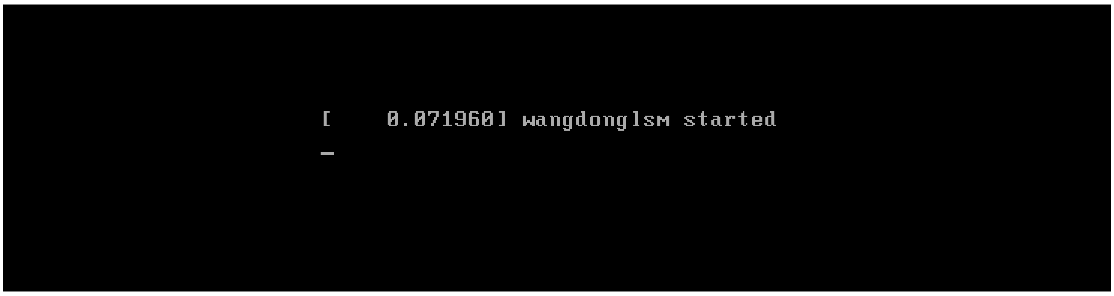
```
root@ubuntu14:/usr/src/linux-3.17.4# update-grub
```

30、然后重启，如果 reboot 无法重启，可强行重启



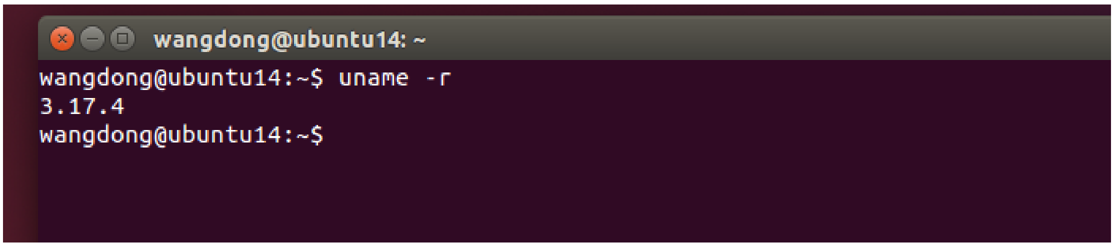
```
root@ubuntu14:/usr/src/linux-3.17.4# update-grub
Generating grub configuration file ...
Warning: Setting GRUB_TIMEOUT to a non-zero value when GRUB_HIDDEN_TIMEOUT is set is no longer supported
Found linux image: /boot/vmlinuz-3.17.4
Found initrd image: /boot/initrd-3.17.4.img
Found linux image: /boot/vmlinuz-3.13.0-32-generic
Found initrd image: /boot/initrd.img-3.13.0-32-generic
Found memtest86+ image: /boot/memtest86+.elf
Found memtest86+ image: /boot/memtest86+.bin
done
root@ubuntu14:/usr/src/linux-3.17.4# reboot
```

31、重启过程中可看到我们的内核安全模块被加载



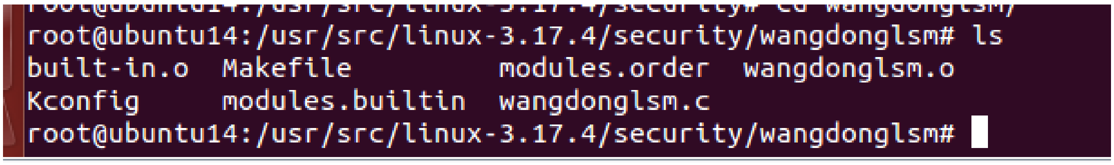
```
[    0.071960] wangdonglsm started
_
```

32、重启进去系统后可检查一下内核版本



```
wangdong@ubuntu14:~$ uname -r
3.17.4
wangdong@ubuntu14:~$
```

33、我们可以进入之前创建的 wangdonglsm 目录下查看，发现多了几个文件，即该内核安全模块被编译进内核后产生了其他的一些必须文件。



```
root@ubuntu14:/usr/src/linux-3.17.4/security# cd wangdonglsm/
root@ubuntu14:/usr/src/linux-3.17.4/security/wangdonglsm# ls
built-in.o  Makefile        modules.order  wangdonglsm.o
Kconfig     modules.builtin  wangdonglsm.c
root@ubuntu14:/usr/src/linux-3.17.4/security/wangdonglsm#
```

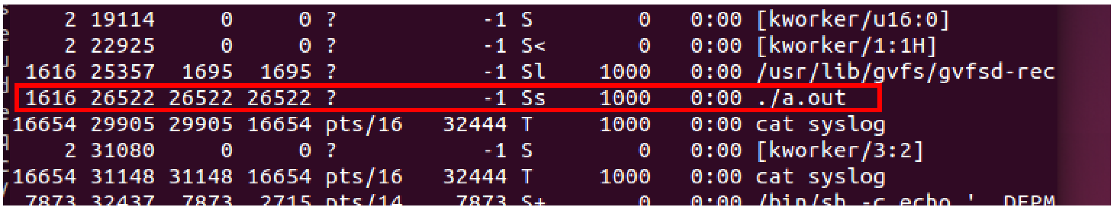
34、接下来使用子任务 2 编写的守护进程测试一下内核安全模块是否能正确工作，注意需要在普通用户下编写守护进程，即 uid=1000 的用户

参考代码

```
#include <sys/types.h>
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/stat.h>
#include <syslog.h>

int main(void){
   pid_t pid;
   pid = fork();
   if(pid <0){
      perror("fork failed\n");
```

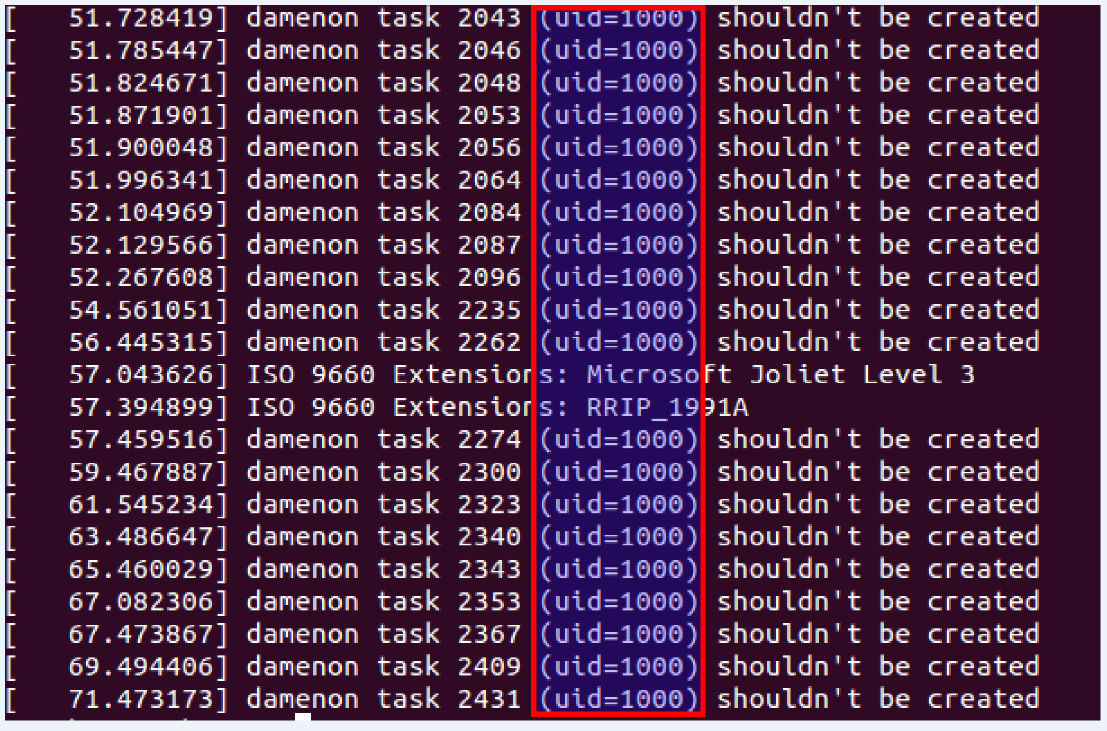
```
        exit(-1);
    }
    else if( pid > 0){
        exit(0);
    }
    int sid;
    if((sid =setsid()) < 0 ){
        syslog(LOG_INFO, "dameon ZXK PROCESS %d can't be created!\n", getpid());
        return -1;
    }
    close(0);
    close(1);
    close(2);
    umask(0);
    chdir("/");

    syslog(LOG_INFO, "GUER PROCESS %d\n", getpid());

    while(1){

    }

    return 0;
}
```

在实现我们的内核安全模块之前，是可以在 uid=1000 的状态下编写守护进程的，即执行上面的代码：gcc guer.c，然后得到 a.out，接着执行./a.out，然后我们通过 ps –axj 可以找到这个进程，如下图所示。



```
    2 19114     0     0 ?           -1 S        0   0:00 [kworker/u16:0]
    2 22925     0     0 ?           -1 S<       0   0:00 [kworker/1:1H]
 1616 25357  1695  1695 ?           -1 Sl    1000   0:00 /usr/lib/gvfs/gvfsd-rec
 1616 26522 26522 26522 ?           -1 Ss    1000   0:00 ./a.out
16654 29905 29905 16654 pts/16   32444 T     1000   0:00 cat syslog
    2 31080     0     0 ?           -1 S        0   0:00 [kworker/3:2]
16654 31148 31148 16654 pts/16   32444 T     1000   0:00 cat syslog
 7873 32437  7873  2715 pts/14    7873 S+       0   0:00 /bin/sh -c echo '   DEPM
```

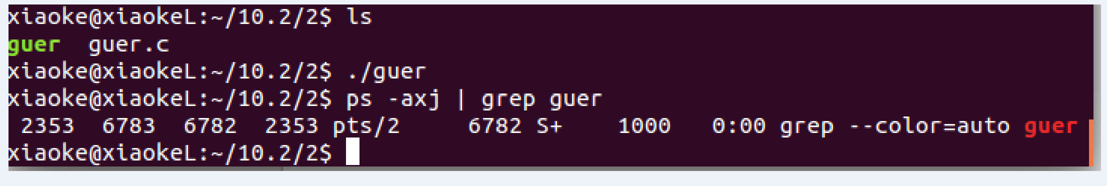
35、在内核安全模块加载之后，系统重启之后，通过 dmesg 查看，可以看到以下信息：



```
[   51.728419] damenon task 2043 (uid=1000) shouldn't be created
[   51.785447] damenon task 2046 (uid=1000) shouldn't be created
[   51.824671] damenon task 2048 (uid=1000) shouldn't be created
[   51.871901] damenon task 2053 (uid=1000) shouldn't be created
[   51.900048] damenon task 2056 (uid=1000) shouldn't be created
[   51.996341] damenon task 2064 (uid=1000) shouldn't be created
[   52.104969] damenon task 2084 (uid=1000) shouldn't be created
[   52.129566] damenon task 2087 (uid=1000) shouldn't be created
[   52.267608] damenon task 2096 (uid=1000) shouldn't be created
[   54.561051] damenon task 2235 (uid=1000) shouldn't be created
[   56.445315] damenon task 2262 (uid=1000) shouldn't be created
[   57.043626] ISO 9660 Extensions: Microsoft Joliet Level 3
[   57.394899] ISO 9660 Extensions: RRIP_1991A
[   57.459516] damenon task 2274 (uid=1000) shouldn't be created
[   59.467887] damenon task 2300 (uid=1000) shouldn't be created
[   61.545234] damenon task 2323 (uid=1000) shouldn't be created
[   63.486647] damenon task 2340 (uid=1000) shouldn't be created
[   65.460029] damenon task 2343 (uid=1000) shouldn't be created
[   67.082306] damenon task 2353 (uid=1000) shouldn't be created
[   67.473867] damenon task 2367 (uid=1000) shouldn't be created
[   69.494406] damenon task 2409 (uid=1000) shouldn't be created
[   71.473173] damenon task 2431 (uid=1000) shouldn't be created
```

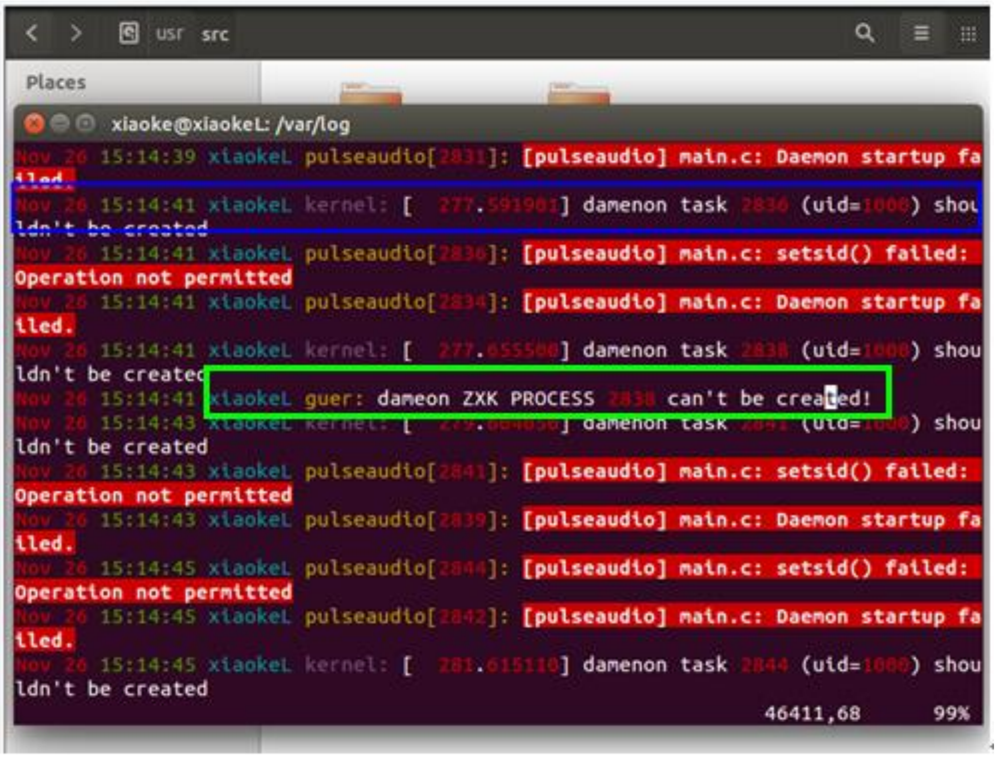
即表示 uid=1000 的进程被禁止成为守护进程。

36、然后我们使用普通用户执行以下之前的那个守护进程：gcc –o guer guer.c；然后执行./guer，然后通过 ps –axj 查找刚才的这个进程，发现找不到，这与我们预期是一致的。



```
xiaoke@xiaokeL:~/10.2/2$ ls
guer  guer.c
xiaoke@xiaokeL:~/10.2/2$ ./guer
xiaoke@xiaokeL:~/10.2/2$ ps -axj | grep guer
 2353  6783  6782  2353 pts/2     6782 S+    1000   0:00 grep --color=auto guer
xiaoke@xiaokeL:~/10.2/2$ 
```

37、与此同时，我们去/var/log/syslog 查看也可以看到相应的错误提示信息：



```
xiaoke@xiaokeL: /var/log
Nov 26 15:14:39 xiaokeL pulseaudio[2831]: [pulseaudio] main.c: Daemon startup fa
iled.
Nov 26 15:14:41 xiaokeL kernel: [  277.591901] damenon task 2836 (uid=1000) shou
ldn't be created
Nov 26 15:14:41 xiaokeL pulseaudio[2836]: [pulseaudio] main.c: setsid() failed:
Operation not permitted
Nov 26 15:14:41 xiaokeL pulseaudio[2834]: [pulseaudio] main.c: Daemon startup fa
iled.
Nov 26 15:14:41 xiaokeL kernel: [  277.655500] damenon task 2838 (uid=1000) shou
ldn't be created
Nov 26 15:14:41 xiaokeL guer: dameon ZXK PROCESS 2838 can't be created!
Nov 26 15:14:43 xiaokeL kernel: [  279.604656] damenon task 2841 (uid=1000) shou
ldn't be created
Nov 26 15:14:43 xiaokeL pulseaudio[2841]: [pulseaudio] main.c: setsid() failed:
Operation not permitted
Nov 26 15:14:43 xiaokeL pulseaudio[2839]: [pulseaudio] main.c: Daemon startup fa
iled.
Nov 26 15:14:45 xiaokeL pulseaudio[2844]: [pulseaudio] main.c: setsid() failed:
Operation not permitted
Nov 26 15:14:45 xiaokeL pulseaudio[2842]: [pulseaudio] main.c: Daemon startup fa
iled.
Nov 26 15:14:45 xiaokeL kernel: [  281.615110] damenon task 2844 (uid=1000) shou
ldn't be created
                                                              46411,68      99%
```

38、到此为止，实训 4 就算是完成了。