先了解一下整个 inode_create, inode_permission, task_create 在内核代码中被调用的位置，查看代码后，当模块注册后，会调用函数进行创建相关内容，其实三个都差不多的过程，如 inode 会先指定.inode = ***，这在 security.h 函数中的 security_operations 结构体中定义了相关的钩子函数。

```
struct security_operations {
        char name[SECURITY_NAME_MAX + 1];

        int (*ptrace_access_check) (struct task_struct *child, unsigned int m
        int (*ptrace_traceme) (struct task_struct *parent);
        int (*capget) (struct task_struct *target,
                       kernel_cap_t *effective,
                       kernel_cap_t *inheritable, kernel_cap_t *permitted);
        int (*capset) (struct cred *new,
                       const struct cred *old,
                       const kernel_cap_t *effective,
                       const kernel_cap_t *inheritable,
                       const kernel_cap_t *permitted);
        int (*capable) (const struct cred *cred, struct user_namespace *ns,
                        int cap, int audit);
        int (*quotactl) (int cmds, int type, int id, struct super_block *sb);
        int (*quota_on) (struct dentry *dentry);
        int (*syslog) (int type);
        int (*settime) (const struct timespec *ts, const struct timezone *tz);
        int (*vm_enough_memory) (struct mm_struct *mm, long pages);

        int (*bprm_set_creds) (struct linux_binprm *bprm);
        int (*bprm_check_security) (struct linux_binprm *bprm);
        int (*bprm_secureexec) (struct linux_binprm *bprm);
        void (*bprm_committing_creds) (struct linux_binprm *bprm);
        void (*bprm_committed_creds) (struct linux_binprm *bprm);

        int (*sb_alloc_security) (struct super_block *sb);
        void (*sb_free_security) (struct super_block *sb);
        int (*sb_copy_data) (char *orig, char *copy);
```

然后在 security.c 下***的函数

```
int security_inode_create(struct inode *dir, struct dentry *dentry, umode_t mode)
{
        if (unlikely(IS_PRIVATE(dir)))
                return 0;
        return security_ops->inode_create(dir, dentry, mode);
}
```

它其实可能会被 VFS_create 函数调用等（在创建文件时），最后找到相应的钩子点。

下载源码后进行解压。

加入我们的安全审计模块，在/linux-3.17.4/security 目录下新建自己的目录来保存自己的安全模块，命名的是 file，在新建的目录下新建的 file.c 文件，用来打印该文件的进程 ID，父进程的 ID，还有审计一个进程中创建目录的数目是否超过 10000.

```c
#include <linux/init.h>
#include <linux/kernel.h>
#include <linux/security.h>
#include <linux/dcache.h>
#include <linux/fs.h>
#include <linux/types.h>
#include <linux/sched.h>
static int pidnum[32769] = {0};
static int filetxt_inode_create(struct inode *dir, struct dentry *dentry, umode_t mode)
{
        int pid = current->pid;
        int parent_pid = current->real_parent->pid;
        printk("current dentry name %s,pid is %d\n",dentry->d_name.name,pid);
        printk("current parent pid is %d\n",parent_pid);
        if(pidnum[pid] > 10000){
                printk("create file beyond 10000\n");
                return -1;
        }
        else{
                (pidnum[pid])++;
        }
        return 0;
}
static struct security_operations filetxt_security_ops = {
        .name                   = "filetxt",
        .inode_create           = filetxt_inode_create,
};
static __init int file_init(void)
{
        if (register_security(&filetxt_security_ops)){
                panic("Failure registering FILETXT Linux");
        }
        printk(KERN_ALERT "FILETXT Linux initialized\n");
        return 0;
}
security_initcall(file_init);
```

在该目录下创建配置文件 KCONFIG

```
config SECURITY_FILETXT
        bool "FIFETXT LSM Protection"
        default n
        help
          This module is create by user,it can not do anything.
```

还有 Makefile 文件

```
obj-$(CONFIG_SECURITY_FILETXT) += file.o
```

在返回到上级目录 security 下对刚创建的 file 目录的安全模块进行登记。

Kconfig 下

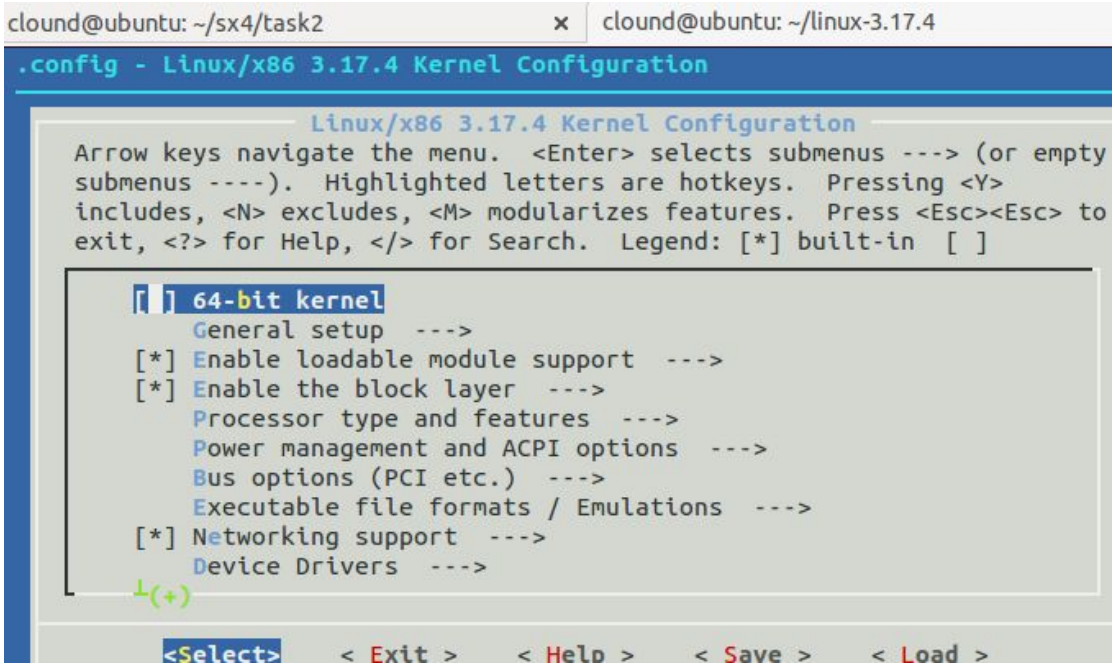
```
source security/smack/Kconfig
source security/tomoyo/Kconfig
source security/apparmor/Kconfig
source security/yama/Kconfig
source security/integrity/Kconfig
source security/file/Kconfig
source security/dmcheck/Kconfig
choice
```

Makefile 下

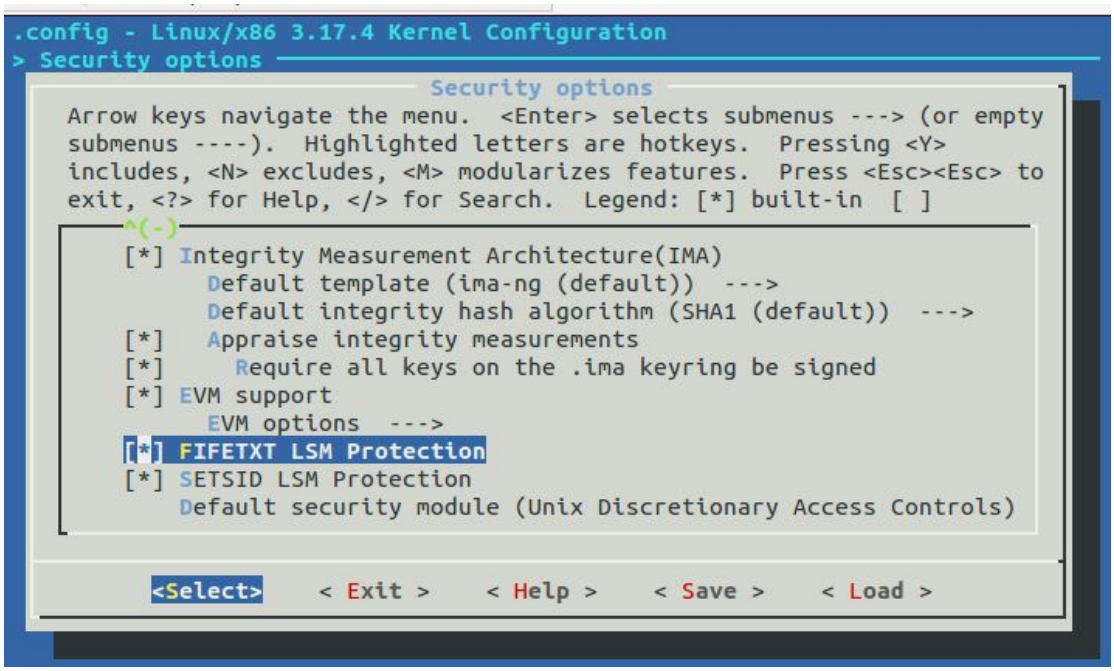
```
obj-$(CONFIG_KEYS)                      += keys/
subdir-$(CONFIG_SECURITY_SELINUX)       += selinux
subdir-$(CONFIG_SECURITY_SMACK)         += smack
subdir-$(CONFIG_SECURITY_TOMOYO)        += tomoyo
subdir-$(CONFIG_SECURITY_APPARMOR)      += apparmor
subdir-$(CONFIG_SECURITY_YAMA)          += yama
subdir-$(CONFIG_SECURITY_FILETXT)       += file
subdir-$(CONFIG_SECURITY_SETSID)        += dmcheck
# 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-$(CONFIG_CGROUP_DEVICE)             += device_cgroup.o
obj-$(CONFIG_SECURITY_FILETXT)          += file/
obj-$(CONFIG_SECURITY_SETSID)           += dmcheck/
# Object integrity file lists
subdir-$(CONFIG_INTEGRITY)              += integrity
obj-$(CONFIG_INTEGRITY)                 += integrity/
```

然后进行编译，我编译的路径是根据源码的 README，先 make mrproper，然后 make O=/home/clound/build/linux.3.17.4 menuconfig 里面，出现错误，说是需要 nurses libraries，再安装 sudo apt-get install libncurses5-dev 后出现图形界面可供选择模块编译。



然后选择我们的加载安全模块



然后进行编译 make O=/home/clound/build/linux.3.17.4

更 make 完，再进行模块 install

Sudo make O=/home/clound/build/linux.3.17.4 modules_install install

结束后 sudo update-grub

重新启动测试该模块是否能执行。

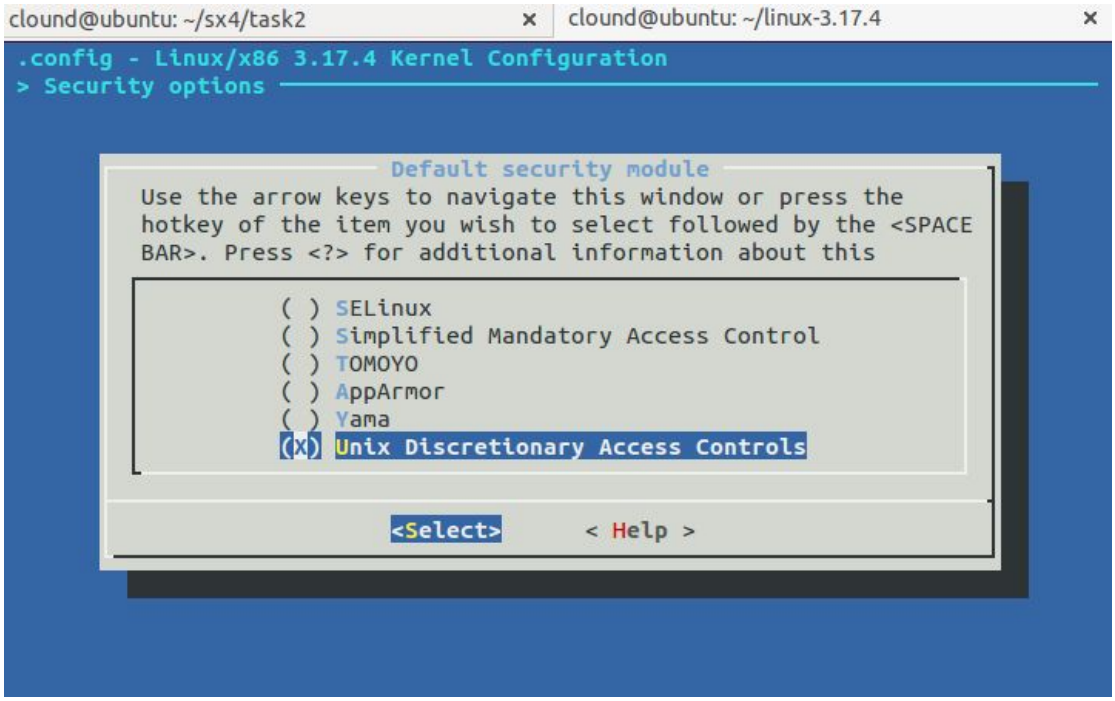
```
clound@ubuntu:~$ sudo dmesg -C
[sudo] password for clound:
clound@ubuntu:~$ rm -rf 1
^[[5~clound@ubuntu:~$ ls
build       Downloads         kernel.txt           Music     sx          txt
Desktop     examples.desktop  linux-3.17.4         Pictures  sx4         Videos
Documents   hello.c           linux-3.17.4.tar.xz  Public    Templates
clound@ubuntu:~$ mkdir 1
clound@ubuntu:~$ dmesg | grep current
[  221.169792] current dentry name LCK..ttyS0,pid is 2881
[  221.169799] current parent pid is 1862
clound@ubuntu:~$
```

```
root@ubuntu:/home/clound/1# rm -f xa*
root@ubuntu:/home/clound/1# dd if=/dev/zero of=masterfile bs=1 count=150000
150000+0 records in
150000+0 records out
150000 bytes (150 kB) copied, 0.693203 s, 216 kB/s
root@ubuntu:/home/clound/1# split -b 10 -a 10 masterfile
split: xaaaaaaaour: Operation not permitted
root@ubuntu:/home/clound/1# dmesg|grep creat
[ 1034.939111] create file beyond 10000
```

测试完成。

在实训中

在编译的过程中出现启动不进去，因为多个安全模块冲突



选取自由选择的进入控制模式，即可注册该编写的模块了。