空天信息安全与可信计算
教育部重点实验室

Linux分析与安全设计






第6章　Selinux及其实现机制分析






Selinux及其实现机制分析

- Selinux基本原理
- LSM
- Selinux实现机制分析

SELinux（ Security-Enhanced Linux）

- SELinux是一种基于 域-类型模型（Domain-Type）的强制访问控制（MAC）安全系统
- 它由NSA(The National Security Agency)编写并设计成内核模块包含到内核中
- 目前已经广泛应用到高安全操作系统中（四级及以上）

传统Linux的不足

- 对于文件的访问权的划分不够细
- ＤＡＣ(Discretionary Access Control)问题
- 存在特权用户root
- SUID程序的权限升级

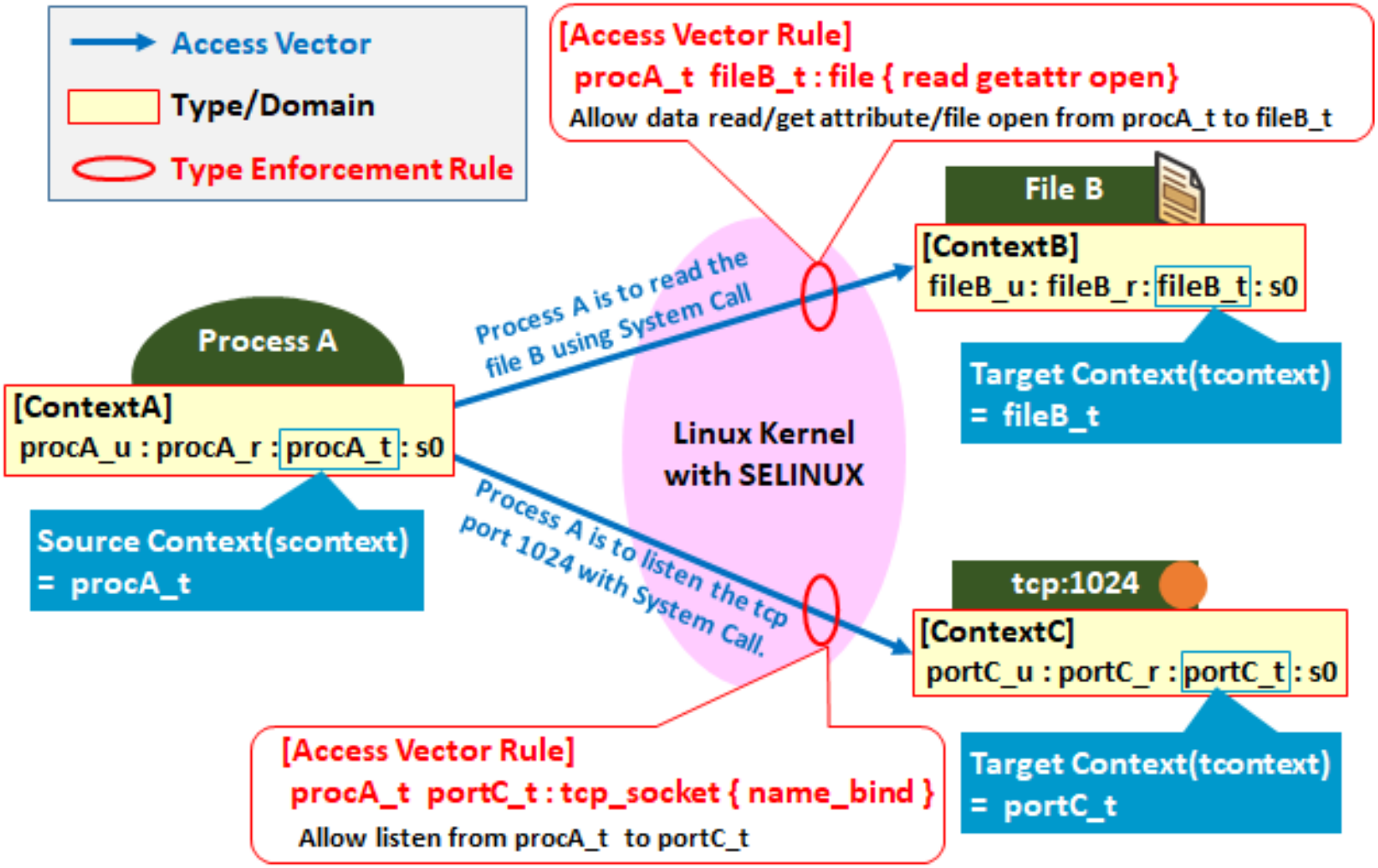
Selinux

- **MAC**

  系统的访问控制基于安全标签

  - ✓ **TE（Type Enforment)**
    - 对所有的文件和进程都赋予一个叫type的文件类型标签，对进程赋予的type通常称为Domain
    - 系统的访问控制主要基于type标签实施
    - 策略由系统管理员基于标签定义

Selinux



Selinux的优点

1）对访问的控制彻底化

对于所有的文件，目录，端口这类的资源的访问，都可以是基于策略设定的，这些策略是由管理员定制的、一般用户是没有权限更改的。

2）对于进程只赋予最小的权限

3）防止权限升级

4）对于用户只赋予与最小的权限

对于用户来说，被划分成一些ROLE，即使是ROOT用户，你要是不在sysadm_r里，也还是不能实行sysadm_t管理操作的。

Selinux安全标签

```
bash-3.2# ps -efZ
LABEL                                UID        PID  PPID  C STIME TTY          TIME CMD
system_u:system_r:init_t:s0-s15:c0.c255 secadm 1    0  0 08:17 ?        00:00:00 init [5]
system_u:system_r:kernel_t:s15:c0.c255 secadm 2     1  0 08:17 ?        00:00:00 [migration/0]
system_u:system_r:kernel_t:s15:c0.c255 secadm 3     1  0 08:17 ?        00:00:00 [ksoftirqd/0]
system_u:system_r:kernel_t:s15:c0.c255 secadm 4     1  0 08:17 ?        00:00:00 [watchdog/0]
system_u:system_r:kernel_t:s15:c0.c255 secadm 5     1  0 08:17 ?        00:00:00 [migration/1]
system_u:system_r:kernel_t:s15:c0.c255 secadm 6     1  0 08:17 ?        00:00:00 [ksoftirqd/1]
system_u:system_r:kernel_t:s15:c0.c255 secadm 7     1  0 08:17 ?        00:00:00 [watchdog/1]
system_u:system_r:kernel_t:s15:c0.c255 secadm 8     1  0 08:17 ?        00:00:00 [events/0]
system_u:system_r:kernel_t:s15:c0.c255 secadm 9     1  0 08:17 ?        00:00:00 [events/1]
system_u:system_r:kernel_t:s15:c0.c255 secadm 10    1  0 08:17 ?        00:00:00 [khelper]
system_u:system_r:kernel_t:s15:c0.c255 secadm 45    1  0 08:17 ?        00:00:00 [kthread]
system_u:system_r:kernel_t:s15:c0.c255 secadm 50   45  0 08:17 ?        00:00:00 [kblockd/0]
system_u:system_r:kernel_t:s15:c0.c255 secadm 51   45  0 08:17 ?        00:00:00 [kblockd/1]
system_u:system_r:kernel_t:s15:c0.c255 secadm 52   45  0 08:17 ?        00:00:00 [kacpid]
```

- SELinux对系统中的许多命令做了修改，我们通过添加一个-Z 选项显示客体和主体的安全上下文，

  Ls -Z 显示文件系统客体的安全上下文

  Ps -Z 显示进程的安全上下文

  Id -Z 它显示的当前用户的shell的安全上下文

Selinux安全标签

- 在SELinux当中，所有操作系统访问控制都是以关联的客体和主体的某种访问控制属性为基础。
- 访问控制属性叫做安全上下文

所有的客体以及主体都有与其有关联的安全上下文

- 一个安全上下文由三部分组成：用户、角色 和类型标识符

```
system_u:system_r:init_t:s0-s15:c0.c255 secadm 1   0  0 08:17 ?        00:00:00 init [5]
```

Selinux安全策略

- 因为SELinux默认不允许任何访问，所有的访问都必须明确授权，不管用户／组ID是什么，在SELinux中，通过allow 语句对主体授权对客体的访问权限
  - **Allow** 规则由四部分组成

    源类型(Source type(s)) 通常是尝试访问进程的域类型

    目标类型(Target type(s)) 被进程访问的客体的类型

    客体类别(Object class(es)) 指定允许访问的客体的类型，如file,dir,socket等

    许可(Pemission(s)) 象征目标类型允许源类型访问客体类型的访问种类

Selinux安全策略

例如:

allow user_t bin_t:file {read execute getattr}

整条规则解释如下:

拥有域类型user_t的进程可以读 / 执行或获取具体有bin_t类型的文件客体的属性。

Selinux角色

- SELinux通过SC中的role实现了“基于角色的访问控制”(RBAC－Role Based Access Control)。在SELinux中，并不直接建立用户和type之间的联系，而是通过角色作为桥梁。
- 用户声明语句和角色声明语句，比如：

user joe roles {user_r};

role user_r types {user_t passwd_t);

安全策略开启，关闭

在安全操作系统上，SELinux其状态有三种:

Enforcing

Permissive

Disabled

查看SELinux状态可用sestatus 以及getenforce命令

如何开启，关闭SELinux，也即在三种状态之间切换？
使用secadm用户登录安全操作系统
#setenforce 0 ----将SELinux状态调为permissive
#setenforce 1 ----将SELinux状态调为Enforcing

如要将SELinux策略置为disabled，则需要修改SELinux配置文件，然后重启系统

SELinux配置文件

SELinux配置文件有:
/boot/grug/grub.conf
/etc/selinux/config

password --md5 1aTSOBfOa$akTgOrnat5HQgg7nYXlkc0
title NeoKylin Linux Security OS (2.6.18-164.el5)
root (hd0,0)
kernel /vmlinuz-2.6.18-164.el5 ro root=LABEL=/1 rhgb quiet selinux=1 enforcing=0
initrd /initrd-2.6.18-164.el5.img
其中selinux = 1 表示开机启动SELinux，enforcing = 0 表示SELinux的状态为permissive

SELinux配置文件

```
bash-3.2# more /etc/selinux/config
# This file controls the state of SELinux on the system.
# SELINUX= can take one of these three values:
#       enforcing - SELinux security policy is enforced.
#       permissive - SELinux prints warnings instead of enforcing.
#       disabled - SELinux is fully disabled.
SELINUX=enforcing
# SELINUXTYPE= type of policy in use. Possible values are:
#       targeted - Only targeted network daemons are protected.
#       strict - Full SELinux protection.
SELINUXTYPE=refpolicy
```

系统对两个文件的读取方式:

当grub.conf中有相SELinux相关选项时，则读取grub.conf中的配置信息，而不读取/etc/selinux/config中的信息，否则读取/etc/selinux/config中的信息

SELINUX =enforcing　　表示SELinux状态为enforcing

SELINUXTYPE=refpolicy　表示策略类型为 refpolicy

SELinux常用命令

getenforce	查看当前系统SELinux状态
setenforce	设置当前系统SELinux状态
seinfo	查看当前系统SELinux信息
sestatus	查看当前系统SELinux详细状态
semoudle	对SELinux模块进行管理
chcon	更改文件、目录、设备的安全上下文
restorecon	恢复文件、目录、设备的安全上下文
sesearch	查看系统详细策略信息

SELinux常用命令

#apt-get install httpd
#ls –ldZ /usr/sbin/httpd /var/www/html

```
[root@livecd ~]# ls -dZ /usr/sbin/httpd  /var/www/html
-rwxr-xr-x. root root system_u:object_r:httpd_exec_t:s0 /usr/sbin/httpd
drwxr-xr-x. root root system_u:object_r:httpd_sys_content_t:s0 /var/www/html
[root@livecd ~]#
```

SELinux常用命令

#cp /etc/hosts /root
#ls –ldZ /etc/hosts /root/hosts /root

```
[root@livecd ~]# ls -dZ /usr/sbin/httpd  /var/www/html
-rwxr-xr-x. root root system_u:object_r:httpd_exec_t:s0 /usr/sbin/httpd
drwxr-xr-x. root root system_u:object_r:httpd_sys_content_t:s0 /var/www/html
[root@livecd ~]# cp /etc/hosts  /root
[root@livecd ~]# ls -ldZ /etc/hosts /root/hosts /root
-rw-r--r--. root root system_u:object_r:net_conf_t:s0  /etc/hosts
dr-xr-x---. root root system_u:object_r:admin_home_t:s0 /root
-rw-r--r--. root root unconfined_u:object_r:admin_home_t:s0 /root/hosts
```

SELinux常用命令

#chcon　[-R] [-t type] [-u user] [-r role] 文件
#chron –t net_conf_t /tmp/hosts

```
[root@livecd ~]# chcon -t net_conf_t  /root/hosts
[root@livecd ~]# ls -lZ /root/hosts
```

SELinux常用命令

#restorecon [-Rv] 文件或目录

```
[root@livecd ~]# restorecon /root/hosts
[root@livecd ~]# ls- ldZ /root/hosts
-bash: ls-: command not found
[root@livecd ~]# ls -ldZ /root/hosts
-rw-r--r--. root root unconfined_u:object_r:admin_home_t:s0 /root/hosts
[root@livecd ~]#
```

SELinux常用命令

#semanage {login|user|port|interface|fcontext|translation} –l
#semanage fcontext -{a|d|m} –t type files

#semanage fcontext -l
#semanage fcontext -a –t public_content_t “/root/test”

#yum install policycoreutils-python

SELinux常用命令

#seinfo
#yum install setools-console

#seinfo –b |grep httpd

SELinux常用命令

#sesearch
#sesearch –all –t httpd_sys_content_t

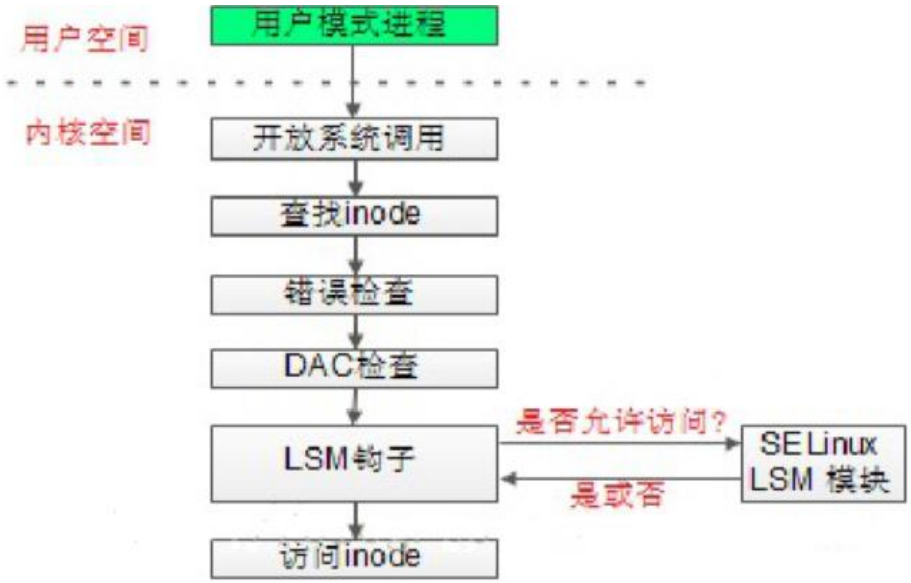
#getsebool –a
#setsebool –P httpd_enable_homedirs=1

SELinux日志

```
# cat /var/log/messages
```

LSM

- LSM（Linux security module)- LSM框架的思想是允许安全模块以插件形式进入内核，以便更严格地控制Linux默认的基于身份的自主访问控制（DAC）安全性。LSM在内核系统调用逻辑中提供了一套钩子（hooks），这些钩子通常放在标准Linux访问检查后、且内核调用访问真实资源之前.



LSM

LSM的优势

- 在关键的特定内核数据结构中加入了安全域；
- 在内核源码中不同的关键点处插入对安全钩子函数的调用；
- 提供了一个通用的安全系统调用；
- 提供了注册和注销函数，使得访问控制策略可以以内核模块方式实现；

LSM Hook函数

- Hook 函数（钩子函数）
  - 钩子函数也叫回调函数，是通过函数指针来实现的。
- LSM Hook函数包括：
  - Task hook
  - Program loading hook
  - IPC hook
  - File system hook
  - Network hook

http://www.cs.unibo.it/~renzo/doc/papers/LinuxSecurityModules.pdf

内核关键数据结构

- Task_struct(进程结构)
- Linux_binprm(二进制代码装入)
- Super_block(超级块)
- Inode(索引节点)
- File(文件)
- Dentry(目录项)
- SK_buff(套接字缓冲区)
- Net_device（网络设备）
- Kern_ipc_perm(ipc消息队列)

Task Hooks

- struct task_security_struct {
  - struct task_struct *task; /* 指向原task对象 */
  - u32 osid; /* 最后一次execve前的SID*/
  - u32 sid; /* 当前进程的SID */
  - u32 exec_sid; /* exec SID */
  - u32 create_sid; /* fscreate SID */
  - u32 keycreate_sid; /* keycreate SID */
  - u32 sockcreate_sid; /* 套接字 SID */
  - u32 ptrace_sid; /* SID of ptrace */
  - };
- int (*task_create) (unsigned long clone_flags);
  - 该函数在调用fork时调用。它调用static int task_has_perm(struct task_struct *tsk1,struct task_struct *tsk2,u32 perms) 进行权限检查。
- int (*task_alloc_security) (struct task_struct * p);
- void (*task_free_security) (struct task_struct * p);

Program Loading Hooks

- struct bprm_security_struct {
  - struct linux_binprm *bprm; /*指向bprm对象*/
  - u32 sid; /* 二进制程序SID */
  - unsigned char set;
  - char unsafe;//为bprm_post_apply_creds()提供共享失败信息，该信息来自于bprm_apply_creds()
  - };
- 下面是主要hook函数的说明：
  - int (*bprm_alloc_security) (struct linux_binprm * bprm);//分配安全域
  - void (*bprm_free_security) (struct linux_binprm * bprm);//释放安全域

Super Block Hooks

- struct superblock_security_struct {
  - struct super_block *sb; /* 指向所属super_block结构*/
  - struct list_head list; /*superblock_security_struct链表 */
  - u32 sid; /* SID of file system superblock*/
  - u32 def_sid; /* default SID for labeling */
  - u32 mntpoint_sid; /*文件系统挂载点的安全上下文 */
  - unsigned int behavior; /*打标签操作*/
  - unsigned char initialized; /*初始化标志*/
  - unsigned char proc; /*proc文件系统*/
  - struct mutex lock;//信号锁
  - struct list_head isec_head;//inode安全域链表
  - spinlock_t isec_lock;//自旋锁
  - };
- 介绍完了安全结构，我们开始介绍Super Block Hooks：
  - int (*sb_alloc_security) (struct super_block * sb);
  - void (*sb_free_security) (struct super_block * sb);

Inode Hooks

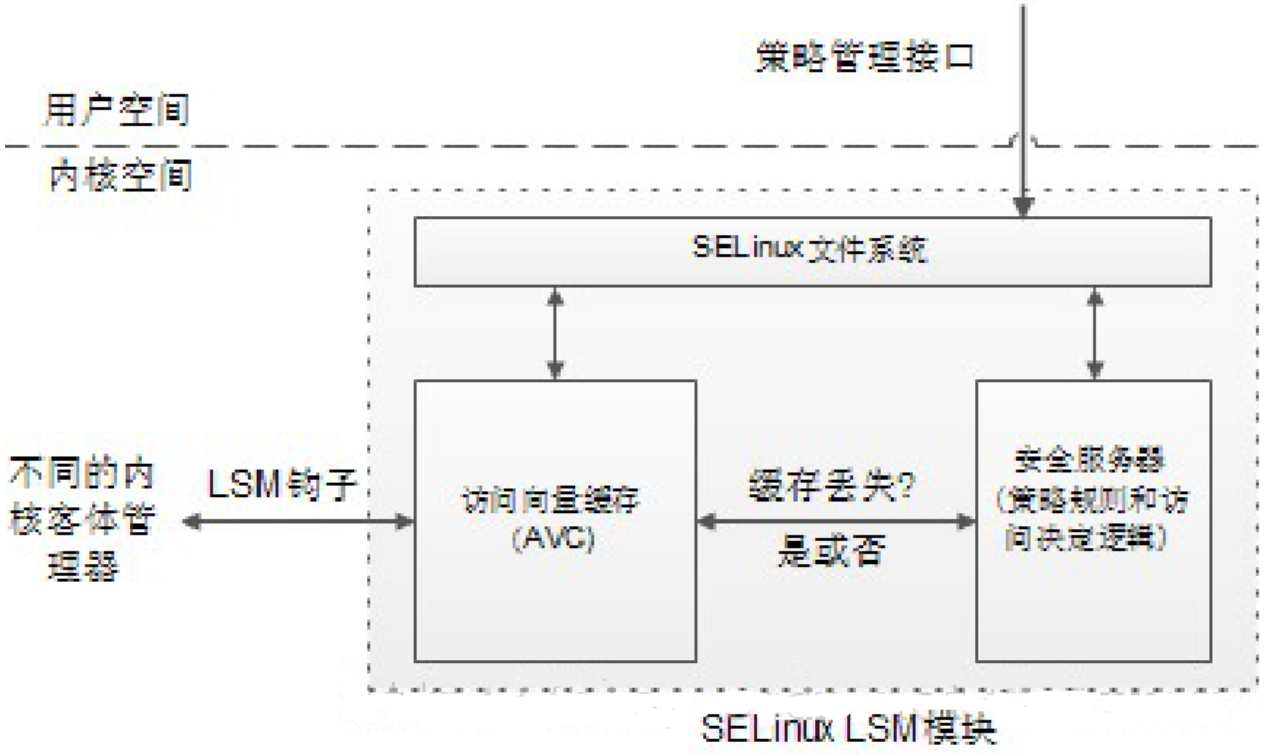
- struct inode_security_struct {
  - struct inode *inode; /* 所属的主体 */
  - struct list_head list; /*inode_security_struct的链表 */
  - u32 task_sid; /* 创建该节点的进程*/
  - u32 sid; /* 所属主体的sid */
  - u16 sclass; /*该主体的安全类*/
  - unsigned char initialized; /* 初始化标识 */
  - struct mutex lock;
  - unsigned char inherit; /*从父目录项继承来的sid*/
  - };
- int (*inode_alloc_security) (struct inode *inode);
- void (*inode_free_security) (struct inode *inode);

File Hooks

- struct file_security_struct {
  - struct file *file; /* 指向所属主体 */
  - u32 sid; /* 打开文件的描述符sid */
  - u32 fown_sid; /* 文件所有者的sid */
  - };
- int (*file_permission) (struct file * file, int mask);
- int (*file_alloc_security) (struct file * file);
- void (*file_free_security) (struct file * file);

Selinux实现

- SELinux LSM模块架构



安全模块LSM的注册

在src/init/main.c的start_kernel（）里：

- fork_init(num_physpages);
- proc_caches_init();
- buffer_init();
- unnamed_dev_init();
- key_init();
- security_init();
- vfs_caches_init(num_physpages);
- radix_tree_init();
- signals_init();

安全模块LSM的注册

security_init()的具体实现/src/security/security.c中：

```
/**
 * security_init - initializes the security framework
*/
int __init security_init(void)
{
    printk(KERN_INFO "Security Framework initialized\n");

    security_fixup_ops(&default_security_ops);
    security_ops = &default_security_ops;
    do_security_initcalls();

    return 0;
}
```

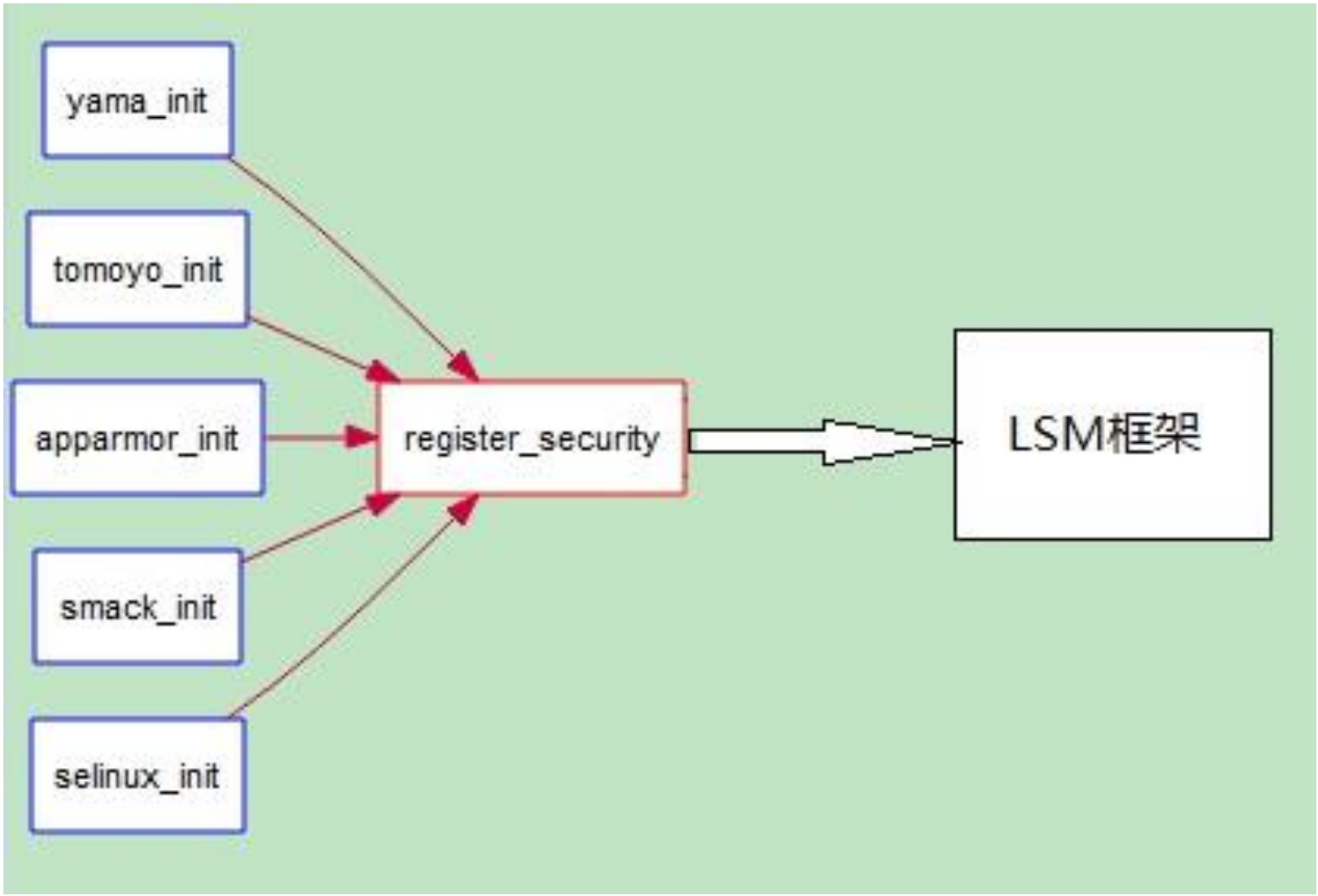
安全模块LSM的注册

```
static void __init do_security_initcalls(void)
{
        initcall_t *call;
        call = __security_initcall_start;
        while (call < __security_initcall_end) {
                (*call) ();
                call++;
        }
}
```

安全模块LSM的注册

- security_initcall(selinux_init);//security/selinux/hooks.c **SELinux 安全策略**
- security_initcall(apparmor_init);//security/apparmor/lsm.c **Apparmor 安全策略**
- security_initcall(smack_init);//security/smack/smack_lsm.c **Smack 安全策略**
- security_initcall(tomoyo_init);//security/tomoyo/tomoyo.c **TOMOYO Linux 安全策略**
- security_initcall(yama_init);//security/yama/yama_lsm.c **YAMA Linux 安全策略**

安全模块LSM的注册



Selinux的初始化

Selinux初始化代码在security/selinux/hooks.c中

```
static __init int selinux_init(void)
{
    //判断SELinux是否为配置的默认安全模块
    if (!security_module_enable(&selinux_ops)) {
        selinux_enabled = 0;
        return 0;
    }
    //如果没有配置SELinux为默认安全模块，则退出
    if (!selinux_enabled) {
        printk(KERN_INFO "SELinux:  Disabled at
boot.\n");
        return 0;
    }
```

Selinux的初始化

```
printk(KERN_INFO "SELinux:  Initializing.\n");

// 设置当前进程的安全状态
cred_init_security();

default_noexec = !(VM_DATA_DEFAULT_FLAGS & VM_EXEC);

// 给inode安全属性对象sel_inode_cache分配空间
sel_inode_cache = kmem_cache_create("selinux_inode_security",sizeof(struct inode_security_struct),     0, SLAB_PANIC, NULL);
//初始化访问向量缓存（AVC）
```

Selinux的初始化

```
avc_init();
        //将SELinux注册到LSM中
    if (register_security(&selinux_ops))
        panic("SELinux: Unable to register with kernel.\n");

       // 显示SE Linux的强制模式
    if (selinux_enforcing)
        printk(KERN_DEBUG "SELinux:  Starting in enforcing mode\n");
    else
        printk(KERN_DEBUG "SELinux:  Starting in permissive mode\n");
     return 0;
}
```

Sid

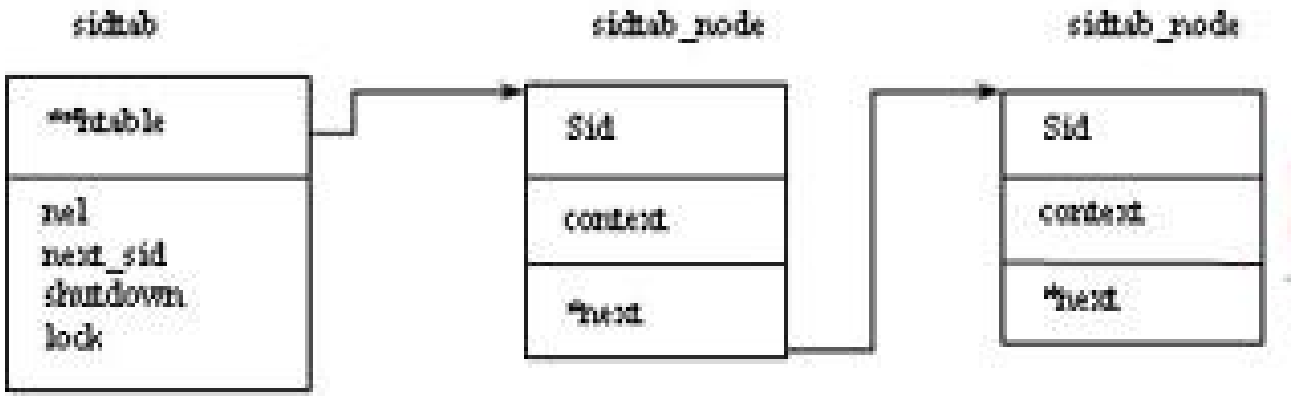
- 安全标识符（security identifier，简称SID）是内核中激活的安全上下文的序号。本质上类似SContext的key值。。
- Sid是在selinux初始化和运行过程中动态分配的32位无符号整数，具体代码主要在/security/selinux/ss/sidtab.c目录下

Sid

- 结构context是安全上下文的描述结构，安全上下文包括用户、角色、类型和安全级别，结构context列出如下(在selinux/ss/context.h中)：

```
struct context {
u32 user; //用户
u32 role; //角色
u32 type; //类型
struct mls_range range; //级别
};
```

Sid



AV

- AV
  - 访问向量（Access vectors，Avs），用来表示策略的规则。如允许域访问各种系统客体，一个AV是一套许可。一个基本的AV规则是主体和客体的类型对，AV规则的语法如下：
- <av_kind> <source_type(s)> <target_type(s)>:<class(es)> <permission(s)>

源代码在：security/selinux/ss/avtab.c和security/selinux/avc.c

AV

<av_kind>在以下四种设置：

- allow　　表示允许主体对客体执行允许的操作。
- neverallow　　表示不允许主体对客体执行指定的操作。
- auditallow　　表示允许操作并记录访问决策信息。
- dontaudit　　表示不记录违反规则的决策信息，且违反规则不影响运行。

AV

- AV的例子：
  - allow init_t apache_exec_t : file execute;
  - allow userdomain shell_exec_t: file{ read getattr lock execute ioctl};

AVC

- AVC提供了从安全服务器获得的访问策略的缓冲区（cache），提高了安全机制的运行性能。它提供了hook函数高效检查授权的接口，提供了安全服务器管理cache的接口。
  - 与hook函数的接口定义在include/avc.h中，与服务器的接口定义在include/avc_ss.h中，AVC代码在avc.c中。

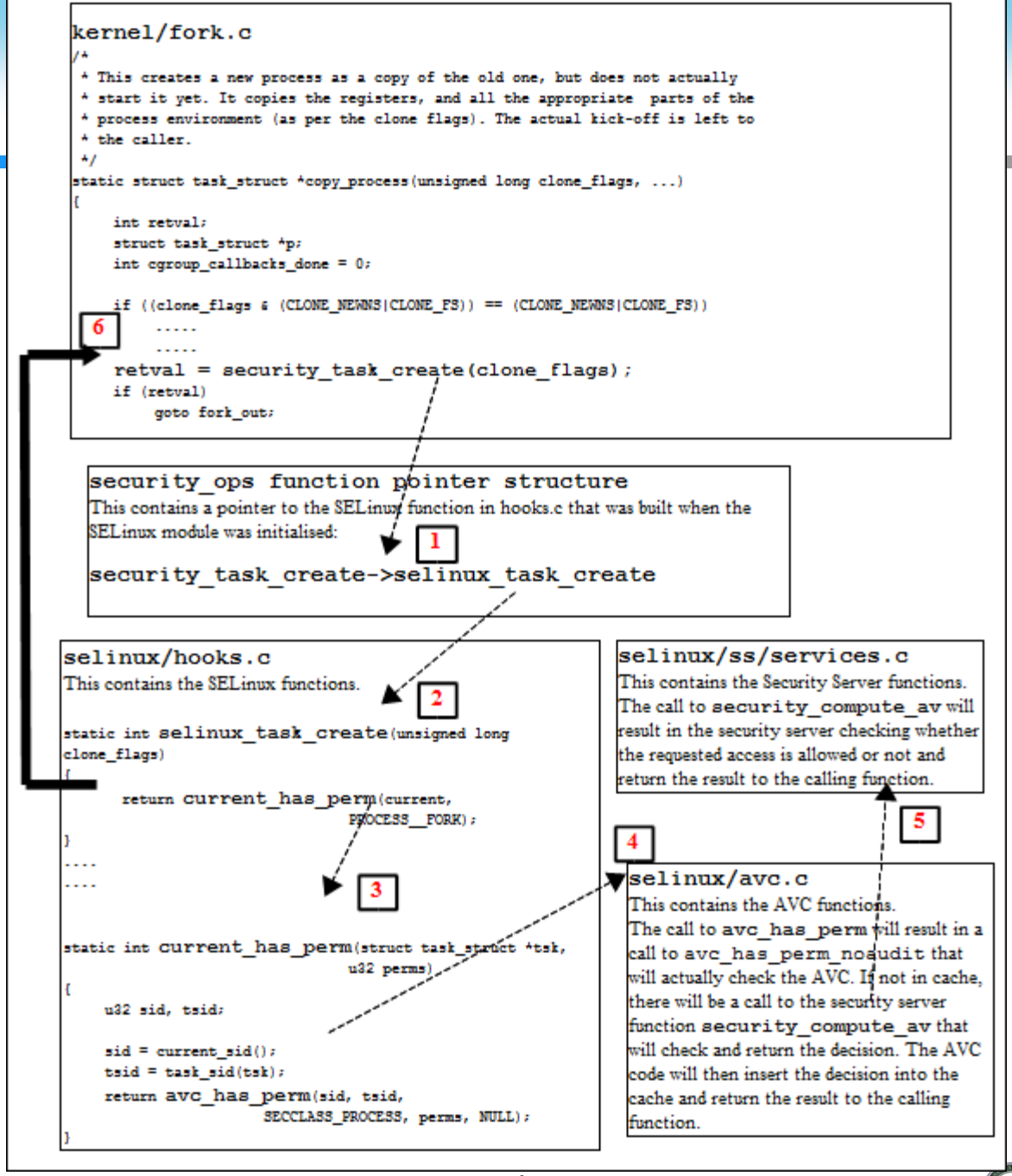
AVC

- Avc接口函数主要是被Hooks函数调用，进行权限检查。例如一个主要的Avc接口函数：

```
- static int task_has_perm(struct task_struct *tsk1, struct
  task_struct *tsk2, u32 perms)
- {
-   struct task_security_struct *tsec1, *tsec2;
-   tsec1 = tsk1->security;
-   tsec2 = tsk2->security;
-   return avc_has_perm(tsec1->sid, tsec2->sid,
-                       SECCLASS_PROCESS, perms, NULL);
- }
```

AVC

- int avc_has_perm(u32 ssid, u32 tsid, u16 tclass, u32 requested, struct avc_audit_data *auditdata)
  - {
  - struct av_decision avd;
  - int rc;
  - rc = avc_has_perm_noaudit(ssid, tsid, tclass, requested, 0, &avd);
  - avc_audit(ssid, tsid, tclass, requested, &avd, rc, auditdata);
  - return rc;
  - }



其它强制访问控制工具

AppArmor

- -使用文件路径作为安全标签
- -相比于 SELinux 它提供更多的特性，更容易使用
- -包括一个学习模式，可以让系统“学习”一个特定应用的行为，以及通过配置文件设置限制实现安全的应用使用。
- -Ubuntu默认支持AppArmor
- -安全性不如Selinux

AppArmor

- AppArmor Policy rule example:
- /usr/sbin/httpd{

/var/www/** r,

}

- /usr/sbin/httpd can read the files under /var/www

其它强制访问控制工具

SVirt

基于SELinux，但不同之处是基于KVM的虚拟化强制访问工具，主要的用途是给虚拟机打上标签，防止虚拟机越权访问宿主机和其它虚拟机。

Grsecurity

不止是SELinux，包含了一套保护linux内核的工具集,并具有策略学习机制。

Selinux或AppArmor实现分析报告

- 安装LINUX源代码，结合Selinux原理，分析Selinux或AppArmor源代码，撰写实验报告（一些关键部分，可以结合截图说明）

Selinux实现原理分析报告

- 问题：
  - Selinux_ops与具体Hook函数的关系？
  - 如何将自己编写的Hook函数注册到LINUX内核中？
  - SID是如何创建的？
  - 通过LSM Hook可以对网络通信进行更细粒度的安全检查吗？
  - Selinux对防御缓冲区溢出攻击及其危害有用吗？