

LSM钩子函数

来自Kylin security

目录

- 1 LSM钩子函数
 - 1.1 钩子说明
 - 1.2 钩子分类
 - 1.2.1 任务钩子
 - 1.2.2 程序装载钩子
 - 1.2.3 进程间通信IPC钩子
 - 1.2.4 文件系统钩子
 - 1.2.5 网络钩子
 - 1.2.6 其他的钩子
 - 1.3 钩子分析
 - 1.3.1 inode类钩子
 - 1.3.2 path类钩子
 - 1.3.3 file类钩子
 - 1.3.4 super_block类钩子
 - 1.3.5 bprm (linux_binprm) 类钩子
 - 1.3.6 task类钩子
 - 1.3.7 netlink类钩子
 - 1.3.8 unix domain类钩子
 - 1.3.9 socket类钩子
 - 1.3.10 XFRM类钩子
 - 1.3.11 key类钩子
 - 1.3.12 System V IPC类钩子
 - 1.3.13 System V IPC消息队列中持有的单个message类钩子
 - 1.3.14 System V IPC消息队列类钩子
 - 1.3.15 System V共享内存段类钩子
 - 1.3.16 System V信号量类钩子
 - 1.3.17 binder类钩子
 - 1.3.18 ptrace类钩子
 - 1.3.19 权能类钩子
 - 1.3.20 审计类钩子
 - 1.3.21 其他钩子
 - 1.4 常用的安全钩子说明
 - 1.4.1 文件操作钩子
 - 1.4.2 可执行程序控制钩子
 - 1.5 安全模块编写注意事项

LSM钩子函数

- 以下的LSM钩子函数及相关信息分析是基于linux-4.4.0版本的内核

钩子说明

- Linux安全模块（LSM）提供了两类对安全钩子函数的调用：一类管理内核对象的安全域，另一类仲裁对这些内核对象的访问。
- 为此给部分内核数据结构添加了安全域的概念（安全域中存放相关的安全信息），添加了安全域的内核数据结构如下：

```
* super_block: 代表文件系统 (include/linux/fs.h) --> void *s_security
* inode: 代表管道,文件,或者Socket套接字 (include/linux/fs.h) --> void *i_security
* file: 代表打开的文件 (include/linux/fs.h) --> void *f_security
* kern_ipc_perm: 代表Semaphore信号量,共享内存段,或者消息队列 (include/linux/ipc.h) --> void *security
* msg_msg: 代表单个的消息 (include/linux/msg.h) --> void *msg_security
* cred: 任务的安全上下文 (include/linux/cred.h) --> void *security
* sock: 网络层的socket表示 (include/net/sock.h) --> void *sk_security
* key: 密钥 (include/linux/key.h) --> void *key_security
```

钩子分类

任务钩子

- Linux安全模块(LSM)提供了一系列的任务钩子使得安全模块可以管理进程的安全信息并且控制进程的操作：

1. 模块可以使用task_struct->cred和task_struct->real_cred结构中的安全域来维护进程安全信息
2. 任务钩子提供了控制进程间通信的钩子,例如kill()
3. 提供了控制对当前进程进行特权操作的钩子,例如setuid()
4. 提供了对资源管理操作进行细粒度控制的钩子,例如setrlimit()和nice()

程序装载钩子

- Linux安全模块(LSM)提供了一系列程序装载钩子，用在一个execve()操作执行过程的关键点上，用于在一个新程序执行时改变其特权的能力：

1. linux_binprm->cred和linux_binprm->file结构中的安全域允许安全模块维护程序装载过程中的安全信息
2. 提供了钩子用于允许安全模块在装载程序前初始化安全信息和执行访问控制
3. 提供了钩子允许模块在新程序成功装载后更新任务的安全信息
4. 提供了钩子用来控制程序执行过程中的状态继承

进程间通信IPC钩子

- 安全模块可以使用进程间通信IPC钩子来对System V IPC的安全信息进行管理,以及执行访问控制:

1. IPC对象数据结构共享一个子结构kern_ipc_perm, 其中加入了一个安全域, 并且这个子结构中只有一个指针传给现存的ipcperms()函数进行权限检查
2. 为了支持单个消息的安全信息,Linux安全模块(LSM)还在msg_msg结构中加入了一个安全域
3. Linux安全模块(LSM)在现存的ipcperms()函数中插入了一个钩子, 使得安全模块可以对每个现存的Linux IPC权限执行检查
4. Linux安全模块(LSM)也在单个的IPC操作中插入了钩子
5. 还有钩子支持对通过System V消息队列发送的单个消息进行细粒度的访问控制

文件系统钩子

- 对于文件操作, 定义了三种钩子: 文件系统(super_block)钩子, inode结点钩子和文件(file)钩子, Linux安全模块(LSM)在对应的三个内核数据结构中均加入了安全域:

1. 超级块文件系统钩子使得安全模块能够控制对整个文件系统进行的操作, 例如挂载, 卸载, 还有statfs()
2. inode节点钩子提供了很多对inode结点进行细粒度访问控制的操作
3. 文件钩子中的一些允许安全模块对read()和write()这样的文件操作进行额外的检查
4. 文件钩子允许安全模块控制通过socket IPC接收打开文件描述符
5. 其他的文件钩子提供对像fcntl()和ioctl()这样的操作的细粒度访问控制

网络钩子

- 对网络的应用层访问使用一系列的socket套接字钩子来进行仲裁, 这些钩子基本覆盖了所有基于socket套接字的协议, 包括socket类钩子, XFRM和netlink类钩子:

1. socket套接字钩子对有关进程的网络访问提供了一个通用的仲裁, 从而显著扩展了内核的网络访问控制框架(这在网络层是已经由Linux内核防火墙netfilter进行处理的), 例如sock_rcv_skb钩子允许在进入内核的包排队到相应的用户空间socket套接字之前, 按照其目的应用来对其进行仲裁
2. LSM也为IPv4, UNIX域, 以及Netlink协议实现了细粒度的钩子
3. 网络数据以包的形式被封装在sk_buff结构(socket套接字缓冲区)中游历协议栈, LSM在sk_buff->sk结构中加入了一个安全域, 使得能够在包的层次上对通过网络层的数据的安全信息进行管理, 并提供了一系列的钩子用于维护这个安全域的整个生命周期

其他的钩子

- Linux安全模块(LSM)还提供了ptrace, binder, audit, 权能, syslog, settimer, quota, secctx与secid间的转换, d_instantiate (dentry类钩子) , proc属性获取和设置钩子, key。

钩子分析

- 钩子名是指结构体security_hook_list.hook中的具体的成员钩子名字, 是需要在安全模块中根据需求具体实现的功能挂载点
- security接口名是指对对应的钩子的调用的函数名字 (对安全钩子的封装), 内核各资源通过调用security接口就可以间接的调用到当前加载的安全模块中实现的对应钩子的功能
- 钩子中的alloc, free和init等相关的钩子函数功能是对安全域资源的管理, 而其他的钩子函数几乎都是起仲裁功能, 主要是对相关权限的判断
- 具体的钩子函数分析, 主要从钩子函数的作用和钩子函数被调用的点 (位置) 两方面进行详细的分析:

inode类钩子

- inode节点操作

| 钩子名 | security接口名 | 作用 | 调用点 |
|----------------------|---|---|---|
| inode_alloc_security | security_inode_alloc() | 分配和附加一个安全结构到inode->i_security上, 当inode结构被分配时i_security域会被初始化为NULL (该钩子是在创建新文件给文件创建分配inode节点时会被调用) | inode_init_always() (初始化inode结构体) ---> alloc_inode() |
| inode_free_security | security_inode_free() | 释放inode的安全结构, 同时将inode->i_security设置为NULL (该钩子是在删除文件销毁该文件的inode节点时会被调用) | _destroy_inode() ---> destroy_inode() (fs/inode.c) |
| inode_init_security | security_inode_init_security()和security_old_inode_init_security() | 获取安全属性value和name suffix (后缀) 去设置一个新的 inode 和它的安全域. 该钩子是在inode节点创建事务的fs代码和提供原子标记的inode节点中调用, 不像post_create/mkdir...等钩子是在VFS中被调用. 该钩子函数中期望通过kmalloc去分配name和value, 同时调用者在使用他们后就必须调用kfree去释放. 如果安全模块没有使用安全属性或者不希望给这些特定的inode节点设置安全属性时, 该钩子函数就应该返回 "-EOPNOTSUPP" 值去跳过该处理 | *_xattr_security_init()或*_security_init() (在各文件系统中对应的这两个函数中的一个) |
| inode_create | security_inode_create() | 在创建普通文件时检查权限 | 1) cachefiles_check_cache_dir() ---> cachefiles_determine_cache_security() (fs/cachefiles/security.c) 2) vfs_create() (fs/namei.c) 3) may_o_create() ---> atomic_open() ---> lookup_open() ---> do_last() ---> path_openat() ---> do_filp_open()/do_file_open_root() (fs/namei.c) |
| inode_link | security_inode_link() | 给文件创建一个新硬链接前检查权限 | vfs_link() ---> linkat() ---> link() (fs/namei.c, linkat()/link()是系统调用) |
| inode_unlink | security_inode_unlink() | 删除文件的一个硬链接时检查权限 | vfs_unlink() ---> do_unlinkat() ---> unlinkat()/unlink() (fs/namei.c, unlinkat()和unlink()是系统调用) |
| inode_symlink | security_inode_symlink() | 给一个文件创建软链接 (符号链接) 时检查权限 | vfs_symlink() ---> symlinkat() (fs/namei.c, symlinkat()是系统调用) |
| inode_mkdir | security_inode_mkdir() | 在与参数dir (inode节点结构体) 相关的已存在的目录中创建一个新目录时检查权限 | vfs_mkdir() ---> mkdirat() (fs/namei.c, mkdirat()是系统调用) |
| inode_rmdir | security_inode_rmdir() | 在删除一个目录时检查权限 | vfs_rmdir() ---> do_rmdir() ---> rmdir()/unlinkat() (fs/namei.c, rmdir()和unlinkat()是系统调用) |
| inode_mknod | security_inode_mknod() | 当创建一个特殊文件 (通过mknod系统调用创建的socket / fifo等文件) 时检查权限. 请注意, 当用mknod操作去创建一个普通文件时只会调用create钩子, 而不会调用该钩子 | vfs_mknod() ---> mknodat() (fs/namei.c, mknodat()是系统调用) |
| inode_rename | security_inode_rename() | 给一个文件或目录重命名时检查权限 | vfs_rename() ---> renameat2()/renameat() / rename() (fs/namei.c, rename*()都是系统调用) |
| inode_readlink | security_inode_readlink() | 在读软链接时检查权限 | readlinkat()/readlink() (fs/stat.c, readlinkat()和readlink()都是系统调用) |
| inode_follow_link | security_inode_follow_link() | 在遵循软链接查找路径名时检查权限 | get_link() ---> ... ---> kern_path() / vfs_path_lookup() / user_path_at_empty() / kern_path_create() / user_path_create() / user_path_mountpoint_at() / kern_path_mountpoint() / do_filp_open() / do_file_open_root() (fs/namei.c) |
| inode_permission | security_inode_permission() | 访问一个inode节点前检查权限. 该钩子能被已经存在的Linux权限功能调用, 因此一个安全模块可以用它为已经存在的Linux权限检查提供额外的检查. 注: 当一个文件被打开 (以及其他的操作) 的时候会去调用该钩子, 而当执行实际的读写操作时会去调用file_security_ops权限钩子 | _inode_permission() ---> inode_permission() ---> safe_hardlink_source() ---> may_linkat() ---> linkat() / link() (fs/namei.c, linkat()和link()都是系统调用) |
| inode_setattr | security_inode_setattr() | 在设置文件属性前检查权限. 注意每当文件属性发生改变, notify_change()内核调用是从几个点进行的 (如当一个文件被截断, chown/chmod操作, 将磁盘配额等) | 1) notify_change() (修改文件系统对象的属性, fs/attr.c) 2) fat_ioctl_set_attributes() ---> fat_generic_ioctl() (fat文件操作, fs/fat/file.c) |
| inode_getattr | security_inode_getattr() | 在获取文件属性前检查权限 | vfs_getattr() ---> vfs_fstat() / vfs_fstatat() ---> fstat() / newfstat() / fstat64() / vfs_stat() / vfs_lstat() / newfstatat() / fstatat64() (fs/stat.c, 其中newfstat/ fstat64/newfstatat/fstatat64都是系统调用) |
| inode_setxattr | security_inode_setxattr() | 在设置扩展属性前检查权限 | vfs_setxattr() ---> setxattr() ---> path_setxattr() / fsetxattr() ---> lsetxattr() / setxattr() (fs/xattr.c, 其中fsetxattr/lsetxattr/setxattr都是系统调用) |
| inode_post_setxattr | security_inode_post_setxattr() | 在setxattr操作成功后更新inode节点的安全域 | _vfs_setxattr_noperm() (执行setxattr操作时不进行权限检查) ---> vfs_setxattr() ---> ... (fs/xattr.c) |
| inode_getxattr | security_inode_getxattr() | 在获取扩展属性前检查权限 | vfs_getxattr() ---> getxattr() ---> path_getxattr() / fgetxattr() ---> getxattr() / lgetxattr() (fs/xattr.c, 其中fgetxattr/getxattr/lgetxattr都是系统调用) |
| inode_listxattr | security_inode_listxattr() | 在获取扩展属性列表前检查权限 | vfs_listxattr() ---> listxattr() ---> flistxattr() / path_listxattr() ---> listxattr() / llistxattr() (fs/xattr.c, flistxattr/listxattr/llistxattr都是系统调用) |
| inode_removexattr | security_inode_removexattr() | 在移除扩展属性前检查权限 | vfs_removexattr() ---> removexattr() ---> fremovexattr() / path_removexattr() ---> removexattr() / lremovexattr() (fs/xattr.c, 其中fremovexattr/removexattr/lremovexattr都是系统调用) |
| inode_need_killpriv | security_inode_need_killpriv() | 当一个inode节点被改变时会被调用 | 1) dentry_needs_remove_privs() (fs/inode.c) 2) notify_change() (fs/attr.c) |

| 钩子名 | security接口名 | 作用 | 调用点 |
|--------------------|-------------------------------|--|---|
| inode_killpriv | security_inode_killpriv() | setuid位将被移除，也移除类似的安全标签，并且在被调用时持有dentry->d_inode->i_mutex互斥锁 | notify_change() (fs/attr.c) |
| inode_getsecurity | security_inode_getsecurity() | 通过参数@buffer去检索关联名字@name的inode节点 @inode的安全标签的扩展属性副本。注意名字@name是指除去security前缀后剩下的属性名字。参数@alloc是用来指示函数调用是应该通过buffer返回一个值，还是仅仅返回长度值（成功时buffer的大小） | xattr_getsecurity() --> vfs_getxattr() (fs/xattr.c) |
| inode_setsecurity | security_inode_setsecurity() | 设置安全标签关联的名字为@name的inode节点@inode的扩展属性值为@value。参数@size表示@value的字节大小，参数@flags可能的值有XATTR_CREATE，XATTR_REPLACE和0。注意名字@name是指属性名字中除去security前缀后剩余的名字 | 1) __vfs_setxattr_noperm() --> vfs_setxattr() (fs/xattr.c) 2) kernfs_iop_setxattr() (fs/kernfs/inode.c) |
| inode_listsecurity | security_inode_listsecurity() | 拷贝inode节点@inode关联的安全标签的扩展属性名字到@buffer中。参数@buffer的大小最大值是被参数@buffer_size指定 | 1) vfs_listxattr() (fs/xattr.c) 2) fs/nfs/nfs4proc.c 3) net/socket.c |
| inode_getsecid | security_inode_getsecid() | 获取与节点关联的secid | 只在kernel/audit.c和security/integrity/ima/ima_policy.c中有调用 |
| inode_notifysecctx | security_inode_notifysecctx() | 通知安全模块一个inode节点的安全上下文应该是什么。为该inode节点初始化被安全模块管理的incore安全上下文 | 1) kernfs_refresh_inode() --> kernfs_iop_getattr() / kernfs_init_inode() / kernfs_iop_permission() --> kernfs_get_inode() (fs/kernfs/inode.c) 2) nfs_setsecurity() --> nfs_fhget() / __nfs_revalidate_inode() --> nfs_revalidate_inode() / __nfs_revalidate_mapping() / nfs_getattr() (fs/nfs/inode.c) |
| inode_setsecctx | security_inode_setsecctx() | 更改一个inode节点的安全上下文。更新被安全模块管理的incore安全上下文，同时在需要的时候调用文件系统代码（通过__vfs_setxattr_noperm）去更新任何支持的扩展属性(xattr)代表的上下文 | 1) nfs4_set_nfs4_label() (fs/nfsd/vfs.c) 2) nfs4_security_inode_setsecctx() --> do_open_lookup() / nfsd4_create() --> nfsd4_open() (fs/nfsd/nfs4proc.c) |
| inode_getsecctx | security_inode_getsecctx() | 获取给定inode节点@inode的安全上下文 | 1) kernfs_iop_setxattr() (fs/kernfs/inode.c) 2) nfsd4_encode_fattr() --> nfsd4_encode_fattr_to_buf().. (fs/nfsd/nfs4xdr.c) |

path类钩子

■ path操作

| 钩子名 | security接口名 | 作用 | 调用点 |
|---------------|--------------------------|--|--|
| path_unlink | security_path_unlink() | 移除一个文件的一个硬链接时检查权限 | 1) do_unlinkat() --> unlinkat() / unlink() (fs/namei.c, unlinkat/unlink是系统调用) 2) call_unlink() --> vfs_sub_unlink() (fs/aufs/vfsub.c) 3) cachefiles_bury_object() --> cachefiles_delete_object() / cachefiles_walk_to_object() / cachefiles_cull() (fs/cachefiles/namei.c) |
| path_mkdir | security_path_mkdir() | 在与参数@path (path结构体) 关联的已存在的目录中创建一个新目录时检查权限 | 1) mkdirat() --> mkdir() (fs/namei.c, mkdirat/mkdir都是系统调用) 2) vfs_sub_mkdir() (fs/aufs/vfsub.c) 3) fs/cachefiles/namei.c |
| path_rmdir | security_path_rmdir() | 删除一个目录时检查权限 | 1) do_rmdir() --> rmdir() / unlinkat() (fs/namei.c, rmdir/unlinkat是系统调用) 2) vfs_sub_rmdir() (fs/aufs/vfsub.c) |
| path_mknod | security_path_mknod() | 创建一个文件时检查权限。注意即使利用mknod去操作一个普通文件，该钩子仍会被调用 | 1) may_o_create() --> atomic_open() --> lookup_open() --> do_last() --> path_openat() --> do_fopen() / do_file_open_root() (fs/namei.c) 2) mknodat() --> mknod() (fs/namei.c, mknodat/mknod都是系统调用) 3) vfs_sub_create() / vfs_sub_mknod() (fs/aufs/vfsub.c) 4) fs/cachefiles/namei.c |
| path_truncate | security_path_truncate() | 在截取文件前检查权限 | 1) handle_truncate() --> do_last() -->... (fs/namei.c) 2) vfs_truncate() / do_sys_ftruncate() --> ftruncate() / ftruncate64() (fs/open.c, ftruncate / ftruncate64是系统调用) 3) vfs_sub_trunc() (fs/aufs/vfsub.c) |
| path_symlink | security_path_symlink() | 给一个文件创建一个软链接（符号链接）时检查权限 | 1) symlinkat() --> symlink() (fs/namei.c, symlinkat/symlink是系统调用) 2) vfs_sub_symlink() (fs/aufs/vfsub.c) |
| path_link | security_path_link() | 为文件创建一个新的硬链接前检查权限 | 1) linkat() --> link() (fs/namei.c, linkat/link都是系统调用) 2) vfs_sub_link() (fs/aufs/vfsub.c) |
| path_rename | security_path_rename() | 给文件或目录重命名时检查权限 | 1) renameat2() --> renameat() / rename() (fs/namei.c, renameat2/renameat / rename都是系统调用) 2) vfs_sub_rename() (fs/aufs/vfsub.c) 3) fs/cachefiles/namei.c |
| path_chmod | security_path_chmod() | 在更改文件或目录的DAC权限时检查权限 | 1) chmod_common() --> fchmod() / fchmodat() --> chmod() (fs/open.c, chmod/fchmod / fchmodat都是系统调用) 2) aufs setattr() (fs/aufs/i_op.c) |
| path_chown | security_path_chown() | 在更改文件或目录的owner/group时检查权限 | 1) chown_common() --> fchownat() / fchown() --> chown() / lchown() (fs/open.c, fchownat / fchown/chown/lchown是系统调用) 2) aufs setattr() (fs/aufs/i_op.c) |
| path_chroot | security_path_chroot() | 在更改根目录时检查权限 | chroot() (fs/open.c, chroot是系统调用) |

file类钩子

■ 打开的文件操作

| 钩子名 | security接口名 | 作用 | 调用点 |
|---------------------|--------------------------------|--|---|
| file_permission | security_file_permission() | 在访问一个打开的文件前检查权限。该钩子被各种操作调用，像读写文件。同样在这些操作上一个安全模块能使用该钩子去执行额外的检查，例如：用来验证权限，支持特权等级和策略的变化。注意该钩子是在执行真实的读／写操作时被使用，而钩子inode_security_ops则是在文件打开（也有一些其他操作）时被调用。警告：尽管该钩子能被用来验证文件读写等多个系统调用操作的权限，但它不会处理文件内存映射(memory-mapped)的权限验证。如果需要这种验证，则在安全模块中必须处理这种个别现象 | 1) vfs_fallocate() ---> falllocate() (fs/open.c, falllocate是系统调用) 2) iterate_dir() ---> old_readdir()/getdents()/getdents64() (fs/readdir.c, old_readdir/getdents/getdents64是系统调用) 3) rw_verify_area() ---> vfs_read() /vfs_write() /do_read/writev() /do_sendfile() ---> sendfile() /sendfile64() /vfs_readadv() /vfs_writev() ---> read() /write() /readv() /writev() /pread64() /pwrite64() /preadv() /pwritev() (fs/read_write.c, sendfile/sendfile64/read/write/readv/ /writev/preadv...都是系统调用) 4) au_rdu() ---> au_rdu_ioctl() (fs/aufs/rdu.c) 5) au_do_iter() ---> aufs_read_iter() /aufs_write_iter() (fs/aufs/ /f_op.c) |
| file_alloc_security | security_file_alloc() | 分配并关联一个安全结构到file->f_security域上。当该结构第一次创建时，安全域被初始化成NULL（在打开文件给文件创建分配file结构体的时候被调用） | get_empty_filp() ---> alloc_file() (分配和初始化file结构体) (fs/file_table.c) |
| file_free_security | security_file_free() | 释放存储在file->f_security中的安全结构（在关闭文件销毁该文件的file结构体的时候被调用） | _fput() /put_filp() ---> delayed_fput() /fput()... (fs/file_table.c) |
| file_ioctl | security_file_ioctl() | 在对参数@file进行ioctl操作时检查权限。注意：参数@arg有时代表一个用户空间指针，而又有时它可能只是一个简单的整数值。当参数@arg代表一个用户空间指针时，它不应该在安全模块中被使用 | 1) ioctl() (fs/ioctl.c, ioctl是系统调用) 2) ioctl() (fs/compat_ioctl.c, ioctl是系统调用) |
| file_mprotect | security_file_mprotect() | 在改变内存访问权限前检查权限 | mprotect() (mm/mprotect.c, mprotect是系统调用) |
| file_lock | security_file_lock() | 在执行文件锁操作前检查权限。注：this hook mediates both flock and fcntl style locks | 1) generic_setlease() ---> vfs_setlease() ---> do_fcntl_add_lease() /fcntl_setlease() (fs/locks.c) 2) flock() (fs/locks.c, flock是系统调用) 3) do_lock_file_wait() ---> fcntl_setlk() /fcntl_setlk64() (fs/locks.c) |
| file_fcntl | security_file_fcntl() | 在允许参数@cmd指定的文件操作在文件@file上执行前检查权限。注意：参数@arg有时代表一个用户空间指针，而又有时它可能只是一个简单的整数值。当参数@arg代表一个用户空间指针时，它不应该在安全模块中被使用 | fcntl() /fcntl64() (fs/fcntl.c, fcntl/fcntl64是系统调用) |
| file_set_fowner | security_file_set_fowner() | 保存拥有者的安全信息（通常来自current->security）到file->f_security中为之后的send_sigiotask钩子使用 | _f_setown() ---> f_setown() /f_setown_ex() ---> do_fcntl() ---> fcntl() /fcntl64() (fs/fcntl.c, fcntl/fcntl64是系统调用) |
| file_send_sigiotask | security_file_send_sigiotask() | 在文件拥有者@fown发送信号SIGIO或SIGURG到进程@tsk时检查权限。注意：1) 该钩子有时候是通过中断去调用；2) fown_struct结构的参数@fown从来不会超出file结构上下文范围之外，因此相应的file结构（和相关的安全信息）总是可以通过一下方法获取到： container_of(fown, struct file, f_owner) | sigio_perm() ---> send_sigio_to_task() /send_sigurg_to_task() ---> send_sigurg() /send_sigio() ---> kill_fasync_rcu() ---> kill_fasync() (fs/fcntl.c) |
| file_receive | security_file_receive() | 该钩子允许安全模块去控制进程通过socket IPC接受打开的文件描述符的能力 | 1) scm_detach_fds_compat() (net/compat.c) 2) scm_detach_fds() (net/core/scm.c) |
| file_open | security_file_open() | 保存打开时(open-time)的权限检查状态，为之后在file_permission上使用，和重新检查访问（如果在inode_permission阶段任何东西发生了改变） | do_dentry_open() ---> finish_open() /vfs_open() ---> dentry_open() (fs/open.c) |
| mmap_addr | security_mmap_addr() | 在参数@addr上进行mmap操作时检查权限 | 1) do_sys_vm86() ---> vm86old() /vm86() (arch/x86/kernel /vm86_32.c, vm86old/vm86是系统调用) 2) validate_mmap_request() ---> do_mmap() (mm/nommu.c) 3) get_unmapped_area() ---> do_brk() /do_mmap() ---> vm_brk() /brk() (mm/mmap.c, brk是系统调用) 4) expand_downwards() ---> expand_stack() ---> find_extend_vma() (mm/mmap.c) |
| mmap_file | security_mmap_file() | 进行mmap操作时检查权限。参数@file可能是NULL，例如映射匿名内存时 | 1) vm_mmap_pgoff() ---> vm_mmap() (mm/util.c) 2) do_shmat() ---> shmat() (ipc/shm.c, shmat是系统调用) |

super_block类钩子

■ 文件系统操作

| 钩子名 | security接口名 | 作用 | 调用点 |
|----------------------|---------------------------------|--|--|
| sb_alloc_security | security_sb_alloc() | 分配并关联一个安全结构到sb->s_security域上。当该结构被分配(allocated)时，s_security域被初始化成NULL | alloc_super() (创建新的超级块) ---> sget_userns() (查找或者创建一个超级块) ---> sget() /mount_ns_userns() ---> mount_bdev() /mount_nodev() /mount_single() /mount_ns() (fs/super.c) |
| sb_free_security | security_sb_free() | 释放和清除sb->s_security域 | destroy_super() (fs/super.c) |
| sb_statfs | security_sb_statfs() | 在获取参数@mnt挂载点的文件系统统计前检查权限 | statfs_by_dentry() ---> vfs_statfs() /vfs_ustat() ---> user_statfs() /fd_statfs() /ustat() ---> statfs() /statfs64() /fstatfs() /fstatfs64() (fs /statfs.c, 其中的ustat/statfs/statfs64/fstatfs/fstatfs64是系统调用) |
| sb_mount | security_sb_mount() | 在将指定对象@dev_name挂载到@nd之前检查权限。对于一个普通挂载，如果文件系统需求一个设备，则@dev_name表示为一个设备；对于重新挂载(@flags & MS_REMOUNT)而言，@dev_name是可以忽略的；对于loopback/bind挂载(@flags @ MS_BIND)而言，@dev_name表示将被挂载对象的路径 | do_mount() ---> mount() (fs/namespace.c, mount是系统调用) |
| sb_copy_data | security_sb_copy_data() | 允许挂载选项数据在被文件系统解析之前拷贝下来，以便安全模块能干净的提取安全性(security-specific)的挂载选项参数（因为文件系统可能会修改这些数据，例如调用函数strsep()）。这里也允许将原始的挂载数据中的安全性选项进行剥离，以避免让文件系统知道他们 | 1) parse_security_options() ---> btrfs_mount() /btrfs_remount() (fs/btrfs/super.c) 2) mount_fs() (fs/super.c) 3) nfs_parse_mount_options() ---> nfs_validate_text_mount_data() /nfs_remount() ---> nfs_fs_mount() (fs/nfs/super.c) |
| sb_remount | security_sb_remount() | 提取安全系统特定的挂载选项，并验证这些选项没有被修改 | do_remount() ---> do_mount() ---> mount() (fs/namespace.c, mount是系统调用) |
| sb_umount | security_sb_umount() | 在卸载@mnt文件系统前检查权限 | do_umount() ---> umount() (fs/namespace.c, umount是系统调用) |
| sb_pivotroot | security_sb_pivotroot() | 在pivot根文件系统前检查权限 | pivot_root() (fs/namespace.c, pivot_root是系统调用) |
| sb_set_mnt_opts | security_sb_set_mnt_opts() | 设置用于超级块(superblock)的安全相关的挂载选项 | 1) setup_security_options() ---> btrfs_mount() /btrfs_remount() (fs/btrfs/super.c) 2) nfs_set_sb_security() ---> nfs_fs_mount() (fs/nfs/super.c) |
| sb_clone_mnt_opts | security_sb_clone_mnt_opts() | 从一个给定的超级块拷贝所有的安全选项到另一个超级块 | nfs_clone_sb_security() ---> nfs_xdev_mount() (fs/nfs/super.c) |
| sb_parse_opts_str | security_sb_parse_opts_str() | 解析填充在@opts结构体中的安全数据字符串 | 1) parse_security_options() ---> btrfs_mount() /btrfs_remount() (fs/btrfs/super.c) 2) nfs_parse_mount_options() ---> nfs_validate_text_mount_data() /nfs_remount() ---> nfs_fs_mount() (fs/nfs/super.c) |
| sb_kern_mount | security_sb_kern_mount() | | mount_fs() (fs/super.c) |
| sb_show_options | security_sb_show_options() | | show_sb_opts() ---> show_vfsmnt() /show_mountinfo() ---> mountinfo_open() /mounts_open() (fs/proc_namespace.c) |
| dentry_init_security | security_dentry_init_security() | 当inode节点不可用时，为dentry计算上下文。（自从NFSv4有EA支持后，NFSv4不再有标签） | nfs4_label_init_security() ---> nfs4_atomic_open() / nfs4_proc_create() /nfs4_proc_symlink() /nfs4_proc_mkdir() / nfs4_proc_mknod() (fs/nfs/nfs4proc.c) |

bprm (linux_binprm) 类钩子

■ 程序执行操作

| 钩子名 | security接口名 | 作用 | 调用点 |
|-----------------------|----------------------------------|--|---|
| bprm_set_creds | security_bprm_set_creds() | 保存bprm->cred->security域中的安全信息（通常是基于有关bprm->file的信息），是为了给之后的apply_creds钩子使用。该钩子可能也被用于检查权限（例如：用于检查安全域间的转换的权限）。该钩子也可能在一个execve中被调用多次（例如：解释程序）。该钩子能识别是否它已经被调用去检查@bprm->cred->security是否为非空(non-NUL)。如果是这样的话，然后该钩子可以决定是去保留之前保存的安全信息还是替换它 | prepare_binprm() ---> do_execveat_common() ---> do_execve() / do_execveat() ---> execve() /execveat() (fs/exec.c, execve/execveat是系统调用) |
| bprm_check_security | security_bprm_check() | 该钩子介入的点是在一个二进制处理程序将开始搜索时(This hook mediates the point when a search for a binary handler will begin)。它允许去对之前通过调用set_creds钩子设置的@bprm->cred->security值进行检查。它与set_creds钩子不同的地方主要是argv列表和envp列表能从@bprm中真实获取到。该钩子也可能在单个execve中被多次调用，并且在每一次都是set_creds第一个被调用 | search_binary_handler() ---> exec_binprm() ---> do_execveat_common() ... (fs/exec.c) |
| bprm_committing_creds | security_bprm_committing_creds() | 准备给一个进程安装新的安全属性，该属性是由execve操作去转换的，这个过程中的数据来源：旧的credentials是在@current->cred中，新的数据是被钩子bprm_set_creds设置在@bprm->cred中。（@bprm是结构体linux_binprm类型的指针。）该钩子是一个好地方去执行进程的状态改变，例如在关闭打开的文件描述符时，属性发生改变时将不再授予访问权限。该钩子在commit_creds()前会立即被调用 | install_exec_creds() (fs/exec.c) |
| bprm_committed_creds | security_bprm_committed_creds() | 整理一个进程安装后的安全属性，该属性是由execve操作去转换的。该新credentials会在这里被设置到@current->cred中去。该钩子是一个好地方去执行进程的状态改变，例如清理non-inheritable（非遗传性的）信号状态。该钩子在commit_creds()后会立即被调用 | install_exec_creds (fs/exec.c) |
| bprm_secureexec | security_bprm_secureexec() | 返回一个boolean值（0或1）指示是否需要一个"secure exec"。该标志是在ELF解释器的初始堆栈辅助表上传递，用于指示是否libc应该打开安全模式 | NEW_AUX_ENT() (fs/binfmt_elf.c) |

task类钩子

■ 任务操作

| 钩子名 | security接口名 | 作用 | 调用点 |
|------------------------|-----------------------------------|--|---|
| task_create | security_task_create() | 创建子进程前检查权限. (通过clone(2) manual page可以查看参数@clone_flags 的定义) | copy_process() ---> _do_fork()/fork_idle() ---> do_fork()/ kernel_thread()/fork()/vfork()/clone() (kernel/fork.c, fork/vfork/clone是系统调用) |
| task_free | security_task_free() | @task将被释放. 处理释放与任务相关的资源. (注: 该钩子能在中断上下文中被调用.) | __put_task_struct() (kernel/fork.c) |
| task_fix_setuid | security_task_fix_setuid() | 在给当前进程设置一个或多个用户身份属性后更新模块状态. 参数@flags指示该钩子被哪个set*uid系统调用调用 (LSM_SETID_*) | setreuid()/setuid()/setresuid()/setsuid() (kernel/sys.c, setreuid/setuid /setresuid/setsuid是系统调用) |
| task_setpgid | security_task_setpgid() | 在设置进程@p组标识符前检查权限. 参数@p包含task_struct (它将会被修改) , @pgid包含新的pgid | setpgid() (kernel/sys.c, setpgid是系统调用) |
| task_getpgid | security_task_getpgid() | 在获取进程@p组标识符前检查权限 | getpgid() (kernel/sys.c, getpgid是系统调用) |
| task_getsid | security_task_getsid() | 在获取进程@p的会话标识符前检查权限 | getsid() (kernel/sys.c, getsid是系统调用) |
| task_getsecid | security_task_getsecid() | 检索(retrieve)进程@p的安全标识符. 如果失败, 则@secid将被设置为0 | (kernel/auditsc.c, kernel/audit.c, kernel/auditfilter.c) |
| task_setnice | security_task_setnice() | 在将进程@p的nice值设置为@nice前检查权限 | 1) nice() (kernel/sched/core.c, nice是系统调用) 2) proc_sched_autogroup_set_nice() (kernel/sched/auto_group.c) 3) set_one_prio() ---> setpriority() (kernel/sys.c, setpriority是系统调用) |
| task_setioprio | security_task_setioprio() | 在将进程@p的ioprio值设置为@ioprio前检查权限 | set_task_ioprio() ---> ioprio_set() (block/ioprio.c, ioprio_set是系统调用) |
| task_getioprio | security_task_getioprio() | 在获取进程@p的ioprio值前检查权限 | get_task_ioprio() ---> ioprio_get() (block/ioprio.c, ioprio_get是系统调用) |
| task_setrlimit | security_task_setrlimit() | 在将当前进程资源@resource的resource limits值设置为@new_rlim前检查权限. 原来的resource limit值能被引用 (current->signal->rlim + resource)检查 | do_prlimit() ---> prlimit64()/setrlimit()/getrlimit() (kernel/sys.c, prlimit64 /setrlimit/getrlimit是系统调用) |
| task_setscheduler | security_task_setscheduler() | 在基于@policy和@lp设置进程@p的调度策略和调度参数前检查权限 | 1) __sched_setscheduler() ---> __sched_setscheduler()/sched_setattr()/ normalize_rt_tasks() ---> sched_setscheduler()/sched_setscheduler_nocheck() ---> do_sched_setscheduler() ---> sched_setscheduler()/sched_setparam() (kernel/sched/core.c, sched_setscheduler/sched_setparam是系统调用) 2) sched_setaffinity() ---> sched_setaffinity() (kernel/sched/core.c, sched_setaffinity是系统调用) 3) cpuset_can_attach() (kernel/cpuset.c) |
| task_getscheduler | security_task_getscheduler() | 在获取进程@p调度信息前检查权限 | sched_getscheduler()/sched_getparam()/sched_getattr()/ sched_getaffinity()/sched_rr_get_interval() (kernel/sched/core.c, 都是系统调用) |
| task_movememory | security_task_movememory() | 在move 属于进程@p的memory前检查权限 | 1) move_pages() (mm/migrate.c, move_pages是系统调用) 2) migrate_pages() (mm/mempolicy.c, migrate_pages是系统调用) |
| task_kill | security_task_kill() | 发送信号@sig到进程@p前检查权限. 参数@info可以是NULL, 常数1, 或者一个指向siginfo结构体的指针. 如果@info是1或者SI_FROMKERNEL(info)值是真, 则该信号signal应该被当作来自内核, 也通常应该被允许 | 1) check_kill_permission() ---> group_send_sig_info()/do_send_specific() ---> __kill_pgpr_info()/kill_pid_info()/kill_something_info()/do_tkill() ... ---> ... ---> rt_tgsigqueueinfo()/tgkill()/kill() /tkill() /rt_sigqueueinfo()... (kernel /signal.c, kill等都是系统调用) 2) kill_pid_info_as_cred() (kernel/signal.c) |
| task_wait | security_task_wait() | 在允许一个进程获得一个子进程@p和获取该子进程状态信息前检查权限 | wait_consider_task() ---> do_wait_thread()/ptrace_do_wait() ---> do_wait() ---> waitid() /wait4() ---> waitpid() (kernel/exit.c, waitid/wait4/waitpid是系统调用) |
| task_prctl | security_task_prctl() | 在当前进程上执行进程控制操作前检查权限 | prctl() (kernel/sys.c, prctl是系统调用) |
| task_to_inode | security_task_to_inode() | 基于一个相关的task的安全属性给一个inode节点设置安全属性, 例如: /proc /pid相关的inode节点 | 1) tid_fd_revalidate() ---> proc_fd_instantiate() /proc_fdinfo_instantiate() (proc文件和inode操作接口) (fs/proc/fd.c) 2) proc_pid_make_inode() (fs/proc/base.c) 3) pid_revalidate() (fs/proc/base.c) 4) map_files_d_revalidate() (fs/proc/base.c) |
| cred_alloc_blank | security_cred_alloc_blank() | 仅分配足够的内存并付给@cred, 这样 cred_transfer()就不会获取到ENOMEM值 | cred_alloc_blank() (kernel/cred.c) |
| cred_free | security_cred_free() | 释放和清除在credentials集合中的 cred->security域 | put_cred_rcu() ---> __put_cred() (kernel/cred.c) |
| cred_prepare | security_prepare_creds() | 准备一个新的credentials集合, 通过从 old集合中拷贝数据 | prepare_creds()/clone_cred() ---> prepare_kernel_cred() / copy_creds() /prepare_exec_creds() (kernel/cred.c) |
| cred_transfer | security_transfer_creds() | 将数据从原始的creds (old) 中转移到新creds中 | key_change_session_keyring() (security/keys/process_keys.c) |
| kernel_act_as | security_kernel_act_as() | 给一个内核服务设置credentials当作 (主体上下文). 当前的task必须是在@secid 中指定的 | set_security_override() ---> set_security_override_from_ctx() (kernel/cred.c) |
| kernel_create_files_as | security_kernel_create_files_as() | 在credentials集合中设置文件creation上下文, 它与指定inode节点的客体上下文一样. 当前的task必须是在@secid中指定的 | set_create_files_as() (kernel/cred.c) |
| kernel_fw_from_file | security_kernel_fw_from_file() | 从用户空间加载firmware (不会为内置的 firmware调用) | fw_read_file_contents() /firmware_loading_store() ---> fw_get_filesystem_firmware()... (drivers/base/firmware_class.c) |

| 钩子名 | security接口名 | 作用 | 调用点 |
|-------------------------|------------------------------------|---|--|
| kernel_module_request | security_kernel_module_request() | 触发内核到用户空间为用户空间自动回调加载给定名字的内核模块的能力. (如modprobe命令根据内核模块名字自动加载内核模块) | _request_module() (kernel/kmod.c) |
| kernel_module_from_file | security_kernel_module_from_file() | 从用户空间加载内核模块 | copy_module_from_user()/copy_module_from_fd() ---> init_module()/finit_module() (kernel/module.c, init_module/finit_module是系统调用) |

netlink类钩子

- Netlink messaging

| 钩子名 | security接口名 | 作用 | 调用点 |
|--------------|-------------------------|--|--|
| netlink_send | security_netlink_send() | 为一个netlink消息保存安全信息, 以便在消息被处理时能进行权限检查. 可以用作提供细粒度的消息传输控制. 如果信息成功保存, 并且消息允许传递时返回0 | 1) netlink_mmap_sendmsg() ---> netlink_sendmsg() 2) netlink_sendmsg() (proto_ops.sendmsg) (net/netlink/af_netlink.c) |

unix domain类钩子

- unix domain networking

| 钩子名 | security接口名 | 作用 | 调用点 |
|---------------------|--------------------------------|---|---|
| unix_stream_connect | security_unix_stream_connect() | 在@sock和@other间建立一个unix domain stream连接前检查权限 | unix_stream_connect() (proto_ops.connect) (net/unix/af_unix.c) |
| unix_may_send | security_unix_may_send() | 在从@sock连接或发送数据报文到@other前检查权限 | 1) unix_dgram_connect() (proto_ops.connect) (net/unix/af_unix.c) 2) unix_dgram_sendmsg() ---> unix_seqpacket_sendmsg() (proto_ops.sendmsg) (net/unix/af_unix.c) |

* 注: @unix_stream_connect和@unix_may_send钩子都是必须的, 因为Linux提供了一种替代传统文件名字空间的Unix域套接字.
然而在文件名字空间中的绑定和连接是由典型的文件权限介导的 (这些文件权限是通过在inode_security_ops中的mknode和permission钩子获取的), 在抽象名字空间中的绑定和连接是完全不会被介导的.
在抽象名字空间仅仅使用socket层钩子是不可能充分控制Unix域套接字的, 因为我们需要知道实际的目标socket, 该socket不会被查询直到进入到af_unix代码中去.

socket类钩子

- socket操作

| 钩子名 | security接口名 | 作用 | 调用点 |
|--------------------------|-------------------------------------|--|--|
| socket_create | security_socket_create() | 在创建一个新的socket前检查权限 | 1) sock_create_lite() ---> kernel_accept() (net/socket.c) 2) __sock_create() ---> sock_create()/sock_create_kern() ---> socket() / socketpair() ---> socketcall (net/socket.c, socket/socketpair/socketcall是系统调用) |
| socket_post_create | security_socket_post_create() | 该钩子允许一个模块去跟新或分配一个per-socket安全结构。注意：安全域不是直接添加到socket结构上，而是将socket安全信息存储在相关的inode节点中。通常inode钩子alloc_security将分配和关联安全信息到sock->file->f_inode->i_security上。该钩子可能用额外的信息（这些信息不能在分配inode节点时获取到）去更新 sock->file->f_inode->i_security域 | 同security_socket_create() |
| socket_bind | security_socket_bind() | 在socket协议层执行bind操作和将套接字@sock绑定到参数@address指定的地址前检查权限 | bind() ---> socketcall() (net/socket.c, bind/socketcall是系统调用) |
| socket_connect | security_socket_connect() | 在socket协议层connect操作尝试去将套接字@sock连接到一个远程地址@address前检查权限 | connect() ---> socketcall() (net/socket.c, connect /socketcall是系统调用) |
| socket_listen | security_socket_listen() | 在socket协议层listen操作前检查权限 | listen() ---> socketcall() (net/socket.c, listen/socketcall是系统调用) |
| socket_accept | security_socket_accept() | 在接受一个新的连接前检查权限。注意：尽管新套接字@newsock被创建和拷贝了一些信息到其中，但是accept操作并没有被真实的执行 | accept4() ---> accept() /socketcall() (net/socket.c, accept4 /accept /socketcall是系统调用) |
| socket_sendmsg | security_socket_sendmsg() | 在传输一则消息到另一个socket前检查权限 | sock_sendmsg() ---> kernel_sendmsg() /sock_write_iter() /sendto() /__sys_sendmsg() ---> __sys_sendmsg() /__sys_sendmmsg() /socketcall() ---> sendmsg() /sendmmsg() ---> socketcall() (net/socket.c, sendto/sendmsg / sendmmsg /socketcall是系统调用) |
| socket_recvmsg | security_socket_recvmsg() | 从一个socket接收一则消息前检查权限 | sock_recvmsg() ---> kernel_recvmsg() /sock_read_iter() /recvfrom() /__sys_recvmsg() ---> __sys_recvmsg() /__sys_recvmmsg() /socketcall() ---> recvmsg() /recvmmsg() ---> socketcall() (net/socket.c, recvfrom/recvmsg / recvmmsg /socketcall是系统调用) |
| socket_getsockname | security_socket_getsockname() | 在获取socket对象的本地地址（名字）前检查权限 | getsockname() ---> socketcall() (net/socket.c, getsockname /socketcall是系统调用) |
| socket_getpeername | security_socket_getpeername() | 在获取socket对象的远程地址（名字）前检查权限 | getpeername() ---> socketcall() (net/socket.c, getpeername /socketcall是系统调用) |
| socket_getsockopt | security_socket_getsockopt() | 在获取套接字@sock相关的options（选项）前检查权限 | 1) getsockopt() ---> socketcall() (net/socket.c, getsockopt /socketcall是系统调用) 2) getsockopt() ---> socketcall() (net/compat.c, getsockopt /socketcall是系统调用) |
| socket_setsockopt | security_socket_setsockopt() | 在设置套接字@sock相关的options（选项）前检查权限 | 1) setsockopt() ---> socketcall() (net/socket.c, setsockopt /socketcall是系统调用) 2) setsockopt() ---> socketcall() (net/compat.c, setsockopt /socketcall是系统调用) |
| socket_shutdown | security_socket_shutdown() | 在套接字@sock上全部或部分连接关闭前检查权限 | shutdown() ---> socketcall() (net/socket.c, shutdown /socketcall是系统调用) |
| socket_sock_rcv_skb | security_sock_rcv_skb() | 检查传入的网络包的权限。该钩子不同于Netfilter的IP输入钩子，因为它是关联了一个特定套接字@sk的传入sk_buff @skb的第一次（过滤和检查点）。在该钩子中一定不要sleep，因为一些调用者持有自旋锁 | sk_filter() (net/core/filter.c) |
| socket_getpeersec_stream | security_socket_getpeersec_stream() | 该钩子允许安全模块通过getsockopt SO_GETPEERSEC为 unix 或连接到用户空间的tcp套接字提供对等的socket安全状态。对于tcp套接字而言，如果套接字关联了ipsec SA则它能变得有意义（meaningful） | sock_getsockopt() (net/core/sock.c) ---> getsockopt() (net/socket.c) |
| socket_getpeersec_dgram | security_socket_getpeersec_dgram() | 该钩子允许安全模块通过getsockopt SO_GETPEERSEC为基于用户空间的udp套接字每一个数据包提供对等的socket安全状态。该应用必须首先通过getsockopt指定IP_PASSSEC选项。它能够通过附加的消息类型SCM_SECURITY为一个数据包(packet)获取(retrieve)从该钩子返回的安全状态 | ip_cmsg_recv_security() ---> ip_cmsg_recv_offset() (net/ipv4/ip_sockglue.c) |
| sk_alloc_security | security_sk_alloc() | (sk: struct sock) 分配和附加一个安全结构到 sk->sk_security域上，该域是被用于在本地套接字流(stream)间拷贝安全属性 | sk_prot_alloc() ---> sk_alloc() /sk_clone_lock() (net/core /sock.c) |
| sk_free_security | security_sk_free() | 释放安全结构 | sk_prot_free() ---> sk_destruct() ---> __sk_free() ---> sock_wfree() / sk_free() (net/core/sock.c) |
| sk_clone_security | security_sk_clone() | 克隆或拷贝安全结构 | 1) sock_copy() ---> sk_clone_lock() (net/core/sock.c) 2) sco_sock_init() ---> sco_sock_create() /sco_conn_ready() ---> sco_connect_cfm() (net/bluetooth/sco.c) 3) rfcomm_sock_init() ---> rfcomm_sock_create() /rfcomm_connect_ind() (net /bluetooth/rfcomm/sock.c) 4) l2cap_sock_init() ---> l2cap_sock_new_connection_cb() / l2cap_sock_create() (net/bluetooth/l2cap_sock.c) |

| 钩子名 | security接口名 | 作用 | 调用点 |
|-------------------------------|-----------------------------------|--|---|
| | | | 5) sctp_copy_sock() --> sctp_do_peeloff() --> sctp_getsockopt_peeloff() --> sctp_getsockopt() (net/sctp /socket.c) 6) tipc_accept() (net/tipc/socket.c) |
| sk_getsecid | security_sk_classify_flow() | 获取LSM指定的secid给sock, 去打开网络认证缓存 | 1) ip_route_output_ports()/ip_route_connect() /ip_route_newports() (include/net/route.h) 2) ping_v4_sendmsg() (struct proto.sendmsg) (net/ipv4 /ping.c) 3) raw_sendmsg() (net/ipv4/raw.c) 4) udp_sendmsg() --> udp_sendpage() (net/ipv4/udp.c) 5) dccp_v6_connect() (net/dccp/ipv6.c) 6) ping_v6_sendmsg() (net/ipv6/ping.c) 7) tcp_v6_connect() (net/ipv6/tcp_ipv6.c) 8) inet6_sk_rebuild_header() (net/ipv6/af_inet6.c) 9) __ip6_datagram_connect() --> ip6_datagram_connect() --> ip6_datagram_connect_v6_only() (net/ipv6 /datagram.c) 10) icmpv6_flow_init() (net/ipv6/icmp.c) 11) rawv6_sendmsg() (net/ipv6/raw.c) 12) udpv6_sendmsg() (net/ipv6/udp.c) 13) inet6_csk_route_socket() --> inet6_csk_xmit() /inet6_csk_update_pmtu() (net/ipv6 /inet6_connection_sock.c) 14) l2tp_ip6_sendmsg() (net/l2tp/l2tp_ip6.c) |
| sock_graft | security_sock_graft() | Sets the socket's isec sid to the sock's sid.) 设置套接字的isec sid到sock的sid | sock_graft() (include/net/sock.h) 1) --> rose_accept() (net/rose/af_rose.c) (struct proto_ops.accept) 2) --> pn_socket_accept() (net/photon/socket.c) 3) --> inet_accept() (net/ipv4/af_inet.c) 4) --> nfc_llcp_accept_dequeue() --> llcp_sock_accept() (net/nfc/llcp_sock.c) 5) --> nr_accept() (net/netrom/af_netrom.c) 6) --> vsock_accept() (net/vmw_vsock/af_vsock.c) 7) --> x25_accept() (net/x25/af_x25.c) 8) --> unix_accept() (net/unix/af_unix.c) 9) --> iucv_accept_dequeue() --> iucv_sock_accept() /iucv_sock_cleanup_listen() --> iucv_sock_close() (net/iucv/af_iucv.c) 10) --> bt_accept_dequeue() (net/bluetooth /af_bluetooth.c) 11) --> ax25_accept() (net/ax25/af_ax25.c) 12) --> llc_ui_sk_init() --> llc_ui_create() /llc_ui_accept() (net/llc/af_llc.c) 13) --> af_alg_accept() --> alg_accept() (crypto/af_alg.c) |
| inet_conn_request | security_inet_conn_request() | Sets the openreq's sid to socket's sid with MLS portion taken from peer sid. | 1) tcp_conn_request() (net/ipv4/tcp_input.c) 2) cookie_v4_check() (net/ipv4/syncookies.c) 3) dccp_v6_conn_request() (net/dccp/ipv6.c) 4) dccp_v4_conn_request() (net/dccp/ipv4.c) 5) cookie_v6_check() (net/ipv6/syncookies.c) |
| inet_csk_clone | security_inet_csk_clone() | 设置新的子套接字的sid到openreq sid | inet_csk_clone_lock() (net/ipv4 /inet_connection_sock.c) |
| inet_conn_established | security_inet_conn_established() | 在skb上设置连接的对等sid(peersid)到secmark | tcp_finish_connect() --> tcp_rcv_synsent_state_process() --> tcp_rcv_state_process() (net/ipv4/tcp_input.c) |
| secmark_relabel_packet | security_secmark_relabel_packet() | 检查进程是否应该被允许用给定的secid去标记 packets | checkentry_lsm() --> secmark_tg_check() (net/netfilter /xt_SECMARK.c) |
| security_secmark_refcount_inc | security_secmark_refcount_inc() | 告诉LSM去增加被加载的secmark标记规则数。 (tells the LSM to increment the number of secmark labeling rules loaded) | checkentry_lsm() --> secmark_tg_check() (net/netfilter /xt_SECMARK.c) |
| security_secmark_refcount_dec | security_secmark_refcount_dec() | 告诉LSM去减少被加载的secmark标记规则数 | secmark_tg_destroy() (net/netfilter/xt_SECMARK.c) |
| req_classify_flow | security_req_classify_flow() | 设置flow的sid到openreq的sid | 1) inet_csk_route_req() (net/ipv4 /inet_connection_sock.c) 2) inet_csk_route_child_sock() (net/ipv4 /inet_connection_sock.c) 3) cookie_v4_check() (net/ipv4/syncookies.c) 4) dccp_v6_send_response() --> dccp_v6_conn_request() (net/dccp/ipv6.c) 5) cookie_v6_check() (net/ipv6/syncookies.c) 6) inet6_csk_route_req() (net/ipv6 /inet6_connection_sock.c) |
| tun_dev_alloc_security | security_tun_dev_alloc_security() | 该钩子允许一个模块去给一个TUN设备分配一个安全结构 | tun_set_if() --> __tun_chr_ioctl() --> tun_chr_ioctl() (drivers/net/tun.c) |
| tun_dev_free_security | security_tun_dev_free_security() | 该钩子允许一个模块去给一个TUN设备释放安全结构 | tun_free_netdev() --> tun_setup() (drivers/net/tun.c) |
| tun_dev_create | security_tun_dev_create() | 在创建一个新的TUN设备前先检查权限 | tun_set_if() -->...--> tun_chr_ioctl() (drivers/net/tun.c) |
| tun_dev_attach_queue | security_tun_dev_attach_queue() | 在添加到一个TUN设备队列前先检查权限 | tun_set_queue() -->...--> tun_chr_ioctl() (drivers/net /tun.c) |

| 钩子名 | security接口名 | 作用 | 调用点 |
|----------------|---------------------------|----------------------------------|---|
| tun_dev_attach | security_tun_dev_attach() | 该钩子能被模块用于更新TUN设备的sock结构相关的任何安全状态 | tun_attach() ---> tun_set_if() / tun_set_queue() ---> tun_chr_ioctl() (drivers/net/tun.c) |
| tun_dev_open | security_tun_dev_open() | 该钩子能被模块用于更新TUN设备的安全结构相关的任何安全状态 | tun_set_if() ---> ... ---> tun_chr_ioctl() (drivers/net/tun.c) |

XFRM类钩子

■ XFRM操作：基于策略的高扩展性的网络安全架构

| 钩子名 | security接口名 | 作用 | 调用点 |
|-----------------------------|--|--|---|
| xfrm_policy_alloc_security | security_xfrm_policy_alloc() | 分配一个安全结构到xp->security域，当xfrm_policy被分配时该安全域被初始化为NULL. (xp是struct xfrm_policy) | 1) copy_from_user_sec_ctx() ---> xfrm_policy_construct() ---> xfrm_add_policy() / xfrm_add_acquire() (struct xfrm_link .doit) (net/xfrm /xfrm_user.c) 2) xfrm_get_policy() (net/xfrm/xfrm_user.c) 3) xfrm_add_pol_expire() (net/xfrm/xfrm_user.c) 4) pfkey_spdadd() (pfkey_handler) (net/key/af_key.c) 5) pfkey_spddelel() (net/key/af_key.c) 6) pfkey_compile_policy() (struct xfrm_mgr .compile_policy) (net/key /af_key.c) |
| xfrm_policy_clone_security | security_xfrm_policy_clone() | 在@new_ctxt上分配一个安全结构，包含来自@old_ctxt结构的信息 | clone_policy() ---> __xfrm_sk_clone_policy() (net/xfrm/xfrm_policy.c) ---> xfrm_sk_clone_policy() (include/net/xfrm.h) ---> sk_clone_lock() (net/core /sock.c) |
| xfrm_policy_free_security | security_xfrm_policy_free() | 释放xp->security | 1) xfrm_policy_destroy_rcu() ---> xfrm_policy_destroy() (net/xfrm /xfrm_policy.c) 2) pfkey_spddelel() (net/key/af_key.c) |
| xfrm_policy_delete_security | security_xfrm_policy_delete() | 授权删除xp->security | 1) xfrm_policy_bysel_ctxt() (net/xfrm/xfrm_policy.c) 2) xfrm_policy_byid() (net/xfrm/xfrm_policy.c) 3) xfrm_policy_flush_secctx_check() ---> xfrm_policy_flush() ---> xfrm_policy_fini() ---> xfrm_net_exit() (struct pernet_operations .exit) (net/xfrm/xfrm_policy.c) |
| xfrm_state_alloc | security_xfrm_state_alloc() | 分配一个安全结构给x->security域，当xfrm_state被分配时安全域被初始化为NULL. (x是struct xfrm_state) 为secid设置相应上下文 | 1) xfrm_state_construct() ---> xfrm_add_policy() / xfrm_add_acquire() (net/xfrm /xfrm_user.c) 2) pfkey_msg2xfrm_state() ---> pfkey_add() (net/key/af_key.c) |
| xfrm_state_alloc_acquire | security_xfrm_state_alloc_acquire() | 分配一个安全结构给x->security域，当xfrm_state被分配时安全域被初始化为NULL. 为secid设置相应上下文 | xfrm_state_find() (net/xfrm/xfrm_state.c) ---> xfrm_tmpl_resolve_one() ---> xfrm_tmpl_resolve() ---> xfrm_resolve_and_create_bundle() ---> xfrm_bundle_lookup() / xfrm_lookup() ---> xfrm_lookup_route() / _xfrm_route_forward() / xfrm_policy_queue_process() ---> xfrm_policy_alloc() (net/xfrm/xfrm_policy.c) |
| xfrm_state_free_security | security_xfrm_state_free() | 释放x->security | xfrm_state_gc_destroy() ---> xfrm_state_gc_task() ---> xfrm_state_init() (net/xfrm/xfrm_state.c) |
| xfrm_state_delete_security | security_xfrm_state_delete() | 授权删除x->security | 1) xfrm_del_sa() (net/xfrm/xfrm_user.c) 2) xfrm_state_flush_secctx_check() ---> xfrm_state_flush() ---> xfrm_state_fini() (net/xfrm/xfrm_state.c) ---> xfrm_net_exit() (net/xfrm/xfrm_policy.c) 3) pfkey_delete() (net/key/af_key.c) |
| xfrm_policy_lookup | security_xfrm_policy_lookup() | 在一个packet (包) 中，当一个flow (流) 选择一个xfrm_policy策略去处理XFRMs时检查权限。当无论是选择一个per-socket策略还是一个通用的xfrm策略时该钩子都会被调用 | 1) xfrm_policy_match() ---> xfrm_policy_lookup_bytype() ---> __xfrm_policy_lookup() ---> __xfrm_policy_check() / xfrm_expand_policies() ---> xfrm_bundle_lookup() / xfrm_lookup() (net/xfrm/xfrm_policy.c) 2) xfrm_sk_policy_lookup() ---> xfrm_lookup() / __xfrm_policy_check() (net/xfrm /xfrm_policy.c) |
| xfrm_state_pol_flow_match | security_xfrm_state_pol_flow_match() | 状态，策略，流的匹配检查。参数@x包含需要去匹配的状态，@xp包含一个匹配检查的策略，@fl包含一个匹配检查的流(flow)。如果这里存在一个匹配则返回1 | xfrm_state_look_at() ---> xfrm_state_find() (net/xfrm/xfrm_state.c) |
| xfrm_decode_session | security_xfrm_decode_session() security_skb_classify_flow() | 参数@skb指向需要解码(decode)的skb，@secid指向需要设置的flow key secid，@ckall表示是否所有被使用的xfrms应该用相同的secid去检查。如果@ckall是0，或者所有被使用的xfrms有相同的secid则返回0 | 1) __xfrm_decode_session() ---> __xfrm_policy_check() (net/xfrm /xfrm_policy.c) 2) ip_send_unicast_reply() (net/ipv4/ip_output.c) 3) icmp_reply() / icmp_route_lookup() ---> icmp_send() / icmp_echo() / icmp_timestamp() (net/ipv4/icmp.c) 4) dccp_v4_ctl_send_reset() (net/dccp/ipv6.c) 5) dccp_v4_route(skb) ---> dccp_v4_ctl_send_reset() (net/dccp/ipv4.c) 6) tcp_v6_send_response() ---> tcp_v6_send_reset() / tcp_v6_send_ack() ---> tcp_v6_timewait_ack() / tcp_v6_reqsk_send_ack() / tcp_v6_do_rcv() / tcp_v6_rcv() (net/ipv6/tcp_ip6.c) 7) synproxy_send_tcp() ---> synproxy_send_client_synack() / synproxy_send_server_syn() / synproxy_send_server_ack() / synproxy_send_client_ack() (net/ipv6/netfilter/ip6t_SYNPROXY.c) 8) nf_send_reset6() (net/ipv6/netfilter/nf_reject_ip6.c) 9) icmp6_send() ---> icmpv6_param_prob() (net/ipv6/icmp.c) 10) icmpv6_echo_reply() ---> icmpv6_rcv() (struct inet6_protocol .handler) (net/ipv6/icmp.c) |

key类钩子

- 密钥管理操作：这类钩子影响所有的密钥管理操作

| 钩子名 | security接口名 | 作用 | 调用点 |
|-----------------|----------------------------|---|---|
| key_alloc | security_key_alloc() | 许可分配一个key和分配安全数据。注：在此时key没有分配一个序列号 | key_alloc() --> key_create_or_update() (security/keys/key.c) |
| key_free | security_key_free() | 销毁通知；释放安全数据 | key_gc_unused_keys() --> key_garbage_collector() (security/keys/gc.c) |
| key_permission | security_key_permission() | 查看一个特定的操作权限是否授予给一个key（密钥）上的进程 | key_task_permission() (security/keys/permission.c) |
| key_getsecurity | security_key_getsecurity() | 获取遵循KEYCTL_GETSECURITY的key相关联的文本格式的安全上下文。该功能给NUL-terminated string分配存储空间，同时调用者也应该去释放它。返回string的长度，或者错误时返回-ve，如果这里不存在label时可能返回0（和一个NULL buffer指针） | keyctl_get_security() --> keyctl() (security/keys/keyctl.c, keyctl是系统调用) |

System V IPC类钩子

- System V IPC操作（IPC：进程间通信）

| 钩子名 | security接口名 | 作用 | 调用点 |
|----------------|---------------------------|------------------------------------|---|
| ipc_permission | security_ipc_permission() | 为访问IPC检查权限 | ipcperms() --> ipc_check_perms() --> ipcget_public() --> ipcget() (ipc/util.c) --> shmget() (ipc/shm.c) /msgget() (ipc/msg.c) /semget() (ipc/sem.c) (shmget/msgget/semget是系统调用) |
| ipc_getsecid | security_ipc_getsecid | 获取ipc对象相关联的secid。如果失败了@secid将被设置成0 | _audit_ipc_obj() (kernel/auditsc.c) --> audit_ipc_obj() (include/linux/audit.h) --> shmctl() (ipc/shm.c, shmctl是系统调用) /ipcperms()/ipccctl_pre_down_nolock() (ipc/util.c) |

System V IPC消息队列中持有的单个message类钩子

- 这类安全钩子用于消息队列中单个消息

| 钩子名 | security接口名 | 作用 | 调用点 |
|------------------------|--------------------------|---|--|
| msg_msg_alloc_security | security_msg_msg_alloc() | 分配和附加一个安全结构到msg->security域中。在该结构第一次被创建时安全域被初始化为NULL | load_msg() (ipc/msgutil.c) --> mq_timedsend() (ipc/mqueue.c, mq_timedsend是系统调用) /do_msghndl() /prepare_copy() --> msgsnd() / do_msgrcv() (ipc/msg.c, msgsnd/msgrcv是系统调用) |
| msg_msg_free_security | security_msg_msg_free() | 为这个消息释放安全结构 | free_msg() (ipc/msgutil.c) |

System V IPC消息队列类钩子

- System V IPC Message Queues

| 钩子名 | security接口名 | 作用 | 调用点 |
|--------------------------|--------------------------------|---|--|
| msg_queue_alloc_security | security_msg_queue_alloc() | 分配和附加一个安全结构给msq->q_perm.security域。在该结构第一次被创建时安全域被初始化为NULL | newque() --> msgget() (ipc/msg.c, msgget是系统调用) |
| msg_queue_free_security | security_msg_queue_free() | 为这个消息队列释放安全结构 | msg_rcu_free() --> freeque() --> msgctl_down() --> msgctl() (ipc/msg.c, msgctl是系统调用) |
| msg_queue_associate | security_msg_queue_associate() | 当通过msgget系统调用请求一个消息队列时检查权限。该钩子只有在为一个已经存在的消息队列返回消息队列标识符时被调用，在创建一个新的消息队列时该钩子不会被调用 | msg_security() --> msgget() (ipc/msg.c, msgget是系统调用) |
| msg_queue_msgctl | security_msg_queue_msgctl() | 在消息队列@msq上执行@cmd指定的消息控制操作时检查权限。参数@msq可能是NULL，如为IPC_INFO或MSG_INFO | 1) msgctl_down() --> msgctl() (ipc/msg.c, msgctl是系统调用) 2) msgctl_nolock() --> msgctl() (ipc/msg.c, msgctl是系统调用) |
| msg_queue_msgsnd | security_msg_queue_msgsnd() | 在一个消息@msg进入消息队列@msq前检查权限 | do_msghndl() --> msgsnd() (ipc/msg.c, msgsnd是系统调用) |
| msg_queue_msgrcv | security_msg_queue_msgrcv() | 在将一个消息@msg从消息队列@msq中移除前检查权限。任务结构@target包含一个指向将会接受消息的进程的指针（当内部接受被执行时，该进程不等于当前进程） | 1) find_msg() --> do_msgrcv() --> msgrcv() (ipc/msg.c, msgrcv是系统调用) 2) pipelined_send() --> do_msghndl() --> msgsnd() (ipc/msg.c, msgsnd是系统调用) |

System V共享内存段类钩子

- System V Shared Memory Segments

| 钩子名 | security接口名 | 作用 | 调用点 |
|--------------------|--------------------------|---|---|
| shm_alloc_security | security_shm_alloc() | 分配和附加一个安全结构到shp->shm_perm.security域。当该结构第一次被创建的时候安全域被初始化为NULL | newseg() --> shmget() (ipc/shm.c, shmget是系统调用) |
| shm_free_security | security_shm_free() | 为这个内存段释放安全结构 | shm_rcu_free() --> shm_destroy() (ipc/shm.c) |
| shm_associate | security_shm_associate() | 在通过shmget系统调用请求一个共享内存区域时检查权限。该钩子只有在为一个已经存在的内存区域返回共享内存区域标识符时被调用，在创建一个新的共享内存区域时该钩子不会被调用 | shm_security() --> shmget() (ipc/shm.c, shmget是系统调用) |
| shm_shmctl | security_shm_shmctl() | 在共享内存区域@shp上执行@cmd指定的共享内存控制操作时检查权限。参数@shp可能是NULL，如为IPC_INFO或SHM_INFO | 1) shmctl_down() --> shmctl() (ipc/shm.c, shmctl是系统调用) 2) shmctl_nolock() --> shmctl() (ipc/shm.c, shmctl是系统调用) 3) shmctl() (ipc/shm.c, shmctl是系统调用) |
| shm_shmat | security_shm_shmat() | 在允许shmctl系统调用将共享内存段@shp附加给调用进程的数据段前检查权限。该附加地址是由@shmaddr指定 | do_shmat() --> shmat() (ipc/shm.c, shmat是系统调用) |

System V 信号量类钩子

■ System V Semaphores

| 钩子名 | security接口名 | 作用 | 调用点 |
|--------------------|--------------------------|--|--|
| sem_alloc_security | security_sem_alloc() | 分配和附加一个安全结构到sma->sem_perm.security域. 当该结构第一次被创建的时候安全域被初始化为NULL | newary() --> semget() (ipc/sem.c, semget是系统调用) |
| sem_free_security | security_sem_free() | 为这个信号量释放安全结构 | sem_rcu_free() --> freeary() (ipc/sem.c) |
| sem_associate | security_sem_associate() | 在通过semget系统调用请求一个信号量时检查权限. 该钩子只有在为一个已经存在的信号量返回该信号量标识符时被调用, 在创建一个新的信号量时该钩子不会被调用 | sem_security() --> semget() (ipc/sem.c, semget是系统调用) |
| sem_semctl | security_sem_semctl() | 在信号量@sma上执行@cmd指定的信号量控制操作时检查权限. 参数@sma可能是NULL, 如为IPC_INFO或SEM_INFO | 1) semctl_nolock() --> semctl() (ipc/sem.c, semctl是系统调用) 2) semctl_setval() --> semctl() (ipc/sem.c, semctl是系统调用) 3) semctl_main() --> semctl() (ipc/sem.c, semctl是系统调用) 4) semctl_down() --> semctl() (ipc/sem.c, semctl是系统调用) |
| sem_semop | security_sem_semop() | 在信号量集@sma的成员上执行操作前检查权限. 如果@alter标识是非0, 则信号量集可能被修改 | semtimedop() --> semop() (ipc/sem.c, semtimedop/semop是系统调用) |

binder类钩子

| 钩子名 | security接口名 | 作用 | 调用点 |
|------------------------|-----------------------------------|---|---|
| binder_set_context_mgr | security_binder_set_context_mgr() | 检查是否允许@mgr成为binder context manager | binder_ioctl_set_ctx_mgr() --> binder_ioctl() (drivers/android/binder.c) |
| binder_transaction | security_binder_transaction() | 检查是否允许@from调用一个binder (transaction)事务调用到@to | binder_transaction() --> binder_thread_write() (drivers/android/binder.c) |
| binder_transfer_binder | security_binder_transfer_binder() | 检查是否允许@from去转移(transfer)一个binder引用(reference)到@to | binder_transaction() |
| binder_transfer_file | security_binder_transfer_file() | 检查是否允许@from转移@file到@to | binder_transaction() |

ptrace类钩子

| 钩子名 | security接口名 | 作用 | 调用点 |
|---------------------|--------------------------------|--|---|
| ptrace_access_check | security_ptrace_access_check() | 在允许当前进程去trace (追踪, 查找) @child进程前检查权限. 在execve (binprm_security_ops-->bprom_set_creds-->set_security/apply_creds) 执行期间, 安全模块可能也想去执行一个进程跟踪检查, 如果该进程已经被追踪那它的安全属性将可能会通过execve来改变 | __ptrace_may_access() --> ptrace_may_access()/ptrace_attach() --> ptrace() (kernel/ptrace.c, ptrace是系统调用) 1. ptrace_may_access() --> proc_ns_follow_link() /proc_ns_readdir() (struct inode_operations) (fs/proc/namespaces.c) 2. ... --> do_task_stat() --> proc_tid_stat() /proc_tgid_stat() (fs/proc/array.c) 3. ... --> proc_pid_wchan() (struct pid_entry) /lock_trace() /proc_fd_access_allowed() /has_pid_permissions() /proc_map_files_lookup() (struct inode_operations .lookup) /proc_map_files_readdir() (struct file_operations .iterate) /do_io_accounting() --> proc_pid_follow_link() /do_proc_readdir() /proc_pid_permission() /pid_getattr() /proc_pid_readdir() /proc_tid_io_accounting() /proc_tgid_io_accounting() (struct pid_entry) (fs/proc/base.c) 4. ... --> mm_access() (kernel/fork.c) 5. ... --> get_robust_list() (kernel/futex.c, get_robust_list是系统调用) 6. ... --> kcmp() (kernel/kcmp.c, kcmp是系统调用) 7. ... --> perf_event_open() (kernel/events/core.c, perf_event_open是系统调用) |
| ptrace_traceme | security_ptrace_traceme() | 在允许当前进程向@parent进程追踪前检查@parent进程有充分的权限去追踪当前进程 | ptrace_traceme() --> ptrace() (kernel/ptrace.c, ptrace是系统调用) |

权限钩子

| 钩子名 | security接口名 | 作用 | 调用点 |
|---------|--|---|--|
| capget | security_capget() | 为进程@target获取权能集@effective, @inheritable和@permitted (effective有效的, inheritable可继承的, permitted允许的) . 该钩子也可能执行权限检查, 用于决定是否当前进程被允许查看进程@target的权能集 | cap_get_target_pid() ---> capget() (kernel/capability.c, capget是系统调用) |
| capset | security_capset() | 给当前进程设置权能集@effective, @inheritable和@permitted | capset() (kernel/capability.c, capset是系统调用) |
| capable | security_capable() security_capable_noaudit() | 检查进程@tsk是否具有在所指示凭证中的权能@cap. (Check whether the @tsk process has the @cap capability in the indicated credentials.) | 1) has_ns_capability()/has_ns_capability_noaudit()/ns_capable_common() file_ns_capable() ---> has_capability()/has_capability_noaudit()/ns_capable() ns_capable_noaudit() (kernel/capability.c) 2) seccomp_prepare_filter() ---> seccomp_prepare_user_filter() ---> seccomp_set_mode_filter() ---> do_seccomp() ---> seccomp() prctl_set_seccomp() (kernel/seccomp.c, seccomp是系统调用) |

审计类钩子

| 钩子名 | security接口名 | 作用 | 调用点 |
|------------------|-----------------------------|---|---|
| audit_rule_init | security_audit_rule_init() | 分配和初始化一个LSM审计规则结构. (@field标志定义在include/linux/audit.h头文件中) | 1) audit_data_to_entry() ---> audit_rule_change() (kernel/auditfilter.c) 2) audit_dupe_lsm_field() ---> audit_dupe_rule() (kernel/auditfilter.c) |
| audit_rule_known | security_audit_rule_known() | 指出给定的@rule是否包含在当前LSM的任何fields | update_lsm_rule() ---> audit_update_lsm_rules() (kernel/auditfilter.c) |
| audit_rule_match | security_audit_rule_match() | 决定给定的@secid是否匹配一个之前已经被audit_rule_known证明过的规则 | 1) audit_filter_user_rules() ---> audit_filter_user() (kernel/auditfilter.c) 2) audit_filter_rules() (kernel/auditsc.c) |
| audit_rule_free | security_audit_rule_free() | 释放之前被audit_rule_init分配的规则结构 | audit_free_lsm_field() ---> audit_free_rule() (kernel/auditfilter.c) |

其他钩子

| 钩子名 | security接口名 | 作用 | 调用点 |
|------------------|--------------------------------|---|--|
| quotactl | security_quotactl() | quota: 配额 | 1) check_quotactl_permission() ---> do_quotactl() ---> quotactl() (fs/quota/quota.c, quotactl是系统调用) 2) quota_sync_all() ---> quotactl() (fs/quota/quota.c, quotactl是系统调用) |
| quota_on | security_quota_on() | | 1) dquot_quota_on() (struct quotactl_ops.quota_on) (fs/quota/dquot.c) 2) dquot_quota_on_mount() (fs/quota/dquot.c) |
| syslog | security_syslog() | 在访问内核消息环或更改日志到控制台前检查权限 | check_syslog_permissions() ---> devkmsg_open()/do_syslog() ---> syslog() (kernel/printk/printk.c, syslog是系统调用) |
| settime | security_settime() | 更改系统时间时检查权限。结构体timespec和timezone都定义在头文件include/linux/time.h中 | 1) stime() (kernel/time/time.c, stime是系统调用) 2) do_sys_settimeofday() ---> settimeofday() (kernel/time/time.c, settimeofday是系统调用) |
| vm_enough_memory | security_vm_enough_memory_mm() | 分配一个新的虚拟内存映射时检查权限 | 1) dup_mmap() ---> dup_mm() ---> copy_mm() ---> copy_process() (kernel/fork.c) 2) __frontswap_unuse_pages() ---> __frontswap_shrink() ---> frontswap_shrink() (mm/frontswap.c) 3) swapoff() (mm/swapfile.c, swapoff是系统调用) 4) mprotect_fixup() ---> mprotect() (mm/mprotect.c, mprotect是系统调用) 5) vma_to_resize() ---> mremap_to() /mremap() (mm/mremap.c, mremap是系统调用) 6) shmem_acct_size() ---> __shmem_file_setup() ---> shmem_kernel_file_setup() /shmem_file_setup() /shmem_zero_setup() ---> memfd_create() (mm/shmem.c, memfd_create是系统调用) 7) shmem_reacct_size() ---> shmem_setattr() (mm/shmem.c) 8) shmem_acct_block() (mm/shmem.c) 9) mmap_region() ---> do_mmap() (mm/mmap.c) 10) acct_stack_growth() ---> expand_upwards() /expand_downwards() ---> expand_stack() ---> find_extend_vma() (mm/mmap.c) 11) do_brk() ---> vm_brk() /brk() (mm/mmap.c, brk是系统调用) 12) insert_vml_struct() ---> __install_special_mapping() ---> _install_special_mapping() /install_special_mapping() (mm/mmap.c) |
| ismaclabel | security_ismaclabel() | 检查通过@name指定的扩展属性是否代表一个MAC标记 | 1) nfs4_xattr_set_nfs4_label() (struct xattr_handler.set) (fs/nfs/nfs4proc.c) 2) nfs4_xattr_get_nfs4_label() (struct xattr_handler.get) (fs/nfs/nfs4proc.c) |
| secid_to_secctx | security_secid_to_secctx() | 将secid转换为安全上下文。如果seadata是NULL，则结果的长度将会在seclen中返回，但没有seadata返回 | 1) scm_passec() ---> scm_recv() (include/net/scm.h) ---> netlink_recvmsg() (net/netlink/af_netlink.c) /unix_dgram_recvmsg() /unix_stream_recvmsg() (net/unix/af_unix.c) 2) audit_log_pid_context() /show_special() ---> audit_log_exit() ... (kernel/audit.c) ---> audit_syscall_exit() (include/linux/audit.h) 3) audit_receive_msg() /audit_log_name() /audit_log_task_context() / audit_log_secctx() (kernel/audit.c) 4) ip_cmsg_recv_security() (net/ipv4/ip_sockglue.c) 5) netlbl_audit_start_common() (net/netlabel/netlabel_user.c) 6) (net/netlabel/netlabel_unlabeled.c) 7) (net/netfilter/*.c) |
| secctx_to_secid | security_secctx_to_secid() | 将安全上下文转换为secid | 1) set_security_override_from_ctx() (kernel/cred.c) 2) (net/netlabel/netlabel_unlabeled.c) 3) checkentry_lsm() ---> secmark_tg_check() (struct xt_target.checkentry) (net/netfilter/xt_SECMARK.c) |
| release_secctx | security_release_secctx() | 释放安全上下文 | |
| getprocattr | security_getprocattr() | | proc_pid_attr_read() (struct file_operations.read) (fs/proc/base.c) |
| setprocattr | security_setprocattr() | | proc_pid_attr_write() (struct file_operations.write) (fs/proc/base.c) |
| d_instantiate | security_d_instantiate() | | 1) nfs_get_root() (fs/nfs/getroot.c) ---> nfs_fs_mount_common() (fs/nfs/super.c) 2) d_instantiate() (给一个dentry填充inode信息) ---> d_tmpfile() / d_make_root() (fs/dcache.c) 3) d_instantiate_unique() (实例化一个没有别名的(non-aliased)dentry) (fs/dcache.c) 4) d_instantiate_no_dalias() (实例化一个没有别名的(non-aliased)dentry) (fs/dcache.c) 5) __d_obtain_alias() ---> d_obtain_alias() (为一个给定的inode查找或分配一个 DISCONNECTED dentry) /d_obtain_root() (为一个给定的inode查找或分配一个 dentry) (fs/dcache.c) 6) d_splice_alias() (如果存在一个树，就将一个断开的dentry连接到该树上) ---> d_add_ci() (fs/dcache.c) |

常用的安全钩子说明

文件操作钩子

- **inode钩子**
- inode节点代表是一个已经存在的文件，在打开文件时（或其他操作）进行权限检查
- inode节点是在文件创建的时候就进行分配，并在文件删除的时候进行销毁，因此inode节点中的安全域是在文件创建的时候就需要分配，在文件删除的时候就会释放

```
inode_alloc_security
inode_free_security
inode_init_security
```

- 主要用到的权限检查钩子（在访问inode时进行权限检查，如文件被打开时）

```
inode_permission
```

- 对文件主要的具体操作时的权限检查，如文件创建，删除，重命名，链接等等

```
inode_create
inode_ls
inode_mkdir
inode_symlink
inode_mkdir
inode_rmdir
inode_mknod
inode_rename
inode_readdir (读软链接)
inode_follow_link (遵循软链接查找路径名)
inode_setattr
inode_getattr
inode_setxattr (在给指定名字扩展属性设置值前检查权限)
inode_post_Setxattr (该钩子在setxattr操作成功后调用，用于更新inode节点的安全域)
inode_getxattr (在获取指定名字扩展属性前检查权限)
inode_listxattr
inode_removexattr
inode_getsecurity (获取指定名字的扩展属性的值)
inode_setsecurity (给指定名字的扩展属性设置值)
```

■ file钩子

- file代表的是一个已经打开的文件，在操作一个已经打开的文件时进行权限检查，如真实的文件读写等操作
- file结构是在打开文件的时候就进行分配，并在关闭文件的时候进行销毁，因此file结构中的安全域是在打开文件的时候就需要分配，在文件关闭的时候就会释放

```
file_alloc_security
file_free_security
```

- 此类钩子主要用到的权限检查钩子是

```
file_permission
```

- 其他使用较少的权限检查钩子，则分别对应的是某个特定的动作

```
mmap_file
file_mprotect
file_ioctl
file_lock
...
```

■ path钩子

- path钩子都是权限控制类的钩子，它一般作为可选的方案
- 在涉及目录和文件名时，用path钩子会更方便直接

可执行程序控制钩子

■ cred钩子

- cred结构体中包含了安全域，这类钩子主要是对cred中安全域的分配，释放，及新旧cred间的安全域的转换

```
cred_alloc_blank
cred_free
cred_prepare
cred_transfer
```

■ task钩子

- task_struct结构体中包含cred，其中存在安全域，主要是对进程任务的安全域的维护

```
task_to_inode (基于进程的安全属性给文件设置安全属性，在涉及进程安全属性对文件的安全属性的影响时一般会用到该钩子)
/* 其他钩子几乎都是进程权限检查相关的 */
task_create
task_free
task_setpgid
task_getpgid
task_getsid
task_kill
...
kernel_module_request (利用模块名加载模块时检查权限)
kernel_module_from_file (加载ko文件时检查权限)
...
```

■ bprm钩子

- linux_binprm结构体中包含cred和file，其中存在安全域
- 该类钩子主要是用于维护，初始化设置，更新程序的安全域信息及在程序执行过程中控制状态的继承

```
bprm_set_creds (涉及到进程任务时该钩子普遍都会用到)
bprm_check_security
bprm_committing_creds
bprm_committed_creds
```

安全模块编写注意点

■ Kconfig文件的编写

1. 在父目录下的Kconfig文件中添加source项，将该模块的Kconfig文件导入（在执行make menuconfig时就会加载该模块的Kconfig，进行配置）


```
/ security/Kconfig
/source security/mylsm/Kconfig
```
2. 安全模块自己的Kconfig文件编写


```
1. 在父目录下的Makefile文件中添加对该模块的引用和关联编译（在obj项后面接的目录名时符号'/'不能省略）
        / security/Makefile
        subdir ${CONFIG_SECURITY_MYLSM}      += mylsm
        obj-$(CONFIG_SECURITY_MYLSM)          += mylsm/
2. 安全模块自己的Makefile文件编写
        
```

■ Makefile文件的编写

1. 在父目录下的Makefile文件中添加对该模块的引用和关联编译（在obj项后面接的目录名时符号'/'不能省略）


```
/ security/Makefile
        subdir ${CONFIG_SECURITY_MYLSM}      += mylsm
        obj-$(CONFIG_SECURITY_MYLSM)          += mylsm/
2. 安全模块自己的Makefile文件编写
        
```

■ 安全模块的实现

1. 在模块初始化函数中需要注册该模块（包括模块名和实现的安全钩子）
2. 各功能的安全钩子的实现

■ lsm安全模块例子

1. 登陆192.168.68.50的ftp服务器
2. 进入到目录“学习资料/报告文档”中下载lsm安全钩子中的mylsm例子

取自 “<http://172.20.189.51/wiki/index.php?title=LSM钩子函数&oldid=270>”

- 本页面最后修改于2017年9月11日 (星期一) 11:55。
- 本页面已经被访问过138次。