L I S E X

GIDIS

Manual de referencia para el programador

Lisex 0.0

Gisela Giusti
Felipe Manzano
Maximiliano Cristiá

GIDIS
Grupo de Investigación y Desarrollo en Ingeniería de Software

Índice General

Capítulo 1

Todas las llamadas al sistema involucradas en Lisex 0.0

1.1 Llamadas al sistema existentes que se modificaron

Funciones

- asmlinkage long sys_uselib (const char ∗library)

  Llamada al sistema uselib. No fue modificada pero usa may_open() y permission().

- int do_execve (char ∗filename, char ∗∗argv, char ∗∗envp, struct pt_regs ∗regs)

  Llamada al sistema execve. No fue modificada pero usa, prepare_binprm() y open_exec() la que a su vez usa permission().

- asmlinkage long sys_mknod (const char ∗filename, int mode, dev_t dev)

  Llamada al sistema mknod. No fue modificada pero usa vfs_mknod() la cual usa may_create().

- asmlinkage long sys_mkdir (const char ∗pathname, int mode)

  Llamada al sistema mkdir. No fue modificada pero usa vfs_mkdir() la cual usa may_create().

- asmlinkage long sys_rmdir (const char ∗pathname)

  Llamada al sistema rmdir. No fue modificada pero usa funciones que usan may_delete()).

- asmlinkage long sys_unlink (const char ∗pathname)

  Llamada al sistema unlink. No fue modificada pero usa funciones que usan may_delete().

- asmlinkage long sys_symlink (const char ∗oldname, const char ∗newname)

  Llamada al sistema symlink. No fue modificada pero usa funciones que usan may_create().

- asmlinkage long sys_link (const char ∗oldname, const char ∗newname)

  Llamada al systema link. No fue modificada pero usa vfs_link() que a su vez usa may_create().

- asmlinkage long sys_rename (const char ∗oldname, const char ∗newname)

  Llamada al sistema rename. No fue modificada pero usa otras funciones que usan may_create() y may_delete().

- asmlinkage long sys_truncate (const char ∗path, unsigned long length)

  Llamada al sistema truncate. No fue modificada pero usa permission().

- asmlinkage long sys_truncate64 (const char ∗path, loff_t length)

  Llamada al sistema truncate64. No fue modificada pero usa permission().

- asmlinkage long sys_access (const char ∗filename, int mode)

  Llamada al sistema access. Esta función no fue modificada, sin embargo usa permission().

- asmlinkage long sys_chdir (const char ∗filename)

  Llamada al sistema chdir.

- asmlinkage long sys_fchdir (unsigned int fd)

  Llamada al systema fchdir. No fue modificada pero usa permission().

- asmlinkage long sys_chroot (const char ∗filename)

  Llamada al sistema chroot. No fue modificada pero usa permission().

- asmlinkage long sys_fchmod (unsigned int fd, mode_t mode)

  Llamada al sistema fchmod.

- asmlinkage long sys_chmod (const char ∗filename, mode_t mode)

  Llamada al sistema chmod.

- asmlinkage long sys_chown (const char ∗filename, uid_t user, gid_t group)

  Llamada al sistema chown.

- asmlinkage long sys_lchown (const char ∗filename, uid_t user, gid_t group)

  Llamada al sistema lchown.

- asmlinkage long sys_fchown (unsigned int fd, uid_t user, gid_t group)

  Llamada al sistema fchown.

- asmlinkage long sys_open (const char *filename, int flags, int mode)

  Llamada al sistema open. No fue modificada pero usa (indirectamente) may_open().

- asmlinkage long sys_creat (const char *pathname, int mode)

  Llamada al sistema creat. No fue modificada pero usa sys_open().

- asmlinkage long sys_close (unsigned int fd)

  Esta llamada no fue modificada. Se la incluye por completitud. Está especificada en el modelo.

- asmlinkage long sys_stat (char *filename, struct __old_kernel_stat *statbuf)

  Llamada al sistema stat.

- asmlinkage long sys_newstat (char *filename, struct stat *statbuf)

  Llamada al sistema newstat. Ídem sys_stat() con la diferencia que esta llamada utiliza la función cp_new_stat().

- asmlinkage long sys_lstat (char *filename, struct __old_kernel_stat *statbuf)

  Llamada al sistema lstat.

- asmlinkage long sys_newlstat (char *filename, struct stat *statbuf)

  Llamada al sistema newlstat. Ídem sys_lstat() con la diferencia que esta llamada utiliza la función cp_new_stat().

- asmlinkage long sys_fstat (unsigned int fd, struct __old_kernel_stat *statbuf)

  Llamada al sistema fstat.

- asmlinkage long sys_newfstat (unsigned int fd, struct stat *statbuf)

  Llamada al sistema newfstat. Ídem sys_fstat() con la diferencia que esta llamada utiliza la función cp_new_stat().

- asmlinkage long sys_stat64 (char *filename, struct stat64 *statbuf, long flags)

  Llamada al sistema stat64. Ídem sys_stat() pero para los sistemas de archivos de 64 bits.

- asmlinkage long sys_lstat64 (char *filename, struct stat64 *statbuf, long flags)

  Llamada al sistema lstat64. Ídem sys_lstat() pero para los fileystems de 64 bits.

- asmlinkage long sys_fstat64 (unsigned long fd, struct stat64 *statbuf, long flags)

  Llamada al sistema fstat64. Ídem sys_fstat() pero para los sistemas de archivos de 64 bits.

- asmlinkage long sys_setuid (uid_t uid)

  Cambia el uid sólo si los archivos abiertos verifican seguridad simple, confinamiento y DAC para el nuevo usuario.

1.1.1 Documentación de las funciones

1.1.1.1 asmlinkage long sys_chdir (const char ∗ *filename*)

Llamada al sistema chdir.

Se modificó la semántica de esta llamada en cuanto a los permisos que se necesitan para ejecutarla. Usa permission().

1.1.1.2 asmlinkage long sys_fchmod (unsigned int *fd*, mode_t *mode*)

Llamada al sistema fchmod.

El código de esta llamada se comentó completamente ya que no cumple con la precondición de la operación del modelo chmod.

fchmod ejecutaba el mismo código que sys_chmod() pero sobre un archivo abierto.

Ahora devuelve siempre -EPERM (operación no permitida).

1.1.1.3 asmlinkage long sys_chmod (const char ∗ *filename*, mode_t *mode*)

Llamada al sistema chmod.

Cambia los permisos del grupo ALLGRP y del usuario y grupo dueño tradicional, en la ACL de un archivo.

Parmetros:

filename Archivo sobre el cual se realizará la operación.

mode Permisos en formato tradicional que serán agregados a la ACL del archivo *filename*.

Postcondicin:

En la ACL del archivo *filename* se cambian los permisos correspondientes al usuario dueño tradicional y al grupo primario de *filename* y también los del grupo ALLGRP por nuevos permisos, que dependenden del valor de *mode*.

Devuelve:

- Un número distinto de cero si *filename* no es un nombre valido.
- -EROFS Si *filename* es un archivo de solo lectura.
- -EPERM Si *filename* es inmutable o append???
- -EACCES Si el usuario que realiza la operación no es dueño del archivo.
- -EBUSY Si *filename* esta abierto en el sistema.
- 0 En caso de que la operación termine con éxito.

Bug

Si no hay espacio en la ACL para todos los cambios, no se realiza ninguno.

1.1.1.4 asmlinkage long sys_chown (const char * *filename*, uid_t *user*, gid_t *group*)

Llamada al sistema chown.

Cambia el usuario dueño tradicional y grupo primario tradicional de un archivo. Esta llamada utiliza la funcion chown_common() la cual fue modificada para cumplir con la nueva especificación.

Parmetros:

filename Archivo al cual se le realizará la operación.

user Nuevo usuario dueno tradicional.

group Nuevo grupo primario tradicional.

Postcondicin:

Cambia el usuario dueño tradicional y grupo primario tradicional del archivo filename, por user y group respectivamente. Se agregan a la ACL del archivo filename permisos para user y para group. Estos permisos son los mismos que tenían los antiguos usuario dueño y grupo primario. Estos últimos son eliminados de la ACL del archivo filename.

Devuelve:

- Un número distinto de cero si filename no es un nombre valido.
- -EACCES Si el usuario que realiza la operación no es el dueño del archivo.
- -EBUSY Si el filename esta abierto en el sistema.
- Lo que devuelva chown_common() en cualquier otro caso.

1.1.1.5 asmlinkage long sys_lchown (const char * *filename*, uid_t *user*, gid_t *group*)

Llamada al sistema lchown.

Cambia el usuario dueño tradicional y grupo primario tradicional a un enlace. Esta llamada utiliza la función chown_common() la cual fue modificada para cumplir con la nueva especificación.

Parmetros:

filename Enlace en el cual se le realiza la operación.

user Nuevo usuario dueño tradicional.

group Nuevo grupo primario tradicional.

Postcondicin:

Cambia el usuario dueño tradicional y grupo primario tradicional del enlace al archivo filename, por user y group respectivamente.

Devuelve:

- Un número distinto de cero si filename no es un nombre valido.
- -EACCES Si el usuario que realiza la operación no es el dueño del archivo.
- -EBUSY Si el enlace esta abierto en el sistema.????????
- Lo que devuelva chown_common() en cualquier otro caso.

1.1.1.6 asmlinkage long sys_fchown (unsigned int *fd*, uid_t *user*, gid_t *group*)

Llamada al sistema fchown.

El código de esta llamada se comentó completamente ya que no cumple con la precondición de la operación del modelo chown.

fchown ejecutaba el mismo código que sys_chown() pero sobre un archivo abierto.

Ahora devuelve siempre -EPERM (operación no permitida).

1.1.1.7 asmlinkage long sys_stat (char * *filename*, struct __old_kernel_stat * *statbuf*)

Llamada al sistema stat.

Devuelve información de estado de un archivo. Esta llamada utiliza la función cp_old_stat() la cual fue modificada para cumplir con la nueva especificación.

Parmetros:

filename Archivo del cual se piede la información.

statbuf Apunta a una estructura donde se devuelven los resultados de la llamada.

Postcondicin:

En el párametro *statbuf* queda cargada información sore *filename,* dentro de la cual se encuentran los permisos del usuario dueño tradicional, el grupo primario tradicional y los permisos del grupo ALLGRP.

Devuelve:

- Un número distinto de cero si *filename* no es un nombre válido.???
- -EACCES Si el usuario que realiza la operación no tiene permiso de lectura sobre *filename*.
- Lo que devuelva cp_old_stat() en cualquier otro caso.

1.1.1.8 asmlinkage long sys_lstat (char * *filename*, struct __old_kernel_stat * *statbuf*)

Llamada al sistema lstat.

Devuelve información del enlace que se obtiene al seguir los enlaces del path que se pasa como parámetro. Esta llamada utiliza la función cp_old_stat() la cual fue modificada para cumplir con la nueva especificación.

Parmetros:

filename Enlace del cual se piede la información.

statbuf Apunta a una estructura donde se devuelven los resultados de lstat.

Postcondicin:

Ídem sys_stat().

Devuelve:

Ídem sys_stat().

1.1.1.9 asmlinkage long sys_fstat (unsigned int *fd*, struct __old_kernel_stat * *statbuf*)

Llamada al sistema fstat.

Devuelve información del inodo al que apunta el descripor de archivo que se pasa como parámetro.

Esta llamada utiliza la función cp_old_stat() la cual fue modificada para cumplir con la nueva especificación.

Parmetros:

fd Descriptor de archivo que apunta al inodo del cual se piede información.

statbuf Apunta a una estructura donde se devuelven los resultados de la llamada.

Postcondicin:

En el parámetro *statbuf* queda cargada información sore el archivo al que apunta *fd,* dentro de la cual se encuentran los permisos del usuario dueno tradicional, el grupo primario tradicional y los permisos del grupo ALLGRP.

Devuelve:

- -EBADF Si *fd* no apunta a un inodo válido. ???
- -EACCES Si el usuario que realiza la operación no tiene permiso de lectura sobre el archivo al que apunta *fd.*
- Lo que devuelva cp_old_stat() en cualquier otro caso.

1.1.1.10 asmlinkage long sys_setuid (uid_t *uid*)

Cambia el uid sólo si los archivos abiertos verifican seguridad simple, confinamiento y DAC para el nuevo usuario.

Este cambio se incluyó debido a que era posible que un usuario se logueara en una terminal con clase de acceso superior a la de aquel (en general debido a que es posible que usando setuid se viole seguridad simple o DAC). El control que se implementó podría debilitarse controlando únicamente seguridad simple y DAC.

Es necesario pensar y diseñar cuidadosamente el mecanismo SUID si va a ser incorporado en Lisex.

Parmetros:

uid El nuevo usuario.

Devuelve:

Se documentan únicamente los errores agregados por Lisex:

- 0 si hay éxito
- ERROR igual al retornado por may_open()

1.2 Nuevas llamadas al sistema

Funciones

- asmlinkage long sys_acladd (char ∗filename, int id, int mode, int type)

  Llamada al sistema acladd. Agrega permisos a un usuario o grupo en la ACL.

- asmlinkage long sys_acldel (char ∗filename, int id, int mode, int type)

  Llamada al sistema acldel. Quita permisos a un usuario o un grupo de una ACL.

- asmlinkage long sys_ownerclose (int pid, char ∗filename)

  Llamada al sistema ownerclose. Permite que un dueño cierre uno de sus archivos que ha sido abierto por otro usuario.

- asmlinkage long sys_oscstat (char ∗filename, int ∗level, int ∗categories, int size)

  Llamada al sistema oscstat. Retorna la clase de acceso de un archivo.

- asmlinkage long sys_chobjsc (char ∗filename, int level, int ∗categories, int size)

  Llamada al sistema chobjsc. Cambia la SC de un archivo.

- asmlinkage long sys_aclstat (char ∗filename, struct acl_entry ∗acl_statbuf, int len)

  Llamada al sistema aclstat. Devuelve información almacenada en una ACL.

- asmlinkage long sys_aclfstat (unsigned int fd, struct acl_entry ∗acl_statbuf, int len)

Llamada al sistema aclfstat. Devuelve información de una ACL.

- asmlinkage long sys_chsubsc (uid_t uid, int level, int ∗categories, int size)
  Llamada al sistema chsubsc. Cambia la clase de acceso de un usuario.
- asmlinkage long sys_sscstat (uid_t uid, int ∗level, int ∗categories, int size)
  Llamada al sistema sscstat. Retorna la clase de acceso de uid.

1.2.1 Documentación de las funciones

1.2.1.1 asmlinkage long sys_acladd (char ∗ *filename*, int *id*, int *mode*, int *type*)

Llamada al sistema acladd. Agrega permisos a un usuario o grupo en la ACL.

Agrega permisos a un usuario o grupo en la ACL del inodo de un archivo, siempre que esto sea posible.

Parmetros:

filename Nombre del archivo sobre el cual se realizará la operacion.

id Identificación del usuario/grupo al que se le agregan permisos a la ACL del inodo del archivo filename.

mode Permisos que se le agregarán a los permisos que tenga el usuario/grupo id en la ACL del inodo del archivo filename.

type Determina si id es la identificación de un usuario o de un grupo.

Postcondicin:

- Se agrega a la ACL del archivo filename el usuario/grupo con identificación id y los permisos mode.

Devuelve:

- Un número distinto de cero si filename no es un nombre valido.
- -EROFS Si el archivo filename es read-only.
- -EPERM Si el archivo filename es inmutable o append.???
- -EACCES Si el usuario que realiza la operación no es dueño del archivo filename.
- -EBUSY Si el archivo filename esta abierto en el sistema.
- -ACL_OK Si la operación se realizó con éxito.
- -ENOMEM Si la ACL del inodo del archivo filename estaba llena y el usuario/grupo id no tenía ningún permiso en la ACL de este inodo antes de realizarse la operación.

1.2.1.2 asmlinkage long sys_acldel (char ∗ *filename*, int *id*, int *mode*, int *type*)

Llamada al sistema acldel. Quita permisos a un usuario o un grupo de una ACL.

Quita permisos a un usuario o un grupo en la ACL del inodo de un archivo, siempre que esto sea posible.

Parmetros:

filename Nombre del archivo sobre el cual se realizará la operación.

id Identificación del usuario/grupo al que se le quitan permisos en la ACL del inodo del archivo filename.

mode Permisos que se le quitarán a los permisos que tenga el usuario/grupo id en la ACL del inodo del archivo filename.

type Determina si id es la identificación de un usuario o de un grupo.

Postcondicin:

La ACL del archivo filename cambia por una nueva ACL donde el usuario/grupo con identificación id tiene los permisos que tenía antes menos los permisos que están en mode, el resto de la ACL queda sin cambios. Si id es el usuario dueno tradicional del archivo filename, y mode contiene el permiso de dueño, la ACL y el archivo cambian de la siguiente forma: El dueño tradicional de filename pasa a ser ROOT, se agrega a la ACL el usuario ROOT con permisos de dueño. Se quitan de la ACL los permisos del usuario id que están en mode.

Devuelve:

- Un número distinto de cero si filename no es un nombre valido.
- -EROFS Si el inodo del archivo filename es read-only.
- -EPERM Si el inodo del archivo filename es inmutable o append?????
- -EPERM Si se intenta borrar al grupo ROOTGRP del conjunto de grupos dueños de la ACL del archivo filename.
- -EACCES Si el usuario que realiza la operación no es dueño de *filename*.
- -EBUSY Si el archivo filename esta abierto en el sistema.
- -EINVAL Si el usuario/grupo id no existía en la ACL del inodo del archivo filename antes de realizarse la operación.
- -ENOMEM Si la operación no pudo agregar a ROOT en la ACL en el caso de que id sea el dueño tradicional de filename.
- -ACL_OK Si la operación se realizó con éxito.

1.2.1.3 asmlinkage long sys_ownerclose (int *pid*, char ∗ *filename*)

Llamada al sistema ownerclose. Permite que un dueño cierre uno de sus archivos que ha sido abierto por otro usuario.

En Lisex las llamadas al sistema que permiten cambiar atributos de seguridad de archivos abiertos retornan siempre con error (ver por ejemplo sys_fchmod()). Por esta razón se provee esta llamada que permite que un dueño cierre uno de sus archivos cuando ha sido abierto por otro usuario. De esta forma, el dueño podrá, inmediatamente después, cambiar los atributos de seguridad de ese archivo.

Notar que el dueño debe invocar ownerclose para cada *pid* que tenga abierto *filename,* para luego poder cambiar algún atributo de seguridad.

Parmetros:

pid Proceso que tiene abierto el archivo *filename*

filename Archivo que se quiere cerrar

Precondicin:

El usuario que invoca la llamada debe ser dueño de *filename.*

Postcondicin:

filename no será un archivo abierto por *pid.*

1.2.1.4 asmlinkage long sys_oscstat (char ∗ *filename*, int ∗ *level*, int ∗ *categories*, int *size*)

Llamada al sistema oscstat. Retorna la clase de acceso de un archivo.

Precondicin:

- La clase de acceso del usuario que realiza la operacion domina a la clase de seguridad del archivo.
- size debe ser el tamaño reservado en el buffer categories y debe ser al menos la cantidad de entradas válidas que hay en la clase de acceso de filename.

Parmetros:

filename Archivo que contiene la clase de seguridad que se utilizará para cargar los datos.

level Donde se cargará el nivel de la clase de acceso de filename.

categories Donde se cargarán las categorías de la clase de acceso de filename.

size Cantidad de categorías que pide cargar en categories.

Postcondicin:

- En el párametro level está el nivel de la clase de seguridad de filename.
- En el párametro categories están las categorías de la clase de seguridad de filename.

Devuelve:

- Un número distinto de cero si filename no es un nombre válido.
- -EACCES Si no se cumple la precondición.
- n Donde n es la cantidad de entradas válidas que hay en la clase de acceso de filename, si len es 0.

- -EINVAL Si size es menor que la cantidad de entradas válidas que hay en la clase de acceso de filename.
- m Si la operación se realizó con éxito. m es la cantidad de entradas válidas que se cargaron en categories.

1.2.1.5 asmlinkage long sys_chobjsc (char ∗ *filename*, int *level*, int ∗ *categories*, int *size*)

Llamada al sistema chobjsc. Cambia la SC de un archivo.

Cambia la SC de un archivo, con los datos que recibe como parámetro.

Precondicin:
El usuario que realiza la operacin debe ser un administrador MAC. El archivo no debe estar abierto en el sistema.

Parmetros:
filename Archivo al cual se le cambia la SC.

level Nivel que se cargará en la SC del archivo filename.

categories Conjunto de categorías que se cargarán en la SC del archivo filename.

size Cantidad de categorías que cargarán en la SC del archivo filename.

Devuelve:

- Un número distinto de cero si filename no es un nombre válido.
- -EROFS Si el archivo filename es read-only
- -EPERM Si el archivo filename es inmutable o append.????
- -EACCES Si el usuario que realiza la operación no es un administrador MAC.
- -EBUSY Si el archivo filename está abierto en el sistema.
- 0 Si la operación se realizó con éxito.

1.2.1.6 asmlinkage long sys_aclstat (char ∗ *filename*, struct acl_entry ∗ *acl_statbuf*, int *len*)

Llamada al sistema aclstat. Devuelve información almacenada en una ACL.

Esta llamada utiliza la función cp_aclstat().

Devuelve información almacenada en la ACL de un archivo, siempre que el usuario que pide dicha información tiene permisos de lectura sobre el archivo.

Parmetros:
filename Archivo del cual se piede la información.

acl_statbuf Apunta a una estructura donde se devuelven los resultados de aclstat.

len Cantidad de entradas de la ACL que se piden obtener como resultado.

Postcondicin:

- En el parámetro statbuf quedan cargadas todas las entradas válidas de la ACL del archivo filename.

Devuelve:

- Un número distinto de cero si filename no es un nombre válido.
- -EACCES Si el usuario que realiza la operación no tiene permiso de lectura sobre el archivo filename.
- Lo que devuelva cp_aclstat() en cualquier otro caso.

1.2.1.7 asmlinkage long sys_aclfstat (unsigned int *fd*, struct acl_entry ∗ *acl_statbuf*, int *len*)

Llamada al sistema aclfstat. Devuelve información de una ACL.

Devuelve información de la ACL del inodo al que apunta el descripor de archivo que se pasa como parámetro. Esta llamada utiliza la función cp_aclstat().

Parmetros:

fd Descriptor de archivo que apunta al inodo del cual se piede la información.

acl_statbuf Apunta a una estructura donde se devuelven los resultados de sys_fstat().

len Cantidad de entradas de la ACL que se piden obtener como resultado.

Postcondicin:

Ídem sys_aclstat()

Devuelve:

Ídem sys_aclstat()

1.2.1.8 asmlinkage long sys_chsubsc (uid_t *uid*, int *level*, int ∗ *categories*, int *size*)

Llamada al sistema chsubsc. Cambia la clase de acceso de un usuario.

Precondicin:

- El usuario que invoca la llamada debe ser un oficial MAC.
- El usuario al cual se le cambiará la clase de acceso no debe tener archivos abiertos.

Parmetros:

uid Identificación del usuario sobre el cual se realiza la operación.

level Nivel que se le asignará a la SC del usuario.

categories Conjunto de categorías que se le asignarán a la SC del usuario.

size Cantidad de categorías que se le asignarán a la SC del usuario.

Postcondicin:

- SC de *uid* cambia por una nueva SC con nivel *level* y conjunto de categorías *categories.*

Devuelve:

- -EBUSY Si *uid* esta corriendo algún proceso en el momento de realizar la operación.
- -EACCES Si el usuario que realiza esta operación no pertenece al grupo SecAdm.
- 0 Si la operación se realizó con éxito.

1.2.1.9 asmlinkage long sys_sscstat (uid_t *uid*, int ∗ *level*, int ∗ *categories*, int *size*)

Llamada al sistema sscstat. Retorna la clase de acceso de *uid.*

Parmetros:

uid Identificación del usuario del cual se quiere conocer la clase de seguridad.

level Donde se cargará el nivel de la clase de acceso de *uid.*

categories Donde se cargarán las categorías de la clase de acceso de *uid.*

size Cantidad de categorías que se pide cargar en categories.

Postcondicin:

- El parámetro *level* contiene el nivel de la clase de acceso de *uid.*
- En el parámetro *categories* están las categorías de la clase de acceso de *uid,* si *size* es mayor que 0.

Devuelve:

- -EACCES Si La clase de acceso del usuario que realiza la operación no domina a la de *uid.*
- Cantidad de entradas válidas que hay en la clase de acceso de *uid,* si *size* es 0. En este caso se carga sólo el nivel.
- -EINVAL Si *size* es menor que la cantidad de entradas válidas que hay en la clase de acceso de *uid.*
- m Si la operación se realizó con éxito. *m* es la cantidad de entradas válidas que se cargaron en *categories.*

1.3 Llamadas al sistema que deberían haber sido modificadas

Funciones

- asmlinkage long sys_mount (char *dev_name, char *dir_name, char *type, unsigned long flags, void *data)

  Llamada al sistema mount.

- asmlinkage long sys_setreuid (uid_t ruid, uid_t euid)

  Llamada al sistema setreuid. Debería haber sido modificada como sys_setuid().

- asmlinkage long sys_setresuid (uid_t ruid, uid_t euid, uid_t suid)

  Llamada al sistema setresgid. Debería haber sido modificada como sys_setuid().

- asmlinkage long sys_setfsuid (uid_t uid)

  Llamada al sistema setrfsuid. Debería haber sido modificada como sys_setuid().

1.3.1 Documentación de las funciones

1.3.1.1 asmlinkage long sys_mount (char * *dev_name*, char * *dir_name*, char * *type*, unsigned long *flags*, void * *data*)

Llamada al sistema mount.

Debería ser imposible montar sistemas de archivos que no sean compatibles con Lisex File System. Tal vez ahora sea imposible hacerlo pero sólo porque no fue tenido en cuenta.

Capítulo 2

Todas las funciones auxiliares involucradas en Lisex 0.0

2.1 Funciones auxiliares existentes que se modificaron

Funciones

- int inode_change_ok (struct inode ∗inode, struct iattr ∗attr)

  Función indirectamente usada por sys_fchmod(), sys_truncate() y sys_truncate64().

- int inode_setattr (struct inode ∗inode, struct iattr ∗attr)

  Función indirectamente usada por sys_fchmod(), sys_truncate() y sys_truncate64().

- file ∗ open_exec (const char ∗name)

  No fue modificada pero usa permission().

- int prepare_binprm (struct linux_binprm ∗bprm)

  Función usada por do_execve().

- void clear_inode (struct inode ∗inode)

  Limpia un inodo.

- void clean_inode (struct inode ∗inode)

  Esta función inicializa los campos del inodo con valores conocidos antes de retornarlo.

- int vfs_permission (struct inode ∗inode, int mask)

  Consulta permisos.

- int permission (struct inode *inode, int mask)

  Es sinónimo de vfs_permission().

- int link_path_walk (const char *name, struct nameidata *nd)

  Resolución de nombres.

- int may_delete (struct inode *dir, struct dentry *victim, int isdir)

  Verifica si es posible o no eliminar el enlace victim del directorio dir, verifica si el tipo de victim es correcto.

- int may_create (struct inode *dir, struct dentry *child)

  Verifica si es posible crear un objeto con dentry child en el directorio dir.

- int mount_is_safe (struct nameidata *nd)

  No fue modificada paro usa permission().

- int chown_common (struct dentry *dentry, uid_t user, gid_t group)

  Esta función es utilizada por sys_chown() y sys_lchown().

- int cp_old_stat (struct inode *inode, struct __old_kernel_stat *statbuf)

  Esta función es utilizada por las llamadas sys_stat() y sys_fstat().

- int cp_new_stat (struct inode *inode, struct stat *statbuf)

  Esta función es utilizada por las llamadas sys_newstat() y sys_newfstat().

- long cp_new_stat64 (struct inode *inode, struct stat64 *statbuf)

  Esta función es utilizada por las llamadas sys_stat64(), sys_fstat64() y sys_lstat64().

2.1.1 Documentación de las funciones

2.1.1.1 int prepare_binprm (struct linux_binprm * *bprm*)

Función usada por do_execve().

Se modificó el chequeo de permisos, que se efectuaba directamente leyendo el modo del inodo, de forma que ahora se hace utilizando permission().

2.1.1.2 void clear_inode (struct inode * *inode*)

Limpia un inodo.

Es llamada por el sistema de archivos para indicar que el inodo ya no es útil.

Parmetros:

inode inodo a limpiar

2.1.1.3 void clean_inode (struct inode $*$ *inode*) `[static]`

Esta función inicializa los campos del inodo con valores conocidos antes de retornarlo.

i_sb, i_ino, i_count, i_state y las listas han sido inicializados en otra parte.

2.1.1.4 int vfs_permission (struct inode $*$ *inode*, int *mask*)

Consulta permisos.

Determina si los permisos que se encuentran en una máscara que se pasa como párametro se encuentran en la ACL de un inodo.

Parmetros:

inode Puntero al inodo al cual se le quieren consultar los permisos.

mask Permisos que se van a comparar con los permisos resultantes de la ACL de *inode*. Estos pueden ser:

- ACL_READ_BIT,
- ACL_WRITE_BIT,
- ACL_OWNER_BIT,
- ACL_EXEC_BIT,

y combinaciones. Se ha cambiado la utilización de permissions() para que se usen los nuevos parámetros.

Devuelve:

- -EROFS Si *mask* contiene permiso de escritura e *inode* está en un sistema de archivos de sólo lectura, y el modo UNIX de *inode* indica que: o es un registro, o un directorio o un link.
- -EACCES Si *mask* contiene permiso de escritura e *inode* es inmutable.
- 0 Si todos los permisos que contiene mask se encuentran en la ACL de *inode*. En caso de que *inode* sea un inodo de un directorio no se tienen en cuenta los permisos de ejecutabilidad.

2.1.1.5 int link_path_walk (const char $*$ *name*, struct nameidata $*$ *nd*)

Resolución de nombres.

Esta es la función básica encargada de la resolución de nombres, convirtiendo un camino en un dentry.

Documentación original: ”We expect ’base’ to be positive and a directory”.

Se cambió la semántica para que se requiera lectura en los directorios intermedios (en lugar de EXEC). Dependiendo del valor de current->link_count sigue o no los links.

Parmetros:

name El camino que se debe resolver.

nd Buffer donde el camino que se resolvió es devuelto.

2.1.1.6 int may_delete (struct inode ∗ *dir*, struct dentry ∗ *victim*, int *isdir*) `[inline, static]`

Verifica si es posible o no eliminar el enlace victim del directorio dir, verifica si el tipo de victim es correcto.

1. No es posible hacerlo si dir es de sólo lectura (efectuado en permission()).
2. Es necesario tener permiso de lecto-escritura en dir (Lisex !!!).
3. No es posible eliminar nada de dir si es append-only.
4. No es posible hacer nada si dir es inmutable (efectuado en permission()).
5. Si el sticky bit está prendido en dir, es necesario:
   - ser el dueño de dir, o
   - ser el dueño de victim.
6. Si victim es append-only o inmutable no es posible hacer nada con los enlaces que apuntan a él.
7. Si se pide remover un directorio y victim no lo es, -ENOTDIR.
8. Si se pide remover algo que no es un directorio y victim no es de ese tipo, -EISDIR.
9. No es posible remover una raíz o un punto de montaje.

Parmetros:

dir Directorio padre del enlace que se quiere borrar.

victim dentry que apunta al enlace que se quiere eliminar.

isdir Determina si *victim* es un directorio o no.

Devuelve:

- 0 Si el usuario que realiza la operación puede abrir dir.
- ERROR Si el usuario que realiza la operacion NO puede abrir dir.

2.1.1.7 int may_create (struct inode * *dir*, struct dentry * *child*) [inline, static]

Verifica si es posible crear un objeto con dentry child en el directorio dir.

1. No es posible hacerlo si child ya existe (sys_open() tiene un tratamiento especial para este caso, pero como esta función es inline está bien)
2. No es posible hacerlo si dir es de sólo lectura (efectuado en permission())
3. No es posible hacerlo si el usuario no puede abrir el directorio padre en modo read-write (Lisex!!!)
4. No es posible hacerlo si dir es inmutable (efectuado en permission())

Parmetros:

dir Directorio padre del archivo que se quiere borrar

child dentry que apunta al archivo que se quiere eliminar

Devuelve:

- 0 Si el usuario que realiza la operación puede abrir el archivo.
- ERROR Si el usuario que realiza la operación NO puede abrir el archivo.

2.1.1.8 int chown_common (struct dentry * *dentry*, uid_t *user*, gid_t *group*) [static]

Esta función es utilizada por sys_chown() y sys_lchown().

Parmetros:

dentry Dentry que contiene un puntero al inodo sobre el cual se le realizará la operación.

user Identificación del usuario que pasará a ser usuario dueño del inodo.

group Identificacion del grupo que pasara a ser grupo primario del inodo.

Postcondicin:

Cambia el usuario dueño tradicional y grupo primario tradicional del inodo al que apunta *dentry,* por *user* y *group* respectivamente. Se agregan a la ACL del inodo al que apunta *dentry* permisos para *user* y *group.* Estos permisos son los mismos que tenían los antiguos usuario dueño y grupo primario. Estos últimos son eliminados de la ACL.

Devuelve:

- -ENOENT Si *dentry* no apunta a un inodo válido.
- -EROFS Si el inodo al que apunta *dentry* es read-only.
- -EPERM Si el inodo al que apunta *dentry* es inmutable o append???

- ACL_OK Si la operación se realizó con éxito.

Bug

No verifica si va a haber espacio para la operación!!

2.1.1.9 int cp_old_stat (struct inode ∗ *inode*, struct __old_kernel_stat ∗ *statbuf*) [static]

Esta función es utilizada por las llamadas sys_stat() y sys_fstat().

Se cambió el código que realizaba el cálculo del modo, el nuevo código calcula el modo utilizando la ACL del inodo que se pasa como párametro. Esta función carga en statbuf información que obtiene del inodo inode.

Parmetros:

inode Inodo sobre el cual se realiza la operación.

statbuf Buffer donde se cargan los resultados.

Devuelve:

- -EFAULT Si no se pudo concretar la copia de los datos a statbuf
- 0 Si la operación se realizó con éxito.

2.1.1.10 int cp_new_stat (struct inode ∗ *inode*, struct stat ∗ *statbuf*) [static]

Esta función es utilizada por las llamadas sys_newstat() y sys_newfstat().

Se cambió el código que realizaba el cálculo del modo, el nuevo código calcula el modo utilizando la ACL del inodo que se pasa como párametro. Los párametros y lo que devuelve es igual a cp_old_stat().

2.1.1.11 long cp_new_stat64 (struct inode ∗ *inode*, struct stat64 ∗ *statbuf*) [static]

Esta función es utilizada por las llamadas sys_stat64(), sys_fstat64() y sys_lstat64().

Se cambió código que realizaba el cálculo del modo, el nuevo código calcula el modo utilizando la ACL del inodo que se pasa como parámetro. Los parámetros y lo que devuelve es igual a cp_old_stat().

2.2 Nuevas funciones auxiliares

Funciones

- int may_open (struct inode ∗inode, int perms)

Determina si un archivo puede ser abierto por un sujeto o no. Usa permission() y mac_permission().

- int acl_chmod_ok (struct inode ∗inode, int mode)

  Calcula si se puede o no realizar sys_chmod() sobre un inodo.

- int is_open (struct inode ∗inode)

  Determina si un archivo está o no abierto en el sistema.

- int mac_permission (struct inode ∗inode, int mode)

  Determina si se tiene permiso para acceder a inode *según la clase de acceso. Implementa seguridad MAC y MLS.*

- int cp_aclstat (struct inode ∗inode, struct acl_entry ∗acl_statbuf, int len)

  Carga en una estructura los datos de la ACL de un inodo.

- int logged (uid_t uid)

  Determina si un sujeto está corriendo algún proceso en el sistema.

2.2.1 Documentación de las funciones

2.2.1.1 int may_open (struct inode ∗ *inode*, int *perms*)

Determina si un archivo puede ser abierto por un sujeto o no. Usa permission() y mac_permission().

Parmetros:

inode Puntero al inodo del archivo que se quiere abrir.

perms Entero que guarda el modo en que se quiere abrir el archivo (lectura y/o escritura).

Devuelve:

- 0 Si el usuario que realiza la operación puede abrir el archivo.
- ERROR Si el usuario que realiza la operación NO puede abrir el archivo.

2.2.1.2 int acl_chmod_ok (struct inode ∗ *inode*, int *mode*)

Calcula si se puede o no realizar sys_chmod() sobre un inodo.

sys_chmod() podrá ejecutarse sobre un inodo si la ACL de este soporta los cambios en los modos del usuario dueño, grupo primario y el grupo ALLGRP.

Parmetros:

inode inodo sobre el cual se quiere ejecutar sys_chmod().

mode Nuevo modo que se intenta establecer para el usuario dueño, grupo primario y grupo ALLGRP.

Devuelve:

- 1 Si es posible realizar la operación sys_chmod() con el modo *mode* sobre el inodo *inode*.
- 0 Si no es posible realizar sys_chmod() con el modo *mode* sobre *inode*.

2.2.1.3 int is_open (struct inode ∗ *inode*)

Determina si un archivo está o no abierto en el sistema.

Parmetros:

inode Puntero a un inodo del VFS, se recorrerán todos los archivos abiertos del sistema para determinar si inodo es o no alguno de ellos.

Devuelve:

- 0 Si *inode* no es el inodo de alguno de los archivos abiertos del sistema.
- 1 Si *inode* es el inodo de un archivo abierto del sistema.

2.2.1.4 int mac_permission (struct inode ∗ *inode*, int *mode*)

Determina si se tiene permiso para acceder a *inode* según la clase de acceso. Implementa seguridad MAC y MLS.

Bug

Hay que bloquear subjectsc
Mal programada, mal indentada, mal concevida. MAL!

Parmetros:

inode Objeto al cual se quiere acceder

mode Los permisos que se requieren

Devuelve:

- 0 Si el usuario que realiza la operación verifica seguridad simple y el proceso verifica confinamiento para el mode solicitado y sobre el inodo inode.
- 1 En caso contrario.

2.2.1.5 int cp_aclstat (struct inode * *inode*, struct acl_entry * *acl_statbuf*, int *len*) [static]

Carga en una estructura los datos de la ACL de un inodo.

Carga en una estructura los datos de la ACL de un inodo que se pasa como parámetro.

Esta función es utilizada por las llamadas sys_aclstat() y sys_aclfstat().

Parmetros:

inode Inodo del cual se quiere obtener la ACL.

acl_statbuf Estructura donde se cargan los resultados

len Cantidad de entradas de la ACL que se piden obtener como resultado.

Devuelve:

- -ENVAL Si len es <0 o si len es menor que la cantidad de entradas válidas que se encuentran en la ACL de inode.
- n Donde n es la cantidad de entradas válidas que hay en la ACL del inodo inode, si len es 0.
- -EFAULF Si no se pudo realizar la carga de la ACL en statbuf.
- m Si la operación se realizó con éxito. m es la cantidad de entradas validas que se cargaron en statbuf.

Ejemplos:

iterador_acl_entry.c.

2.2.1.6 int logged (uid_t *uid*)

Determina si un sujeto está corriendo algún proceso en el sistema.

Determina si un usario está logueado (corriendo algún proceso) o no en el sistema.

Parmetros:

uid Usuario sobre el cual se realiza la operación.

Devuelve:

- 0 Si el usuario con identificación *uid* no está corriendo ningún proceso en el sistema.
- 1 Si el usuario está corriendo al menos un proceso en el sistema.

2.3 Funciones auxiliares a nivel de EXT2 que fueron modificadas

Funciones para transformar el modo EXT2 <–> VFS

Las usan ext2_read_inode() y ext2_update_inode()

- #define ext2tovfs_mode(mode)

  Macro para transformar la representación del modo.

- #define vfstoext2_mode(mode, t)

  Macro para transformar la representación del modo.

Funciones

- inode ∗ ext2_new_inode (const struct inode ∗dir, int mode)

  Se agregó código para inicializar idoneamente las ACL y la SC del nuevo inodo.

- int ext2_update_inode (struct inode ∗inode, int do_sync)

  Graba en un inodo físico del superbloque de EXT2 los datos de un inodo del VFS.

- void ext2_read_inode (struct inode ∗inode)

  Lee un inodo físico del superbloque de EXT2 y lo ajusta a un inodo del vfs.

2.3.1 Documentación de las definiciones

2.3.1.1 #define ext2tovfs_mode(mode)

Valor:

```
(char) (((mode & EXT2_ACL_EXEC_BIT)?ACL_EXEC_BIT:0) | \
                    ((mode & EXT2_ACL_READ_BIT)?ACL_READ_BIT:0) | \
                    ((mode & EXT2_ACL_WRITE_BIT)?ACL_WRITE_BIT:0) | \
                    ((mode & EXT2_ACL_OWNER_BIT)?ACL_OWNER_BIT:0) )
```

Macro para transformar la representación del modo.

Macro usada en ext2_read_inode() para transformar la representación de los los permisos que se utilizan a nivel EXT2 en la representación que se utiliza a nivel VFS.

2.3.1.2 #define vfstoext2_mode(mode, t)

Valor:

```
(__u8) (((mode & ACL_EXEC_BIT)?EXT2_ACL_EXEC_BIT:0) | \
                    ((mode & ACL_READ_BIT)?EXT2_ACL_READ_BIT:0) | \
                    ((mode & ACL_WRITE_BIT)?EXT2_ACL_WRITE_BIT:0) | \
                    ((mode & ACL_OWNER_BIT)?EXT2_ACL_OWNER_BIT:0) | \
                        ((t==ACL_USR)?EXT2_ACL_USR_BIT:0) )
```

Macro para transformar la representación del modo.

Macro usada en ext2_update_inode() para transformar la representación de los los permisos que se utilizan a nivel VFS en la representación que se utiliza a nivel VFS.

El parámetro *t* sirve para indicar si se trata de permisos de usuario o de grupo.

2.3.2 Documentación de las funciones

2.3.2.1 int ext2_update_inode (struct inode * *inode*, int *do_sync*) `[static]`

Graba en un inodo físico del superbloque de EXT2 los datos de un inodo del VFS.

Sólo se modificó lo necesario para que se guarden apropiadamente los nuevos miembros del (vfs_)inode en los nuevos miembros del ext2_inode.

2.3.2.2 void ext2_read_inode (struct inode * *inode*)

Lee un inodo físico del superbloque de EXT2 y lo ajusta a un inodo del vfs.

Sólo se modificó lo necesario para que levante los nuevos miembros del ext2_inode y los ponga en los nuevos miembros del (vfs_)inode.

Capítulo 3

TADs y estructuras modificadas o agregadas

3.1 TAD vfs_acl

vfs_acl representa la ACL a nivel del VFS.

Clases

- struct vfs_acl
  ACL a nivel VFS. Se utiliza dentro del inodo del VFS.

Funciones privadas de vfs_acl

- #define ACL_INUSE(e) (e.mode != ACL_NULL_BITS)
  Determina si la entrada e *de la ACL esta o no en uso.*
- int search_free_slot (struct vfs_acl ∗vfs_acl)
  Busca una entrada libre dentro de la ACL.
- int search_id (struct vfs_acl ∗vfs_acl, int id, int type)
  Busca la entrada en la ACL que corresponde a al usuario o grupo id.

Constantes (públicas) de vfs_acl

Debe considerarse que el nombre de la constante es público y su valor privado.

- #define ACL_OK 0

  Retorno de funciones que manejan las ACL.

Iterador para las acl_entry de un objeto vfs_acl

También forma parte de la interfaz de vfs_acl.

Ver acl_for_each para un ejemplo.

- #define ACL_IT_ERROR -1

  Retorno de las funciones que iteran sobre la ACL de un inodo.

- #define acl_for_each(acl, i) for(i=acl_first(acl);i!=ACL_IT_ERROR;i=acl_-next(acl,i))

  Macro que itera entre todas las entradas válidas de una ACL.

- acl_iterator acl_first (struct vfs_acl ∗vfs_acl)

  Busca la primer entrada válida dentro de la ACL.

- acl_iterator acl_next (struct vfs_acl ∗vfs_acl, acl_iterator it)

  Busca la próxima entrada válida en la ACL a partir de una posición que se pasa como parámetro.

- acl_entry acl_getentry (struct vfs_acl ∗vfs_acl, acl_iterator it)

  Devuelve una entrada de la ACL.

Funciones de la interfaz de vfs_acl

- umode_t acl_calc_xmode (struct vfs_acl ∗vfs_acl, uid_t uid, gid_t gid)

  Calcula el modo tradicional de un inodo a partir de su ACL.

- int xmode_to_acl (int xmode, int owner)

  Calcula los permisos compatibles con el modo.

- void acl_aclinit (struct vfs_acl ∗vfs_acl)

  Inicializa una ACL.

- void acl_acldestroy (struct vfs_acl ∗vfs_acl)

  Aun no está implementada, pero debería liberar la ACL que recibe como parámetro.

- int acl_setmode (struct vfs_acl ∗vfs_acl, int id, int mode, int type)

  Establece el modo de un usuario o grupo en la ACL.

- int acl_getmode (struct vfs_acl ∗vfs_acl, int id, int type)

Obtiene el modo de un usuario/grupo.

- int acl_getsize (struct vfs_acl ∗vfs_acl)

  Devuelve la cantidad de entradas válidas que existen en la ACL.

3.1.1 Descripción detallada

vfs_acl representa la ACL a nivel del VFS.

Este TAD oculta la representación de las listas de control de acceso a nivel del VFS, así como los algoritmos que se usan para establecer y obtener los atributos privados.

La estructura oculta *vfs_acl* no debe ser accedida excepto por las funciones de la interfaz aquí documentada. Como esto no es posible imponerlo en tiempo de compilación, los programadores son los responsables de mantener esta disciplina.

3.1.2 Documentación de las definiciones

3.1.2.1 #define ACL_OK 0

Retorno de funciones que manejan las ACL.

Indica que la operación se realizó con éxito.

3.1.2.2 #define ACL_IT_ERROR -1

Retorno de las funciones que iteran sobre la ACL de un inodo.

Indica que no existen más entradas válidas en la ACL a partir de la posición pasada como parámetro.

3.1.2.3 #define acl_for_each(acl, i) for(i=acl_first(acl);i!=ACL_IT_-ERROR;i=acl_next(acl,i))

Macro que itera entre todas las entradas válidas de una ACL.

Parmetros:

acl ACL sobre la cual se va a iterar.

i Es el puntero iterador.

Ejemplos:

iterador_acl_entry.c.

3.1.3 Documentación de las funciones

3.1.3.1 int search_free_slot (struct vfs_acl * *vfs_acl*)

Busca una entrada libre dentro de la ACL.

Parmetros:

vfs_acl ACL donde se realizará la búsqueda.

Devuelve:

Posición dentro de *vfs_acl* donde se encontró el primer lugar libre, en caso de que la ACL esté llena, este valor sera MAX_ACL_LEN.

3.1.3.2 int search_id (struct vfs_acl * *vfs_acl*, int *id*, int *type*)

Busca la entrada en la ACL que corresponde a al usuario o grupo id.

Parmetros:

vfs_acl ACL donde se realizará la búsqueda.

id Identificación del usuario o grupo a buscar.

type Indica si *id* es de un usuario o de un grupo.

Devuelve:

Posición dentro de *vfs_acl* donde se encontró el usuario o grupo. Si esta es igual a MAX_ACL_LEN significará que no se encontró tal usuario/grupo.

3.1.3.3 umode_t acl_calc_xmode (struct vfs_acl * *vfs_acl*, uid_t *uid*, gid_t *gid*)

Calcula el modo tradicional de un inodo a partir de su ACL.

Parmetros:

vfs_acl ACL sobre la cual se calculará el modo tradicional.

uid Identificación del usuario dueño tradicional del inodo.

gid Identificación del grupo dueño tradicional del inodo.

Devuelve:

Un modo que se calcula a partir los permisos que tienen:

- el usuario dueno tradicional,
- el grupo tradicional y AllGrup;

dentro de *vfs_acl*.

3.1.3.4 int xmode_to_acl (int *xmode*, int *owner*)

Calcula los permisos compatibles con el modo.

Calcula los permisos compatibles con el modo utilizado en una ACL, a partir de un modo tradicional y de un entero que indica si son permisos de dueño o no.

Parmetros:

xmode Modo tradicional.

owner Los permisos que se calculan corresponden a un dueño o no.

Devuelve:

El modo en formato ACL.

3.1.3.5 void acl_aclinit (struct vfs_acl * *vfs_acl*)

Inicializa una ACL.

Parmetros:

vfs_acl Es la ACL a inicializar, se establece un valor nulo para los modos de las entradas de *vfs_acl* y se asigna 0 en size::vfs_acl.

3.1.3.6 int acl_setmode (struct vfs_acl * *vfs_acl*, int *id*, int *mode*, int *type*)

Establece el modo de un usuario o grupo en la ACL.

En caso de que el usuario/grupo exista en la ACL, cambia su modo siempre que el modo pasado como parámetro sea un modo válido. Si no existe tal usuario/grupo y hay lugar en la ACL, lo agrega.

Parmetros:

vfs_acl ACL sobre la que se realizará la operación.

id Identificación del usuario o grupo que se busca dentro de la ACL.

mode Nuevo modo que deberá quedar cargado en la ACL para el usuario/grupo con identificacion *id* luego de realizar la operación, siempre que esto sea posible.

type Indica si *id* representa la identificación de un usuario o de un grupo.

Devuelve:

- ACL_OK En caso de que el usuario o grupo ya exista en *vfs_acl* y *mode* sea un modo válido.
- ACL_OK En caso de que el usuario o grupo no se encuentre en *vfs_acl* pero que exista un lugar vacío y *mode* sea un modo válido.

- ACL_OK En caso de que el usuario o grupo no se encuentre en *vfs_acl* pero que el modo sea el modo nulo.
- ENOMEM En caso de que el usuario o grupo no se encuentre en *vfs_acl* y esta esté llena.

3.1.3.7 int acl_getmode (struct vfs_acl ∗ *vfs_acl*, int *id*, int *type*)

Obtiene el modo de un usuario/grupo.

Parmetros:

vfs_acl ACL sobre la cual se realizará la operación.

id Identifiación del usuario del cual se desea obtener su modo.

type Indica si *id* se refiere a la identificación de un grupo o de un usuario.

Devuelve:

- El modo del usuario/grupo con identificación *id* en la ACL *vfs_acl*.

3.1.3.8 int acl_getsize (struct vfs_acl ∗ *vfs_acl*) `[inline]`

Devuelve la cantidad de entradas válidas que existen en la ACL.

Parmetros:

vfs_acl ACL de la cual se obtiene la cantidad de entradas válidas almacenadas.

Devuelve:

La cantidad de entradas válidas dentro de la ACL.

Ejemplos:

iterador_acl_entry.c.

3.1.3.9 acl_iterator acl_first (struct vfs_acl ∗ *vfs_acl*)

Busca la primer entrada válida dentro de la ACL.

Parmetros:

vfs_acl ACL donde se realizará la búsqueda.

Devuelve:

- La posición dentro de *vfs_acl* donde se encontró la primer entrada válida.
- ACL_IT_ERROR En caso de que *vfs_acl* este vacia.

3.1.3.10 acl_iterator acl_next (struct vfs_acl * *vfs_acl*, acl_iterator *it*)

Busca la próxima entrada válida en la ACL a partir de una posición que se pasa como parámetro.

Parmetros:

vfs_acl ACL donde se realizará la búsqueda.

it Posición a partir de la cual se realizará la búsqueda, previamente obtenido con acl_first() o acl_next().

Devuelve:

- La posición dentro de *vfs_acl* donde se encontró la práxima entrada válida.
- ACL_IT_ERROR En caso de que *it* sea la posición de la última entrada válida dentro de *vfs_acl*.

3.1.3.11 struct acl_entry acl_getentry (struct vfs_acl * *vfs_acl*, acl_iterator *it*)

Devuelve una entrada de la ACL.

Parmetros:

vfs_acl ACL a la que pertenece la entrada que se devuelve.

it Posición, dentro de *vfs_acl,* de la entrada que se quiere obtener.

Devuelve:

La entrada ubicada en la posición *it* de *vfs_acl*.

Ejemplos:

iterador_acl_entry.c.

3.1.4 Estructura vfs_acl

ACL a nivel VFS. Se utiliza dentro del inodo del VFS.

```
#include <fs_acl.h>
```

Atributos públicos

- acl_entry acl_entry [MAX_ACL_LEN]
- int size

3.1.4.1 Descripción detallada

ACL a nivel VFS. Se utiliza dentro del inodo del VFS.

Esta estructura debe ser accedida sólo a través de las funciones de su interfaz.

3.1.4.2 Documentación de los datos miembro

3.1.4.3 struct acl_entry vfs_acl::acl_entry[MAX_ACL_LEN]

Arreglo de MAX_ACL_LEN estructuras acl_entry.

3.1.4.4 int vfs_acl::size

Representa la cantidad de entradas válidas en el arreglo anterior.

La documentación para esta estructura fué generada a partir del siguiente archivo:

- fs_acl.h

3.2 TAD acl_entry

acl_entry representa una terna de la forma (ID, MODO, USR_GRP).

Clases

- struct acl_entry
  Entrada de la ACL a nivel VFS. Se utiliza dentro del la estructura vfs_acl.

Constantes (públicas) de acl_entry

Debe considerarse que el nombre de la constante es público y su valor privado.

- #define ACL_NULL_BITS 0
  Para representar el modo nulo (ningun permiso) en la ACL.
- #define ACL_EXEC_BIT 1
  Para representar el permiso de ejecutabilidad en la ACL.

- #define ACL_WRITE_BIT 2

  Para representar el permiso de escritura en la ACL.

- #define ACL_READ_BIT 4

  Para representar el permiso de lectura en la ACL.

- #define ACL_OWNER_BIT 8

  Para representar el permiso de propiedad (dueño) en la ACL.

- #define ACL_RWXO_BIT ACL_EXEC_BIT|ACL_READ_BIT|ACL_WRITE_-BIT|ACL_OWNER_BIT

  Para representar el modo que tiene todos los permisos y también es dueño.

- #define ACL_GRP 0

  Para indicar que se trata de los permisos de un grupo en una entrada de la ACL.

- #define ACL_USR 1

  Para indicar que se trata de los permisos de un usuario en una entrada de la ACL.

Funciones de la interfaz de acl_entry

- #define ACL_TYPE(e) e.type

  Devuelve el campo type *de una entrada de ACL.*

- #define ACL_MODE(e) e.mode

  Devuelve el campo mode *de una entrada ACL.*

- #define ACL_ID(e) ((ACL_TYPE(e)==ACL_USR)?e.id.uid:e.id.gid)

  Devuelve el el campo id *de una entrada ACL.*

3.2.1 Descripción detallada

acl_entry representa una terna de la forma (ID, MODO, USR_GRP).

Este TAD oculta la representación de las ternas (ID, MODO, USR_GRP) que se usarán en las ACL a nivel del VFS, así como los algoritmos que se usan para obtener los atributos privados.

La estructura oculta *acl_entry* no debe ser accedida excepto por las funciones de la interfaz aquí documentada. Como esto no es posible imponerlo en tiempo de compilación, los programadores son los responsables de mantener esta disciplina.

3.2.2 Documentación de las definiciones

3.2.2.1 #define ACL_GRP 0

Para indicar que se trata de los permisos de un grupo en una entrada de la ACL.

Se utiliza para establecer el campo acl_entry::type de la estructura ACL.

3.2.2.2 #define ACL_USR 1

Para indicar que se trata de los permisos de un usuario en una entrada de la ACL.

Se utiliza para establecer el campo acl_entry::type de la estructura ACL.

3.2.2.3 #define ACL_ID(e) ((ACL_TYPE(e)==ACL_USR)?e.id.uid:e.id.gid)

Devuelve el el campo *id* de una entrada ACL.

Como acl_entry::id es una union puede retornarse un entero que representa un UID o GID, dependiendo del valor de acl_entry::type.

3.2.3 Estructura acl_entry

Entrada de la ACL a nivel VFS. Se utiliza dentro del la estructura vfs_acl.

`#include <fs_acl.h>`

Atributos públicos

- union {
  uid_t uid
  gid_t gid
  } id
- char mode
- char type

3.2.3.1 Descripción detallada

Entrada de la ACL a nivel VFS. Se utiliza dentro del la estructura vfs_acl.

Comentarios:
Esta estructura debe ser accedida sólo a través de las funciones de su interfaz.

Ejemplos:
iterador_acl_entry.c.

3.2.3.2 Documentación de los datos miembro

3.2.3.3 uid_t acl_entry::uid

Identifica a un usuario.

3.2.3.4 gid_t acl_entry::gid

Identifica a un grupo.

3.2.3.5 union { ... } acl_entry::id

Determina si la entrada corresponde a un usuario o grupo. Dependerá del campo *type* si *id* es *uid* o *gid*.

3.2.3.6 char acl_entry::mode

Se utiliza para representar los permisos del usuario/grupo (id) sobre el inodo.

3.2.3.7 char acl_entry::type

Puede tomar los valores ACL_USR o ACL_GRP, dependiendo si se trata de una entrada para un usuario o para un grupo.

La documentación para esta estructura fué generada a partir del siguiente archivo:

- fs_acl.h

3.3 TAD sc

sc representa las clases de seguridad a nivel de VFS.

Clases

- struct sc
  SC a nivel VFS. Se utiliza dentro del inodo del VFS.

Constantes (públicas) de sc

Debe considerarse que el nombre de la constante es público y su valor privado.

- #define SC_CAT_FULL -1

Retorno de funciones que manejan las SC.

- #define SC_OK 0

  Retorno de funciones que manejan las SC.

- #define SCADMIN 65535

  Categoría que deben tener los administradores de seguridad para cambiar clases de acceso de sujetos y objetos.

Iterador para las categorías de un objeto sc

También forma parte de la interfaz de sc.

Ver sc_for_each_cat para un ejemplo que muestra cómo usarlo.

- #define SC_IT_ERROR -1

  Retorno de las funciones que iteran sobre las categorías de una SC de un inodo.

- #define sc_for_each_cat(sc, i) for(i=sc_cat_first(sc);i!=SC_IT_ERROR;i=sc_cat_next(sc,i))

  Macro que itera entre todas las categorías de una SC.

- sc_cat_iterator sc_cat_first (struct sc ∗sc)

  Obtiene un iterador o selector que corresponde a la primer categoría válida dentro de las categorías de una SC.

- sc_cat_iterator sc_cat_next (struct sc ∗sc, sc_cat_iterator it)

  Obtiene el iterador correspondiente a la próxima categoría válida dentro de una SC con respecto a una posición de referencia pasada como parámetro.

- int sc_getcat (struct sc ∗sc, sc_cat_iterator it)

  Obtiene la categoría referenciada por un iterador.

Funciones de la interfaz de sc

- void sc_destroy (struct sc ∗sc)

  Libera los recursos asignados a una SC.

- int sc_getsize (struct sc ∗sc)

  Obtiene la cantidad de categorías válidas dentro de una SC.

- int sc_getlevel (struct sc ∗sc)

  Obtiene el nivel de una SC.

- void sc_setlevel (struct sc ∗sc, unsigned int l)

Establece el nivel de una SC.

- void sc_clearcat (struct sc ∗sc)

  Borra todas las categorías de una SC.

- int sc_addcat (struct sc ∗sc, int cat)

  Agrega una categoría las categorías de una SC.

- int sc_compare (struct sc ∗sc1, struct sc ∗sc2)

  Determina si sc2 *domina a* sc1.

- int sc_copy (struct sc ∗sc1, struct sc ∗sc2)

  Copia el contenido de una clase de seguridad en otra.

- int sc_usergetncat (struct sc ∗sc, int ∗cat, int n)

  Carga un conjunto de categorías que provienen del espacio de usuario a una SC.

- int sc_userputcat (struct sc ∗sc, int ∗cat)

  Carga un conjunto de categorías que provienen de una SC en espacio de kernel a un conjunto de categorías que se encuentra en espacio de usuario.

Tipos definidos

- typedef int sc_cat_iterator

  Renombre del tipo int, utilizado como iterador dentro de las categorías de una SC.

Funciones

- sc ∗ sc_init (void)

  Inicializa una SC.

3.3.1 Descripción detallada

sc representa las clases de seguridad a nivel de VFS.

Este TAD oculta la representación de las clases de acceso (o de seguridad) a nivel de VFS, así como los algoritmos que se usan para establecer y obtener los atributos privados.

La estructura oculta *sc* no debe ser accedida excepto por las funciones de la interfaz aquí documentada. Como esto no es posible imponerlo en tiempo de compilación, los programadores son los responsables de mantener esta disciplina. En caso de que no existan las funciones apropiadas para acceder los atributos privados, los programadores son libres de hacerlo como lo crean conveniente.

3.3.2 Documentación de las definiciones

3.3.2.1 #define SC_CAT_FULL -1

Retorno de funciones que manejan las SC.

Indica que la operación no pudo realizarse porque la SC del inodo está llena.

3.3.2.2 #define SC_OK 0

Retorno de funciones que manejan las SC.

Indica que la operación se realizó con éxito.

3.3.2.3 #define SC_IT_ERROR -1

Retorno de las funciones que iteran sobre las categorías de una SC de un inodo.

Indica que no existen más entradas válidas en la ACL a partir de la posición pasada como parámetro.

3.3.2.4 #define sc_for_each_cat(sc, i) for(i=sc_cat_first(sc);i!=SC_IT_ERROR;i=sc_cat_next(sc,i))

Macro que itera entre todas las categorías de una SC.

Parmetros:

sc SC que tiene las categorías sobre las que se va a iterar

i Es el puntero iterador.

Bug

el 'arreglo' de categorías podría ser un TAD por sí mismo.

Ejemplos:

iterador_sc_cat.c.

3.3.3 Documentación de las funciones

3.3.3.1 sc∗ sc_init (void)

Inicializa una SC.

Se debe tener en cuenta lo siguiente:

- Esta función prepara y reserva recursos para que un objeto sc pueda ser utilizado.
- Ningún objeto sc debe ser utilizado sin haber ejecutado antes esta función.
- Inicializa a la clase de seguridad más baja.

3.3.3.2 sc_cat_iterator sc_cat_first (struct sc ∗ *sc*)

Obtiene un iterador o selector que corresponde a la primer categoría válida dentro de las categorías de una SC.

Parmetros:

sc SC en la cual se busca la primer categoría válida.

Devuelve:

- 0 Si la cantidad de categrías válidas en *sc* es mayor que cero.
- SC_IT_ERROR Si no existe ninguna categoría válida en *sc*.

3.3.3.3 sc_cat_iterator sc_cat_next (struct sc ∗ *sc*, sc_cat_iterator *it*)

Obtiene el iterador correspondiente a la próxima categoría válida dentro de una SC con respecto a una posición de referencia pasada como parámetro.

Parmetros:

sc SC en la cual se busca la próxima categoría válida.

it Posición dentro de las categorías de *sc* donde se encuentra la categoría de referencia.

Devuelve:

- *it+1* Si hay suficientes categorías válidas en *sc*.
- SC_IT_ERROR Si no existe ninguna categoría válida luego de la referenciada por *it*.

3.3.3.4 int sc_getcat (struct sc ∗ *sc*, sc_cat_iterator *it*)

Obtiene la categoría referenciada por un iterador.

La SC debe ser válida y el iterador debe ser uno obtenido utilizando sc_cat_first() y sc_cat_next() sobre la misma SC y distinto de SC_IT_ERROR.

Parmetros:

sc SC donde se encuentra la categoría a devolver.

it Iterador correspondiente a la categoría que se desea obtener.

Devuelve:
La categoría ubicada en la posición *it* de *sc*.

Ejemplos:
iterador_sc_cat.c.

3.3.3.5 void sc_destroy (struct sc ∗ *sc*)

Libera los recursos asignados a una SC.

Destruye memoria del núcleo utilizada en una SC.

Parmetros:
sc SC que será eliminada.

3.3.3.6 int sc_getsize (struct sc ∗ *sc*)

Obtiene la cantidad de categorías válidas dentro de una SC.

Parmetros:
sc SC sobre la cual se realiza la operación.

Devuelve:
- Cantidad de categorías válidas, siempre que exista alguna.
- -1 Si no existen categorías válidas.

3.3.3.7 int sc_getlevel (struct sc ∗ *sc*)

Obtiene el nivel de una SC.

El parámetro *sc* debe estar inicializado.

Parmetros:
sc SC de la cual se obtiene el nivel.

Devuelve:
El nivel de *sc*.

3.3.3.8 void sc_setlevel (struct sc * *sc*, unsigned int *l*)

Establece el nivel de una SC.

El parametro *sc* debe estar inicializado.

Parmetros:

sc SC a la cual se asignará el nivel.

l El nuevo nivel.

3.3.3.9 void sc_clearcat (struct sc * *sc*)

Borra todas las categorías de una SC.

Parmetros:

sc SC de la cual se borrarán todas las categorías.

3.3.3.10 int sc_addcat (struct sc * *sc*, int *cat*)

Agrega una categoría las categorías de una SC.

Parmetros:

sc SC sobre la cual se realizará la inserción de una categoría.

cat Categoría a agregar a las categorías de *sc*.

Devuelve:

- SC_CAT_FULL Si no hay más lugar para agregar una categoría a las categorias de *sc*.
- SC_OK Si la operación se realizó con éxito.

Bug

Siempre agrega la categoría por más que ya esté.

3.3.3.11 int sc_compare (struct sc * *sc1*, struct sc * *sc2*)

Determina si *sc2* domina a *sc1*.

Parmetros:

sc1 Primer parámetro de la comparación.

sc2 Segundo parámetro de la comparación.

Devuelve:

- 1 Si *sc2* no domina a *sc1*.
- 0 Si *sc2* domina a *sc1*.

Bug

Si alguna de las SC a comparar no son válidas, el resultado es impredecible.

3.3.3.12 int sc_copy (struct sc ∗ *sc1*, struct sc ∗ *sc2*)

Copia el contenido de una clase de seguridad en otra.

Parmetros:

sc1 SC destino.

sc2 SC origen.

Devuelve:

- SC_OK Si la operación se realizó con éxito.
- SC_CAT_FULL Si ocurrió algún error.

3.3.3.13 int sc_usergetncat (struct sc ∗ *sc*, int ∗ *cat*, int *n*)

Carga un conjunto de categorías que provienen del espacio de usuario a una SC.

Parmetros:

sc SC a la cual se le cargarán las categorías.

cat Conjunto de categorías a cargar en *sc*.

n Cantidad de categorías válidas que estan en *cat*.

Devuelve:

- SC_CAT_FULL Si no es posible cargar *n* categorías a *sc* porque no hay al menos *n* lugares libres.
- 0 Si la operación se realizó con éxito
- Un número distinto de 0 si no se pudo concretar la operación por algún otro motivo.

3.3.3.14 int sc_userputcat (struct sc * *sc*, int * *cat*)

Carga un conjunto de categorías que provienen de una SC en espacio de kernel a un conjunto de categorías que se encuentra en espacio de usuario.

No se realiza ningún control para asegurar que la cantidad de categorías en la SC de espacio de kernel es menor o igual a la cantidad de espacio reservado en espacio de usuario para almacenarlas.

Parmetros:

sc SC de la cual se le obtendrán las categorías.

cat Conjunto de categorías a cargar en *sc*.

Devuelve:

- 0 Si la operación se realizó con éxito.
- Un número distinto de 0 si no se pudo concretar la operación por algún motivo.

3.3.4 Estructura sc

SC a nivel VFS. Se utiliza dentro del inodo del VFS.

`#include <sc.h>`

Atributos públicos

- int level
- int categories [MAX_CAT_LEN]
- int size

3.3.4.1 Descripción detallada

SC a nivel VFS. Se utiliza dentro del inodo del VFS.

Esta estructura debe ser accedida sólo a través de las funciones de su interfaz.

Ejemplos:

iterador_sc_cat.c.

3.3.4.2 Documentación de los datos miembro

3.3.4.3 int sc::level

Nivel de la SC.

3.3.4.4 int sc::categories[MAX_CAT_LEN]

Las categorías de la SC.

3.3.4.5 int sc::size

Cantidad de categorías válidas que están almacenadas en *categories*.

La documentación para esta estructura fué generada a partir del siguiente archivo:

- sc.h

3.4 TAD subjectsc

subjectsc representa las clases de acceso de los usuarios (sujetos).

Funciones de la interfaz de subjectsc

- int __init subjectsc_init (void)
  Inicializa las SC de todos los sujetos.
- int __exit subjectsc_exit (void)
  Libera las SC de todos los sujetos.
- sc ∗ subjectscget (uid_t uid)
  Obtiene la SC de un usuario.
- int subjectscadd (uid_t uid, sc ∗sc)
  Agrega una categoría a la SC de un usuario.
- sc ∗ subjectscdel (uid_t uid)
  Elimina una categoría de la SC de un usuario.

Estructuras de datos privadas (ocultas)

- sc ∗ subjectsc [SC_MAX_BUCKETS]
  Aquí se almacenan las SC de los sujetos.
- sc ∗ lowest_sc
  Apunta a la SC más baja.

3.4.1 Descripción detallada

subjectsc representa las clases de acceso de los usuarios (sujetos).

El diseño de este TAD no está completo y muchas de sus funciones son sólo prototipos; futuras versiones de Lisex completarán el diseño y re-implementarán las funciones. Notar que la interfaz del TAD es muy poco probable que se modifique por lo que puede programarse sobre este sin demasiado riesgo.

Se pretende ocultar la representación de las clases de seguridad de los usuarios, así como los algoritmos que se usan para establecer y obtener los atributos privados. En principio se construye sobre el TAD *sc*.

Se considera que los usuarios son aquellos que tienen UID menor que SC_MAX_BUCKETS. Esto se debe a que Lisex 0.0 es un prototipo; en futuras versiones se inicializará la clase de acceso de cada usuario presente en el sistema (i.e. en `/etc/passwd)`. Por lo tanto, el administrador debe crear cuentas de usuario con UID menor que SC_MAX_BUCKETS pues de lo contrario no tendrán clase de acceso lo que producirá un comportamiento errático del sistema.

Las estructuras ocultas *subjectsc* y *lowest_sc* no deben ser accedidas excepto por las funciones de la interfaz aquí documentada. Como esto no es posible imponerlo en tiempo de compilación, los programadores son los responsables de mantener esta disciplina. En caso de que no existan las funciones apropiadas para acceder los atributos privados, los programadores son libres de hacerlo como lo crean conveniente.

3.4.2 Documentación de las funciones

3.4.2.1 int __init subjectsc_init (void)

Inicializa las SC de todos los sujetos.

Se considera que los usuarios son aquellos que tienen UID menor que SC_MAX_BUCKETS. Esto se debe a que Lisex 0.0 es un prototipo; en futuras versiones se inicializará la clase de acceso de cada usuario presente en el sistema (i.e. en `/etc/passwd)`.

3.4.2.2 sc* subjectscget (uid_t *uid*)

Obtiene la SC de un usuario.

Se considera que los usuarios son aquellos que tienen UID menor que SC_MAX_BUCKETS. Esto se debe a que Lisex 0.0 es un prototipo; en futuras versiones se inicializará la clase de acceso de cada usuario presente en el sistema (i.e. en `/etc/passwd)`.

Parmetros:

uid Identificación del usuario sobre el que se quiere realizar la operación.

Devuelve:
La SC del usuario con identificación *uid*.

Bug
Debería buscar dentro del bucket (una lista enlazada?)

3.4.2.3 int subjectscadd (uid_t *uid*, sc ∗ *sc*)

Agrega una categoría a la SC de un usuario.

Devuelve:
- -EPERM si *uid* no es un usuario registrado (ver subjectsc_init()).
- 0 si la operación se realizó con éxito.

3.4.3 Documentación de las variables

3.4.3.1 sc∗ subjectsc[SC_MAX_BUCKETS] `[static]`

Aquí se almacenan las SC de los sujetos.

Esta estructura debe ser accedida sólo a través de las funciones de su interfaz.

Ver SC_MAX_BUCKETS.

3.4.3.2 sc∗ lowest_sc `[static]`

Apunta a la SC más baja.

Usado por todos los sujetos que no tengan SC asignada. Una vez que se inicializa subjectsc (subjectsc_init()) todas sus componentes apuntan a *lowest_sc,* de forma tal que cada nuevo usuario tendrá *lowest_sc* como clase de acceso hasta tanto se le asigne una.

3.5 TAD ext2_acl_entry

ext2_acl_entry representa las ternas de la forma (ID, MODE, UID_GID).

Clases

- struct ext2_acl_entry

  Representación de las ternas (ID, MODO, USR_GRP) a nivel de EXT2.

Constantes (públicas) de ext2_acl_entry

El campo ext2_acl_entry::mode sólo debría almacenar combinaciones (usando | o &) de estas constantes.

Debe considerarse que el nombre de la constante es público y su valor privado.

- #define EXT2_ACL_NULL_BIT 0

  Modo nulo (ningún permiso) en la ACL.

- #define EXT2_ACL_EXEC_BIT 1

  Permiso de ejecutabilidad en la ACL.

- #define EXT2_ACL_READ_BIT 2

  Permiso de lectura en la ACL.

- #define EXT2_ACL_WRITE_BIT 4

  Permiso de escritura en la ACL.

- #define EXT2_ACL_OWNER_BIT 8

  Modo dueño en la ACL.

- #define EXT2_ACL_USR_BIT 16

  Diferenciación entre usuarios y grupos en la ACL.

3.5.1 Descripción detallada

ext2_acl_entry representa las ternas de la forma (ID, MODE, UID_GID).

Este TAD representa las ternas (ID, MODO, USR_GRP) que se usarán en las ACL a nivel de EXT2. En cada *ext2_inode* se incluirá un "objeto" *ext2_acl_entry* por cada UID o GID que tenga algún permiso sobre el archivo, directorio, dispositivo, etc. que representa el inodo.

Por hora el TAD no tiene funciones en su interfaz y sólo oculta la representación de los permisos básicos por medio de constantes simbólicas.

3.5.2 Estructura ext2_acl_entry

Representación de las ternas (ID, MODO, USR_GRP) a nivel de EXT2.

```
#include <ext2_fs.h>
```

Atributos públicos

- __u32 id
- __u8 mode

3.5.2.1 Descripción detallada

Representación de las ternas (ID, MODO, USR_GRP) a nivel de EXT2.

En cada ext2_inode se incluirá un "objeto" *ext2_acl_entry* por cada UID o GID que tenga algún permiso sobre el archivo, directorio, dispositivo, etc. que representa el inodo.

Notar la diferencia de representación con acl_entry a nivel de VFS.

3.5.2.2 Documentación de los datos miembro

3.5.2.3 __u32 ext2_acl_entry::id

Representa un UID o un GID; depende del valor del campo *mode*.

3.5.2.4 __u8 ext2_acl_entry::mode

Contiene los permisos de *id*. Entre los permisos se codifica si *id* es un UID (tiene EXT2_ACL_USR_BIT) o un GID (no tiene EXT2_ACL_USR_BIT).

La documentación para esta estructura fué generada a partir del siguiente archivo:

- ext2_fs.h

3.6 Estructura ext2_inode

Representa el inodo físico de un sistema de archivos EXT2. Sólo se comentan los cambios efectuados por Lisex.

`#include <ext2_fs.h>`

Atributos públicos

- ext2_acl_entry ext2_acl [EXT2_ACL_LEN]

  Lista de control de acceso (ver EXT2_ACL_LEN).

- __u32 sc_level

  Nivel de la clase de seguridad.

- __u32 categories [EXT2_CAT_LEN]

  Conjunto de categorías de la clase de seguridad (ver EXT2_CAT_LEN).

- __u32 cat_len

  Cantidad de categorías que están siendo utilizadas en el arreglo (las primeras cat_len *son las usadas).*

3.6.1 Descripción detallada

Representa el inodo físico de un sistema de archivos EXT2. Sólo se comentan los cambios efectuados por Lisex.

La documentación para esta estructura fué generada a partir del siguiente archivo:

- ext2_fs.h

Capítulo 4

Constantes o parmetros de compilación

4.1 Constantes o parámetros de compilación definidos en Lisex 0.0

Definiciones

- #define __NR_aclstat 226

  Número de la llamada al sistema sys_aclstat().

- #define __NR_aclfstat 227

  Número de la llamada al sistema sys_aclfstat().

- #define __NR_acladd 228

  Número de la llamada al sistema sys_acladd().

- #define __NR_acldel 229

  Número de la llamada al sistema sys_acldel().

- #define __NR_chsubsc 230

  Número de la llamada al sistema sys_chsubsc().

- #define __NR_sscstat 231

  Número de la llamada al sistema sys_sscstat().

- #define __NR_chobjsc 232

  Número de la llamada al sistema sys_chobjsc().

- #define __NR_ownerclose 233

  Número de la llamada al sistema sys_ownerclose().

- #define __NR_oscstat 234

  Número de la llamada al sistema sys_oscstat().

- #define EXT2_ACL_LEN 8

  Cantidad máxima de entradas posibles a la ACL de un inodo EXT2.

- #define EXT2_CAT_LEN 8

  Cantidad máxima de categorías posibles a la SC de un inodo a nivel EXT2.

- #define MAX_ACL_LEN 8

  Cantidad máxima de entradas posibles a la ACL de un inodo del VFS.

- #define ALLGRP 666

  Identificación del grupo AllGrp.

- #define ROOTID 0

  Identificación del usuario Root.

- #define ROOTGRP 0

  Identificación del grupo Root.

- #define LISEXDEBUG(x)

  Maneja los mensajes del núcleo.

- #define SC_MAX_BUCKETS 600

  Cantidad máxima de SC de sujetos.

- #define SUBSC_HASH(x) x

  Función de dispersión; por ahora un arreglo completo.

4.1.1 Documentación de las definiciones

4.1.1.1 #define EXT2_ACL_LEN 8

Cantidad máxima de entradas posibles a la ACL de un inodo EXT2.

La cantidad máxima de entradas a la ACL de un inodo del EXT2 no puede ser mayor a la cantidad máxima de entradas a la ACL de un inodo del VFS. En otras palabras, EXT2_ACL_LEN no puede ser mayor a MAX_ACL_LEN que esta definido en fs.h

4.1.1.2 #define EXT2_CAT_LEN 8

Cantidad máxima de categorías posibles a la SC de un inodo a nivel EXT2.

Esta definición sirve para evitar que se defina EXT2_CAT_LEN con un tamaño mayor a MAX_CAT_LEN que esta definido en sc.h, es decir, la cantidad máxima de categorías almacenadas en un inodo del EXT2 no puede ser mayor a la cantidad máxima de categorías con las que trabaja el núcleo en general. Esto debe ir en el superbloque del EXT2 (Lisex FS) y ser dinámicos.

4.1.1.3 #define LISEXDEBUG(x)

Maneja los mensajes del núcleo.

Se utiliza durante la fase de corrección para manejar los de mensajes de las porciones del núcleo que fueron modificadas o agregadas por Lisex.

4.1.1.4 #define SC_MAX_BUCKETS 600

Cantidad máxima de SC de sujetos.

Se considera que los usuarios son aquellos que tienen UID menor que *SC_MAX_BUCKETS*. Esto se debe a que Lisex 0.0 es un prototipo.

Capítulo 5

Archivos

5.1 Referencia del Archivo acl.c

Definición de las funciones relativas a las listas de control de acceso.

```
#include <linux/errno.h>
#include <linux/fs_acl.h>
#include <linux/stat.h>
#include <linux/fs.h>
#include <linux/mm.h>
#include <asm/uaccess.h>
#include <linux/sched.h>
#include <linux/smp_lock.h>
```

Funciones privadas de vfs_acl

- #define ACL_INUSE(e) (e.mode != ACL_NULL_BITS)

  Determina si la entrada e *de la ACL esta o no en uso.*

- int search_free_slot (struct vfs_acl ∗vfs_acl)

  Busca una entrada libre dentro de la ACL.

- int search_id (struct vfs_acl ∗vfs_acl, int id, int type)

  Busca la entrada en la ACL que corresponde a al usuario o grupo id.

Funciones de la interfaz de vfs_acl

- umode_t acl_calc_xmode (struct vfs_acl ∗vfs_acl, uid_t uid, gid_t gid)

  Calcula el modo tradicional de un inodo a partir de su ACL.

- int xmode_to_acl (int xmode, int owner)

  Calcula los permisos compatibles con el modo.

- void acl_aclinit (struct vfs_acl ∗vfs_acl)

  Inicializa una ACL.

- void acl_acldestroy (struct vfs_acl ∗vfs_acl)

  Aun no está implementada, pero debería liberar la ACL que recibe como parámetro.

- int acl_setmode (struct vfs_acl ∗vfs_acl, int id, int mode, int type)

  Establece el modo de un usuario o grupo en la ACL.

- int acl_getmode (struct vfs_acl ∗vfs_acl, int id, int type)

  Obtiene el modo de un usuario/grupo.

- int acl_getsize (struct vfs_acl ∗vfs_acl)

  Devuelve la cantidad de entradas válidas que existen en la ACL.

Iterador para las acl_entry de un objeto vfs_acl

También forma parte de la interfaz de vfs_acl.

Ver acl_for_each para un ejemplo.

- acl_iterator acl_first (struct vfs_acl ∗vfs_acl)

  Busca la primer entrada válida dentro de la ACL.

- acl_iterator acl_next (struct vfs_acl ∗vfs_acl, acl_iterator it)

  Busca la próxima entrada válida en la ACL a partir de una posición que se pasa como parámetro.

- acl_entry acl_getentry (struct vfs_acl ∗vfs_acl, acl_iterator it)

  Devuelve una entrada de la ACL.

Funciones

- asmlinkage long sys_acladd (char ∗filename, int id, int mode, int type)

  Llamada al sistema acladd. Agrega permisos a un usuario o grupo en la ACL.

- asmlinkage long sys_acldel (char ∗filename, int id, int mode, int type)
  Llamada al sistema acldel. Quita permisos a un usuario o un grupo de una ACL.

5.1.1 Descripción detallada

Definición de las funciones relativas a las listas de control de acceso.

5.2 Referencia del Archivo attr.c

Se cambió la forma en que se verifican los permisos: se eliminaron las capacidades y se introdujo la idea de los permisos por ACL.

```
#include <linux/sched.h>
#include <linux/mm.h>
#include <linux/string.h>
#include <linux/smp_lock.h>
#include <linux/dnotify.h>
#include <linux/fcntl.h>
#include <linux/quotaops.h>
```

Funciones

- int inode_change_ok (struct inode ∗inode, struct iattr ∗attr)
  Función indirectamente usada por sys_fchmod(), sys_truncate() y sys_truncate64().
- int inode_setattr (struct inode ∗inode, struct iattr ∗attr)
  Función indirectamente usada por sys_fchmod(), sys_truncate() y sys_truncate64().

5.2.1 Descripción detallada

Se cambió la forma en que se verifican los permisos: se eliminaron las capacidades y se introdujo la idea de los permisos por ACL.

5.3 Referencia del Archivo exec.c

Contiene las llamadas do_execve() y sys_uselib() que fueron modificadas.

```
#include <linux/config.h>
#include <linux/slab.h>
#include <linux/file.h>
#include <linux/mman.h>
#include <linux/a.out.h>
#include <linux/stat.h>
#include <linux/fcntl.h>
#include <linux/smp_lock.h>
#include <linux/init.h>
#include <linux/pagemap.h>
#include <linux/highmem.h>
#include <linux/spinlock.h>
#include <linux/personality.h>
#include <linux/module.h>
#include <asm/uaccess.h>
#include <asm/pgalloc.h>
#include <asm/mmu_context.h>
```

Funciones

- asmlinkage long sys_uselib (const char ∗library)

  Llamada al sistema uselib. No fue modificada pero usa may_open() y permission().

- file ∗ open_exec (const char ∗name)

  No fue modificada pero usa permission().

- int prepare_binprm (struct linux_binprm ∗bprm)

  Función usada por do_execve().

- int do_execve (char ∗filename, char ∗∗argv, char ∗∗envp, struct pt_regs ∗regs)

  Llamada al sistema execve. No fue modificada pero usa, prepare_binprm() y open_exec() la que a su vez usa permission().

5.3.1 Descripción detallada

Contiene las llamadas do_execve() y sys_uselib() que fueron modificadas.

5.4 Referencia del Archivo ext2_fs.h

Constantes y estructuras de ext2.

`#include <linux/types.h>`

`#include <linux/fs_acl.h>`

Clases

- struct ext2_acl_entry

  Representación de las ternas (ID, MODO, USR_GRP) a nivel de EXT2.

- struct ext2_inode

  Representa el inodo físico de un sistema de archivos EXT2. Sólo se comentan los cambios efectuados por Lisex.

Funciones para transformar el modo EXT2 <–> VFS

Las usan ext2_read_inode() y ext2_update_inode()

- #define ext2tovfs_mode(mode)

  Macro para transformar la representación del modo.

- #define vfstoext2_mode(mode, t)

  Macro para transformar la representación del modo.

Constantes (públicas) de ext2_acl_entry

El campo ext2_acl_entry::mode sólo debría almacenar combinaciones (usando | o &) de estas constantes.

Debe considerarse que el nombre de la constante es público y su valor privado.

- #define EXT2_ACL_NULL_BIT 0

  Modo nulo (ningún permiso) en la ACL.

- #define EXT2_ACL_EXEC_BIT 1

  Permiso de ejecutabilidad en la ACL.

- #define EXT2_ACL_READ_BIT 2

  Permiso de lectura en la ACL.

- #define EXT2_ACL_WRITE_BIT 4

  Permiso de escritura en la ACL.

- #define EXT2_ACL_OWNER_BIT 8

  Modo dueño en la ACL.

- #define EXT2_ACL_USR_BIT 16

  Diferenciación entre usuarios y grupos en la ACL.

Definiciones

- #define EXT2_ACL_LEN 8

  Cantidad máxima de entradas posibles a la ACL de un inodo EXT2.

- #define EXT2_CAT_LEN 8

  Cantidad máxima de categorías posibles a la SC de un inodo a nivel EXT2.

Funciones

- inode ∗ ext2_new_inode (const struct inode ∗, int)

  Se agregó código para inicializar idoneamente las ACL y la SC del nuevo inodo.

- void ext2_read_inode (struct inode ∗)

  Lee un inodo físico del superbloque de EXT2 y lo ajusta a un inodo del vfs.

5.4.1 Descripción detallada

Constantes y estructuras de ext2.

Se agregaron las definiciones y estructuras que se utilizarán para el manejo de las listas de control de acceso (ACL) y las clases de seguridad (SC) de los inodos dl ext2.

En la estructura ext2_inode se agregaron los campos para el manejo de SC y ACL. Estos campos son:

1. ext2_acl Un arreglo de EXT2_ACL_LEN para almancenar la ACL del inodo.
2. sc_level Un entero para guardar el nivel de la SC del inodo.
3. categories Un arreglo de EXT2_CAT_LEN enteros para almacenar las categorías de la SC del inodo.
4. cat_len Un entero para indicar la cantidad de categorías válidas ingresadas en *categories*.

5.4.2 Documentación de las definiciones

5.4.2.1 #define test_opt(sb, opt)

Valor:

```
((sb)->u.ext2_sb.s_mount_opt & \
                     EXT2_MOUNT_##opt)
```

5.4.2.2 #define EXT2_FEATURE_RO_COMPAT_SUPP

Valor:

```
(EXT2_FEATURE_RO_COMPAT_SPARSE_SUPER| \
                     EXT2_FEATURE_RO_COMPAT_LARGE_FILE| \
                     EXT2_FEATURE_RO_COMPAT_BTREE_DIR)
```

5.4.2.3 #define EXT2_DIR_REC_LEN(name_len)

Valor:

```
(((name_len) + 8 + EXT2_DIR_ROUND) & \
                     ~EXT2_DIR_ROUND)
```

5.4.3 Documentación de las funciones

5.4.3.1 void ext2_read_inode (struct inode * *inode*)

Lee un inodo físico del superbloque de EXT2 y lo ajusta a un inodo del vfs.

Sólo se modificó lo necesario para que levante los nuevos miembros del ext2_inode y los ponga en los nuevos miembros del (vfs_)inode.

5.5 Referencia del Archivo fs.h

Definiciones, estructuras y prototipos de funciones que manejan el VFS. En particular se encuentra la estructura inode donde se realizaron los cambios.

```
#include <linux/config.h>
#include <linux/linkage.h>
#include <linux/limits.h>
#include <linux/wait.h>
#include <linux/types.h>
#include <linux/vfs.h>
#include <linux/net.h>
#include <linux/kdev_t.h>
#include <linux/ioctl.h>
#include <linux/list.h>
#include <linux/dcache.h>
#include <linux/stat.h>
#include <linux/cache.h>
#include <linux/stddef.h>
#include <linux/string.h>
#include <asm/atomic.h>
#include <asm/bitops.h>
#include <linux/fs_acl.h>
#include <linux/sc.h>
#include <asm/semaphore.h>
#include <asm/byteorder.h>
#include <linux/pipe_fs_i.h>
#include <linux/minix_fs_i.h>
#include <linux/ext2_fs_i.h>
#include <linux/hpfs_fs_i.h>
#include <linux/ntfs_fs_i.h>
#include <linux/msdos_fs_i.h>
#include <linux/umsdos_fs_i.h>
#include <linux/iso_fs_i.h>
#include <linux/nfs_fs_i.h>
#include <linux/sysv_fs_i.h>
#include <linux/affs_fs_i.h>
#include <linux/ufs_fs_i.h>
#include <linux/efs_fs_i.h>
```

```
#include <linux/coda_fs_i.h>
#include <linux/romfs_fs_i.h>
#include <linux/shmem_fs.h>
#include <linux/smb_fs_i.h>
#include <linux/hfs_fs_i.h>
#include <linux/adfs_fs_i.h>
#include <linux/qnx4_fs_i.h>
#include <linux/reiserfs_fs_i.h>
#include <linux/bfs_fs_i.h>
#include <linux/udf_fs_i.h>
#include <linux/ncp_fs_i.h>
#include <linux/proc_fs_i.h>
#include <linux/usbdev_fs_i.h>
#include <linux/jffs2_fs_i.h>
#include <linux/cramfs_fs_sb.h>
#include <linux/quota.h>
#include <linux/mount.h>
#include <linux/fcntl.h>
#include <linux/minix_fs_sb.h>
#include <linux/ext2_fs_sb.h>
#include <linux/hpfs_fs_sb.h>
#include <linux/ntfs_fs_sb.h>
#include <linux/msdos_fs_sb.h>
#include <linux/iso_fs_sb.h>
#include <linux/nfs_fs_sb.h>
#include <linux/sysv_fs_sb.h>
#include <linux/affs_fs_sb.h>
#include <linux/ufs_fs_sb.h>
#include <linux/efs_fs_sb.h>
#include <linux/romfs_fs_sb.h>
#include <linux/smb_fs_sb.h>
#include <linux/hfs_fs_sb.h>
#include <linux/adfs_fs_sb.h>
#include <linux/qnx4_fs_sb.h>
```

```
#include <linux/reiserfs_fs_sb.h>
#include <linux/bfs_fs_sb.h>
#include <linux/udf_fs_sb.h>
#include <linux/ncp_fs_sb.h>
#include <linux/usbdev_fs_sb.h>
#include <linux/jffs2_fs_sb.h>
```

Funciones

- int mac_permission (struct inode ∗inode, int mode)

  Determina si se tiene permiso para acceder a inode *según la clase de acceso. Implementa seguridad MAC y MLS.*

- asmlinkage long sys_oscstat (char ∗filename, int ∗level, int ∗categories, int len)

  Llamada al sistema oscstat. Retorna la clase de acceso de un archivo.

- asmlinkage long sys_chobjsc (char ∗filename, int level, int ∗categories, int size)

  Llamada al sistema chobjsc. Cambia la SC de un archivo.

- asmlinkage long sys_open (const char ∗, int, int)

  Llamada al sistema open. No fue modificada pero usa (indirectamente) may_open().

- asmlinkage long sys_close (unsigned int)

  Esta llamada no fue modificada. Se la incluye por completitud. Está especificada en el modelo.

- int is_open (struct inode ∗inode)

  Determina si un archivo está o no abierto en el sistema.

- int permission (struct inode ∗, int)

  Es sinónimo de vfs_permission().

- int vfs_permission (struct inode ∗, int)

  Consulta permisos.

- file ∗ open_exec (const char ∗)

  No fue modificada pero usa permission().

- void clear_inode (struct inode ∗)

  Limpia un inodo.

- int inode_change_ok (struct inode ∗, struct iattr ∗)

  Función indirectamente usada por sys_fchmod(), sys_truncate() y sys_truncate64().

- int inode_setattr (struct inode ∗, struct iattr ∗)

  Función indirectamente usada por sys_fchmod(), sys_truncate() y sys_truncate64().

5.5.1 Descripción detallada

Definiciones, estructuras y prototipos de funciones que manejan el VFS. En particular se encuentra la estructura inode donde se realizaron los cambios.

- Se agregó la estructura vfs_acl definida en fs_acl.h dentro del inodo del VFS.
- Se agregó la estructura sc definida en sc.h dentro del inodo del VFS.
- Se agregaron los prototipos de la función mac_permission() y de las llamadas al sistema oscstat() y chobjsc().
- Se quitaron las definiciones MAY_XXXX que se usaban para llamar a permission() (ahora se usan ACL_XXX_BIT).

5.5.2 Documentación de las definiciones

5.5.2.1 #define MS_RMT_MASK

Valor:

```
(MS_RDONLY|MS_SYNCHRONOUS|MS_MANDLOCK|MS_NOATIME|\
             MS_NODIRATIME)
```

5.5.2.2 #define DECLARE_FSTYPE(var, type, read, flags)

Valor:

```
struct file_system_type var = { \
    name:        type, \
    read_super: read, \
    fs_flags:   flags, \
    owner:      THIS_MODULE, \
}
```

5.5.2.3 #define fops_get(fops)

Valor:

```
(((fops) && (fops)->owner)  \
        ? ( try_inc_mod_count((fops)->owner) ? (fops) : NULL ) \
        : (fops))
```

5.5.2.4 #define fops_put(fops)

Valor:

```
do {    \
    if ((fops) && (fops)->owner) \
        __MOD_DEC_USE_COUNT((fops)->owner); \
} while(0)
```

5.5.3 Documentación de las funciones

5.5.3.1 int mac_permission (struct inode ∗ *inode*, int *mode*)

Determina si se tiene permiso para acceder a *inode* según la clase de acceso. Implementa seguridad MAC y MLS.

Bug

Hay que bloquear subjectsc
Mal programada, mal indentada, mal concevida. MAL!

Parmetros:

inode Objeto al cual se quiere acceder

mode Los permisos que se requieren

Devuelve:

- 0 Si el usuario que realiza la operación verifica seguridad simple y el proceso verifica confinamiento para el mode solicitado y sobre el inodo inode.
- 1 En caso contrario.

5.5.3.2 asmlinkage long sys_oscstat (char ∗ *filename*, int ∗ *level*, int ∗ *categories*, int *size*)

Llamada al sistema oscstat. Retorna la clase de acceso de un archivo.

Precondicin:

- La clase de acceso del usuario que realiza la operacion domina a la clase de seguridad del archivo.
- size debe ser el tamaño reservado en el buffer categories y debe ser al menos la cantidad de entradas válidas que hay en la clase de acceso de filename.

Parmetros:

filename Archivo que contiene la clase de seguridad que se utilizará para cargar los datos.

level Donde se cargará el nivel de la clase de acceso de filename.

categories Donde se cargarán las categorías de la clase de acceso de filename.

size Cantidad de categorías que pide cargar en categories.

Postcondicin:

- En el párametro level está el nivel de la clase de seguridad de filename.
- En el párametro categories están las categorías de la clase de seguridad de filename.

Devuelve:

- Un número distinto de cero si filename no es un nombre válido.
- -EACCES Si no se cumple la precondición.
- n Donde n es la cantidad de entradas válidas que hay en la clase de acceso de filename, si len es 0.
- -EINVAL Si size es menor que la cantidad de entradas válidas que hay en la clase de acceso de filename.
- m Si la operación se realizó con éxito. m es la cantidad de entradas válidas que se cargaron en categories.

5.5.3.3 asmlinkage long sys_chobjsc (char ∗ *filename*, int *level*, int ∗ *categories*, int *size*)

Llamada al sistema chobjsc. Cambia la SC de un archivo.

Cambia la SC de un archivo, con los datos que recibe como parámetro.

Precondicin:

El usuario que realiza la operacin debe ser un administrador MAC. El archivo no debe estar abierto en el sistema.

Parmetros:

filename Archivo al cual se le cambia la SC.

level Nivel que se cargará en la SC del archivo filename.

categories Conjunto de categorías que se cargarán en la SC del archivo filename.

size Cantidad de categorías que cargarán en la SC del archivo filename.

Devuelve:

- Un número distinto de cero si filename no es un nombre válido.
- -EROFS Si el archivo filename es read-only
- -EPERM Si el archivo filename es inmutable o append.????
- -EACCES Si el usuario que realiza la operación no es un administrador MAC.
- -EBUSY Si el archivo filename está abierto en el sistema.
- 0 Si la operación se realizó con éxito.

5.5.3.4 int is_open (struct inode ∗ *inode*)

Determina si un archivo está o no abierto en el sistema.

Parmetros:

inode Puntero a un inodo del VFS, se recorrerán todos los archivos abiertos del sistema para determinar si inodo es o no alguno de ellos.

Devuelve:

- 0 Si *inode* no es el inodo de alguno de los archivos abiertos del sistema.
- 1 Si *inode* es el inodo de un archivo abierto del sistema.

5.5.3.5 int vfs_permission (struct inode ∗ *inode*, int *mask*)

Consulta permisos.

Determina si los permisos que se encuentran en una máscara que se pasa como párametro se encuentran en la ACL de un inodo.

Parmetros:

inode Puntero al inodo al cual se le quieren consultar los permisos.

mask Permisos que se van a comparar con los permisos resultantes de la ACL de *inode.* Estos pueden ser:

- ACL_READ_BIT,
- ACL_WRITE_BIT,
- ACL_OWNER_BIT,

- ACL_EXEC_BIT,

y combinaciones. Se ha cambiado la utilización de permissions() para que se usen los nuevos parámetros.

Devuelve:

- -EROFS Si *mask* contiene permiso de escritura e *inode* está en un sistema de archivos de sólo lectura, y el modo UNIX de *inode* indica que: o es un registro, o un directorio o un link.
- -EACCES Si *mask* contiene permiso de escritura e *inode* es inmutable.
- 0 Si todos los permisos que contiene mask se encuentran en la ACL de *inode*. En caso de que *inode* sea un inodo de un directorio no se tienen en cuenta los permisos de ejecutabilidad.

5.5.3.6 void clear_inode (struct inode ∗ *inode*)

Limpia un inodo.

Es llamada por el sistema de archivos para indicar que el inodo ya no es útil.

Parmetros:

inode inodo a limpiar

5.6 Referencia del Archivo fs_acl.h

Definiciones, estructuras y prototipos de funciones que manejan las ACL de los inodos del VFS.

```
#include <linux/types.h>
```

Clases

- struct acl_entry

  Entrada de la ACL a nivel VFS. Se utiliza dentro del la estructura vfs_acl.

- struct vfs_acl

  ACL a nivel VFS. Se utiliza dentro del inodo del VFS.

Constantes (públicas) de vfs_acl

Debe considerarse que el nombre de la constante es público y su valor privado.

- #define ACL_OK 0

  Retorno de funciones que manejan las ACL.

Constantes (públicas) de acl_entry

Debe considerarse que el nombre de la constante es público y su valor privado.

- #define ACL_NULL_BITS 0

  Para representar el modo nulo (ningun permiso) en la ACL.

- #define ACL_EXEC_BIT 1

  Para representar el permiso de ejecutabilidad en la ACL.

- #define ACL_WRITE_BIT 2

  Para representar el permiso de escritura en la ACL.

- #define ACL_READ_BIT 4

  Para representar el permiso de lectura en la ACL.

- #define ACL_OWNER_BIT 8

  Para representar el permiso de propiedad (dueño) en la ACL.

- #define ACL_RWXO_BIT ACL_EXEC_BIT|ACL_READ_BIT|ACL_WRITE_-BIT|ACL_OWNER_BIT

  Para representar el modo que tiene todos los permisos y también es dueño.

- #define ACL_GRP 0

  Para indicar que se trata de los permisos de un grupo en una entrada de la ACL.

- #define ACL_USR 1

  Para indicar que se trata de los permisos de un usuario en una entrada de la ACL.

Funciones de la interfaz de acl_entry

- #define ACL_TYPE(e) e.type

  Devuelve el campo type *de una entrada de ACL.*

- #define ACL_MODE(e) e.mode

  Devuelve el campo mode *de una entrada ACL.*

- #define ACL_ID(e) ((ACL_TYPE(e)==ACL_USR)?e.id.uid:e.id.gid)

  Devuelve el el campo id *de una entrada ACL.*

Iterador para las acl_entry de un objeto vfs_acl

También forma parte de la interfaz de vfs_acl.

Ver acl_for_each para un ejemplo.

- #define ACL_IT_ERROR -1

  Retorno de las funciones que iteran sobre la ACL de un inodo.

- #define acl_for_each(acl, i) for(i=acl_first(acl);i!=ACL_IT_ERROR;i=acl_-next(acl,i))

  Macro que itera entre todas las entradas válidas de una ACL.

Definiciones

- #define MAX_ACL_LEN 8

  Cantidad máxima de entradas posibles a la ACL de un inodo del VFS.

- #define ALLGRP 666

  Identificación del grupo AllGrp.

- #define ROOTID 0

  Identificación del usuario Root.

- #define ROOTGRP 0

  Identificación del grupo Root.

Tipos definidos

- typedef int acl_iterator

  Renombre del tipo int, utilizado como iterador dentro de una ACL.

Funciones

- void acl_aclinit (struct vfs_acl ∗vfs_acl)

  Inicializa una ACL.

- void acl_acldestroy (struct vfs_acl ∗vfs_acl)

  Aun no está implementada, pero debería liberar la ACL que recibe como parámetro.

- int acl_setmode (struct vfs_acl ∗vfs_acl, int id, int mode, int type)

  Establece el modo de un usuario o grupo en la ACL.

- int acl_getsize (struct vfs_acl ∗vfs_acl)

  Devuelve la cantidad de entradas válidas que existen en la ACL.

- int acl_getmode (struct vfs_acl ∗vfs_acl, int id, int type)

  Obtiene el modo de un usuario/grupo.

- int xmode_to_acl (int xmode, int owner)

  Calcula los permisos compatibles con el modo.

- umode_t acl_calc_xmode (struct vfs_acl ∗vfs_acl, uid_t uid, gid_t gid)

  Calcula el modo tradicional de un inodo a partir de su ACL.

- acl_iterator acl_first (struct vfs_acl ∗vfs_acl)

  Busca la primer entrada válida dentro de la ACL.

- acl_iterator acl_next (struct vfs_acl ∗vfs_acl, acl_iterator it)

  Busca la próxima entrada válida en la ACL a partir de una posición que se pasa como parámetro.

- acl_entry acl_getentry (struct vfs_acl ∗vfs_acl, acl_iterator it)

  Devuelve una entrada de la ACL.

5.6.1 Descripción detallada

Definiciones, estructuras y prototipos de funciones que manejan las ACL de los inodos del VFS.

5.6.2 Documentación de las funciones

5.6.2.1 void acl_aclinit (struct vfs_acl ∗ *vfs_acl*)

Inicializa una ACL.

Parmetros:

vfs_acl Es la ACL a inicializar, se establece un valor nulo para los modos de las entradas de *vfs_acl* y se asigna 0 en size::vfs_acl.

5.6.2.2 int acl_setmode (struct vfs_acl * *vfs_acl*, int *id*, int *mode*, int *type*)

Establece el modo de un usuario o grupo en la ACL.

En caso de que el usuario/grupo exista en la ACL, cambia su modo siempre que el modo pasado como parámetro sea un modo válido. Si no existe tal usuario/grupo y hay lugar en la ACL, lo agrega.

Parmetros:

vfs_acl ACL sobre la que se realizará la operación.

id Identificación del usuario o grupo que se busca dentro de la ACL.

mode Nuevo modo que deberá quedar cargado en la ACL para el usuario/grupo con identificacion *id* luego de realizar la operación, siempre que esto sea posible.

type Indica si *id* representa la identificación de un usuario o de un grupo.

Devuelve:

- ACL_OK En caso de que el usuario o grupo ya exista en *vfs_acl* y *mode* sea un modo válido.
- ACL_OK En caso de que el usuario o grupo no se encuentre en *vfs_acl* pero que exista un lugar vacío y *mode* sea un modo válido.
- ACL_OK En caso de que el usuario o grupo no se encuentre en *vfs_acl* pero que el modo sea el modo nulo.
- ENOMEM En caso de que el usuario o grupo no se encuentre en *vfs_acl* y esta esté llena.

5.6.2.3 int acl_getsize (struct vfs_acl * *vfs_acl*) `[inline]`

Devuelve la cantidad de entradas válidas que existen en la ACL.

Parmetros:

vfs_acl ACL de la cual se obtiene la cantidad de entradas válidas almacenadas.

Devuelve:

La cantidad de entradas válidas dentro de la ACL.

5.6.2.4 int acl_getmode (struct vfs_acl * *vfs_acl*, int *id*, int *type*)

Obtiene el modo de un usuario/grupo.

Parmetros:

vfs_acl ACL sobre la cual se realizará la operación.

id Identifiación del usuario del cual se desea obtener su modo.

type Indica si *id* se refiere a la identificación de un grupo o de un usuario.

Devuelve:

- El modo del usuario/grupo con identificación *id* en la ACL *vfs_acl*.

5.6.2.5 int xmode_to_acl (int *xmode*, int *owner*)

Calcula los permisos compatibles con el modo.

Calcula los permisos compatibles con el modo utilizado en una ACL, a partir de un modo tradicional y de un entero que indica si son permisos de dueño o no.

Parmetros:

xmode Modo tradicional.

owner Los permisos que se calculan corresponden a un dueño o no.

Devuelve:

El modo en formato ACL.

5.6.2.6 umode_t acl_calc_xmode (struct vfs_acl ∗ *vfs_acl*, uid_t *uid*, gid_t *gid*)

Calcula el modo tradicional de un inodo a partir de su ACL.

Parmetros:

vfs_acl ACL sobre la cual se calculará el modo tradicional.

uid Identificación del usuario dueño tradicional del inodo.

gid Identificación del grupo dueño tradicional del inodo.

Devuelve:

Un modo que se calcula a partir los permisos que tienen:

- el usuario dueno tradicional,
- el grupo tradicional y AllGrup;

dentro de *vfs_acl*.

5.6.2.7 acl_iterator acl_first (struct vfs_acl ∗ *vfs_acl*)

Busca la primer entrada válida dentro de la ACL.

Parmetros:

vfs_acl ACL donde se realizará la búsqueda.

Devuelve:

- La posición dentro de *vfs_acl* donde se encontró la primer entrada válida.
- ACL_IT_ERROR En caso de que *vfs_acl* este vacia.

5.6.2.8 acl_iterator acl_next (struct vfs_acl * *vfs_acl*, acl_iterator *it*)

Busca la próxima entrada válida en la ACL a partir de una posición que se pasa como parámetro.

Parmetros:

vfs_acl ACL donde se realizará la búsqueda.

it Posición a partir de la cual se realizará la búsqueda, previamente obtenido con acl_first() o acl_next().

Devuelve:

- La posición dentro de *vfs_acl* donde se encontró la práxima entrada válida.
- ACL_IT_ERROR En caso de que *it* sea la posición de la última entrada válida dentro de *vfs_acl*.

5.6.2.9 struct acl_entry acl_getentry (struct vfs_acl * *vfs_acl*, acl_iterator *it*)

Devuelve una entrada de la ACL.

Parmetros:

vfs_acl ACL a la que pertenece la entrada que se devuelve.

it Posición, dentro de *vfs_acl,* de la entrada que se quiere obtener.

Devuelve:

La entrada ubicada en la posición *it* de *vfs_acl*.

5.7 Referencia del Archivo ialloc.c

Se modificó levemente la función ext2_new_inode().

```
#include <linux/config.h>
#include <linux/fs.h>
#include <linux/ext2_fs.h>
#include <linux/locks.h>
#include <linux/quotaops.h>
```

Funciones

- inode * ext2_new_inode (const struct inode *dir, int mode)

  Se agregó código para inicializar idoneamente las ACL y la SC del nuevo inodo.

5.7.1 Descripción detallada

Se modificó levemente la función ext2_new_inode().

5.8 Referencia del Archivo inode.c

Funciones encargadas de manejar los inodos físicos de ext2. También están las que se encargan de cargar el inodo físico al inodo del VFS y viceversa.

```
#include <linux/fs.h>
#include <linux/ext2_fs.h>
#include <linux/locks.h>
#include <linux/smp_lock.h>
#include <linux/sched.h>
#include <linux/highuid.h>
#include <linux/quotaops.h>
#include <linux/module.h>
```

Funciones

- int ext2_update_inode (struct inode *inode, int do_sync)

  Graba en un inodo físico del superbloque de EXT2 los datos de un inodo del VFS.

- void ext2_read_inode (struct inode *inode)

  Lee un inodo físico del superbloque de EXT2 y lo ajusta a un inodo del vfs.

5.8.1 Descripción detallada

Funciones encargadas de manejar los inodos físicos de ext2. También están las que se encargan de cargar el inodo físico al inodo del VFS y viceversa.

Se han realizado dos modificaciones:

1. En la función ext2_read_inode() se agregó código que proyecta las estructuras ACL y SC del inodo del ext2 en las estructuras ACL y SC del inodo del VFS, respectivamente.

2. En la función ext2_update_inode() se agregó código que proyecta las estructuras ACL y SC del inodo del VFS en las estructuras ACL y SC del inodo del ext2, respectivamente.

5.8.2 Documentación de las variables

5.8.2.1 struct address_space_operations ext2_aops

Valor inicial:

```
 {
    readpage: ext2_readpage,
    writepage: ext2_writepage,
    sync_page: block_sync_page,
    prepare_write: ext2_prepare_write,
    commit_write: generic_commit_write,
    bmap: ext2_bmap
}
```

5.9 Referencia del Archivo inode.c

Funciones que manejan los inodos que se encuentran en el VFS.

```
#include <linux/config.h>
#include <linux/fs.h>
#include <linux/string.h>
#include <linux/mm.h>
#include <linux/dcache.h>
#include <linux/init.h>
#include <linux/quotaops.h>
#include <linux/slab.h>
#include <linux/cache.h>
#include <linux/swap.h>
#include <linux/swapctl.h>
#include <linux/prefetch.h>
```

`#include <linux/locks.h>`

`#include <linux/fs_acl.h>`

`#include <linux/sc.h>`

Funciones

- void clear_inode (struct inode ∗inode)

  Limpia un inodo.

- void clean_inode (struct inode ∗inode)

  Esta función inicializa los campos del inodo con valores conocidos antes de retornarlo.

5.9.1 Descripción detallada

Funciones que manejan los inodos que se encuentran en el VFS.

Se agregaron llamadas a las funciones:

- acl_destroy() y sc_destroy() en la función clear_inode().
- acl_aclinit() y sc_init() en la función clean_inode().

5.9.2 Documentación de las definiciones

5.9.2.1 #define CAN_UNUSE(inode)

Valor:

```
((((inode)->i_state | (inode)->i_data.nrpages) == 0)  && \
     !inode_has_buffers(inode))
```

5.10 Referencia del Archivo namei.c

Contiene las funciones donde se implementaron los nuevos controles de acceso para ACLs.

`#include <linux/init.h>`

`#include <linux/slab.h>`

```
#include <linux/fs.h>
#include <linux/quotaops.h>
#include <linux/pagemap.h>
#include <linux/dnotify.h>
#include <linux/smp_lock.h>
#include <linux/personality.h>
#include <asm/namei.h>
#include <asm/uaccess.h>
```

Funciones

- int vfs_permission (struct inode ∗inode, int mask)

  Consulta permisos.

- int permission (struct inode ∗inode, int mask)

  Es sinónimo de vfs_permission().

- int link_path_walk (const char ∗name, struct nameidata ∗nd)

  Resolución de nombres.

- int may_open (struct inode ∗inode, int perms)

  Determina si un archivo puede ser abierto por un sujeto o no. Usa permission() y mac_permission().

- int may_delete (struct inode ∗dir, struct dentry ∗victim, int isdir)

  Verifica si es posible o no eliminar el enlace victim del directorio dir, verifica si el tipo de victim es correcto.

- int may_create (struct inode ∗dir, struct dentry ∗child)

  Verifica si es posible crear un objeto con dentry child en el directorio dir.

- asmlinkage long sys_mknod (const char ∗filename, int mode, dev_t dev)

  Llamada al sistema mknod. No fue modificada pero usa vfs_mknod() la cual usa may_create().

- asmlinkage long sys_mkdir (const char ∗pathname, int mode)

  Llamada al sistema mkdir. No fue modificada pero usa vfs_mkdir() la cual usa may_create().

- asmlinkage long sys_rmdir (const char ∗pathname)

  Llamada al sistema rmdir. No fue modificada pero usa funciones que usan may_delete()).

- asmlinkage long sys_unlink (const char ∗pathname)

Llamada al sistema unlink. No fue modificada pero usa funciones que usan may_delete().

- asmlinkage long sys_symlink (const char ∗oldname, const char ∗newname)

  Llamada al sistema symlink. No fue modificada pero usa funciones que usan may_create().

- asmlinkage long sys_link (const char ∗oldname, const char ∗newname)

  Llamada al systema link. No fue modificada pero usa vfs_link() que a su vez usa may_create().

- asmlinkage long sys_rename (const char ∗oldname, const char ∗newname)

  Llamada al sistema rename. No fue modificada pero usa otras funciones que usan may_create() y may_delete().

5.10.1 Descripción detallada

Contiene las funciones donde se implementaron los nuevos controles de acceso para ACLs.

Se realizaron los siguientes cambios:

1. Se modificaron las funciones vfs_permission() y permission() para que las verificaciones de permisos se realicen utilizando la ACL del inodo.
2. Se agregó la función may_open() que será utilizada por las llamadas al sistema sys_open() y sys_creat().
3. Se modificó la función link_path_walk() para que pida lectura de directorios en lugar de ejecutabilidad al recorrer un camino.
4. Se cambiaron los permisos MAY_XXX por ACL_XXX_BIT.
5. open_namei() ??

5.10.2 Documentación de las variables

5.10.2.1 struct inode_operations page_symlink_inode_operations

Valor inicial:

```
 {
    readlink:   page_readlink,
    follow_link:    page_follow_link,
}
```

5.11 Referencia del Archivo namespace.c

Contiene algunas llamadas y funciones auxiliares ligeramente modificadas o que deberían haber sido modificadas.

```
#include <linux/config.h>
#include <linux/slab.h>
#include <linux/smp_lock.h>
#include <linux/init.h>
#include <linux/quotaops.h>
#include <linux/acct.h>
#include <linux/module.h>
#include <linux/devfs_fs_kernel.h>
#include <asm/uaccess.h>
#include <linux/nfs_fs.h>
#include <linux/nfs_fs_sb.h>
#include <linux/nfs_mount.h>
```

Funciones

- int mount_is_safe (struct nameidata *nd)

  No fue modificada paro usa permission().

- asmlinkage long sys_mount (char *dev_name, char *dir_name, char *type, unsigned long flags, void *data)

  Llamada al sistema mount.

5.11.1 Descripción detallada

Contiene algunas llamadas y funciones auxiliares ligeramente modificadas o que deberían haber sido modificadas.

5.11.2 Documentación de las variables

5.11.2.1 struct proc_fs_info mnt_info[ ] `[static]`

Valor inicial:

```
{
   { MNT_NOSUID, ",nosuid" },
   { MNT_NODEV, ",nodev" },
   { MNT_NOEXEC, ",noexec" },
   { 0, NULL }
}
```

5.11.2.2 struct file_operations rootfs_dir_operations [static]

Valor inicial:

```
{
   read:       generic_read_dir,
   readdir:    dcache_readdir,
}
```

5.11.2.3 struct inode_operations rootfs_dir_inode_operations [static]

Valor inicial:

```
{
   lookup:     rootfs_lookup,
}
```

5.12 Referencia del Archivo open.c

Funciones que manejan archivos abiertos del VFS.

#include <linux/string.h>

#include <linux/mm.h>

#include <linux/utime.h>

#include <linux/file.h>

#include <linux/smp_lock.h>

#include <linux/quotaops.h>

#include <linux/dnotify.h>

#include <linux/module.h>

#include <linux/slab.h>

#include <linux/tty.h>

```
#include <linux/iobuf.h>
#include <asm/uaccess.h>
```

Funciones

- asmlinkage long sys_truncate (const char *path, unsigned long length)

  Llamada al sistema truncate. No fue modificada pero usa permission().

- asmlinkage long sys_truncate64 (const char *path, loff_t length)

  Llamada al sistema truncate64. No fue modificada pero usa permission().

- asmlinkage long sys_access (const char *filename, int mode)

  Llamada al sistema access. Esta función no fue modificada, sin embargo usa permission().

- asmlinkage long sys_chdir (const char *filename)

  Llamada al sistema chdir.

- asmlinkage long sys_fchdir (unsigned int fd)

  Llamada al systema fchdir. No fue modificada pero usa permission().

- asmlinkage long sys_chroot (const char *filename)

  Llamada al sistema chroot. No fue modificada pero usa permission().

- asmlinkage long sys_fchmod (unsigned int fd, mode_t mode)

  Llamada al sistema fchmod.

- int acl_chmod_ok (struct inode *inode, int mode)

  Calcula si se puede o no realizar sys_chmod() sobre un inodo.

- asmlinkage long sys_chmod (const char *filename, mode_t mode)

  Llamada al sistema chmod.

- int chown_common (struct dentry *dentry, uid_t user, gid_t group)

  Esta función es utilizada por sys_chown() y sys_lchown().

- asmlinkage long sys_chown (const char *filename, uid_t user, gid_t group)

  Llamada al sistema chown.

- asmlinkage long sys_lchown (const char *filename, uid_t user, gid_t group)

  Llamada al sistema lchown.

- asmlinkage long sys_fchown (unsigned int fd, uid_t user, gid_t group)

  Llamada al sistema fchown.

- asmlinkage long sys_open (const char *filename, int flags, int mode)

Llamada al sistema open. No fue modificada pero usa (indirectamente) may_open().

- asmlinkage long sys_creat (const char *pathname, int mode)

  Llamada al sistema creat. No fue modificada pero usa sys_open().

- asmlinkage long sys_close (unsigned int fd)

  Esta llamada no fue modificada. Se la incluye por completitud. Está especificada en el modelo.

- asmlinkage long sys_ownerclose (int pid, char *filename)

  Llamada al sistema ownerclose. Permite que un dueño cierre uno de sus archivos que ha sido abierto por otro usuario.

- int is_open (struct inode *inode)

  Determina si un archivo está o no abierto en el sistema.

5.12.1 Descripción detallada

Funciones que manejan archivos abiertos del VFS.

- Se agregó la función is_open() y acl_chmod_ok().
- Se modificaron las llamadas al sistema sys_chmod(), sys_fchmod(), sys_chown(), sys_fchown() y sys_lchmod().
- Se modificó la función chown_common().
- Se cambiaron los MAY_XXX por ACL_XXX_BIT.

5.13 Referencia del Archivo sc.c

Funciones y llamadas al sistema que manejan las clases de acceso de los inodos del VFS.

```
#include <linux/fs.h>
#include <asm/uaccess.h>
#include <linux/subjectsc.h>
#include <linux/smp_lock.h>
#include <linux/file.h>
#include <linux/sched.h>
#include <linux/mm.h>
```

Funciones

- int mac_permission (struct inode *inode, int mode)

  Determina si se tiene permiso para acceder a inode *según la clase de acceso. Implementa seguridad MAC y MLS.*

- asmlinkage long sys_oscstat (char *filename, int *level, int *categories, int size)

  Llamada al sistema oscstat. Retorna la clase de acceso de un archivo.

- asmlinkage long sys_chobjsc (char *filename, int level, int *categories, int size)

  Llamada al sistema chobjsc. Cambia la SC de un archivo.

5.13.1 Descripción detallada

Funciones y llamadas al sistema que manejan las clases de acceso de los inodos del VFS.

5.14 Referencia del Archivo sc.c

Definiciones de las funciones declaradas en sc.h y algunas funciones auxiliares.

```
#include <linux/sc.h>
#include <linux/slab.h>
#include <asm/uaccess.h>
#include <linux/subjectsc.h>
```

Iterador para las categorías de un objeto sc

También forma parte de la interfaz de sc.

Ver sc_for_each_cat para un ejemplo que muestra cómo usarlo.

- sc_cat_iterator sc_cat_first (struct sc *sc)

  Obtiene un iterador o selector que corresponde a la primer categoría válida dentro de las categorías de una SC.

- sc_cat_iterator sc_cat_next (struct sc *sc, sc_cat_iterator it)

  Obtiene el iterador correspondiente a la próxima categoría válida dentro de una SC con respecto a una posición de referencia pasada como parámetro.

- int sc_getcat (struct sc *sc, sc_cat_iterator it)

Obtiene la categoría referenciada por un iterador.

Funciones de la interfaz de sc

- void sc_destroy (struct sc ∗sc)

  Libera los recursos asignados a una SC.

- sc ∗ sc_init (void)

  Inicializa una SC.

- int sc_getsize (struct sc ∗sc)

  Obtiene la cantidad de categorías válidas dentro de una SC.

- int sc_getlevel (struct sc ∗sc)

  Obtiene el nivel de una SC.

- void sc_setlevel (struct sc ∗sc, unsigned int l)

  Establece el nivel de una SC.

- void sc_clearcat (struct sc ∗sc)

  Borra todas las categorías de una SC.

- int sc_addcat (struct sc ∗sc, int cat)

  Agrega una categoría las categorías de una SC.

- int sc_compare (struct sc ∗sc1, struct sc ∗sc2)

  Determina si sc2 *domina a* sc1.

- int sc_copy (struct sc ∗sc1, struct sc ∗sc2)

  Copia el contenido de una clase de seguridad en otra.

- int sc_usergetncat (struct sc ∗sc, int ∗cat, int n)

  Carga un conjunto de categorías que provienen del espacio de usuario a una SC.

- int sc_userputcat (struct sc ∗sc, int ∗cat)

  Carga un conjunto de categorías que provienen de una SC en espacio de kernel a un conjunto de categorías que se encuentra en espacio de usuario.

Funciones adicionales

Funciones implementadas con las funciones de la interfaz.

- int is_secadm (void)

Determina si un usuario es un administrador de seguridad.

- int cat_in (int cat, sc ∗sc)

  Operador de pertenencia sobre conjuntos de categorías.

5.14.1 Descripción detallada

Definiciones de las funciones declaradas en sc.h y algunas funciones auxiliares.

Bug

Los conjuntos de categorías deben implementarse con arreglos o conjuntos de bits cada uno representando una categoría, ver capability.h.

5.14.2 Documentación de las funciones

5.14.2.1 int is_secadm (void)

Determina si un usuario es un administrador de seguridad.

Determina si el usuario que llama a esta función es o no un oficial MAC.

Devuelve:

- 1 Si es oficial MAC.
- 0 Si no es oficial MAC.

Bug

No debería ir en este archivo!

5.14.2.2 int cat_in (int *cat*, sc ∗ *sc*)

Operador de pertenencia sobre conjuntos de categorías.

Implementa la operacion de pertenencia de una categoria al conjunto de categorias de una SC.

Determina si *cat* pertenece a *sc* o no.

Parmetros:

cat Una categoría.

sc Una clase de seguridad.

Devuelve:

- 1 Si *cat* pertenece a *sc*.

- 0 Si *cat* no pertenece a *sc* .

Ejemplos:
iterador_sc_cat.c.

5.15 Referencia del Archivo sc.h

Definiciones, estructuras y prototipos de funciones para manejar las clases de seguridad (SC) en general.

```
#include <linux/errno.h>
```

Clases

- struct sc

  SC a nivel VFS. Se utiliza dentro del inodo del VFS.

Constantes (públicas) de sc

Debe considerarse que el nombre de la constante es público y su valor privado.

- #define SC_CAT_FULL -1

  Retorno de funciones que manejan las SC.

- #define SC_OK 0

  Retorno de funciones que manejan las SC.

- #define SCADMIN 65535

  Categoría que deben tener los administradores de seguridad para cambiar clases de acceso de sujetos y objetos.

Iterador para las categorías de un objeto sc

También forma parte de la interfaz de sc.

Ver sc_for_each_cat para un ejemplo que muestra cómo usarlo.

- #define SC_IT_ERROR -1

  Retorno de las funciones que iteran sobre las categorías de una SC de un inodo.

- #define sc_for_each_cat(sc, i) for(i=sc_cat_first(sc);i!=SC_IT_ERROR;i=sc_cat_-next(sc,i))

  Macro que itera entre todas las categorías de una SC.

Definiciones

- #define MAX_CAT_LEN 8

  Cantidad maxima de categorias posibles en la SC de un inodo del VFS.

Tipos definidos

- typedef int sc_cat_iterator

  Renombre del tipo int, utilizado como iterador dentro de las categorías de una SC.

Funciones

- sc ∗ sc_init (void)

  Inicializa una SC.

5.15.1 Descripción detallada

Definiciones, estructuras y prototipos de funciones para manejar las clases de seguridad (SC) en general.

5.16 Referencia del Archivo stat.c

Llamadas al sistema de la familia stat.

```
#include <linux/config.h>
#include <linux/mm.h>
#include <linux/errno.h>
#include <linux/file.h>
#include <linux/smp_lock.h>
#include <linux/highuid.h>
#include <asm/uaccess.h>
#include <linux/fs_acl.h>
```

Funciones

- int cp_old_stat (struct inode ∗inode, struct __old_kernel_stat ∗statbuf)

  Esta función es utilizada por las llamadas sys_stat() y sys_fstat().

- int cp_new_stat (struct inode ∗inode, struct stat ∗statbuf)

  Esta función es utilizada por las llamadas sys_newstat() y sys_newfstat().

- asmlinkage long sys_stat (char ∗filename, struct __old_kernel_stat ∗statbuf)

  Llamada al sistema stat.

- asmlinkage long sys_newstat (char ∗filename, struct stat ∗statbuf)

  Llamada al sistema newstat. Ídem sys_stat() con la diferencia que esta llamada utiliza la función cp_new_stat().

- asmlinkage long sys_lstat (char ∗filename, struct __old_kernel_stat ∗statbuf)

  Llamada al sistema lstat.

- asmlinkage long sys_newlstat (char ∗filename, struct stat ∗statbuf)

  Llamada al sistema newlstat. Ídem sys_lstat() con la diferencia que esta llamada utiliza la función cp_new_stat().

- asmlinkage long sys_fstat (unsigned int fd, struct __old_kernel_stat ∗statbuf)

  Llamada al sistema fstat.

- asmlinkage long sys_newfstat (unsigned int fd, struct stat ∗statbuf)

  Llamada al sistema newfstat. Ídem sys_fstat() con la diferencia que esta llamada utiliza la función cp_new_stat().

- long cp_new_stat64 (struct inode ∗inode, struct stat64 ∗statbuf)

  Esta función es utilizada por las llamadas sys_stat64(), sys_fstat64() y sys_lstat64().

- asmlinkage long sys_stat64 (char ∗filename, struct stat64 ∗statbuf, long flags)

  Llamada al sistema stat64. Ídem sys_stat() pero para los sistemas de archivos de 64 bits.

- asmlinkage long sys_lstat64 (char ∗filename, struct stat64 ∗statbuf, long flags)

  Llamada al sistema lstat64. Ídem sys_lstat() pero para los fileystems de 64 bits.

- asmlinkage long sys_fstat64 (unsigned long fd, struct stat64 ∗statbuf, long flags)

  Llamada al sistema fstat64. Ídem sys_fstat() pero para los sistemas de archivos de 64 bits.

- int cp_aclstat (struct inode ∗inode, struct acl_entry ∗acl_statbuf, int len)

  Carga en una estructura los datos de la ACL de un inodo.

- asmlinkage long sys_aclstat (char ∗filename, struct acl_entry ∗acl_statbuf, int len)

    Llamada al sistema aclstat. Devuelve información almacenada en una ACL.

- asmlinkage long sys_aclfstat (unsigned int fd, struct acl_entry ∗acl_statbuf, int len)

    Llamada al sistema aclfstat. Devuelve información de una ACL.

5.16.1 Descripción detallada

Llamadas al sistema de la familia stat.

- Estas llamadas son las siguientes:

1. sys_stat()
2. sys_newstat()
3. sys_lstat()
4. sys_newlstat()
5. sys_fstat()
6. sys_newfstat()
7. sys_stat64()
8. sys_lstat64()
9. sys_aclstat()
10. sys_aclfstat()

La llamada sys_newstat() es una implementación más actual de la llamada sys_stat(), esta última sigue existiendo por problmas de compatibilidad. Lo mismo ocurre con sys_newlstat(), sys_newfstat().

5.17 Referencia del Archivo subjectsc.c

Definiciones de las funciones declaradas en subjectsc.h y algunas funciones auxiliares.

```
#include <linux/init.h>
#include <linux/kernel.h>
#include <linux/subjectsc.h>
#include <asm/uaccess.h>
```

Funciones de la interfaz de subjectsc

- int __init subjectsc_init (void)
  Inicializa las SC de todos los sujetos.
- int __exit subjectsc_exit (void)
  Libera las SC de todos los sujetos.
- sc ∗ subjectscget (uid_t uid)
  Obtiene la SC de un usuario.
- int subjectscadd (uid_t uid, sc ∗sc)
  Agrega una categoría a la SC de un usuario.
- sc ∗ subjectscdel (uid_t uid)
  Elimina una categoría de la SC de un usuario.

Estructuras de datos privadas (ocultas)

- sc ∗ subjectsc [SC_MAX_BUCKETS]
  Aquí se almacenan las SC de los sujetos.
- sc ∗ lowest_sc
  Apunta a la SC más baja.

Definiciones

- #define LISEXDEBUG(x)
  Maneja los mensajes del núcleo.
- #define SC_MAX_BUCKETS 600
  Cantidad máxima de SC de sujetos.
- #define SUBSC_HASH(x) x
  Función de dispersión; por ahora un arreglo completo.

Funciones

- asmlinkage long sys_chsubsc (uid_t uid, int level, int ∗categories, int size)
  Llamada al sistema chsubsc. Cambia la clase de acceso de un usuario.
- asmlinkage long sys_sscstat (uid_t uid, int ∗level, int ∗categories, int size)

Llamada al sistema sscstat. Retorna la clase de acceso de uid.

- int logged (uid_t uid)

  Determina si un sujeto está corriendo algún proceso en el sistema.

5.17.1 Descripción detallada

Definiciones de las funciones declaradas en subjectsc.h y algunas funciones auxiliares.

5.17.2 Documentación de las funciones

5.17.2.1 int __init subjectsc_init (void)

Inicializa las SC de todos los sujetos.

Se considera que los usuarios son aquellos que tienen UID menor que SC_MAX_BUCKETS. Esto se debe a que Lisex 0.0 es un prototipo; en futuras versiones se inicializará la clase de acceso de cada usuario presente en el sistema (i.e. en `/etc/passwd)`.

5.17.2.2 sc∗ subjectscget (uid_t *uid*)

Obtiene la SC de un usuario.

Se considera que los usuarios son aquellos que tienen UID menor que SC_MAX_BUCKETS. Esto se debe a que Lisex 0.0 es un prototipo; en futuras versiones se inicializará la clase de acceso de cada usuario presente en el sistema (i.e. en `/etc/passwd)`.

Parmetros:

uid Identificación del usuario sobre el que se quiere realizar la operación.

Devuelve:

La SC del usuario con identificación *uid.*

Bug

Debería buscar dentro del bucket (una lista enlazada?)

5.17.2.3 int subjectscadd (uid_t *uid*, sc ∗ *sc*)

Agrega una categoría a la SC de un usuario.

Devuelve:

- -EPERM si *uid* no es un usuario registrado (ver subjectsc_init()).
- 0 si la operación se realizó con éxito.

5.17.2.4 asmlinkage long sys_chsubsc (uid_t *uid*, int *level*, int ∗ *categories*, int *size*)

Llamada al sistema chsubsc. Cambia la clase de acceso de un usuario.

Precondicin:

- El usuario que invoca la llamada debe ser un oficial MAC.
- El usuario al cual se le cambiará la clase de acceso no debe tener archivos abiertos.

Parmetros:

uid Identificación del usuario sobre el cual se realiza la operación.

level Nivel que se le asignará a la SC del usuario.

categories Conjunto de categorías que se le asignarán a la SC del usuario.

size Cantidad de categorías que se le asignarán a la SC del usuario.

Postcondicin:

- SC de *uid* cambia por una nueva SC con nivel *level* y conjunto de categorías *categories*.

Devuelve:

- -EBUSY Si *uid* esta corriendo algún proceso en el momento de realizar la operación.
- -EACCES Si el usuario que realiza esta operación no pertenece al grupo SecAdm.
- 0 Si la operación se realizó con éxito.

5.17.2.5 asmlinkage long sys_sscstat (uid_t *uid*, int ∗ *level*, int ∗ *categories*, int *size*)

Llamada al sistema sscstat. Retorna la clase de acceso de *uid*.

Parmetros:

uid Identificación del usuario del cual se quiere conocer la clase de seguridad.

level Donde se cargará el nivel de la clase de acceso de *uid*.

categories Donde se cargarán las categorías de la clase de acceso de *uid*.

size Cantidad de categorías que se pide cargar en categories.

Postcondicin:

- El parámetro *level* contiene el nivel de la clase de acceso de *uid.*
- En el parámetro *categories* están las categorías de la clase de acceso de *uid,* si *size* es mayor que 0.

Devuelve:

- -EACCES Si La clase de acceso del usuario que realiza la operación no domina a la de *uid.*
- Cantidad de entradas válidas que hay en la clase de acceso de *uid,* si *size* es 0. En este caso se carga sólo el nivel.
- -EINVAL Si *size* es menor que la cantidad de entradas válidas que hay en la clase de acceso de *uid.*
- m Si la operación se realizó con éxito. *m* es la cantidad de entradas válidas que se cargaron en *categories.*

5.18 Referencia del Archivo subjectsc.h

Definiciones y prototipos de funciones para manejar las SC de los usuarios.

`#include <linux/types.h>`

`#include <linux/errno.h>`

`#include <linux/sc.h>`

Funciones de la interfaz de subjectsc

- int __init subjectsc_init (void)

  Inicializa las SC de todos los sujetos.

- int __exit subjectsc_exit (void)

  Libera las SC de todos los sujetos.

- sc ∗ subjectscget (uid_t uid)

  Obtiene la SC de un usuario.

- int subjectscadd (uid_t uid, sc ∗sc)

  Agrega una categoría a la SC de un usuario.

- sc ∗ subjectscdel (uid_t uid)

  Elimina una categoría de la SC de un usuario.

Funciones

- asmlinkage long sys_chsubsc (uid_t uid, int level, int ∗categories, int size)
  Llamada al sistema chsubsc. Cambia la clase de acceso de un usuario.
- asmlinkage long sys_sscstat (uid_t uid, int ∗level, int ∗categories, int size)
  Llamada al sistema sscstat. Retorna la clase de acceso de uid.

5.18.1 Descripción detallada

Definiciones y prototipos de funciones para manejar las SC de los usuarios.

5.19 Referencia del Archivo sys.c

Contiene la familia de llamdas setuid, entre otras.

```
#include <linux/module.h>
#include <linux/mm.h>
#include <linux/utsname.h>
#include <linux/mman.h>
#include <linux/smp_lock.h>
#include <linux/notifier.h>
#include <linux/reboot.h>
#include <linux/prctl.h>
#include <linux/init.h>
#include <linux/highuid.h>
#include <asm/uaccess.h>
#include <asm/io.h>
#include <linux/fs.h>
#include <linux/file.h>
#include <linux/slab.h>
#include <linux/subjectsc.h>
#include <linux/sched.h>
```

Funciones

- asmlinkage long sys_setreuid (uid_t ruid, uid_t euid)

Llamada al sistema setreuid. Debería haber sido modificada como sys_setuid().

- asmlinkage long sys_setuid (uid_t uid)

  Cambia el uid sólo si los archivos abiertos verifican seguridad simple, confinamiento y DAC para el nuevo usuario.

- asmlinkage long sys_setresuid (uid_t ruid, uid_t euid, uid_t suid)

  Llamada al sistema setresgid. Debería haber sido modificada como sys_setuid().

- asmlinkage long sys_setfsuid (uid_t uid)

  Llamada al sistema setrfsuid. Debería haber sido modificada como sys_setuid().

5.19.1 Descripción detallada

Contiene la familia de llamdas setuid, entre otras.

Se modificó únicamente sys_setuid() aunque se debería haber modificado la familia completa.

5.20 Referencia del Archivo unistd.h

Se agregaron las definiciones necesarias para registrar las nuevas llamadas al sistema.

Definiciones

- #define __NR_aclstat 226

  Número de la llamada al sistema sys_aclstat().

- #define __NR_aclfstat 227

  Número de la llamada al sistema sys_aclfstat().

- #define __NR_acladd 228

  Número de la llamada al sistema sys_acladd().

- #define __NR_acldel 229

  Número de la llamada al sistema sys_acldel().

- #define __NR_chsubsc 230

  Número de la llamada al sistema sys_chsubsc().

- #define __NR_sscstat 231

  Número de la llamada al sistema sys_sscstat().

- #define __NR_chobjsc 232

Número de la llamada al sistema sys_chobjsc().

- #define __NR_ownerclose 233

  Número de la llamada al sistema sys_ownerclose().

- #define __NR_oscstat 234

  Número de la llamada al sistema sys_oscstat().

5.20.1 Descripción detallada

Se agregaron las definiciones necesarias para registrar las nuevas llamadas al sistema.

5.21 Referencia del Archivo unistd.h

Se agregaron las definiciones necesarias para registrar las nuevas llamadas al sistema.

Definiciones

- #define __NR_aclstat 226

  Número de la llamada al sistema sys_aclstat().

- #define __NR_aclfstat 227

  Número de la llamada al sistema sys_aclfstat().

- #define __NR_acladd 228

  Número de la llamada al sistema sys_acladd().

- #define __NR_acldel 229

  Número de la llamada al sistema sys_acldel().

- #define __NR_chsubsc 230

  Número de la llamada al sistema sys_chsubsc().

- #define __NR_sscstat 231

  Número de la llamada al sistema sys_sscstat().

- #define __NR_chobjsc 232

  Número de la llamada al sistema sys_chobjsc().

- #define __NR_ownerclose 233

  Número de la llamada al sistema sys_ownerclose().

- #define __NR_oscstat 234

  Número de la llamada al sistema sys_oscstat().

5.21.1 Descripción detallada

Se agregaron las definiciones necesarias para registrar las nuevas llamadas al sistema.

Capítulo 6

Bugs

6.1 Lista de Bugs

Global mac_permission (struct inode *inode, int mode) Hay que bloquear subjectsc

Mal programada, mal indentada, mal concevida. MAL!

Archivo sc.c Los conjuntos de categorías deben implementarse con arreglos o conjuntos de bits cada uno representando una categoría, ver capability.h.

Global is_secadm (void) No debería ir en este archivo!

Global subjectscget (uid_t uid) Debería buscar dentro del bucket (una lista enlazada?)

Global sys_chmod (const char *filename, mode_t mode) Si no hay espacio en la ACL para todos los cambios, no se realiza ninguno.

Global chown_common (struct dentry *dentry, uid_t user, gid_t group) No verifica si va a haber espacio para la operación!!

Global sc_for_each_cat (sc, i) el 'arreglo' de categorías podría ser un TAD por sí mismo.

Global sc_addcat (struct sc *sc, int cat) Siempre agrega la categoría por más que ya esté.

Global sc_compare (struct sc *sc1, struct sc *sc2) Si alguna de las SC a comparar no son válidas, el resultado es impredecible.

Capítulo 7

Ejemplos

7.1 iterador_acl_entry.c

Este ejemplo muestra cómo usar el iterador para recorrer las entradas de una ACL.

```
static int cp_aclstat(struct inode * inode, struct acl_entry * acl_statbuf, int len)
{
    struct acl_entry tmp[MAX_ACL_LEN];
    acl_iterator i;
    int j=0;

    if(len < 0 || len < acl_getsize(&inode->vfs_acl)) /*si es negativo o si no alcanza*/
    return -EINVAL;
    if(len)     /*si no es cero*/
    {

//EL EJEMPLO

    acl_for_each(&inode->vfs_acl,i){
        if(j>len) break;
        tmp[j++]=acl_getentry(&inode->vfs_acl,i);   /*\bug deberiamos borrar los bits que no le interesan al
    }

//FIN EJEMPLO

        if(copy_to_user(acl_statbuf,tmp,j * sizeof(struct acl_entry)))
        return -EFAULT;
    }
    return acl_getsize(&inode->vfs_acl);
}
```

7.2 iterador_sc_cat.c

Este ejemplo muestra cómo usar el iterador para recorrer las categorías en una clase de acceso.

```
int cat_in (int cat, sc * sc){
sc_cat_iterator i;

    sc_for_each_cat(sc,i)
            if(sc_getcat(sc,i)==cat)
            return 1;
    return 0;
}
```

Índice de Materias