L I S E X

GIDIS

Manual del usuario y del administrador

Lisex 0.0

Maximiliano Cristiá

GIDIS

Grupo de Investigación y Desarrollo en Ingeniería de Software

Índice general

Advertencia

⚠ **¡¡Importante!!**

Este capítulo contiene información esencial sobre la seguridad de Lisex 0.0.

Lisex 0.0 es un prototipo orientado a expertos en el área de seguridad en sistemas operativos; particularmente no está destinado al usuario promedio ni a ser utilizado en entornos de producción.

Implementa las mejoras de seguridad sólo en el sistema de archivos por lo que sigue siendo posible que el sistema sea atacado por medio de caballos de Troya. En particular no se incluye ninguna mejora en el subsistema de red; se espera que Lisex 0.0 sea usado sin conexión a ninguna red.

La implementación actual no ha sido testeada con la profundidad y amplitud debidas. En ese sentido puede decirse que es una versión alfa e incluso anterior.

Lisex 0.0 presenta cierta resistencia contra ataques por medio de caballos de Troya que intentan divulgar datos confidenciales. Por lo tanto, debe quedar claro que otros tipos de ataques, en particular ataques contra la integridad, siguen siendo posibles.

Esta versión de Lisex es una prueba de concepto con el fin de comenzar a trabajar en la construcción de una plataforma de cómputo confiable. Futuras versiones de Lisex abandonarán el carácter de prototipo y ampliarán su alcance.

Lisex 0.0 sólo fue probado sobre arquitectura i386.

Caballos de Troya que pueden atacar Lisex 0.0

Como ya se mencionó existen ciertos tipo de caballos de Troya que pueden dañar la confidencialidad de los datos almacenados en Lisex 0.0. Esto se debe pura y exclusivamente a que el equipo de desarrollo decidió poner tempranamente a disposición del público una versión incompleta, antes que retrasar su aparición en aras de la completitud.

Un caballo de Troya que puede afectar a Lisex 0.0 trabaja de la siguiente forma:

1. Abre un archivo muy confidencial
2. Copia el archivo a un buffer de la memoria
3. Cierra el archivo
4. Abre un archivo escasamente confidencial
5. Escribe el contenido del buffer en el archivo
6. Cierra el archivo y termina

Además, si un caballo de Troya utiliza llamadas al sistema no relacionadas con el sistema de archivos escapará al control MLS implementado.

Otro tipo de caballo de Troya que puede atacar Lisex 0.0 se relaciona con los programas SUID y la existencia de varias terminales lógicas asociadas con la misma terminal física. El escenario se describe a continuación:

1. Un usuario de bajo nivel de seguridad logra que un usuario de alto nivel de seguridad instale un caballo de Troya SUID a este último.
2. El usuario de bajo nivel ejecuta el caballo de Troya.
3. El caballo de Troya cambia de UID al usuario de alto nivel.
4. Abre una terminal lógica (virtual) que el usuario de alto nivel puede abrir y muestra por pantalla el contenido de todos y cada uno de los archivos a los que el usuario de alto nivel tiene acceso y el de bajo nivel no, mientras tanto el usuario de bajo nivel se encuentra sentado frente a la terminal física asociada con esa terminal lógica.

El ataque anterior es posible debido a que todas las terminales lógicas están asociadas con la misma terminal física (la consola) y a que el usuario de bajo nivel de seguridad tiene acceso físico a dicha terminal. La existencia de terminales físicas en diferentes ubicaciones físicas con acceso restringido es un requisito común en ambientes de alta seguridad.

Un tercer tipo de caballo de Troya que puede dañar la confidencialidad de los datos "protegidos" por Lisex 0.0 actuaría como se describe a continuación:

1. El atacante logra que `root` ejecute un programa que, además de realizar alguna función útil, modifica el archivo `/etc/passwd` de forma tal que el intérprete de comandos de `secadm` es otro programa de ataque.
2. Este último programa de ataque permite que `secadm` modifique clases de acceso, como lo haría su *shell* habitual, pero también lo hace de alguna forma conveniente para el atacante.
3. Por lo tanto, cada vez que `secadm` se loguee será víctima inocente de un programa que hace más de lo que parece.

Esta lista de caballos de Troya no es exhaustiva. Es importante aclarar que estos caballos de Troya pueden actuar debido a que Lisex 0.0 es un prototipo. En futuras versiones se incorporarán controles para resistir a estos tipos de ataques.

Capítulo 1

Introducción

Este capítulo está destinado a aquellos que no tienen mayores conocimientos sobre modelos MLS y temas relacionados. Los lectores con más experiencia en este campo pueden leer la tesis [Cri02] para una descripción detallada del modelo de seguridad implementado en Lisex 0.0.. Aquellos para los cuales esta introducción resulte insuficiente pueden consultar [AJP95, Gas88].

Sugerimos leer este capítulo secuencial y completamente.

Lisex es un sistema operativo pensado para resistir ataques a la *confidencialidad* (1.1) efectuados por medio de *caballos de Troya* (1.2) en ambientes donde la seguridad es crítica o al menos muy importante[1]. Según las investigaciones desarrolladas principalmente en Estados Unidos de Norteamérica durante los '70 y '80, tales amenazas se combaten utilizando dos herramientas:

— un *modelo de seguridad multinivel* (MLS, 1.6) en el núcleo del sistema, y

— un desarrollo del sistema que utilice *métodos formales* (1.9).

El modelo MLS que incorpora Lisex 0.0 [Cri02], llamado *Lisex Security Model 0.0* (LSM0.0), es una adaptación del modelo introducido por Bell y La Padula [BL73a, BL73b] en 1973. LSM0.0 ha sido incorporado únicamente al sistema de archivos del núcleo 2.4.14 de Linux y desarrollado utilizando el entorno de pruebas Coq [Coq]. El desarrollo de LSM0.0 incluye especificación y verificación formal. Futuras versiones de Lisex implementarán otros modelos equivalentes en otras porciones del núcleo.

Además, Lisex 0.0 implementa *listas de control de acceso* (ACL) para mejorar el mecanismo DAC (1.7) tradicional de Linux y otras características de seguridad menores.

Por el momento, no está planeado generar versiones POSIX de Lisex aunque algunas de las extensiones de seguridad han sido consideradas.

[1] Los números entre paréntesis, como (1.1) o (1.2), indican secciones de este manual donde se puede encontrar más información sobre el tema que está siendo tratado.

1.1. Confidencialidad

Existen muchas caracterizaciones de seguridad informática. La que se considera en este proyecto es la que se relaciona con el concepto de seguridad en tecnología de la información. Este concepto está definido en [Com91], donde seguridad en tecnología de la información se define como la combinación de:

Confidencialidad prevenir la revelación no autorizada de información

Integridad prevenir la modificación no autorizada de información

Disponibilidad prevenir la negación no autorizada de información o recursos de cómputo.

Por lo tanto, cualquier organización que posea una política de acceso a la información que indique que no todos los individuos pueden leer todos los datos, requiere de un sistema de cómputo capaz de defenderse de ataques contra la confidencialidad.

La principal amenaza contra la confidencialidad en sistemas de cómputo la constituyen las diferentes variedades de caballos de Troya (1.2).

Algunos ejemplos de organizaciones que cuentan con políticas de seguridad orientadas a la confidencialidad son:

— Sistemas bancarios en aquellos estados donde existen leyes de secreto bancario

— Organizaciones vinculadas con la inteligencia y la seguridad

— Oficinas estatales que deban proteger datos de sus ciudadanos o documentos como pliegos licitatorios, documentación judicial, etc.

— Empresas de desarrollo que quieran proteger sus proyectos hasta tanto se transformen en productos en el mercado

— Instituciones de salud (por ejemplo, la UE ha promulgado una norma específica orientada a estandarizar la protección de datos clínicos procesados digitalmente [Cou96]).

1.2. Caballos de Troya

Como su contraparte mitológica, refiere a una técnica para atacar sistemas de cómputo desde adentro en lugar de hacerlo frontalmente contra barreras defensivas poderosas. Más aun, los caballos de Troya pueden atacar sistemas sin necesidad de que estos posean fallas en sus sistemas de protección. Un caballo de Troya es un programa cuya ejecución resulta en efectos secundarios indeseados, generalmente no anticipados por el usuario. Normalmente, un caballo de Troya se presenta como un programa útil que además contiene funciones que atacan de una forma u otra al sistema; estas funciones, obviamente, no están documentadas ni publicitadas. La utilidad manifiesta del caballo de Troya sirve como un cebo para que usuarios inocentes ejecuten el programa y por lo tanto, activen, sin intención ni conocimiento, las funciones de ataque.

Esta forma de ataque es conocida y estudiada desde mediados de la década del 60.

1.2.1. Ejemplo de un caballo de Troya

Imaginemos el siguiente escenario. Usted administra un sitio de comercio electrónico donde se almacenan los números de tarjetas de crédito de sus clientes[2]. Uno de los miembros del equipo de desarrollo descubre en Internet una herramienta que mejorará la productividad del equipo de una u otra forma. El empleado, con absoluta honestidad y queriendo favorecer a la empresa, decide instalarla para que todo el equipo la utilice. Ahora supongamos que tal herramienta fue desarrollada por una banda dedicada al robo de números de tarjeta de crédito, quienes la pusieron a disposición del público precisamente para que los desarrolladores de sitios de *e-commerce* la instalaran inocentemente. La herramienta no sólo presta la función que publicita sino que también, utilizando los permisos de los desarrolladores que la ejecutan, recorre las bases de datos en busca de números de tarjetas de crédito y los envía por e-mail (algo que cualquier desarrollador podría hacer si lo deseara) a una cuenta anónima de algún servidor como Hotmail o Yahoo, evitando de forma sencilla pero contundente el control de acceso del sistema operativo, la base de datos y el firewall.

Puede consultar otro ejemplo en [Cri02, pp. 7].

1.2.2. La esencia de un caballo de Troya

Es conveniente resaltar algunos puntos:

— Ni los administradores ni los usuarios de la organización atacada son conscientes de la existencia de las funciones de ataque presentes en el programa.

— Ni los administradores ni los usuarios de la organización tienen intenciones de violar la política de seguridad revelando información a terceros. A lo sumo algún usuario o administrador que no tiene acceso a cierta información es quien se las arregla para instalar el caballo de Troya en el sistema, pero quienes lo usan (y tienen acceso a la información) no saben que están usando un programa de ataque.

— Cualquier software que ejecute a nivel de usuario puede ser un caballo de Troya: editores de texto, planillas de cálculo, comandos del sistema operativo,

[2] También podría pensarse en cualquier sistema que administre información sensible cuya confidencialidad debe ser preservada; por ejemplo: datos médicos, información bancaria, judicial, estatal, de inteligencia, militar, etc.

programas de administración del sistema, aplicaciones corporativas, servidores o clientes TCP/IP, juegos, software de desarrollo, etc.

— No es preciso que exista una falla en el sistema de protección (en todo caso el sistema está fundamentalmente fallado o no ha sido preparado para soportar este tipo de ataques)

— El software de base (sistema operativo o DBMS) sobre el cual ejecuta el caballo de Troya posee un mecanismo de control de acceso similar al implementado en UNIX (Linux), MS NT o 2000, OS/400, Informix, Oracle, etc.

Digamos que el caballo de Troya es ejecutado por cierto usuario que tiene acceso a la información que interesa al atacante. Al hacerlo, el programa se convierte en un proceso a nombre del usuario como cualquier otro. Por lo tanto, dicho proceso puede acceder todos los recursos que cualquier otro proceso del usuario podría. En particular puede acceder los archivos, tablas, bases de datos, etc. que contienen la información a obtener. Mientras el programa presta la función útil por la cual el usuario lo ejecutó, el caballo de Troya activa sus funciones de ataque las cuales efectúan una o varias de las siguientes acciones dependiendo de la "distancia" que separa al atacante de la víctima y de qué tan profundo sea el daño que se desea realizar:

— copia la información valiosa en archivos, tablas o lo que sea a nombre del atacante en el mismo sistema

— envía la información por mail o cualquier otro protocolo capaz de sortear el o los firewalls interpuestos entre la organización e Internet.

— cambia la información de control de acceso de los archivos, tablas, etc. donde se almacena la información secreta de manera tal que más tarde el atacante la pueda acceder normalmente

— modifica el código objeto de cualquier programa escrito por el usuario de manera tal que estos otros programas se convierten en nuevos caballos de Troya (es decir, aquí se utilizan técnicas propias de los virus pero con una diferencia muy importante: al no ser programas de ataque indiscriminado, sino todo lo contrario, es muy poco probable que alguien lo detecte y lo reporte como un virus de manera que sean reconocidos por programas anti-virus).

De los párrafos precedentes, debe resultar claro que un caballo de Troya no requiere ni la más mínima falla en el sistema para lograr su cometido y que como arma de ataque, es letal y básicamente indetectable ... al menos en los sistemas más usados en la actualidad.

1.3. Control de acceso obligatorio

La comunidad científica del área de Seguridad Informática, luego de años de investigación, estipula que ninguno de los sistemas operativos comerciales de uso masivo, y por extensión otros software de base, brindan protección efectiva contra ataques a la confidencialidad efectuados por medio de caballos de Troya. Su accionar es prácticamente indetectable e indistinguible del funcionamiento de un programa similar sin fines ofensivos. Uno de los motivos que llevan a esta conclusión es que cualquier propietario de un archivo puede alterar los atributos de seguridad que lo protegen.

El control de acceso obligatorio[3] debe implementarse cuando se debe imponer una política de seguridad obligatoria. Una política de seguridad obligatoria es aquella política de seguridad en la cual la definición de la política y la asignación de atributos de seguridad están estrictamente controlados por los administradores. En otras palabras, si un sistema implementa un mecanismo MAC (al menos) parte de la información de control de acceso asociada a recursos y procesos puede ser alterada únicamente por una cantidad muy restringida de usuarios (administradores con ciertos privilegios). Este es el primer requisito para construir un sistema resistente a caballos de Troya. De no verificarse, cualquier programa podría alterar esos atributos sin el consentimiento del propietario, posibilitando así que un atacante acceda información ajena.

Los primeros en pensar en este tipo de ataque fueron investigadores y funcionarios del Departamento de Defensa (DoD) de los Estados Unidos de Norteamérica. Al DoD le interesaba incluir computadoras en su organización siempre y cuando esto no pusiera en riesgo la seguridad (entendida como confidencialidad) de la información que allí se administra. La política de seguridad de la información del DoD debía implementarse sin que existiera la posibilidad de que fuera violada por el mero hecho de que ahora se involucraba un sistema operativo y otros programas.

La implementación segura y confiable de la política de seguridad del DoD dio lugar a numerosas investigaciones que con el tiempo fueron dando forma a un cuerpo teórico y práctico de gran nivel técnico e implicaciones impensadas al comienzo. Uno de los conceptos más importantes en esta teoría son los sistemas de seguridad multinivel. Por esta razón, en las secciones que siguen se introducen con cierto detalle la política de seguridad del DoD, los sistemas multi-nivel y otros conceptos relacionados.

1.4. Política de seguridad del DoD

El Departamento de Defensa de los Estados Unidos de Norteamérica (de aquí en más DoD), tiene una política muy estricta para la administración y almacenamiento manual de información clasificada, que nosotros llamaremos *política de seguridad del DoD*. La política fue diseñada primordialmente para proteger la *confidencialidad* de

[3] Mandatory Access Control, MAC

cierta información militar, de política exterior y de inteligencia. En esta política, cada porción de información (usualmente en forma de documentos) y cada individuo poseen una *clase de acceso*. Para determinar si una persona puede tener acceso a un documento, se comparan ambas clases de acceso.

Las clases de acceso se componen de dos partes:

Nivel de seguridad también llamado *nivel de sensitividad* o simplemente *nivel*, consistente en uno de varios nombres tales como $TOPSECRET, SECRET, CONFIDENTIAL, UNCLASSIFIED$

Conjunto de categorías también llamados *compartimientos*, consistente en uno o más nombres tales como $NATO, NUCLEAR, CIA, F14$, etc.

Ejemplo 1.4.1 *Típicamente una clase de acceso se representa por medio de un par ordenado:*

$$(SECRET, \{NATO, NUCLEAR\}) \tag{1.1}$$

$$(TOPSECRET, \{F14, NUCLEAR, CIA\}) \tag{1.2}$$

donde la clase de acceso 1.1 tiene nivel $SECRET$ y conjunto de categorías $\{NATO, NUCLEAR\}$, mientras que la clase 1.2 tiene nivel $TOPSECRET$ y categorías $F14, NUCLEAR$ y CIA.

El conjunto de niveles de seguridad está linealmente ordenado, por ejemplo:

UNCLASSIFIED ¡CONFIDENTIAL ¡SECRET ¡TOP SECRET

mientras que las categorías son independientes unas de otras y no están ordenadas. Las clases de acceso estructuran la información y los usuarios según su grado de privacidad o responsabilidad; en la sub-sección siguiente se verá que es posible definir una escala u orden entre las clases de acceso, lo que a su vez implica una escala o orden en el grado de privacidad. La idea detrás de esto es determinar el grado de privacidad de la información y de responsabilidad de los usuarios para manejarla: cuanto más "alta" la clase de acceso más privada la información y más responsabilidad se le exige al usuario. Por tal motivo se habla de *altura* de la información y los usuarios.

De esta manera el acceso a cierta porción de información estará legalmente permitido a un individuo sí y sólo sí su clase de acceso *domina* (ver sub-sección siguiente) a la clase de acceso del la información. La clase de acceso de cada porción de información y de cada individuo sólo puede ser alterada por oficiales especialmente designados para esta tarea.

El propósito de esta política es prevenir el compromiso producido por un individuo capaz de leer información para la que no tiene suficiente autorización. En particular, la política no dice nada en cuanto a modificar o destruir esa misma información.

Si esta política se implementa en un sistema de cómputo, por lo general, el sistema maneja dos formas para identificar los niveles de seguridad y las categorías:

Interno que es utilizado por el núcleo del sistema y es básicamente invisible al usuario; usualmente son números enteros de uno o dos bytes o arreglos de bits

Humano que es utilizado por los usuarios y administradores en su trabajo cotidiano; usualmente son cadenas de caracteres que refieren a oficinas, departamentos, cargos, posiciones, etc. dentro de una organización. Por tal motivo se las suele denominar etiquetas legibles por humanos o *human readable labels*.

1.4.1. Orden parcial de las clases de acceso

Definición 1.4.1 *La clase de acceso* (n_1, C_1) *domina, lo que se simboliza con* $\succeq$, *a la clase de acceso* (n_2, C_2) *sí y sólo sí* $n_1 \geq n_2$ *y* $C_1 \supseteq C_2$.

De la definición anterior se deduce que en virtud del orden parcial definido por $\subseteq$, las clases de acceso están *parcialmente ordenadas*. En otras palabras dadas dos clases de acceso, (n_1, C_1) y (n_2, C_2), no siempre será posible decir que $(n_1, C_1) \succeq (n_2, C_2)$ ni que $(n_2, C_2) \succeq (n_1, C_1)$. Claramente esta es una situación diferente a la que se da, por ejemplo, en el conjunto de los números enteros donde siempre, dados dos enteros m_1 y m_2, $m_1 \leq m_2$ es cierto o $m_2 \leq m_1$ lo es.

Ejemplo 1.4.2 *Consideremos las siguientes clases de acceso*

$$
\begin{aligned}
CA_1 &= (3, \{NATO, NUCLEAR\}) \\
CA_2 &= (1, \{NATO, NUCLEAR\}) \\
CA_3 &= (2, \{NATO, CIA\}) \\
CA_4 &= (3, \{NATO\}) \\
CA_5 &= (2, \{NUCLEAR\}) \\
CA_6 &= (1, \{\}) \\
CA_7 &= (1, \{NATO\})
\end{aligned}
$$

entonces

$$
\begin{aligned}
CA_1 &\succeq CA_2 && (1.3) \\
CA_1 &\succeq CA_4 && (1.4) \\
CA_1 &\succeq CA_5 && (1.5) \\
CA_1 &\succeq CA_6 && (1.6) \\
CA_1 &\succeq CA_7 && (1.7) \\
CA_1 &\not\succeq CA_3 && (1.8) \\
CA_3 &\not\succeq CA_1 && (1.9)
\end{aligned}
$$

$$CA_2 \succeq CA_6 \tag{1.10}$$
$$CA_2 \succeq CA_7 \tag{1.11}$$
$$CA_3 \succeq CA_6 \tag{1.12}$$
$$CA_3 \succeq CA_7 \tag{1.13}$$
$$CA_4 \not\succeq CA_3 \tag{1.14}$$
$$CA_3 \not\succeq CA_4 \tag{1.15}$$

Donde 1.3 se justifica dado que el nivel de CA_1 es mayor que el de CA_2 y los conjuntos de categorías son idénticos, mientras que 1.4 vale pues el conjunto de categorías de CA_1 incluye al de CA_4 en tanto los niveles son iguales. Dado que el nivel y el conjunto de categorías de CA_1 superan a los de CA_5, 1.5 es verdadera. Por otro lado, 1.8 y 1.9 valen pues los conjuntos de categorías de CA_1 y CA_3 son incomparables.

Es interesante notar que la segunda condición que define el orden entre clase de acceso, a saber

$$C_1 \supseteq C_2$$

es diferente a la regla que define el control de acceso en ACLs (ver 1.7) pues allí se pide que el usuario tenga *al menos una* entrada en la ACL que le permita acceder el archivo, mientras que en la política del DoD se exige que *todas* las categorías del documento sean también categorías del individuo. Esta es una de las razones por las cuales no es posible reducir la política del DoD a las ACL ni vice-versa.

1.5. Generalización de la política de seguridad del DoD

De la sección anterior surge que individuos e información están clasificados según un criterio que involucra dos dimensiones:

Jerárquica representada por el nivel de seguridad. Dado que se requiere que el conjunto de niveles esté linealmente ordenado se establece en esta dimensión una jerarquía de documentos e individuos. Los documentos clasificados en los niveles superiores de la jerarquía contendrían la información más crítica, secreta, valiosa, etc. para la organización; a medida que se desciende por la jerarquía la importancia o confidencialidad que se le asigna a la información decrece y por lo tanto puede ser vista por mayor cantidad de individuos. Simétricamente, los individuos a los que se les ha asignado un nivel de seguridad alto, se les ha conferido una "altura" tal que les permite ver gran parte de la jerarquía de documentos; cuanto más baja es la "altura" de un individuo menos le está permitido ver.

No jerárquica representada por el conjunto de categorías. Aunque un individuo puede tener un nivel de seguridad muy alto (o una gran "altura") no siempre necesita, en su trabajo cotidiano, ver toda la información que tal "altura" le permite. Por ejemplo, si un comandante (gran "altura") está asignado a áreas de la OTAN no debería tener la necesidad de leer información sobre la CIA. Por lo tanto, se combina con la dimensión jerárquica una no jerárquica que hace referencia a las áreas de interés de cada individuo o a las áreas temáticas que involucra cada documento.

De esta forma, se permite que cada individuo pueda leer toda la información que su "altura" le permita pero que sólo es de interés para su trabajo.

Entonces, cualquier organización que pueda clasificar su información y usuarios según una dimensión jerárquica y otra no jerárquica con la intención de reglamentar el acceso a aquella puede muy bien utilizar una generalización de la política del DoD y, más aun, echar mano de los modelos de seguridad para sistemas de cómputo que se derivan de aquella (1.6). La generalización que proponemos es muy simple y evidente: cambiar los nombres de niveles y categorías por otros que tengan significado en la organización de que se trate.

Ejemplo 1.5.1 *Consideremos una empresa típica en la cual existe una gerencia general, gerencias a cargo de ciertas áreas y departamentos dentro de cada una de estas. Si bien formalmente este organigrama parece rígido y sugiere que no es posible que un empleado de cierta oficina requiera, para su trabajo, información de oficinas en otras gerencias, todos sabemos que en la práctica esto no es así y que la información debe ser utilizada por oficinas en diferentes gerencias. Más aun, algunas empresas optan por una organización basada en proyectos donde empleados de diferentes oficinas cooperan para llevar adelante un proyecto el cual utiliza y genera información propia y corporativa en general.*

En consecuencia definiremos los siguientes nombres para ser utilizados en las clases de acceso:

Jerárquico: *GrteGral, Grte, Jefe, EmpGrcia, Emp, Pub*

No jerárquico: *GrciaGral, GrciaPers, GrciaFin, GrciaVentas, GrciaProd, VentasMayor, VentasMenor, ProdProdA, ProdProdB, CtaX, CtaY, CtaZ, DesProdC, MercadoAsia, etc.*

El orden textual en la dimensión jerárquica es equivalente al orden de mayor a menor. Las abreviaturas de las etiquetas tienen el siguiente significado:

GrteGral *gerente general de la empresa; información destinada únicamente al gerente general.*

Grte *gerente; indica que la información está destinada únicamente a los gerentes o GrteGral, o a un individuo que ocupa cargo de gerente esté o no al frente de una gerencia (por ejemplo si está al frente de un proyecto estratégico para la empresa).*

Jefe *de departamento; información destinada a los jefes de departamento o superiores, los individuos que posean este nivel serán jefes de área.*

EmpGrcia *empleado de gerencia; destinado a los empleados que trabajan junto a algún gerente pero que no están bajo el mando de un Jefe, podrán acceder información levemente superior a la que pueden acceder otros empleados.*

Emp *empleado; información o individuos que forman la base de la escala de mando.*

Pub *público en general; por ejemplo podrían etiquetarse los folletos de la empresa o los usuarios que acceden a través de un servicio web, etc.*

GrciaGral *gerencia general; categoría que identifica a individuos o información manejada por la gerencia general.*

GrciaPers *gerencia de personal; categoría que identifica a individuos o información manejada por la gerencia de personal.*

GrciaFin *gerencia financiera; categoría que identifica a individuos o información manejada por la gerencia financiera.*

GrciaVentas *gerencia de ventas; categoría que identifica a individuos o información manejada por la gerencia de ventas.*

GrciaProd *gerencia de producción; categoría que identifica a individuos o información manejada por la gerencia de producción.*

VentasMayor *categoría que identifica a individuos o información manejada por el departamento de ventas al por mayor.*

VentasMenor *categoría que identifica a individuos o información manejada por el departamento de ventas al por menor.*

ProdProdA *categoría que identifica a individuos o información manejada por el departamento encargado de la producción del producto A.*

ProdProdB *categoría que identifica a individuos o información manejada por el departamento encargado de la producción del producto B.*

CtaX *cuenta X; X es un cliente muy importante de la empresa que es tratado de manera especial por un equipo de empleados liderados por alguien con cargo de gerente aunque no está al frente de ninguna gerencia en particular.*

CtaY *cuenta Y; Y es un cliente muy importante de la empresa que es tratado de manera especial por un equipo de empleados liderados por alguien con cargo de gerente aunque no está al frente de ninguna gerencia en particular.*

DesProdC *desarrollo producto C; la empresa está desarrollando un nuevo producto y ha designado un equipo especial para tal efecto compuesto por empleados de varios departamentos que cruzan el organigrama estándar.*

MercadoAsia *la empresa intenta ampliar su mercado a Asia y para ello ha compuesto un grupo de tareas con ese objetivo*

Teniendo estos niveles y categorías, el gerente general de la empresa tendrá la clase de acceso

$(GrteGral, CATEGORIAS)$

donde $CATEGORIAS$ es el conjunto de todos los nombres de categorías definidos en la empresa. Mientras que el líder de la cuenta X tendrá clase de acceso

$(Grte, \{CtaX, VentasMayor, ProdProdA\})$

pues X compra grandes cantidades del producto A.

1.6. Modelos de seguridad MLS

MLS es acrónimo de *MultiLevel Security*. Los modelos MLS son un subconjunto de los modelos denominados Mandatory Access Control (MAC). Todos estos modelos de seguridad provienen de las investigaciones financiadas por el DoD en las décadas del '70 y '80, con el objetivo de implementar en sistemas de computó la política de seguridad de ese ministerio.

Dado que los modelos MLS son, también, modelos MAC, implica que en Lisex ciertos atributos de control de acceso sólo pueden ser alterados por un grupo de administradores del sistema.

Los modelos MLS son descripciones matemáticas y/o lógicas de adaptaciones a los sistemas de cómputo de la política de seguridad del DoD. El modelo MLS más conocido, estudiado e implementado es el descripto por D. Elliott Bell y Leonard J. La Padula en 1973 [BL73a, BL73b] como parte de las investigaciones que desarrolló Mitre Corp. para el DoD.

La idea esencial de estos modelos es resistir los ataques a la confidencialidad (1.1) por medio de caballos de Troya (1.2). Con este fin se toman conceptualmente tres medidas:

— Ciertos atributos de seguridad de los recursos[4] manejados por el sistema[5] sólo pueden ser alterados por administradores y no por los usuarios ordinarios

— Sobre el conjunto de archivos y procesos se define un orden parcial utilizando aquellos atributos de seguridad que sólo pueden ser modificados por los administradores.

[4] Archivos, directorios, procesos, etc.
[5] Sistema operativo, DBMS, servidor Web, etc.

— El sistema sólo permite que la información fluya en sentido ascendente según el orden mencionado, lo que se logra imponiendo ciertas restricciones sobre las operaciones básicas del sistema

Entonces, digamos que el usuario E es un espía que desea conocer el contenido de cierto documento. Si el usuario E está por debajo del documento[6], el sistema le impedirá el acceso. Por lo tanto, E instala un caballo de Troya con la intención que al ser ejecutado por un usuario que tiene acceso al documento, el programa logre ponerlo al su alcance. Sin embargo, el caballo de Troya no podrá hacer descender la información debido a que ningún proceso puede hacerlo, ni si quiera los del superusuario del sistema (ver más adelante en esta misma sección).

Un modelo MLS típicamente incluye una descripción abstracta del sistema de cómputo que se programará y una descripción abstracta de las propiedades de seguridad que se espera que el sistema posea:

— **Modelo abstracto del sistema de cómputo**

Recursos un conjunto de recursos de cómputo a proteger (usualmente archivos, dispositivos, etc.)

Procesos un conjunto de procesos capaces de acceder a dichos recursos

Modos un conjunto de modos de acceso (usualmente read y write)

Clase de acceso de objetos una función que asocia a cada recurso protegido una clase de acceso

Clase de acceso de procesos una función que asocia a cada proceso una clase de acceso

Operaciones un conjunto de operaciones que son las únicas que los procesos pueden ejecutar para acceder los recursos protegidos. Pueden existir, sin embargo, operaciones que escapan al control del sistema de protección pero que pueden ser usadas por los procesos únicamente una vez que están accediendo a los recursos (el ejemplo típico de esta situación son las operaciones sobre la memoria que permiten copiar datos de un buffer a otro sin intervención del núcleo del sistema pero que son significativas una vez que se ha abierto un archivo).

— **Propiedades del modelo**

MAC predicado que establece que las clases de acceso de procesos y recursos sólo pueden ser alterados por administradores y en condiciones especiales

Seguridad simple predicado que establece que cualquier proceso que está accediendo un recurso lo hace pues la clase de acceso del proceso domina a la del recurso. (1.6.2)

[6] En rigor debería decir "si la clase de acceso de E está por debajo de la del documento". Utilizaremos esta forma más coloquial cuando sea conveniente.

Confinamiento predicado que establece que si un proceso está accediendo varios recursos en modo de lectura y otros en modo de escritura, la clase de acceso de cada uno de los últimos domina a la clase de acceso de cualquiera de los primeros. (1.6.3)

Puede reveerse el ejemplo de un caballo de Troya en el contexto de un sistema MLS en la sección 1.6.1.

Claramente, un sistema MLS verifica dos tipos de permisos antes de otorgar acceso a un recurso:

Estático vinculado a la propiedad de Seguridad simple, que compara la clase de acceso del proceso con la del recurso, las que están almacenadas (persistentemente) en el sistema.

Dinámico vinculado a la propiedad de Confinamiento, que compara la clase de acceso del recurso a abrir con la de los otros recursos aun abiertos por el proceso.

Lisex 0.0 implementa en el sistema de archivos de Linux una adaptación del modelo de Bell-La Padula que hemos denominado Lisex Security Model 0.0 (LSM 0.0). Futuras versiones de Lisex implementarán otros modelos semejantes que abarcarán todo el núcleo de Linux.

La mayor parte de los modelos MLS han sido especificados y verificados utilizando métodos formales ya que se supone que luego serán implementados en sistemas destinados a organizaciones para las cuales la protección de la información es una actividad de gran valor y, en ocasiones, crítica. LSM0.0 no escapa a esa tendencia dado que fue especificado y verificado formalmente utilizando Coq (1.9).

Notar que, en tanto adaptaciones a sistemas de cómputo, los modelos MLS se diferencian de la política de seguridad del DoD en lo siguiente:

Confinamiento pues la política del DoD no impide que un individuo cuyo escritorio está plagado de documentos altamente clasificados escriba, por ejemplo, una carta para un amigo. Precisamente, el confinamiento *aparece* como necesario por características intrínsecas a los sistemas de cómputo [McL90].

Modo de escritura que no es ni si quiera mencionado en la política del DoD dado que esta se orienta básicamente a la confidencialidad. El problema de saber si la información contenida en un documento es cierta o correcta se reduce a un problema de *autenticación*: saber con certeza que quien escribió el documento es quien figura como su autor y en tal caso que el autor es una autoridad en el tema.

1.6.1. Comercio electrónico en sistemas MLS

Aquí revisaremos el ejemplo introducido en la sección 1.2.1.

Supongamos que el sistema operativo instalado en el servidor del sitio de comercio electrónico implementa alguna forma de seguridad MLS. Digamos que el archivo que contiene los números de tarjetas de crédito posee una clase de acceso superior a la del dispositivo de comunicación conectado a Internet (por ejemplo, el dispositivo `eth0`). El caballo de Troya programado por los atacantes no podrá enviar el contenido de ese archivo hacia el exterior dado que el S.O. lo impedirá por la relación existente entre la clase de acceso de la información y la del dispositivo[7]. Ni siquiera será posible que lo haga ejecutando como el super-usuario.

1.6.2. Seguridad simple

Un sistema MLS impone la siguiente regla de control de cesó en el momento en que un proceso solicita la apertura (`open`) de un recurso:

— Un proceso puede abrir un recurso (en cualquier modo) siempre y cuando la clase de acceso del proceso domine a la clase de acceso del recurso.

Ejemplo 1.6.1 *Digamos que F, P, V y L son las etiquetas de categoría asignadas a los departamentos de $FINANZAS$, $PERSONAL$, $VENTAS$ y $LOGISTICA$, respectivamente. Sea $(3, \{F, P, V\})$ la clase de acceso del proceso P. La tabla 1.1 muestra algunos ejemplos de recursos que P puede y no puede abrir debido a que el sistema impone la propiedad de seguridad simple.*

Debe tenerse en cuenta que aunque la regla de seguridad simple indique que en cierta situación un proceso puede acceder a un recurso esto no significa necesariamente que pueda hacerlo ya que el sistema impone, además, la regla de confinamiento (1.6.3) que, en el caso general, impide ciertas solicitudes de apertura.

1.6.3. Confinamiento

Un sistema MLS impone la siguiente regla de control de acceso en el momento en que un proceso solicita la apertura (`open`) de un recurso:

— Un proceso puede abrir un recurso en modo read siempre y cuando la clase de acceso de cualquier recurso abierto en modo write domine a la del recurso que se quiere abrir.

— Un proceso puede abrir un recurso en modo write siempre y cuando la clase de acceso de este recurso domine a la de cada recurso abierto en modo read.

La intención de esta regla es evitar que una vez abierto cierto recurso con una clase de acceso alta se abran otros recursos con clases de acceso inferiores y se copie el contenido del recurso "alto" en los "bajos". Subyacen a esta idea dos supuestos

[7]Por otro lado, el dispositivo de comunicación con la red corporativa podría tener una clase de acceso más alta, permitiendo así enviar esos datos a otros equipos dentro de la empresa, pero nunca hacia fuera.

Recurso	¿Puede?	¿Por qué?
$(1, \{F, P, V\})$	Sí	El nivel del recurso es inferior al de P mientras que los conjuntos de categorías son idénticos, por lo cual la clase de acceso de P domina a la del recurso
$(3, \{F, V\})$	Sí	Los niveles son iguales mientras que el conjunto de categorías del recurso está incluido en el de P, por lo cual la clase de acceso de P domina a la del recurso
$(3, \{\})$	Sí	El nivel del recurso es igual al de P mientras que el conjunto de categorías del recurso es vacío, por lo cual la clase de acceso de P domina a la del recurso
$(4, \{V, F, P\})$	No	El nivel del recurso es mayor al de P, por lo cual la clase de acceso de P no domina a la del recurso
$(3, \{V, P, F, L\})$	No	Aunque el nivel del recurso es igual al de P, el conjunto de categorías de P no incluye al del recurso, por lo cual la clase de acceso de P no domina a la del recurso
$(2, \{L, V, P\})$	No	Aunque el nivel del recurso es inferior al de P, los conjuntos de categorías son incomparables, por lo cual la clase de acceso de P no domina a la del recurso

Tabla 1.1: Ejemplos de aplicación de la propiedad de seguridad simple. La clase de acceso a la que hace referencia la columna izquierda corresponde a la de distintos recursos que P intentaría abrir. Tener en cuenta que el modo en que P intenta abrir el recurso es irrelevante para la seguridad simple.

1. se han considerado dentro del conjunto de recursos a proteger sólo algunos recursos del sistema y no todos, por ejemplo, la memoria usada por los procesos

2. el control de acceso se impone en tiempo de `open`.

En general esto es así, debido a que la implementación de controles semejantes sobre la memoria es mucho más complejo dado que el flujo de información a través de ella escapa al control del núcleo del sistema [McL90].

Del enunciado de esta regla se deduce que no se puede determinar si un proceso pasará el control de confinamiento al intentar abrir un recurso con sólo analizar la relación entre la clase de acceso del proceso y la del recurso. Precisamente, la clase de acceso del proceso no juega, en esta regla, ningún papel.

Ejemplo 1.6.2 *Para dar algunos ejemplos sobre la aplicación de la regla de confinamiento es necesario considerar la clase de acceso del recurso a abrir y la clase de acceso de cada recurso abierto en uno u otro modo según sea el caso.*

En la tabla 1.2 la primer columna de la izquierda indica la clase de acceso del recurso a abrir por cierto proceso P y la segunda el modo en que se intenta abrir el recurso. En la tercera columna se consigna el estado del proceso en cuanto a los archivos abiertos: un par de la forma $[CA, m]$ indica que el proceso aun tiene abierto un recurso con clase de seguridad CA en modo m. Si el proceso no tuviera recursos abiertos, no sería necesario aplicar confinamiento.

La cuarta y quinta columnas indican si el proceso podrá o no abrir el recurso en cuanto a la regla de confinamiento refiere y la razón de tal respuesta, respectivamente. Las categorías utilizadas son las mismas que en el ejemplo de la página 18

Recurso	**M**	**Estado**	**R**	**¿Por qué?**
$(3, \{F, P, V\})$	*	$[(3, \{F, P, V\}), *]$	S	Clases de acceso idénticas, no puede haber flujo de información ilegal
$(3, \{F, P, V\})$	R	$[(2, \{F, V\}), RW]$	N	Podrían copiarse datos del nuevo recurso al ya abierto, flujo ilegal
$(3, \{F, P, V\})$	W	$[(2, \{F, V\}), RW]$	S	No puede haber flujo ilegal, sólo puede escribirse "hacia arriba"
$(3, \{F, P, V\})$	RW	$[(2, \{F, V\}), R]$	S	Copiar datos del recurso ya abierto al nuevo no produce flujo ilegal
$(1, \{F, P, V\})$	R	$[(3, \{F, P, V\}), R]$	S	No puede haber flujo de información en ningún sentido
$(1, \{F, P, V\})$	W	$[(3, \{F, P, V\}), R]$	N	Puede haber flujo de datos hacia el nuevo recurso
$(1, \{F, P, V\})$	R	$[(3, \{F, P, V\}), RW]$	S	Los datos sólo pueden fluir "hacia arriba"
$(1, \{F, P, V\})$	W	$[(3, \{F, P, V\}), RW]$	N	Leer del recurso ya abierto y escribir en el nuevo, flujo ilegal
$(1, \{F, P, V\})$	RW	$[(3, \{F, P, V\}), RW]$	N	Idem anterior
$(2, \{L, V, P\})$	R	$[(3, \{F, P, V\}), RW]$	N	Clases de acceso incomparables, cualquier flujo de datos es ilegal
$(2, \{L, V, P\})$	R	$[(3, \{F, P, V\}), R]$	S	No puede haber flujo de datos
$(2, \{F, P\})$	W	$[(3, \{F, P, V\}), R]$, $[(1, \{F, P, V\}), R]$	N	Posible flujo ilegal desde el primer recurso abierto hacia el nuevo

Tabla 1.2: Los ejemplos que se muestran son muy simples por cuanto el estado del proceso sólo consta de uno o dos recursos abiertos. R: Read, W: Write, RW: Read-Write, *: cualquier modo

Confinamiento al abrir un archivo en modo read-write

Un caso muy importante a tener en cuenta es aquel en el cual el proceso P tiene abierto el recurso f en modo read-write pues luego P sólo podrá abrir recursos en modo:

— read-write con la misma clase de acceso de f

— read con clase de acceso dominada por la de f

— write con clase de acceso que domina a la de f

Este es el caso típico en sistemas UNIX debido a que la terminal (lógica) que utiliza cualquier usuario es un recurso abierto en modo read-write. Por lo tanto, luego de que el usuario ingrese al sistema, podrá utilizar recursos que cumplan alguna de las condiciones anteriores. Claramente, el usuario antes de loguearse debe estimar en cuál terminal es conveniente hacerlo.

Indudablemente esta es un restricción que puede entorpecer el trabajo cotidiano, motivo por el cual el equipo de Lisex planea implementar modelos MLS menos restrictivos pero igualmente seguros (ver [CDG+02]). Sin embargo, en algún sentido, detrás de esta restricción está la esencia misma del significado de seguridad en este tipo de sistemas: el input de usuarios de alto nivel de seguridad no debe llegar como output a usuarios de bajo nivel de seguridad.

Cabe, sin dudas, un refinamiento del enunciado anterior: el input de *alto nivel* generado por usuarios de *alto nivel* no debe llegar como output a usuarios de *bajo nivel* de seguridad. Implícitamente, esta definición de seguridad en sistemas MLS dice que no todo el input de un usuario de alto nivel es de alto nivel. Por ejemplo, no es lo mismo escribir un informe confidencial de la empresa que enviar un mail a un amigo, diferencia que, para un sistema de cómputo, no es fácil de discriminar.

1.7. Control de acceso discrecional

La mayoría del software de base comercial implementa alguna forma de control de acceso discrecional[8]. Un mecanismo de control de acceso es discrecional si los usuarios ordinarios pueden participar en la definición de la política de control de acceso. Este es el caso de UNIX dado que cualquier usuario puede definir los atributos de seguridad de los archivos que posee y, por lo tanto, tiene la capacidad de negar y brindar acceso a los datos de su propiedad.

Uno de los mecanismos DAC más versátiles son las listas de control de acceso (*Access Control List*, ACL) el cual asocia a cada archivo una lista con los usuarios y grupos de usuarios que pueden acceder el archivo. La propiedad básica que define a este modelo es que si un usuario tiene abierto un archivo en cierto modo, se debe a que en el momento de solicitar la apertura él o alguno de los grupos a los que él pertenece figuran en la ACL del archivo con el modo requerido.

Hemos denominado al modelo DAC basado en ACLs modelo DAC extendido debido a que puede pensarse que el modelo DAC de UNIX es una ACL de longitud y estructura fijas.

1.8. Integración entre el modelo MLS y el modelo DAC

Las razones por las que se incluyen o implementan ambos modelos en el sistema son muy diferentes en cada caso:

DAC compatibilidad con Linux, brindar mayor libertad a los usuarios para compartir información

MLS resistir ataques contra la confidencialidad basados en caballos de Troya

[8]Discretionary Access Control, DAC

Por lo tanto, surge la necesidad de responder a la pregunta ¿cómo debe actuar el sistema al controlar el acceso siendo que implementa dos modelos con objetivos muy distintos? Usualmente la respuesta es: un proceso podrá acceder un recurso sí y sólo sí el proceso sortea exitosamente *todas* las reglas siguientes:

— el UID a nombre del cual ejecuta el proceso tiene, en la ACL del recurso, una entrada de la forma (UID, M) o de la forma (GID, M) donde GID es un grupo al cual pertenece UID y M es el modo en que se quiere abrir el recurso

— el proceso verifica seguridad simple con respecto al recurso

— el proceso verifica confinamiento con respecto al recurso que se intenta acceder y en relación con todos los otros recursos que aun tiene abiertos.

En otras palabras, el sistema conjuga ambos modelos de manera tal que se obtiene resistencia a los caballos de Troya y se impone la política de acceso habitual en sistemas UNIX. Obviamente, un sistema de estas características es más restrictivo que un sistema que impone sólo uno de los modelos (es decir, considera una meta política que se expresa con una disyunción).

1.9. Métodos formales en Lisex

Como es posible que para muchos desarrolladores el término "método formal" resulte desconocido sugerimos revisar [CW96] o dirigirse al sitio reservado a métodos formales de la Universidad de Oxford (`http://www.comlab.ox.ac.uk/archive/formal-methods/`) para tener una idea de qué se trata. Luego podrían visitar el sitio de Coq que es el método formal más utilizado en Lisex.

Sin entrar en detalles diremos que aplicar un método formal al desarrollo de un sistema de software implica la utilización más o menos intensiva de variadas técnicas matemáticas y/o lógicas durante varias fases del desarrollo particularmente durante la especificación de requerimientos y verificación.

Hasta el momento se han aplicado métodos formales en cuatro etapas del desarrollo de Lisex:

— Especificación de un modelo abstracto de un sistema de archivos compatible con Linux. Se define el estado del sistema de archivos como un registro que comprende los elementos relevantes a la seguridad (archivos, directorios, clases de acceso, ACLs, archivos abiertos, etc.). Se definen operaciones que representan cada una de las llamadas al sistema que tienen relación con el sistema de archivos. Cada operación se define constructivamente mediante un tipo inductivo en términos de sus precondiciones y postcondiciones. Existen dos tipos de operaciones:

Modificación Aquellas operaciones que llevan el sistema de un estado a otro (por ejemplo, `chmod`)

Consulta Aquellas operaciones que solicitan datos sobre el estado del sistema pero no tienen la capacidad de efectuar un cambio de estado (por ejemplo, `stat`)

— Especificación de las propiedades de seguridad que se espera que el sistema de archivos posea. Se escribe un predicado por cada propiedad que se desea. En general, cada predicado define el significado de estado seguro del sistema desde un punto de vista diferente a los demás. En LSM 0.0 se definieron cuatro predicados:

  — Seguridad discrecional

  — Seguridad simple

  — Confinamiento

  — Cambio de atributos de seguridad

— Verificación del modelo de seguridad. El objetivo es demostrar formalmente que cada operación que parte de un estado seguro, según cada una de las propiedades anteriores, deja al sistema en un estado seguro.

— Testing de la implementación basado en el modelo formal. Se analiza sintácticamente el modelo formal y se generan casos de prueba a nivel del modelo que representan cada uno de los diferentes caminos sintácticos los que, a su vez, representan las diferentes alternativas funcionales. Luego, cada uno de estos casos se proyecta a un caso de prueba a nivel de la implementación. Gran parte de este proceso es automático.

Sugerimos leer el apéndice A antes de continuar a fin de tener una idea general sobre la usabilidad de este tipo de sistemas.

Capítulo 2

Cambios semánticos

El desarrollo de Lisex en general y de Lisex 0.0 en particular tiene como uno de sus objetivos preservar la compatibilidad con Linux tanto como sea posible. Sin embargo, otro de los objetivos es implementar un sistema operativo en el cual la seguridad se ha tenido en cuenta desde el comienzo, principio que, creemos, no se ha aplicado (siempre) en Linux. Por lo tanto, decidimos introducir ciertos cambios semánticos en conceptos vinculados a la seguridad del sistema en tanto no afecten drásticamente la compatibilidad entre Lisex y Linux.

En este capítulo se introducen los cambios mencionados que se relacionan con el permiso de ejecución, el modo de archivos y directorios para el resto de los usuarios (es decir, la tercer terna de bits de permisos), la cuenta `root`, el mecanismo SUID y el concepto de dueño y grupo de un archivo o directorio. Aunque no en todos los casos los cambios son significativos recomendamos leer atentamente este capítulo.

Por otro lado, en este capítulo no se incluye una descripción del cambio semántico más importante que implementa Lisex 0.0, es decir el modelo MLS, pues no existía previamente en Linux.

2.1. Permiso de ejecución

La semántica del permiso de ejecución ha sido modificada en su relación con la ejecución de programas y el acceso a directorios.

En general se ha subsumido el bit de ejecutabilidad en el de lectura. En otras palabras para:

Programas

— El sistema registra únicamente si el archivo es o no ejecutable.

— Si un usuario solicita su ejecución el sistema verifica que este tenga permiso de lectura sobre aquel

— (6.1)

Directorios

— El sistema no considera en absoluto este permiso para los directorios.

— Si un usuario tiene permiso de lectura sobre el directorio podrá listar su contenido y utilizar `cd` para ingresar a él.

— Listar su contenido significa que podrá ver los nombres de archivos y directorios que contiene.

— Si un usuario intenta ver otros atributos de un archivo, el sistema tiene en cuenta los permisos sobre el archivo y no sobre el directorio. (6.6 y 6.8)

2.2. Modo para el resto de los usuarios

En Linux los tres últimos bits en el modo de archivos y directorios se aplican al resto de los usuarios. Se entiende por resto de los usuarios a todos los usuarios del sistema que no son el dueño del archivo o directorio ni pertenecen al grupo asociado al archivo o directorio.

En Lisex 0.0 los tres últimos bits en el modo de archivos y directorios se aplican al grupo de usuarios `allgrp`, más precisamente al grupo con GID 666. La intención es que todos los usuarios del sistema pertenezcan a este grupo. En Lisex 0.0 la pertenencia a este grupo debe ser explícita, es decir que el administrador debe incluir a cada usuario en el grupo. (7.2.1).

Por otro lado, Lisex implementa ACL's en lugar del esquema de permisos UNIX (Dueño/Grupo/Otros). Entonces, el grupo `allgrp` puede o no estar en la ACL de un archivo o directorio con lo cual es posible darle o revocarle permisos a todos los usuarios sobre ese archivo o directorio (6.7).

Nota 2.2.1 *En versiones posteriores de Lisex la pertenencia al grupo* `allgrp` *no será explícita y por lo tanto no será necesario que el administrador incluya a cada usuario en este grupo.*

2.3. Dueño y grupo de un archivo o directorio

Debido a que Lisex 0.0 incorpora listas de control de acceso (ACL) para los archivos o directorios, el concepto de dueño de un archivo o directorio se extiende a una lista de usuarios y/o grupos dueños. De igual forma, ya no hay un único grupo asociado al archivo o directorio sino que puede haber múltiples grupos asociados cada uno con un conjunto diferente de permisos.

De esta forma, cualquiera de los usuarios o grupos dueños de un archivo o directorio tienen la capacidad de alterar su ACL. Más aun, esta es la única tarea que un dueño tiene capacidad de realizar en tanto dueño; es decir, que por el hecho de ser dueño no tiene, necesariamente, el permiso de leer o modificar el archivo o directorio que posee aunque tiene la capacidad de (auto)asignarse cualquiera de esos permisos.

Si un grupo es dueño de un archivo o directorio implica que todos los miembros de ese grupo son dueños del archivo o directorio (3.1).

2.4. Atributos de seguridad de recursos abiertos

Lisex 0.0 impide cambiar los atributos de seguridad de archivos y directorios abiertos. Esto implica que llamadas como `fchmod` fallarán siempre. Este comportamiento está en relación directa con las propiedades de seguridad que se quiso que Lisex 0.0 verificara. En general, se exige que si un usuario tiene abierto un archivo en cierto modo se debe a que ese usuario tiene los permisos necesarios para abrirlo. Notar que es *tiene* y no *tuvo*. Por lo tanto, si se permitiera cambiar los atributos de seguridad de archivos abiertos existiría la posibilidad de que en un momento dado un usuario tenga un archivo abierto sin tener (en ese mismo momento) los permisos necesarios.

Ejemplo 2.4.1 *Digamos que el proceso P tiene abierto el archivo* `highfile.tex`*. Además, $CA($*`highfile.tex`*$) = CA(P) = (SECRET, \{CITYCORP, IMF\})$. Entonces, digamos que se cambia la clase de acceso de* `highfile.tex` *a $(SECRET, \{CITYCORP, IMF\})$ sin que P lo cierre. En ese caso, P dejaría de verificar la propiedad de seguridad simple, lo que significa que el sistema en general entraría en un estado inseguro.*

Por otro lado, Linux y otras variedades UNIX históricamente han permitido este comportamiento. Por lo tanto, durante el desarrollo se debió decidir si se preservaba este comportamiento o se lo modificaba para respetar las propiedades esperadas. Obviamente, se eligió modificarlo más que nada con el fin de probar qué tanto impacto tenía sobre el sistema.

Notas 2.4.1

— *Futuras versiones de Lisex posiblemente permitan modificaciones de los atributos DAC de archivos abiertos.*

— *Permitir lo mismo para los atributos MAC iría en contra de la filosofía de los modelos MLS, por lo que Lisex no progresará por ese camino.*

2.5. Cuenta root

En esta sección cada vez que se mencione Lisex 0.0 se está mencionando, implícitamente, el sistema de archivos de Lisex 0.0 y no el resto del núcleo.

Lisex intenta equiparar la cuenta `root` con el grupo `root`. Es decir, se intenta que no sólo haya un usuario capaz de efectuar las tareas propias de la cuenta `root` sino todo un grupo de usuarios.

Lisex 0.0 está basado sobre el núcleo de Linux 2.4.14. Este núcleo incluye un mecanismo de seguridad denominado *capacidades* (*capabilities*). Cualquier cuenta de usuario adquiere privilegios al tener capacidades. `root`, obviamente, posee todas las capacidades. En Lisex 0.0, el mecanismo de capacidades fue removido del sistema de archivos. En versiones anteriores del núcleo de Linux, `root` tenía mayores privilegios debido a que internamente el núcleo trataba de forma diferente al UID 0.

Lisex 0.0 retoma la filosofía de los núcleos anteriores pero en lugar de aplicarla al UID 0 lo hace sobre el GID 0. Concretamente, `root` adquiere privilegios por pertenecer al grupo `root` (GID 0), caso contrario carece de privilegios especiales sobre archivos y directorios. Al mismo tiempo, es muy importante tener en cuenta que los privilegios especiales de `root`, o más precisamente del grupo `root`, se relacionan únicamente con el control de acceso discrecional (1.7). En otras palabras, ningún miembro del grupo `root` (incluido `root` mismo) podrá violar las reglas MAC (1.3), seguridad simple (1.6.2) y confinamiento (1.6.3).

Concretamente, la implementación de Lisex 0.0 incluye al grupo `root` como dueño en la ACL de cada archivo o directorio del sistema e impide que este grupo sea removido de la ACL de cualquier archivo o directorio. Más precisamente:

— cada vez que un archivo o directorio es creado mediante `creat` o `mkdir`, el sistema incluye de forma automática al grupo `root` como dueño de tal archivo o directorio

— no existe ninguna llamada al sistema que elimine al grupo `root` como grupo dueño de una ACL (en particular la nueva llamada al sistema `acldel` tiene como precondición no eliminar el GID 0 de ninguna ACL sea quien fuere el usuario que lo solicita).

Por lo tanto, dado que el grupo `root` es dueño de todos los archivos y directorios y que no existe forma de evitar esto, cualquier miembro de ese grupo puede acceder, al menos en lo que a DAC se refiere, cualquier archivo o directorio. Precisamente, estos privilegios especiales no le garantizan a `root` ni a ningún otro miembro del grupo `root` que puedan, por ejemplo, borrar todos los archivos del sistema, leer cualquier archivo, etc. pues aun está el control MLS.

En particular `root` ni ningún otro miembro del grupo `root` pueden cambiar las clases de acceso de ningún archivo, directorio ni cuenta de usuario, incluyendo la propia. Estos privilegios corresponden a la porción MLS del modelo de seguridad que implementa Lisex 0.0 y por lo tanto están bajo la jurisdicción de los "oficiales MAC" (7.4.2 y 7.4.4).

Dado que uno de los privilegios extra de `root` en Linux es la posibilidad de ejecutar de forma irrestricta la llamada `setuid` se recomienda leer la sección 2.6.

2.6. Mecanismo SUID

La semántica de la llamada `setuid` se fortaleció de la siguiente forma: la llamada cambiará el UID según le es solicitado si, además de cumplirse las condiciones habituales de Linux, el nuevo UID del proceso verifica seguridad simple (1.6.2) con respecto a todos los archivos abiertos por el proceso. Claramente, esta nueva semántica impide que `root` ejecute en cualesquiera circunstancias la llamada `setuid`.

Ejemplo 2.6.1 *Supongamos que un usuario cualquiera intenta ingresar al sistema utilizando una terminal lógica que, por seguridad simple, no podría utilizar. Entonces, el programa* `login`*, que ejecuta con UID 0, luego de autenticar al usuario llamará a* `setuid` *pero esta retornará con error ya que* $CA(u) \not\succeq CA(term)$*, por lo que el usuario no podrá loguearse.*

Para la llamada `setuid`, `root` no es equivalente al grupo `root`; es decir, ningún miembro de ese grupo, excepto `root`, podrá utilizar de forma especial esta llamada.

Notas 2.6.1

1. *En futuras versiones de Lisex la semántica de la llamada* `setuid` *se podría debilitar al equiparar la cuenta* `root` *con el grupo* `root`*. Es decir, cualquier integrante del grupo* `root` *podría ejecutar la llamada* `setuid` *para lanzar procesos a nombre de cualquier otro usuario.*

2. *La llamada* `setuid` *aun utiliza el mecanismo de capacidades.*

Capítulo 3

Clases de acceso y ACL

3.1. Estructura de las clases de acceso

En Lisex 0.0 la estructura de las clases de acceso es idéntica a la indicada en la sección 1.4 y siguientes. Sin embargo, no se implementaron etiquetas legibles por el usuario para las categorías o niveles utilizados por el sistema (*human readable labels*).

Nota 3.1.1 *En futuras versiones de Lisex se implementarán etiquetas legibles por el usuario para las categorías o niveles utilizados por el sistema.*

En esta distribución se utilizan niveles de seguridad en el intervalo $[1, 5]$ y categorías del conjunto $\{0, 1, 2, 3, 4, 5, 6, 65535\}$. El nivel de seguridad 0 no debe utilizarse, aunque no causaría ningún problema hacerlo, pues se lo reserva para situaciones anómalas dentro del núcleo del sistema. Por lo tanto, la clase de acceso mínima utilizada es $(1, \{\})$ y la máxima $(5, \{0, 1, 2, 3, 4, 5, 6, 65535\})$.

Nota 3.1.2 *Observar que la clase de acceso* $(1, \{0\})$ *es diferente y domina (estrictamente) a la clase de acceso* $(1, \{\})$ *pues el conjunto de categorías de la primera posee un elemento (0) mientras que el de la segunda está vacío.*

Se espera que los niveles de seguridad sean utilizados de la siguiente forma (aunque en cada instalación los administradores pueden definir la forma que ellos consideren adecuada, recomendamos (7.4)):

Nivel 1 se reserva para los oficiales MLS (`secadm` y otros usuarios), no debe haber programas que tengan este nivel excepto aquellos que están certificados para ser usados por estos administradores (7.4).

Nivel 2 es el primer nivel para usuarios ordinarios por lo que todos aquellos programas que se espera que sean utilizados por todos los usuarios deben tener este nivel. Además, se supone que no habrá archivos ordinarios con un nivel inferior a 2. De hecho, al crear un archivo el sistema le asigna automáticamente la clase de acceso del usuario que lo crea. (6.3)

Nivel 3 en adelante se pueden usar para jerarquizar a los distintos usuarios ordinarios (ejemplo 1.5.1).

La categoría 65535 es la única categoría reservada por el sistema. Si un usuario posee esta categoría podrá ejecutar las llamadas al sistema `chobjsc` y `chsubsc` que permiten modificar y asignar clases de acceso a archivos y usuarios, respectivamente (7.4.4 y 7.4.2).

Notas 3.1.3 *En Lisex 0.0 el conjunto de categorías de cada clase de acceso*

— *es de longitud fija configurable en tiempo de compilación por medio de la constante* `MAX_CAT_LEN` *definida en* `sc.h`*; el valor actual es 8 [GM02].*

— *está implementado con un arreglo de longitud fija igual para todos los i-nodos y todas las cuentas de usuario por lo cual si se aumenta la longitud habrá una utilización ineficiente del espacio en disco.*

3.2. Clases de acceso de usuarios y procesos, interacción con el mecanismo SUID y SGID

Tal como se explica en las secciones 1.4 y 1.6 cada cuenta de usuario del sistema tiene asociada una clase de acceso. Tal clase es heredada por todos los procesos lanzados por esa cuenta de usuario a menos que el proceso sea lanzado con un UID diferente, es decir, a menos que se invoque la llamada al sistema `setuid`, en cuyo caso el proceso hereda la clase de acceso del nuevo UID.

El uso del mecanismo SGID no tiene influencia en la clase de acceso del proceso que se lanza.

3.3. Estructura y semántica de las ACL

Una ACL es una lista de ternas ordenadas. Cada terna es de la forma:

$$(U\_G\_ID, type, mode)$$

donde

U_G_ID es un entero que identifica a un usuario o grupo de usuarios

type : indica si el entero de la componente U_G_ID corresponde a un usuario o grupo; se usan las constantes ACL_GRP y ACL_USR, respectivamente.

mode son los permisos que el usuario o grupo tienen sobre el archivo o directorio al cual está asociada la ACL. Este campo se construye relacionando con XOR los permisos elementales: NULL, READ, WRITE, EXEC y OWNER.

Si un grupo tiene una entrada en una ACL significa que cualquier integrante de ese grupo tiene los permisos indicados en la entrada del grupo. En otras palabras, si el grupo g está formado por los usuarios $u_1, u_2, ..., u_n$ y en la ACL a hay una entrada de la forma (g, ACL_GRP, M), entonces es equivalente a reemplazar la entrada de g por n entradas de la forma (u_i, ACL_USR, M).

El significado de cada permiso elemental es muy similar al de Linux:

NULL el usuario o grupo no posee ningún permiso, lo que es equivalente a que el usuario o grupo no está en la ACL

READ el usuario o grupo puede leer el archivo o directorio (2.1)

WRITE el usuario o grupo puede escribir el archivo o directorio (para un directorio esto significa que el usuario o grupo podrá crear y eliminar archivos o directorios)

EXEC el archivo es ejecutable (2.1)

OWNER el usuario o grupo puede modificar la ACL del archivo o directorio, es decir puede agregar o eliminar ternas

Notas 3.3.1

1. *En Lisex 0.0 las ACL*

   — *son de longitud fija configurable en tiempo de compilación por medio de la constante* `MAX_ACL_LEN` *definida en* `fs_acl.h`*; valor actual 8.*

   — *están implementadas con arreglos de longitud fija igual para todos los i-nodos por lo cual si se aumenta la longitud habrá una utilización ineficiente del espacio en disco.*

2. *La estructura de las ACL a nivel del modelo no es igual a la que se eligió para la implementación pero se corresponden unívocamente.*

3.3.1. Compatibilidad de la ACL con el modo

Claramente, una ACL extiende el concepto de modo por lo que fue necesario buscar una forma de preservar lo más posible la compatibilidad hacia atrás. Las reglas de compatibilidad son las siguientes.

Dueño Como en Lisex 0.0 las ACL pueden contener más de un dueño, al dueño tradicional se lo denomina *dueño UNIX*.

Si el dueño UNIX es removido de la ACL por medio de la llamada al sistema `acldel`, el sistema pone como dueño UNIX al usuario `root` (6.7.4).

Permiso UNIX	**Entrada en la ACL**
000	$(UID\ owner, ACL\_USR, OWNER)$
400	$(UID\ owner, ACL\_USR, OWNER \mid READ)$
200	$(UID\ owner, ACL\_USR, OWNER \mid WRITE)$
600	$(UID\ owner, ACL\_USR, OWNER \mid READ \mid WRITE)$

Tabla 3.1: Compatibilidad ACL-modo para el dueño UNIX

La tabla 3.1 resume la correspondencia entre modo y entradas en la ACL para el dueño UNIX. Por ejemplo, si el dueño UNIX tiene permiso de read-write entonces habrá una entrada de la forma $(UID\,UNIX\ owner, ACL\_USR, OWNER \mid READ \mid WRITE)$ que indica que ese usuario es dueño (ninguno en particular) y tiene permiso de read-write.

Grupo Como en Lisex 0.0 las ACL pueden contener más de un grupo, al grupo tradicional se lo denomina *grupo UNIX*. La tabla 3.2 resume la correspondencia entre modo y entradas en la ACL para el grupo UNIX.

Permiso UNIX	**Entrada en la ACL**
000	No tiene entrada
040	$(GID\ grupo, ACL\_GRP, READ)$
020	$(GID\ grupo, ACL\_GRP, WRITE)$
060	$(GID\ grupo, ACL\_GRP, READ \mid WRITE)$

Tabla 3.2: Compatibilidad ACL-modo para el grupo UNIX

Resto Como ya se explicó el resto de los usuarios corresponde al grupo `allgrp` (GID 666) que contiene a todos los usuarios del sistema. La tabla 3.3 resume la correspondencia entre modo y entradas en la ACL para el resto de los usuarios.

Permiso UNIX	**Entrada en la ACL**
000	No tiene entrada
004	$(666, ACL\_GRP, READ)$
002	$(666, ACL\_GRP, WRITE)$
006	$(666, ACL\_GRP, READ \mid WRITE)$

Tabla 3.3: Compatibilidad ACL-modo para el resto de los usuarios

Capítulo 4

Instalación y arranque

El proceso de instalación que describimos en este capítulo supone que quien lo realiza tiene buenos conocimientos y experiencia con Linux y las herramientas asociadas. Además, suponemos que Lisex 0.0 compartirá un equipo junto a otros sistemas operativos, en particular alguna distribución de Linux. Obviamente este último requisito no es imprescindible; Lisex 0.0 puede instalarse como único sistema operativo, pero en tanto esta distribución no es más que un prototipo de prueba, suponemos que nadie destinará un equipo exclusivamente para este fin.

4.1. Descripción de los paquetes de software

Esta distribución cuenta con tres paquetes de software:

Núcleo compilado (`lisex0.0`) listo para ser instalado. Este núcleo es el resultado de modificar el núcleo 2.4.14 de Linux según se describe en [GM02].

Imagen del sistema de archivos (`lisexfs0.0.img.bz`) de Lisex 0.0 conteniendo los programas y archivos de un Linux mínimo, más algunos comandos propios de Lisex 0.0. Debido a que una de las modificaciones principales fue cambiar la estructura del nodo-i de `ext2` y a que aun no contamos con herramientas de instalación automáticas, nos vemos en la necesidad de entregar una imagen de un sistema de archivos Lisex 0.0.

Fuentes (`lisex0.0.tar.bz`) del núcleo y demás aplicaciones que fueron modificadas o creadas por el equipo Lisex.

4.2. Proceso de instalación y arranque

Todos estos pasos deben ser ejecutados por `root`.

1. Iniciar el equipo con Linux.

2. Copiar, utilizando `bzcat`, en una partición sin formato de al menos 12Mb la imagen del sistema de archivos de Lisex 0.0 provisto con la distribución.

   Por ejemplo, si la partición está en el dispositivo `/dev/hda6` y la imagen del sistema de archivos de Lisex 0.0 en el directorio `/tmp/lisex/` se debe ejecutar el comando

   ```
   bzcat /tmp/lisex/lisexfs0.0.img.bz >/dev/hda6
   ```

3. Copiar el núcleo compilado en el directorio Linux donde normalmente se encuentran otras imágenes de núcleos. Por ejemplo, en RedHat este directorio es `/boot`.

4. Instalar el núcleo compilado de Lisex 0.0 en su administrador de arranque (*boot loader*).

   Por ejemplo, si el administrador de arranque es Lilo y la partición donde se copió la imagen del sistema de archivos es `/dev/hda6`:

   a) Editar el archivo `/etc/lilo.conf`

   b) Agregar al final del archivo las siguientes líneas:

   ```
   image=/boot/lisex0.0
   label=Lisex 0.0
   read-write
   root=/dev/hda6
   ```

   c) Ejecutar el comando `lilo`.

   Otro ejemplo es `grub`:

   a) Editar el archivo `/boot/grub/grub.conf`

   b) Agregar al final del archivo las siguientes líneas:

   ```
   title Lisex 0.0
         kernel /boot/lisex0.0 rw root=/dev/hda6
   ```

5. Re-iniciar el equipo.

6. En el menú de su administrador de arranque elija arrancar con el núcleo de Lisex 0.0.

Capítulo 5

Iniciar una sesión

5.1. Consideraciones generales

Es esencial que un sistema MLS asocie una clase de acceso a cada terminal. La clase de acceso asociada a una terminal determina el nivel de entrada y salida máximo que se puede dar en ese dispositivo. La entrada debe ser clasificada pues tarde o temprano terminará almacenada en un archivo o dirigida hacia algún dispositivo de salida (seguramente la misma terminal donde se produjo el *input*). Al mismo tiempo, en tanto la confidencialidad es lo que se intenta proteger, la "altura" que tendrá la salida sobre una terminal dada es vital pues es una de las formas que tiene un atacante de ver información confidencial. Considere el siguiente ejemplo.

Ejemplo 5.1.1 *Imagine un equipo corriendo Lisex 0.0 y conectadas a él por cables punto a punto varias terminales ubicadas en distintas dependencias físicas de cierta organización. Suponga que tal equipo procesa información con distintos grados de confidencialidad. Por ejemplo, hay información que puede ser vista por el público en general, otra que sólo está destinada a personal de la empresa, e información sumamente secreta que sólo puede ser vista por ciertos y determinados ejecutivos.*

Digamos que una de las terminales físicas se encuentra en una oficina que da a la calle. Supongamos que uno de los ejecutivos de alto nivel de la organización se loguea en esa terminal y pretende ver la información altamente secreta de la cual hablábamos antes. Si la terminal se lo permitiera sería muy sencillo para un atacante filmar, fotografiar o simplemente observar desde la calle lo que aparece en la pantalla, con lo cual, incluso sin la complicidad del ejecutivo, se violaría la política de seguridad de la organización.

Por lo tanto, una terminal con esa ubicación física debe impedir que el ejecutivo vea esa información (no es necesario que le impida loguearse pues ese usuario aun podría ver la información pública). Esto se consigue asociando una clase de acceso a la terminal (tanto física como lógica), a la información y al ejecutivo, e imponiendo restricciones semejantes a la seguridad simple y al confinamiento.

En definitiva, los usuarios de sistemas MLS deben estimar, antes de loguearse, con qué información van a trabajar para decidir en qué terminal lo harán (respecto del ejemplo anterior, el ejecutivo no debería haberse logueado en esa terminal si su intención era trabajar con la información más sensible de la compañía). El escenario anterior muestra que esta restricción no es caprichosa ni exagerada si se consideran situaciones en las que los atacantes ponen en juego tecnología al menos tan sofisticada como la que utiliza la víctima para defenderse.

Al mismo tiempo, en una terminal físicamente aislada[1] se debería poder trabajar con información de cualquier "altura". Debido a que Lisex 0.0 es un prototipo de prueba esto no es posible; estimamos que en futuras versiones los usuarios tengan mayor libertad en este sentido.

5.2. Como root

En Lisex 0.0 `root` tiene asociada la clase de acceso:

$$(5, \{0, 1, 2, 3, 4, 5, 6, 65535\})$$

que es la máxima clase de acceso según los niveles y categorías presentes en la distribución. Esto significa que `root` podrá abrir individualmente y en cualquier modo todos los archivos del sistema. Decimos que podrá abrir cualquier archivo individualmente pues ni si quiera `root` puede violar la propiedad de confinamiento (1.6.3) impuesta a nivel del núcleo. En otras palabras si `root` intenta abrir en modo read-write simultáneamente dos archivos con clases de acceso diferentes el núcleo se lo impedirá a pesar que podría abrir cada uno de ellos en momentos diferentes.

¡¡Importante!!

El hecho de que `root` tenga la máxima clase de acceso del sistema es una brecha de seguridad ya que:

—cualquier programa instalado por los administradores, que a priori debe ser considerado como un caballo de Troya, podría divulgar cualquier porción de información, y

—los administradores pueden ver cualquier porción de información siendo que no necesariamente están autorizados.

Nota 5.2.1 *En futuras distribuciones* `root` *no tendrá la máxima clase de acceso sino que poseerá una clase de acceso intermedia como cualquier otro usuario ordinario.*

[1] Por ejemplo, dentro de una sala TEMPEST.

Como `root` puede abrir cualquier archivo en cualquier modo entonces puede loguearse en cualquier terminal (ver tablas 5.2 y 5.3 al final del capítulo).

Nota 5.2.2 *Esto no será posible en futuras versiones por lo que* `root` *podrá loguearse únicamente en las terminales que posean una clase de acceso dominada por la de* `root`*.*

En definitiva, un usuario que desee loguearse como `root` podrá hacerlo en cualquier terminal sin necesidad de contraseña[2]. Sin embargo, lo que pueda hacer `root` luego de loguearse dependerá de la terminal en la que lo haya hecho pues esta será siempre el primer archivo abierto en modo read-write, lo que determina los archivos que `root` podrá abrir de allí en más (1.6.3).

5.3. Como secadm

⚠ **¡¡Importante!!**

Utilice esta cuenta de usuario lo menos posible.

Lea con atención los párrafos que siguen.

La cuenta `secadm` se debe utilizar únicamente para alterar o establecer la clase de acceso de cuentas de usuario, archivos o directorios (7.4).

En esta distribución secadm tiene asociada la clase de acceso:

$$(1, \{65535\})$$

Como la terminal es un archivo que se abre en modo read-write, este usuario sólo puede loguearse en la terminales `tty5` o `tty10` dado que son las únicas cuyas clases de acceso están dominadas por la clase de acceso de `secadm`.

Es muy importante que el lector revise la sección 7.4.

5.4. Como un usuario corriente

En esta distribución se incluyen, a manera de ejemplo, dos cuentas de usuario ordinarias: `lowuser` y `highuser`. En la tabla 5.1 se indica la clase de acceso de cada uno de estos usuarios.

[2]Ninguna de las cuentas provistas en esta distribución requiere contraseña para loguearse (6.10).

Como la terminal es un archivo que se abre en modo read-write, estos usuarios sólo pueden loguearse en las terminales indicadas en la tabla 5.2. Es claro que ambos usuarios pueden loguearse en más de una terminal por lo que deberá seleccionarse la más adecuada según la clase de acceso de los archivos con los que se desee trabajar.

Notas 5.4.1

— *En futuras versiones de Lisex no se incluirán las cuentas* `lowuser` *y* `highuser`.

— *En esta versión se las incluye debido a que el procedimiento para dar de alta cuentas de usuario es innecesariamente complejo (pues Lisex 0.0 es un prototipo).*

Login	**Clase de acceso**
lowuser	$(2, \{\})$
highuser	$(4, \{0, 1, 6\})$

Tabla 5.1: Clases de acceso de dos usuarios ordinarios

Usuario	**Terminal**					
	`tty1`	`tty2`	`tty3`	`tty4`	`tty5`	`tty10`
`highuser`	•	•	•		•	
`lowuser`		•			•	
`secadm`					•	•
`root`	•	•	•	•	•	•

Tabla 5.2: Terminales en las que se pueden loguear los usuarios provistos en es esta distribución.

Terminal	**Clase de acceso**
`tty1`	$(4, \{0, 1, 6\})$
`tty2`	$(2, \{\})$
`tty3`	$(3, \{1\})$
`tty4`	$(5, \{0, 1, 2, 3, 4, 5, 6, 65535\})$
`tty5`	$(1, \{\})$
`tty10`	$(1, \{65535\})$

Tabla 5.3: Clase de acceso de las terminales que están activas en la distribución.

Capítulo 6

El trabajo del usuario corriente

Este capítulo está destinado tanto al usuario corriente como al administrador pues este último debe conocer ciertas tareas básicas comunes a todos los usuarios que se efectúan de una manera diferente en Lisex 0.0 con respecto a como se realizan en Linux.

6.1. Ejecutar comandos o programas

En esta distribución la mayoría de los comandos de usuario son ligas al programa `busybox`. Por lo tanto, de todos ellos, el único archivo que tiene asociada una clase de acceso es `/bin/busybox` dado que las ligas (siguiendo la filosofía UNIX) tienen, automáticamente, la misma clase de acceso que el archivo original. La clase de acceso de `busybox` es:

$$(1, \{\})$$

lo que significa que cualquier usuario puede ejecutar cualquier programa de `busybox` (sin tener en cuenta que pueda haber alguna restricción de acceso en la ACL). Otros comandos presentes en esta distribución, que se supone que pueden ser ejecutados por todos los usuarios en general (como `acladdusr`, `acladdgrp`, etc.), tienen la misma clase de acceso que `busybox`.

Ante la solicitud de ejecución de un programa, Lisex realiza las siguientes verificaciones:

— que el archivo tenga el bit de ejecutabilidad para el dueño UNIX; es decir que a Lisex no le importa si el usuario que solicita la ejecución tiene permiso de ejecución, sólo le importa que el archivo sea ejecutable (2.1 y 2.2)

— que el usuario que hace la solicitud tenga permiso de lectura en la ACL del programa; es decir que se cambia, con respecto a Linux, el permiso de ejecutabilidad por el de lectura

— que la clase de acceso del usuario que hace la solicitud domine a la clase de acceso del programa (cf. seguridad simple, 1.6.2)

— que el usuario que hace la solicitud no tenga abierto un archivo en modo write cuya clase de acceso esté dominada por la del programa (cf. confinamiento, 1.6.3)

Por lo tanto, como la mayoría de los comandos tienen la clase de acceso $(1, \{\})$, cualquier usuario ordinario podrá ejecutar cualquiera de ellos en cualquier situación (ya que es imposible que el usuario tenga abierto en modo write un archivo con clase de acceso inferior, (3.1)).

Nota 6.1.1 *Una verificación que no se menciona en la lista anterior se relaciona con las librerías dinámicas y es la siguiente:*

— *que la clase de acceso del usuario que hace la solicitud de ejecución domine a la clase de acceso de cada librería utilizada por el programa*

pero dado que en esta distribución todas las librerías tienen clase de acceso $(1, \{\})$ *ningún usuario tendrá problemas en abrirlas pues no es posible que el usuario tenga abierto en modo write un archivo con clase de acceso inferior.*

6.2. Acceder a directorios

La política de seguridad que implementa Lisex 0.0 en cuanto al uso de `cd` es muy simple

— un usuario puede ejecutar `cd` para cambiar a otro directorio sólo si tiene permiso de lectura sobre el directorio de destino; los atributos y reglas MLS no se aplican en este caso.

Recordar que el permiso `x` ha sido completamente removido de la semántica de acceso a directorios (2.1).

Esta regla se complementa con la siguiente:

— un usuario, u, puede listar el contenido de un directorio[1] sólo si se dan todas las condiciones que siguen:

  — en la ACL hay una entrada de la forma $(u, ACL\_USR, READ)$ o de la forma $(g, ACL\_GRP, READ)$ donde g es un grupo al cual pertenece u

  — u verifica seguridad simple (1.6.2) con respecto al directorio

  — el proceso que intenta listar el contenido del directorio verifica confinamiento (1.6.3) donde el directorio se considera como un otro recurso más abierto en modo read

[1]Entendemos por contenido de un directorio los nombres de archivos o directorios allí almacenados, y nada más.

Por lo tanto, es muy posible que un usuario no pueda listar el contenido de un directorio a pesar de que se encuentra dentro de él. Más aun, si estas reglas se combinan con las enunciadas en la sección 6.6 que indican quienes pueden listar el estado de un archivo o directorio, es posible que al ejecutar `ls -l` se obtenga un listado parcial: todos los nombres y sólo el estado de aquellos archivos o directorios sobre los cuales el usuario tiene permiso de lectura.

Ejemplo 6.2.1 *Si* `highuser` *ingresa al sistema usando la terminal* `tty2` *no podrá listar el contenido de su propio directorio (*`/home/highuser`*). La razón es la siguiente:* `tty2` *tiene clase de acceso* $(2, \{\})$ *mientras que la de* `/home/highuser` *es* $(4, \{0, 1, 6\})$ *por lo que el contenido del directorio no puede ser mostrado a través de* `tty2` *(la terminal podría estar en un lugar poco protegido).*

6.3. Crear/Eliminar archivos o directorios

Los comandos y llamadas al sistema para crear o eliminar archivos o directorios son los mismos que en el Linux tradicional. Sin embargo, algunas llamadas, y por tanto los comandos asociados, tienen un comportamiento levemente diferente a las tradicionales.

Lisex 0.0 preserva tanto como es posible los principios de diseño de Linux. En particular, los directorios son considerados como archivos que contienen nombres de archivos y/o nombres de directorios. Por tal motivo, Lisex 0.0 realiza la siguiente verificación ante la solicitud de creación o eliminación de un archivo o directorio dentro de un directorio existente:

— que el proceso solicitante tenga abierto en modo read-write el directorio

A su vez, esa comprobación implica haber verificado:

— que el proceso solicitante tenga permiso de read-write en la ACL del directorio

— que la clase de acceso del proceso solicitante domine a la del directorio (cf. seguridad simple, 1.6.2)

— que el proceso solicitante verifique confinamiento (1.6.3) con respecto a todos los archivos aun abiertos y al directorio.

Nota 6.3.1 *El sticky-bit sigue teniendo la misma semántica para los (archivos y) directorios que en Linux por lo que la primera condición de la lista anterior es algo más compleja.*

La ACL de los archivos o directorios que se creen se establecerá según la máscara del usuario en ese momento y según las reglas habituales de Linux, es decir:

— habrá una entrada para el usuario que lo crea con los permisos indicados en la máscara del usuario que lo crea

— habrá una entrada para el grupo primario del usuario que lo crea con los permisos indicados en la máscara del usuario que lo crea

— habrá una entrada para el grupo `allgrp` con los permisos indicados para el "resto" de los usuarios en la máscara del usuario que lo crea

Los directorios o archivos que se creen tienen la misma clase de acceso que el usuario que los crea. Las clases de acceso de archivos o directorios sólo pueden ser modificadas por los oficiales MAC (7.4.2).

Nota 6.3.2 *Esta decisión de diseño respecto a qué clase de acceso asociarle a cada archivo que se crea puede resultar demasiado restrictiva. Más aun, desde un punto de vista teórico, es innecesariamente restrictiva. A menos que se consideren canales encubiertos, crear archivos o directorios con cualquier clase de acceso (incluso superior a la del usuario que los crea) no entraña ningún peligro con respecto a la confidencialidad de la información. Por lo tanto, en futuras versiones de Lisex es posible que se le permita al usuario seleccionar la clase de acceso con la que crear cada archivo. Más aun, es posible que ciertos tipos de archivos, como pipes, tengan una clase de acceso que dependa de la información que transmitan en cada momento.*

De todo lo anterior surge que para crear un archivo o directorio se debe tener en cuenta, aunque parezca inverosímil, la terminal en la cual se lo hace pues esta y el directorio son dos recursos abiertos en modo read-write. Vea el siguiente ejemplo.

Ejemplo 6.3.1 *Digamos que* `highuser` *está logueado en la terminal* `tty2` *la cual tiene clase de acceso* $(2, \{\})$. *Si este usuario desea crear el directorio* `docs` *dentro de su directorio personal (*`/home/highuser`*) el cual tiene clase de acceso* $(4, \{0, 1, 6\})$, *el sistema lo impedirá. La secuencia de controles que efectúa Lisex 0.0 en este caso es más o menos así:*

1. *Verifica que* `highuser` *tenga una entrada adecuada en la ACL del directorio* `/home/highuser`

2. *Verifica que la clase de acceso de* `highuser` *domine a la clase de acceso de* `/home/highuser`

3. *Verifica que la clase de acceso de* `/home/highuser` *sea idéntica*[2] *a la clase de acceso del otro archivo abierto en modo read-write, es decir* `/dev/tty2`

Los pasos 1 y 2 son sorteados con éxito pero el último no se verifica: $(2, \{\})$ *no es idéntica a* $(4, \{0, 1, 6\})$. *Cabe preguntarse, entonces, si esto no es excesivo. ¿Qué relación hay entre la terminal y el directorio? ¿Qué información podría divulgar un caballo de Troya si no se tiene en cuenta el último paso? La respuesta la encontrará en el siguiente ejemplo.*

[2] Esto es una versión simplificada de la regla de confinamiento dado que los dos recursos se abren en modo read-write (1.6.3).

Ejemplo 6.3.2 *En todo este ejemplo suponemos que Lisex 0.0 no efectúa la tercera verificación señalada en el ejemplo anterior.*

Consideremos que `bash_ct` *es un caballo de Troya que además de hacer todo lo que* `bash` *hace ataca de la siguiente forma:*

1. *Abre un directorio de alto nivel de seguridad*
2. *Si la terminal en la que está trabajando no es* `tty2` *la cierra momentáneamente y abre* `tty2`*.*
3. *Vuelca el contenido del directorio (nombres de archivos y directorios) en* `tty2`
4. *Reabre la terminal a la que estaba conectado.*

Entonces si `highuser` *inocentemente ejecuta* `bash_ct` *y* `lowuser` *espera pacientemente frente a la terminal física que tiene asociada* `tty2`*, podrá ver el nombre de cada archivo y directorio almacenado en los directorios de alto nivel de seguridad a los que* `highuser` *tiene acceso y* `lowuser` *no.*

Si la información así divulgada es de importancia o no remite a una discusión sobre si los nombres de archivos o directorios de por sí constituyen datos valiosos o no. Sin embargo, este escenario (si no se considerasen a los directorios como recursos contra los cuales verificar confinamiento) habilita otros ataques. Ahora `bash_ct` *trabaja de la siguiente manera:*

— *Cada palabra que llega por una terminal de alto nivel de seguridad es copiada como nombre de archivo en un directorio de bajo nivel de seguridad (digamos* `/home/lowuser` *que tiene clase de acceso* $(2, \{\})$*)*

Entonces, si `highuser` *se loguea en* `tty1`*, lo que le permite ingresar datos confidenciales, y usa como intérprete de comandos* `bash_ct`*, todo lo que escriba irá a parar al lugar esperado por* `highuser` *pero también se convertirá en nombres de archivos almacenados en un directorio de bajo nivel de seguridad. En ese caso* `lowuser` *logueándose en* `tty2` *podrá listar el contenido de* `/home/lowuser` *que junto a la hora de creación de cada archivo le permite reconstruir la información que ingresó al sistema y a la cual él no debería tener acceso.*

Obviamente, `bash_ct` *puede trabajar de esa forma pues al sistema no le importa que* `tty1` *y* `/home/lowuser` *estén ambos abiertos en modo read-write al mismo tiempo.*

Nota 6.3.3 *Si bien los ataques mostrados en los ejemplos anteriores serían posibles si Lisex no efectuara la verificación discutida, los directorios no son contenedores de información equiparables a los archivos. En efecto, el último ataque mostrado usa a los directorios para un fin que no fueron pensados. En la teoría de seguridad informática el uso de un recurso de formas no pensadas por los ingenieros de manera tal que se viola la política de seguridad, es denominado canal encubierto.*

Lisex 0.0 no hace nada por evitar la existencia de canales encubiertos. Más allá de los modelos de seguridad que postulan la posibilidad de evitar por completo estos canales, la solución estándar es auditar el uso de todos aquellos recursos que se sepa pueden ser usados como canales encubiertos. Es posible que futuras versiones de Lisex no traten a los directorios como recursos a proteger y se los considere como canales encubiertos. Por lo tanto, dichas versiones no efectuarían la verificación que motivó esta discusión, lo que implica que tanto usuarios como caballos de Troya tendrán mayor libertad de acción.

En definitiva, para crear un directorio o archivo es necesario que el usuario se loguee en una terminal que tenga la misma clase de acceso que el directorio donde se crearán los archivos o directorios. Esto vale, también, si se usan los comandos y directivas del *shell* estándar: `cp`, `mv`, `mkdir`, `>`, etc. Para que `root` pueda mantener los directorios estándar (como `/bin`, `/dev`, etc.) la distribución cuenta con la terminal `tty5` con clase de acceso $(1, \{\})$ que es la misma que la de los directorios mencionados.

Ahora pasemos a analizar el árbol de directorios. En sistemas MLS es conveniente que las clases de acceso de los directorios crezcan o permanezcan igual a medida que se desciende por el árbol de directorios. Obviamente también es posible que se creen directorios con clase de acceso incomparable con respecto al directorio padre. En caso de que no se proceda de esta forma, se estará ocultando ramas del árbol de directorio a ciertos usuarios. Esto es así, debido a que si el archivo f con clase de acceso CA_f está dentro del directorio d con clase de acceso CA_d, cualquier usuario u con clase de acceso CA_u tal que $CA_u \succeq CA_f$ y $CA_u \not\succeq CA_d$, no podrá saber que existe el archivo f ya que no podrá listar jamás el contenido de d, siendo que tiene autorización para leer f.

Por esta razón es que el directorio raíz de Lisex 0.0 tiene clase de acceso $(1, \{\})$. Esto significa que, al menos en términos MLS, cualquier usuario podrá crear y borrar archivos y directorios dentro del directorio raíz. De todas formas, aun queda la protección DAC. Es decir, el usuario que quiera modificar la raíz deberá tener el permiso adecuado en la ACL de este directorio. Siendo que lo único que impide a un usuario crear, borrar o reemplazar archivos o directorios en el directorio raíz, es la ACL de la raíz, es posible que un caballo de Troya realice cualquiera de esas acciones como parte de un ataque, aunque le será "imposible" divulgar información. Lisex 0.0 aumenta la protección frente a ataques a la confidencialidad (1.1) y no a la integridad, como es el caso antes descripto.

El mismo análisis de los párrafos anteriores es aplicable a cualquier directorio del árbol. La conclusión es: no espere que Lisex 0.0 proteja la integridad de sus datos más de lo que Linux lo hace, Lisex 0.0 sólo mejora la confidencialidad de su información.

6.3.1. Agregar comandos o programas

Dado que Lisex 0.0 es un prototipo sobre el cual probar la seguridad y utilidad de sistemas MLS, no hemos implementado alguna forma de agregar programas al sistema. No hemos incluido un compilador ni es posible montar otros tipos de sistemas de archivos diferentes a Lisex 0.0. Los únicos programas que pueden agregarse son *shell scripts* que cualquier usuario puede programar (7.5).

6.4. Editar un archivo

En esta distribución se incluye el editor de textos `six` como un *applet* de `busybox`[3]. Editar un archivo es equivalente a abrirlo en modo read-write.

Al editar archivos con Lisex 0.0 deben tenerse en cuenta las siguientes consideraciones:

— Lisex 0.0 es en general más restrictivo que el Linux o UNIX tradicional.

— La terminal es el primer archivo que se edita.

— Sólo se pueden editar archivos que tengan la misma clase de seguridad que la terminal.

— Se pueden editar simultáneamente todos los archivos que se quiera e intercambiar información entre ellos siempre y cuando posean todos la misma clase de seguridad entre sí y con respecto a la terminal.

— Pueden consultarse (abrir en modo read) archivos con una clase de seguridad inferior a la de la terminal.

— Pueden abrirse en modo write-only archivos con una clase de seguridad superior a la de la terminal.

— Las restricciones anteriores se aplican por proceso y no por usuario.

Más técnicamente, Lisex 0.0 realiza las siguientes verificaciones al momento en que un proceso solicita la edición de un archivo:

— que el UID asociado con el proceso tenga permiso de read-write en la ACL del archivo

— que tal proceso verifique seguridad simple con respecto al archivo

— que tal proceso verifique confinamiento (modo read-write) con respecto a todos los archivos aun abiertos y al archivo que se desea abrir

`six` es suficientemente "inteligente" como para abrir cada archivo que se le pasa como parámetro con el mayor modo posible.

Ejemplo 6.4.1 *Supongamos que* `highuser` *está trabajando en la terminal* `tty1` *que tiene la misma clase de acceso de ese usuario. Digamos que*

[3] `six` es una versión simplificada de `vim`; de hecho también se puede ejecutar como `vim`.

1. *el archivo* `f1` *tiene clase de acceso* $(4, \{0, 1, 6\})$*, es decir idéntica a la de la terminal, y*

2. *el archivo* `f2` *tiene clase de acceso* $(2, \{\})$

Entonces si `highuser` *ejecuta*

```
$ six f1 f2
```

el editor tratará de abrir ambos en modo read-write pues ese es el significado implícito del comando. Sin embargo, si bien tendrá éxito con el primero el segundo quedará abierto en modo read-only. Claramente, este comportamiento obedece a las reglas de seguridad simple y confinamiento. Si `f2` *fuese abierto en modo read-write podrían copiarse datos de* `f1` *a* `f2` *violando la seguridad del sistema. Recordar que en la Advertencia al inicio de este manual se señaló que algunos de estos ataques son posibles debido a que Lisex 0.0 es únicamente un prototipo.*

6.5. *Pipes* y re-direccionamiento

Las interconexiones[4] son archivos en el sistema de archivos. Por lo tanto, en principio se le aplican los mismos controles de seguridad que a cualquier otro archivo. Sin embargo, existen dos tipos de interconexiones (nombradas y sin nombre) que son manejados por diferentes llamadas al sistema. Lisex 0.0 aplica los mismos controles que a un archivo corriente, sólo a las interconexiones nombradas.

Nota 6.5.1 *Futuras versiones de Lisex implementarán los controles de seguridad sobre las interconexiones sin nombre.*

Las interconexiones sin nombre, es decir las que no maneja Lisex 0.0, son las que se utilizan en comandos de la forma

```
$ cmd1 file1 | cmd2 > file2
```

Por lo tanto, no se pueden utilizar este tipo de comandos; más precisamente fallarán siempre que se los ejecute. Recomendamos ver el apéndice A.

Lisex 0.0 soporta la posibilidad de re-direccionar (mediante `>`, `>>` y `<`) la entrada, salida o error de comandos. Sin embargo, al combinar esta funcionalidad con el modelo MLS implementado se reduce la usabilidad del sistema en general. Concretamente, para que un comando de la forma[5]:

```
$ cmd1 file1 > file2
```

no falle se deben dar las siguientes condiciones[6]:

[4] De aquí en más utilizaremos *interconexión* o *tubo* como traducción de *pipe*.

[5] Lo mismo vale para $\gg$.

[6] Recordar que $a \preceq b$ significa a está dominada por b y que $a \succeq b$ significa a domina a b (1.4.1).

1. $CA(\texttt{cmd1}) \preceq CA(tty)$
2. $CA(\texttt{file1}) \preceq CA(tty)$
3. $CA(dir) = CA(tty)$ o `file2` ya existe y $CA(\texttt{file2}) \preceq CA(u)$

donde $CA(x)$ es la clase de acceso de x, u es el usuario que ejecuta el comando, tty es la terminal donde está trabajando u, y dir es el directorio donde (posiblemente) se crea `file2`. En Lisex 0.0 la condición 1 se cumple trivialmente pues todos los comandos tienen la clase mínima; a su vez existe una terminal con la misma clase de acceso para cada usuario incluido en la distribución y el directorio personal de cada usuario tiene la misma clase de acceso del usuario.

6.6. Consultar la ACL de un recurso

Esta distribución cuenta con el programa `aclstat` que vuelca la ACL de un único archivo o directorio que se le pasa como parámetro.

Nota 6.6.1 *El programa* `aclstat` *es sumamente rudimentario y no ha sido testeado.*

Para que el usuario u pueda listar la ACL de un archivo o directorio debe tener permiso de lectura sobre el archivo o directorio en cuestión. Esto es equivalente a que la ACL tenga una entrada de la forma $(UID\_u, ACL\_USR, READ \mid OP)$ o $(GID\_g, ACL\_GRP, READ \mid OP)$ donde g es un grupo al cual pertenece el usuario y OP son otros permisos que el usuario o el grupo pueden tener.

Nota 6.6.2 *Luego de haber usado el sistema notamos que la regla anterior debería debilitarse a:*

— *para que el usuario u pueda listar la ACL de un archivo o directorio, r, debe ser dueño o tener permiso de lectura sobre r.*

Debido a que el concepto de dueño de un archivo o directorio está vinculado exclusivamente a la posibilidad de alterar su ACL, la regla actual requiere que un dueño se incluya como lector del archivo o directorio si desea ver su ACL (acción obviamente necesaria para cumplir con su tarea). Evidentemente, esta restricción es contraria a la imposición del principio de mínimo privilegio por lo que en futuras versiones de Lisex optaremos por la versión expuesta en esta nota.

La salida que provee `aclstat` tiene el siguiente formato y significado:

— Se imprime una línea por cada terna o entrada en la ACL

— Cada línea comienza con `Usr` o `Grp` dependiendo si la entrada corresponde a un usuario o grupo, luego sigue el UID o GID del usuario o grupo y finalmente `Mode` seguido del modo, expresado como la suma de los permisos básicos que el usuario o grupo poseen sobre el archivo o directorio. El valor numérico de cada uno de los permisos básicos se puede consultar en la tabla 6.1.

Permiso	**Valor**
Ejecutabilidad	1
Escritura	2
Lectura	4
Propiedad	8

Tabla 6.1: Valor numérico asociado con cada uno de los permisos básicos. Notar que se extiende la numeración estándar con el nuevo permiso básico de *propiedad*. Además, recordar que el "permiso" de ejecutabilidad no es en realidad un permiso sino que simplemente indica si el archivo es o no ejecutable.

Ejemplo 6.6.1 *Si se ejecuta*

```
$ aclstat /
```

la salida que se obtiene es la siguiente

```
Usr: 0, Mode: 14
Grp: 0, Mode: 10
Grp: 666, Mode: 4
```

pues, por ejemplo, el usuario `root` *es dueño (8) de "/" y además puede leer (4) y escribir (2) en ese directorio:* $2+4+8=14$. *También el grupo* `root` *es dueño*[7]*, más allá de tener permiso de lectura y escritura. Notar que:*

1. *no aparece el permiso de ejecución o "stat" sobre el directorio debido a los cambios semánticos que se introdujeron*
2. *que cualquier usuario podrá ejecutar exitosamente el comando anterior dado que* `allgrp` *(GID 666) tiene permiso de lectura sobre el directorio raíz.*

Si se ejecuta

```
$ aclstat /sbin/passwd
```

la salida que se obtiene es la siguiente

```
Usr: 0, Mode: 13
Grp: 0, Mode: 12
Grp: 666, Mode: 4
```

Lo que significa, por ejemplo, que, por la nueva semántica, cualquier usuario podrá ejecutar este comando dado que `allgrp` *(GID 666) tiene permiso de lectura y es un archivo ejecutable (codificado en el 13 = Ejecutabilidad + Lectura + Propiedad). Notar, además, que el grupo* `root` *es dueño del archivo.*

Es interesante comparar la salida de este programa con la provista por `ls -l` *que no ha sido modificado para tener en cuenta ACLs.*

[7] Recordar que el grupo `root` es dueño inmodificable de todos los recursos del sistema (2.5).

El programa `aclstat` no realiza mucho más que una llamada a la función de la librería `liblisex.a` que lleva el mismo nombre, la cual a su vez, invoca a la llamada al sistema `sys_aclstat`. Quien desee programar una versión mejorada de `aclstat` puede consultar [GM02] para una descripción detallada de la llamada `aclstat` la cual implementa la especificación de la operación `aclstat` descripta en el modelo formal [Cri02].

6.7. Modificar la ACL de un recurso

Para que el usuario u pueda modificar la ACL de un archivo o directorio debe ser dueño del archivo o directorio en cuestión. Esto es equivalente a que la ACL tenga una entrada de la forma $(UID\_u, ACL\_USR, OWNER \mid OP)$ o $(GID\_g, ACL\_GRP, OWNER \mid OP)$ donde g es un grupo al cual pertenece el usuario y OP son otros permisos que el usuario o grupo pueden tener.

6.7.1. Mediante chmod

Tanto el comando `chmod` como la llamada homónima tienen la misma función que en Linux. Es decir, pueden alterar el modo de un archivo o directorio en lo que se refiere al dueño (UNIX), grupo (UNIX) y grupo `allgrp` de la misma forma que en Linux. Observar, sin embargo, que se ha modificado la semántica del modo para el resto de los usuarios (2.2) y que cualquier cambio tiene impacto sobre la ACL (3.3.1).

6.7.2. Mediante chown

Sugerimos releer la sección 3.3.1.

Tanto el comando `chown` como la llamada homónima tienen la misma función que en Linux. Es decir, pueden cambiar el dueño (UNIX) y el grupo (UNIX) de un archivo o directorio de la misma forma que en Linux.

6.7.3. Mediante acladdusr o acladdgrp

Estos comandos permiten agregar o modificar una entrada en la ACL de un único archivo o directorio que se le pasa como parámetro. `acladdusr` agrega usuarios mientras que `acladdgrp` agrega grupos. En cualquier caso los parámetros de entrada son:

filename nombre del archivo o directorio al cual se le desea modificar la ACL

id UID o GID (para `acladdusr` y `acladdgrp`, respectivamente) del usuario o grupo que se quiere agregar a la ACL

mode modo, expresado como la suma de los permisos básicos, que se desea agregar para el usuario o grupo

Cualquiera de los dos comandos agregan permisos o entradas en la ACL, es decir que si ya existe una entrada para cierto usuario o grupo entonces estos comandos ampliarán los permisos para ese usuario o grupo, y si no existe agregarán una entrada según el modo que se pase como parámetro.

Ejemplo 6.7.1 *Supongamos que la máscara para creación de archivos de* `lowuser` *vale* 022 *y que crea en* `/home/lowuser` *el archivo* `lowfile`*. Si ese mismo usuario ejecuta*

```
$ aclstat /home/lowuser/lowfile
```

la salida será[8]

```
Usr: 500, Mode: 14
Grp: 500, Mode: 4
Grp: 666, Mode: 4
Grp: 0, Mode 8
```

Entonces si `lowuser` *quiere agregar en la ACL a* `highuser` *con permisos de lectura y escritura debe ejecutar*

```
$ acladdusr /home/lowuser/lowfile 501 6
```

Y lo puede comprobar ejecutando:

```
$ aclstat /home/lowuser/lowfile
```

siendo la salida

```
Usr: 500, Mode: 14
Grp: 500, Mode: 4
Grp: 666, Mode: 4
Grp: 0, Mode 8
Usr: 501, Mode 6
```

Si ahora `lowuser` *quiere agregar a* `highuser` *como dueño del archivo entonces debe ejecutar*

```
$ acladdusr /home/lowuser/lowfile 501 8
```

lo que implica que `highuser` *tendrá un modo total acumulado sobre el archivo igual a 14.*

Por otro lado, el primer comando `acladdusr` *del ejemplo tiene el mismo efecto que el siguiente:*

[8] Notar que el grupo 0 (`root`) es dueño del archivo.

```
$ acladdusr /home/lowuser/lowfile 501 2
```

pues `highuser` *tiene permiso de lectura en virtud de su pertenencia a* `allgrp`. *Aunque, si* `lowuser` *elimina* `allgrp` *de la ACL de* `/home/lowuser/lowfile`, *entonces* `highuser` *sólo tendrá permiso de escritura sobre el archivo en cuestión.*

Nota 6.7.1 *Recordar que en Lisex 0.0 la longitud de las ACL es fija (modificable en tiempo de compilación). Más precisamente, en la distribución estándar se admiten hasta 8 entradas en cada ACL (3.3).*

6.7.4. Mediante acldelusr o acldelgrp

Estos comandos permiten eliminar o modificar una entrada en la ACL de un único archivo o directorio que se le pasa como parámetro. `acldelusr` elimina usuarios mientras que `acldelgrp` elimina grupos. En cualquier caso los parámetros de entrada son:

filename nombre del archivo o directorio al cual se le desea modificar la ACL

id UID o GID (para `acldelusr` y `acldelgrp`, respectivamente) del usuario o grupo que se quiere eliminar de la ACL

mode modo, expresado como la suma de los permisos básicos, que se desea eliminar para el usuario o grupo

Cualquiera de los dos comandos eliminan permisos o entradas en la ACL, es decir que si ya existe una entrada para cierto usuario o grupo entonces estos comandos reducirán los permisos para ese usuario o grupo y eventualmente eliminarán la entrada si el usuario o grupo ya no posee ningún permiso.

Ejemplo 6.7.2 *Continuando con el ejemplo 6.7.1 supongamos que* `lowuser` *quiere eliminar a* `allgrp` *de la ACL de* `/home/lowuser/lowfile`, *entonces ejecuta*

```
$ acldelgrp /home/lowuser/lowfile 666 4
```

Aunque podría haber usado cualquier modo superior a 4; es decir, si uno quiere asegurarse de eliminar a un usuario o grupo de una ACL puede usar como modo el más alto posible.

Como ya dijimos, estos comandos eliminan una entrada de la ACL. Por otro lado, al contrario de `chown`, `acldelusr` no reemplaza la entrada que se elimina con una entrada nueva (recordar que el grupo `root` no puede ser removido como grupo dueño de ninguna ACL). Por tanto, existe la posibilidad que, utilizando `acldelusr` o `acldelgrp`, se elimine el dueño o el grupo UNIX lo cual dejaría a la ACL en un estado inconsistente con respecto a comandos como `chown` o `chmod`. Para evitar esta inconsistencia, si se intenta eliminar el dueño o el grupo UNIX el sistema agrega de forma automática al usuario o el grupo `root` según sea el caso.

6.8. Consultar la clase de acceso de un recurso

Para esta tarea puede usarse el comando `oscstat` que vuelca la clase de acceso de un único archivo o directorio que se le pasa como parámetro.

Nota 6.8.1 *El programa* `oscstat` *es sumamente rudimentario y no ha sido testeado.*

Para que el usuario u pueda listar la clase de acceso de un archivo o directorio debe tener una clase de acceso que domine a la del archivo o directorio.

La salida que provee `oscstat` tiene el siguiente formato y significado:

— Se imprimen dos líneas, una para el nivel y otra para el conjunto de categorías

— La primera línea comienza con `Level:` y a continuación se imprime el nivel del archivo o directorio que se pasó como parámetro.

— La segunda línea comienza con `Cats:` y a continuación, separadas por un espacio, cada una de las categorías del conjunto de categorías del archivo o directorio que se pasó como parámetro.

Ejemplo 6.8.1 *Si habiéndose logueado como* `lowuser` *se ejecuta*

```
$ oscstat /
```

la salida que se obtiene es la siguiente

```
Level: 1
Cats:
```

pues el conjunto de categorías del directorio raíz es vacío. En la misma situación anterior, si se ejecuta

```
$ oscstat /root
```

el comando retorna un error debido a que `lowuser` *no tiene derecho a conocer la clase de acceso de* `/root` *pues este último tiene una "altura" superior a la de* `lowuser`*. Lo único que* `lowuser` *puede deducir es que* `/root` *posee una clase superior o incomparable a la propia.*

Pero si el mismo comando se ejecuta como `root` *la salida que se obtiene es la siguiente:*

```
Level: 5
Cats: 0 1 2 3 4 5 6 65535
```

Más aun, si `root` *ejecuta*

```
$ oscstat /home/lowuser
```

la salida es

```
Level: 2
Cats:
```

El programa `oscstat` no realiza mucho más que una llamada a la función de la librería `liblisex.a` que lleva el mismo nombre, la cual a su vez, invoca a la llamada al sistema `sys_oscstat`. Quien desee programar una versión mejorada de `oscstat` puede consultar [GM02] para una descripción detallada de la llamada `oscstat` la cual implementa la especificación de la operación `oscstat` descripta en el modelo formal [Cri02].

6.9. Consultar la clase de acceso de un usuario

Para esta tarea puede usarse el comando `sscstat` que vuelca la clase de acceso de una única cuenta de usuario que se le pasa como parámetro.

Nota 6.9.1 *El programa* `sscstat` *es sumamente rudimentario y no ha sido testeado.*

Para que el usuario u pueda listar la clase de acceso del usuario $user$, u debe tener una clase de acceso que domine a la de $user$.

La salida que provee `sscstat` tiene el siguiente formato y significado:

— Se imprimen dos líneas, una para el nivel y otra para el conjunto de categorías

— La primera línea comienza con `Level:` y a continuación se imprime el nivel del usuario que se pasó como parámetro.

— La segunda línea comienza con `Cats:` y a continuación, separadas por un espacio, cada una de las categorías del conjunto de categorías del usuario que se pasó como parámetro.

Ejemplo 6.9.1 *Si habiéndose logueado como* `lowuser` *se ejecuta*

```
$ sscstat lowuser
```

la salida que se obtiene es la siguiente

```
Level: 2
Cats:
```

En la misma situación anterior, si se ejecuta

```
$ sscstat highuser
```

el comando retorna un error debido a que `lowuser` *no tiene derecho a conocer la clase de acceso de* `highuser` *pues este último tiene una "altura" superior a la de* `lowuser`*. Lo único que* `lowuser` *puede deducir es que* `highuser` *posee una clase superior o incomparable a la propia.*

Pero si el mismo comando se ejecuta como `root` *la salida que se obtiene es la siguiente:*

```
Level: 4
Cats: 0 1 6
```

Sin embargo, se debe recordar que en futuras versiones de Lisex, `root` *no tendrá la máxima clase de acceso por lo que le será imposible obtener este tipo de información.*

El programa `sscstat` no realiza mucho más que una llamada a la función de la librería `liblisex.a` que lleva el mismo nombre, la cual a su vez, invoca a la llamada al sistema `sys_sscstat`. Quien desee programar una versión mejorada de `sscstat` puede consultar [GM02] para una descripción detallada de la llamada `sscstat` la cual implementa la especificación de la operación `sscstat` descripta en el modelo formal [Cri02].

6.10. Cambiar la contraseña

En Lisex 0.0 ninguna cuenta de usuario requiere contraseña para ingresar al sistema tanto que ni si quiera la distribución incluye el comando `passwd`. Si bien esto puede resultar llamativo en un sistema operativo cuyo objetivo es mejorar la seguridad, debe recordarse que Lisex 0.0 no es más que un prototipo rudimentario e incompleto.

El objetivo principal de Lisex 0.0 es contar con una plataforma donde experimentar con formas más complejas de control de acceso. En futuras versiones se incorporarán los mecanismos de autenticación tradicionales de Linux y en lo posible se los mejorará (7.3).

6.11. Terminar una sesión

Las sesiones se finalizan de la misma forma que en Linux.

Capítulo 7

El trabajo del administrador

Como ya se mencionó, Lisex 0.0 combina el modelo de control de acceso tradicional de Linux, con el modelo de ACL y con un modelo MLS. El modelo MLS es fundamentalmente diferente de los otros dos. Es esencial para la seguridad de un sistema de este tipo que la administración de la porción DAC esté completamente separada de la administración de la porción MLS. Esto significa que ni siquiera `root` debería poder inmiscuirse en los atributos MAC de recursos o cuentas de usuario.

Dado que Lisex 0.0 es un prototipo esta condición básica no se cumple aunque se aproxima lo suficiente como para darle a los usuarios la idea de cómo serían las cosas en un sistema apto para producción. Por tanto, muchas de las recomendaciones generales que se dan a continuación deben considerarse como una simulación de un sistema MLS en producción.

7.1. Crear/Eliminar una cuenta de usuario

En Lisex 0.0 se necesita el trabajo cooperativo de dos cuentas de usuario para crear una nueva cuenta:

1. cualquier miembro del grupo `root` debe crear la cuenta como se hace habitualmente en Linux

2. un oficial MAC debe establecer la clase de acceso para esa cuenta (7.4.2)

Nota 7.1.1 *Lisex 0.0 no incluye comandos para crear cuentas de usuario por lo que se deberán editar los archivos* `/etc/passwd` *y* `/etc/group` *manualmente.*

De esta forma se impide que un caballo de Troya actuando a nombre de `root` pueda crear una cuenta que luego pueda ser utilizada por el atacante para obtener información valiosa.

Nota 7.1.2 *Tener en cuenta que* `root` *puede invocar la llamada al sistema* `setuid` *por lo que un caballo de Troya ejecutado por* `root` *podría alterar las clases de acceso. Esta es una deficiencia de Lisex 0.0 debida a que aun es un prototipo.*

Una cuenta recién creada por `root` a la que aun `secadm` no le asignó una clase de acceso tendrá la clase $(0, \{\})$ con la cual no podrá ni siquiera loguearse ni abrir archivos. Y aunque esto fuese posible sólo podría obtener información de muy poco valor o nulo.

Las cuentas se eliminan en dos fases:

1. cualquier miembro del grupo `root` debe eliminar la cuenta como se hace habitualmente en Linux

2. un oficial MAC debe eliminar la clase de acceso para esa cuenta (7.4.3)

7.2. Agregar/Eliminar grupos a una cuenta

Esto se realiza de la misma manera que en Linux con la salvedad de que lo puede realizar cualquier miembro del grupo `root`. Tener en cuenta que puede ser el caso que los archivos `/etc/group` y `/etc/passwd` no tengan permiso de read-write para el grupo `root`, pero, en tanto este grupo es dueño de ambos, cualquiera de sus miembros podrá otorgar cualquier permiso a cualquier usuario o grupo, en particular al grupo `root`.

7.2.1. Grupos root y allgrp

Como se explicó en la sección 2.5 el núcleo de Lisex considera al grupo `root` como equivalente al usuario `root` de Linux. En otras palabras, la cuenta con UID 0 carece de privilegios especiales a menos que sea miembro del grupo `root` (GID 0). Por lo tanto, se debe poner especial atención en no eliminar la cuenta `root` del grupo `root` pues en tal caso dicha cuenta no podrá realizar las tareas de administración usuales.

Notas 7.2.1

— *Es muy importante aclarar que lo antedicho sólo tiene validez para el sistema de archivos, es decir, para el resto de las tareas de administración la cuenta* `root` *sigue siendo la única capaz de realizarlas (por ejemplo bajar procesos de otros usuarios).*

— *Así mismo, la cuenta* `root` *sigue siendo la única capaz de ejecutar de forma irrestricta la llamada setuid (2.6).*

— *En futuras versiones de Lisex es posible que se reemplace en otras porciones del sistema a la cuenta* `root` *por el grupo* `root`*.*

Paralelamente, se debe poner especial atención en agregar al grupo `root` sólo a aquellos usuarios que en Linux compartirían la contraseña de la cuenta `root` pues, en lo que al sistema de archivos se refiere, cualquiera de estos usuarios podrá hacer lo mismo que `root`.

En la sección 2.2 se explicó que Lisex no considera al *resto* de los usuarios en el modo de un archivo sino a *todos* los usuarios. Esto se implementa por medio del grupo de usuarios `allgrp` (GID 666). En esa misma sección se aclaró que la pertenencia a este grupo debe ser explícita y que es posible que en futuras versiones de Lisex se cambie a pertenencia implícita (es decir, en el futuro el sistema considerará a este GID de manera especial suponiendo que todos los usuarios del sistema pertenecen a él sin importar si fueron o no agregados por el administrador). Por lo tanto, en Lisex 0.0 el administrador que crea una cuenta de usuario debe asegurarse que esta pertenezca al grupo `allgrp` para que la semántica se implemente adecuadamente.

7.3. Cambiar la contraseña de un usuario

Como se mencionó en el capítulo anterior, en Lisex 0.0 no requiere contraseña para ninguna cuenta de usuario. El equipo Lisex es consciente de que esto constituye un grueso error de seguridad pero, al confrontarlo con el hecho de que Lisex 0.0 es un *prototipo donde probar modelos de control de acceso*, se decidió deshabilitar el mecanismo de autenticación.

Implementar adecuadamente un mecanismo de autenticación compatible con un sistema MLS entraña una serie de modificaciones que están más allá del interés actual del equipo Lisex. Valga como ejemplo el siguiente análisis.

El archivo `/etc/passwd` posee clase de acceso $(1, \{\})$. Si allí se almacenaran las contraseñas de los usuarios, esta clase de acceso no reflejaría la importancia de la información contenida en este archivo. Sin embargo, es necesario asignarle esa clase de acceso debido a que se requiere que todos los usuarios puedan leerlo. Si, en cambio, las contraseñas se almacenaran en `/etc/shadow`, este archivo debería tener a lo sumo la clase de acceso de `root` ya que es la única cuenta que se espera lea este archivo. Pero, en un sistema MLS `root` debería ser una cuenta más con acceso a herramientas de administración pero sin acceso a información valiosa. Claramente, la contraseña de un integrante de la gerencia es información sumamente valiosa; por lo tanto si `root` fuese capaz de modificar o deducir esta contraseña, también lo sería un caballo de Troya y en consecuencia la seguridad del sistema se vería comprometida. En definitiva los mecanismos estándar de autenticación presentan ciertas debilidades al ser considerados en un contexto MLS. Por lo tanto, el equipo Lisex pospone la implementación de un mecanismo de autenticación compatible con Linux y un modelo MLS para futuras versiones.

El problema analizado en el párrafo anterior no agota la lista de inconvenientes relativos a autenticación que habría que resolver para tener resistencia a caballos de Troya. Otro problema, por ejemplo, es asegurarle a los usuarios que la rutina criptográfica que encripta sus contraseñas no es (parte de) un caballo de Troya. La solución a este problema requiere la implementación de un mecanismo para establecer un camino confiable (*trusted path*) entre l núcleo del sistema y el usuario.

7.4. Administración MAC

Las recomendaciones que se dan a continuación son más aplicables a un sistema MLS capaz de ser utilizado en producción que para un prototipo como Lisex 0.0. Sin embargo, creemos que son válidas para aquellos que decidan probar Lisex 0.0 de la forma más real posible y así medir la seguridad y usabilidad de este tipo de sistemas.

⚠ **¡¡Importante!!**

El contenido de esta sección es de suma importancia para la seguridad de su sistema.

Como se mencionó en el capítulo introductorio, Lisex es un proyecto cuyo objetivo es construir un sistema operativo resistente a ataques por medio de caballos de Troya. Dado que hasta el momento no se conocen otras soluciones a este problema que no pasen por la implementación de alguna forma de control de acceso obligatorio, Lisex incorpora desde el comienzo (Lisex 0.0) un modelo MLS basado en el trabajo de Bell y La Padula.

La inclusión de este modelo transforma radicalmente la seguridad de Linux. Por tanto, la administración de la seguridad y del sistema en general sufren cambios profundos. Tan es así que la administración de la seguridad tradicional pasa a segundo plano y queda "relegada" a `root` (o al grupo `root`). Si el objetivo de la seguridad en Lisex es la resistencia a ataques que utilizan caballos de Troya, la seguridad tradicional de Linux es de poca o nula efectividad contra esta amenaza, motivo por el cual la seguridad DAC tradicional (e incluso la extensión a ACL) no es considerada crítica.

En definitiva, si se debe administrar la implementación de un nuevo modelo de seguridad se debe crear la infraestructura para quienes vayan a realizarla. En Lisex 0.0 esta nueva cara de la administración se reduce a controlar el cambio de clases de acceso de cuentas de usuario, archivos y directorios. Los usuarios capaces de realizar estas tareas se denominan *oficiales MAC*. Un usuario es un oficial MAC si su clase de acceso posee un conjunto de categorías que incluye la categoría 65535.

Obviamente, la cantidad de usuarios que posean la categoría 65535 en su clase de acceso debe reducirse al mínimo posible y sólo a personal de la más absoluta confianza. Luego de la instalación del sistema la cuenta `secadm` (UID = 1) es la única que posee la categoría 65535 en su clase de acceso (5.3). Por lo tanto, si se desea que haya otros oficiales MAC se necesita crear otras cuentas y que `secadm` les asigne la categoría 65535 en sus clases de acceso respectivas.

⚠ **¡¡Importante!!**

La única defensa real contra ataques que utilicen caballos de Troya es la porción MLS o MAC de Lisex.

Parte de esta defensa se basa en que sólo unos pocos usuarios estén autorizados a modificar las clases de acceso del resto de los usuarios y de todos los archivos o directorios sean o no propietarios de estos.

Además, estos usuarios deben utilizar únicamente los programas *certificados* para la administración MAC. Si un oficial MAC puede ejecutar otros programas que no han sido certificados, y alguno de ellos es un caballo de Troya, la seguridad de todo el sistema estará en peligro. Esto se debe a que dicho caballo de Troya será capaz de modificar la clase de acceso de cualquier cuenta de usuario, archivo y directorio tal como lo podría hacer el individuo que utiliza esa cuenta de usuario. Para el sistema es imposible distinguir si la solicitud de cambio de clase de acceso proviene de un individuo que utiliza conscientemente un programa sin fallas o de un programa que actúa sin el consentimiento y la orden expresa de ese mismo individuo[1].

7.4.1. Precauciones para la configuración de cuentas de oficiales MAC

Si un mismo individuo efectuará tareas de administración tradicionales y/o será un usuario ordinario y, a la vez, será oficial MAC, entonces debe poseer dos cuentas de usuario:

Administración DAC. La cuenta de usuario deberá pertenecer al grupo `root` y no deberá tener la categoría 65535 entre su conjunto de categorías. Observar que ambos requisitos están a cargo de las dos cuentas iniciales de administración, `root` y `secadm`, respectivamente.

Administración MAC. La cuenta de usuario no deberá pertenecer únicamente `allgrp` y deberá tener la categoría 65535 entre su conjunto de categorías; más aun, se espera que estas cuentas tengan el conjunto de categorías $\{65535\}$. El primer requisito no es esencial y, peor aun, es imposible imponerlo si se considera el accionar de un caballo de Troya actuando a nombre de algún miembro del grupo `root`.

Los oficiales MAC sólo deben poder ejecutar los programas certificados para efectuar las tareas de administración de la porción MAC del sistema y ningún otro. Este requisito se impone asociando la categoría 65535 sólo a los programas certificados

[1] En realidad, existe un mecanismo que permite efectuar esta distinción denominado camino confiable (traducción de *trusted path*). Lisex 0.0 no cuenta con ninguna forma de camino confiable. Futuras versiones de Lisex implementarán este mecanismo.

para llevar a cabo la administración MAC y a alguna terminal. De esta forma, los oficiales MAC sólo podrán loguearse en terminales físicas seguras y al hacerlo sólo podrán ejecutar esos programas. Observar que este requisito queda en manos, inicialmente, de `secadm` y, a medida que se suman otros oficiales MAC, de todos ellos. En otras palabras, los individuos en poder de estas cuentas deben cuidar de

— no asignar la categoría 65535 a ningún otro programa que no sean los programas certificados para administración MAC,

— no asignar categorías distintas a la 65535 a sus propias cuentas

⚠ **¡¡Importante!!**

De no verificarse alguno de los requisitos anteriores la seguridad de sus sistema estará en peligro pues los oficiales MAC podrán ejecutar programas que podrían ser caballos de Troya.

Si estos dos requisitos se verifican, la seguridad del sistema recae únicamente en:

— los individuos que actúan como oficiales MAC,

— los programas certificados para administración MAC, y

— el núcleo del sistema operativo.

En Lisex 0.0 fue imposible verificar todos estos requisitos aunque se logró una aproximación aceptable para un prototipo. `secadm` tiene clase de acceso $(1, \{65535\})$ pero todos los programas y librerías del sistema tienen clase de acceso $(1, \{\})$ lo que permite que `secadm` o cualquier otro oficial MAC ejecute cualquier programa. Sin embargo, se configuró la cuenta de `secadm` de forma tal que su intérprete de comandos sea el programa que permite cambiar clases de acceso, con lo cual se evita que pueda ejecutar otros programas. Más aun, el núcleo del sistema operativo controla que el proceso que invoca las llamadas al sistema que permiten alterar clases de acceso posea la categoría 65535 en su clase de acceso. De esta forma, ni si quiera `root` puede alterar clases de acceso a menos que haga SUID a algún oficial MAC, lo que es aun totalmente posible. Finalmente, Lisex 0.0 no incluye ningún programa certificado.

Los programas certificados como mínimo deberían ser los siguientes:

— Un programa para cambiar la clase de acceso de cuentas de usuarios

— Un programa para cambiar la clase de acceso de archivos y directorios

— Un programa que permita a los oficiales MAC cambiar sus contraseñas

— Un programa que actúe como intérprete de comandos; idealmente debería ejecutar sólo los programas anteriores

Separar completamente la administración MLS o MAC de la DAC requiere, por ejemplo, tener dos procesos `init` uno a nombre de `root` y otro a nombre de `secadm`; prohibir que `root` pueda hacer SUID a `secadm`, tener librerías diferentes, etc. Por lo tanto, esta característica esencial se pospone para futuras versiones.

En otro orden, idealmente, las contraseñas de las cuentas de los oficiales MAC deberían estar dividas en dos partes cada una de las cuales debería estar en poder de individuos distintos de forma tal que para loguearse se necesita la cooperación (o colusión) de dos personas.

7.4.2. Asignar y modificar la clase de acceso de una cuenta de usuario

La clase de acceso de cuentas de usuario no debe ser alterada excepto en casos excepcionales. Se supone que estos cambios se dan en muy raras ocasiones, por ejemplo, cuando un usuario es promovido, descendido, cambiado de sección, etc.

En caso de que sea necesario sólo se lo puede hacer iniciando el equipo con otro sistema operativo, modificando los fuentes del núcleo de Lisex, recompilando el núcleo e iniciando nuevamente con Lisex (ver las instrucciones precisas más abajo)[2]. Este proceso poco común se debe a que Lisex 0.0 es un prototipo. En futuras versiones de Lisex, esta tarea podrá ser llevada a cabo por un oficial MAC mientras el sistema esté en funcionamiento. Sin embargo, se podrá cambiar la clase de acceso de una cuenta de usuario sólo si dicha cuenta no posee ningún archivo abierto lo que es equivalente a que el usuario no esté logueado.

En Lisex 0.0 la clase de acceso de un usuario se asigna o modifica de la siguiente forma:

1. Debido a que Lisex 0.0 es un prototipo esta tarea de administración es compleja y más que nada pobremente implementada.

2. Para cambiar o establecer la clase de acceso de un usuario es necesario iniciar la computadora con Linux.

3. Los fuentes de la distribución de Lisex 0.0 se encuentran en algún punto del árbol de directorios del Linux con que se arrancó la computadora. Digamos que es `/linux-2-4-14(Lisex)`.

4. Editar el archivo `/linux-2-4-14(Lisex)/init/main.c`

[2] También se puede cambiar la clase de acceso de una cuenta de usuario usando el comando `chssc`, pero el cambio tendrá efecto sólo mientras no se reinicie el equipo (ver los detalles en la sección 7.4.4.

5. En ese archivo buscar el comentario

   ```
   /*Aqui se se deben establecer las clases de acceso de los
   usuarios del sistema*/
   ```

   Luego de dicho comentario encontrará el código C que se utilizó para establecer la clase de acceso de los cuatro usuarios que vienen por defecto con la distribución, es decir, `root`, `lowuser`, `highuser` y `secadm`.

6. Si desea establecer la clase de acceso de usuario nuevo se deben agregar líneas C según se explica más abajo; si desea modificar la clase de acceso de algún usuario existente sólo tiene que editar las líneas C adecuadas como se explica más abajo.

7. Primero se debe incluir una línea C para reservar el espacio en la tabla que asocia usuarios con clases de acceso:

   ```
   subjectscadd(uid, sc_init());
   ```

   donde `uid` es el UID del nuevo usuario.

8. Luego se debe indicar el nivel con una línea de la forma:

   ```
   sc_setlevel(subjectscget(uid), level);
   ```

   donde `uid` es el UID del usuario que tendrá nivel `level`. `level` es un `int`.

9. Finalmente se debe incluir una línea C por cada categoría que se desee poner en el conjunto de categorías:

   ```
   sc_addcat(subjectscget(uid), cat);
   ```

   donde `uid` es el UID del usuario que tendrá la categoría `cat`. `cat` es un `int`.

10. Una vez incluidas todas las líneas C necesarias, se debe grabar el archivo y recompilar el núcleo.

11. Ahora se necesita que ese nuevo núcleo sea el que utilice Lisex para iniciar el equipo por lo que deberá instalarlo según las instrucciones del administrador de arranque que utilice (por ejemplo, `lilo`, `grub`, etc.) (4.2)

Ejemplo 7.4.1 *Digamos que se desea crear una nueva cuenta de usuario con login* `jperon`*, uid 503 y clase de acceso* $(3, \{5, 6\})$*, entonces* `root` *edita* `/etc/passwd` *y agrega un registro con los datos pertinentes:*

```
jperon::503:503:Juan Domingo Per\'{o}n:/home/jperon/:/bin/bash
```

Luego, se debe bajar el sistema (Lisex) y re-iniciar el equipo con Linux. Cualquier usuario de Linux capaz de editar el archivo `/linux-2-4-14(Lisex)/init/main.c` *debe agregar, luego del comentario* `/*Aqui se se deben... */`*, lo siguiente:*

```
subjectscadd(503, sc_init());
sc_setlevel(subjectscget(503), 3);
sc_addcat(subjectscget(503), 5);
sc_addcat(subjectscget(503), 6);
```

Finalmente, se debe re-compilar el núcleo de Lisex y re-instalarlo de forma tal que la próxima vez que se inicie el equipo con Lisex sea con este núcleo que se acaba de compilar.

Ejemplo 7.4.2 *Supongamos ahora que se quiere cambiar la clase de acceso de* `lowuser` *a* $(2, \{0, 1, 2\})$*, entonces se debe bajar el sistema (Lisex) y re-iniciar el equipo con Linux. Cualquier usuario del Linux capaz de editar* `/linux-2-4-14(Lisex)/init/main.c` *debe buscar la línea de código de C:*

```
subjectscadd(500, sc_init());
```

pues 500 es el UID de `lowuser`*. A continuación de esta línea encontrará la siguiente:*

```
sc_setlevel(subjectscget(500), 2);
```

que establece el nivel del usuario[3]*. Como la nueva clase de acceso para* `lowuser` *tiene el mismo nivel (2) esa línea debe quedar tal como está y se deben agregar, justo después, las siguientes:*

```
sc_addcat(subjectscget(500), 0);
sc_addcat(subjectscget(500), 1);
sc_addcat(subjectscget(500), 2);
```

Finalmente, se debe re-compilar el núcleo de Lisex y re-instalarlo de forma tal que la próxima vez que se inicie el equipo con Lisex sea con este núcleo que se acaba de compilar.

Recomendamos leer la sección 7.4.5.

[3]No hay otras líneas para ese usuario pues la clase de acceso que viene con la distribución es $(2, \{\})$.

7.4.3. Eliminar la clase de acceso de una cuenta de usuario

En Lisex 0.0 eliminar la clase de acceso de una cuenta de usuario es equivalente a eliminar las líneas de código C que establecen el nivel y las categorías en el archivo `/linux-2-4-14(Lisex)/init/main.c`. Más concretamente se debe hacer lo siguiente:

1. Debido a que Lisex 0.0 es un prototipo esta tarea de administración es compleja y más que nada pobremente implementada.

2. Para eliminar la clase de acceso de un usuario es necesario iniciar la computadora con Linux.

3. Los fuentes de la distribución de Lisex 0.0 se encuentran en algún punto del árbol de directorios del Linux con que se arrancó la computadora. Digamos que es `/linux-2-4-14(Lisex)`.

4. Editar el archivo `/linux-2-4-14(Lisex)/init/main.c`

5. En ese archivo buscar el comentario

   ```
   /*Aqui se se deben establecer las clases de acceso de los
   usuarios del sistema*/
   ```

   Luego de dicho comentario encontrará el código C que se utilizó para establecer la clase de acceso de los usuarios del sistema.

6. Ahora debe localizar la línea de código C que inicializa la clase de acceso del usuario que quiere eliminar. Tal línea tiene la forma:

   ```
   subjectscadd(uid, sc_init());
   ```

   donde `uid` es el UID del usuario que se quiere eliminar.

7. Luego debe borrar esa línea y todas las líneas que le sigan que sean de la forma

   ```
   sc_setlevel(subjectscget(uid), level);
   ```

   o

   ```
   sc_addcat(subjectscget(uid), cat);
   ```

   donde `uid` es el UID del usuario que se quiere eliminar.

8. Una vez eliminadas todas las líneas C necesarias, se debe grabar el archivo y recompilar el núcleo.

9. Ahora se necesita que ese nuevo núcleo sea el que utilice Lisex para iniciar el equipo por lo que deberá instalarlo según las instrucciones del administrador de arranque que utilice (por ejemplo, `lilo`, `grub`, etc.)

Ejemplo 7.4.3 *Siguiendo con el ejemplo 7.4.1 supongamos que se quiere eliminar el usuario* `jperon` *del sistema. El primer paso es, entonces, que* `root` *o algún otro miembro del grupo* `root` *elimine el registro correspondiente del archivo* `/etc/passwd`*. Luego se debe bajar el equipo y reiniciarlo con Linux. Cualquier usuario del Linux capaz de editar el archivo* `/linux-2-4-14(Lisex)/init/main.c` *debe buscar la línea de código de C:*

```
subjectscadd(503, sc_init());
```

pues 503 es el UID de `jperon`*. A continuación de esta línea encontrará las siguientes:*

```
subjectscadd(503, sc_init());
sc_setlevel(subjectscget(503), 3);
sc_addcat(subjectscget(503), 6);
sc_addcat(subjectscget(503), 7);
```

que establece el nivel y las categorías de `jperon`*. Esas cuatro líneas deben ser borradas del archivo. Luego de grabarlo, se debe re-compilar el núcleo de Lisex y re-instalarlo de forma tal que la próxima vez que se inicie el equipo con Lisex sea con este núcleo que se acaba de compilar.*

7.4.4. El intérprete de comandos de `secadm`

El intérprete de comandos de `secadm` permite realizar dos tareas: cambiar persistentemente clases de acceso de recursos y cambiar temporalmente clases de acceso de cuentas de usuarios.

La clase de acceso de archivos o directorios no debe ser alterada excepto en casos excepcionales. Se supone que estos cambios se dan en muy raras ocasiones por ejemplo, cuando cierta información se vuelve obsoleta, cuando de un archivo se eliminan todas las referencias y/o menciones a cierto tema, cuando en un archivo se agregan datos sobre cuestiones que antes no se mencionaban, etc.

En caso de que sea necesario lo pueden hacer únicamente los oficiales MAC utilizando los programas certificados para esta tarea. Un oficial MAC podrá cambiar la clase de acceso de un archivo o directorio sólo si dicho recurso no está abierto por ningún usuario.

⚠ **¡¡Importante!!**

El programa para modificar clases de acceso debe estar certificado. Es decir, debe verificarse completamente que su implementación corresponde con su especificación y que no implementa nada más que lo descrito en ella. Caso contrario, este programa debe considerarse como un caballo de Troya en potencia.

A diferencia de lo que ocurre con las cuentas de usuario, la clase de acceso de un archivo o directorio puede ser alterada de forma persistente mientras el sistema está en ejecución.

El intérprete de comandos de `secadm` no está certificado. Este programa es muy rudimentario y no ha sido testeado. El intérprete de comandos de `secadm` se denomina `secshell`. Este programa presenta tres opciones y espera a que el usuario seleccione una de ellas:

```
1) chosc
2) chssc
3) Salir
?)
```

La primera opción permite alterar (de forma persistente) clases de acceso de archivos o directorios; la segunda puede utilizarse para cambiar temporalmente (hasta que se reinicie el equipo) la clase de acceso de una cuenta de usuario y la tercera para finalizar la sesión.

Se espera que `secshell` sea asignado como intérprete de comandos de cada nuevo oficial MAC que se configure.

chosc

Una vez que se selecciona esta opción se solicitan los siguientes datos:

Ingrese el nombre del archivo: o directorio al cual se le desea modificar la clase de acceso; debe escribirse el camino completo.

Ingrese el nivel: que se quiere asignar al archivo o directorio.

Cuantas categorías? cantidad de categorías que se quiere asignar al archivo o directorio.

Ingrese una categoría: que se quiere asignar al recurso; el programa repetirá este pedido según la cantidad ingresada en el ítem anterior; si el número fue cero este dato no será requerido.

Esta opción sobrescribe completamente la clase de acceso del archivo o directorio que se le pasa como parámetro, en otras palabras no agrega categorías al conjunto que ya estaba asociado con el recurso, simplemente "pisa" la clase de acceso anterior.

Nota 7.4.1 *Luego de haber trabajado con el sistema observamos que no permitir que los oficiales MAC puedan listar la clase de acceso de cualquier archivo, directorio o usuario complica demasiado la tarea de administración asignada a estas cuentas (es decir, modificar clases de acceso). Por lo tanto, es posible que en futuras versiones de Lisex se permita que los oficiales MAC puedan consultar la clase de acceso de cualquier recurso del sistema.*

Ejemplo 7.4.4 *Digamos que* `highuser` *crea* `/home/highuser/lowfile.tex`*. Como el sistema asigna automáticamente la clase de acceso del usuario que lo crea,* `lowfile.tex` *tendrá la clase de acceso de* `highuser` *es decir* $(4, \{0, 1, 6\})$*. Supongamos que tal clase de acceso es excesivamente alta en relación con la información que contendrá el archivo. Por lo tanto,* `highuser` *deberá solicitar a un oficial MAC que modifique la clase de acceso del archivo* `lowfile.tex` *a, por ejemplo,* $(2, \{\})$*. De esta forma el archivo podrá ser consultado por* `lowuser` *y cualquier otro usuario con clase de acceso superior*[4]*.*

En consecuencia un oficial MAC se logueará y seleccionará la opción 1) de su menú e ingresará los siguientes datos:

```
Ingrese el nombre del archivo: /home/highuser/lowfile.tex
Ingrese el nivel: 2
Cuantas categorias? 0
```

Ejemplo 7.4.5 *Si se quiere cambiar la clase de acceso de* `tty1` *de* $(4, \{0, 1, 6\})$ *a* $(4, \{0, 1, 6, 2\})$*, entonces un oficial MAC deberá loguearse e ingresar lo siguiente luego de seleccionar la opción 1:*

```
Ingrese el nombre del archivo: /dev/tty1
Ingrese el nivel: 4
Cuantas categorias? 4
Ingrese una categoria: 1
Ingrese una categoria: 0
Ingrese una categoria: 6
Ingrese una categoria: 2
```

Notar que las categorías pueden ingresarse en cualquier orden.

Recomendamos leer la sección 7.4.5.

[4] Tener en cuenta que además de la restricción MLS sobre `lowfile.tex` puede haber restricciones DAC (es decir, `lowuser` puede no estar en la ACL del archivo) o restricciones en el camino de directorios que llevan a `lowfile.tex` que impidan que otros usuarios lo consulten. De todas formas, la restricción MLS es la única que brinda algún grado de confianza en que el archivo no podrá ser consultado por usuarios no autorizados, es decir, por atacantes que utilicen caballos de Troya.

chssc

Recordamos que esta opción cambia la clase de acceso de una cuenta de usuario sólo hasta que el sistema se reinicia.

Una vez que se selecciona esta opción se solicitan los mismos datos que en la opción 1 excepto que el primer dato pide un UID en lugar de un nombre de archivo:

Ingrese el uid: al cual se le desea modificar la clase de acceso; no acepta nombres de login.

7.4.5. Crear/eliminar niveles de seguridad o categorías

En Lisex 0.0 no se crean ni eliminan niveles de seguridad o categorías. Simplemente, si a una cuenta de usuario, a un archivo o a un directorio se le asigna una clase de acceso que contiene un nivel o una categoría no utilizada hasta el momento, el sistema hará todo lo necesario para que tal categoría pueda ser asignada.

Simétricamente, se elimina un nivel o una categoría si ninguna cuenta de usuario, archivo o directorio la utilizan.

Por lo tanto, si se desea mantener los niveles o las categorías en cierto conjunto y evitar errores humanos en la asignación de clases de acceso, se lo debe hacer a nivel de usuario. En otras palabras, deben ser los programas certificados para cambiar clases de acceso (tanto de cuentas de usuario como de archivos o directorios) los que impongan esta restricción en caso necesario.

7.5. Instalar nuevo software

Dado que Lisex 0.0 es un prototipo sobre el cual probar la seguridad y utilidad de sistemas MLS, no hemos implementado alguna forma de agregar programas al sistema. No hemos incluido un compilador ni es posible montar otros tipos de sistemas de archivos diferentes a Lisex 0.0. Los únicos programas que pueden agregarse son *shell scripts* que cualquier usuario puede programar.

La única forma de agregar otros programas es iniciar la computadora con Linux, copiar con `debugfs` los programas en la partición destinada a Lisex 0.0, y, también utilizando `debugfs`, configurar las ACL y clases de acceso de cada uno de ellos. Más aun, en todo este proceso debe utilizarse una versión especial de `debugfs`, modificada por el equipo de desarrollo de Lisex. Esta versión de `debugfs` no ha sido incluida en la distribución de Lisex 0.0 pero puede solicitarse una copia al equipo de desarrollo de Lisex en lisex@sol.info.unlp.edu.ar.

Apéndice A

Usabilidad de Lisex 0.0

Lisex 0.0 implementa un modelo MLS muy semejante al presentado en los trabajos originales de Bell y La Padula (BLP). La razón fundamental que nos llevó a optar por este modelo, habiendo otros que lo superaron, fue que BLP es, tal vez, el modelo más conocido, estudiado e implementado hasta el momento. Además, a raíz de comunicaciones personales con otros investigadores en el tema, nos interesó evaluar la usabilidad de un sistema que implementara el modelo mencionado.

Luego de haber implementado el modelo y luego de haber usado el prototipo por cierto tiempo resulta evidente que mucho de lo que se gana en seguridad se pierde en usabilidad. Más precisamente, muchas acciones o secuencias de acciones que deberían estar permitidas pues no violan la política de seguridad, el sistema las prohibe. En otras palabras, comandos que en UNIX son ejecutados regularmente, en Lisex fallarán muy a menudo.

Por ejemplo, de haber incluido en las llamadas al sistema que trabajan con tubos sin nombre los controles MLS, cada vez que un usuario ejecutara un comando tan simple como:

```
$ sort file1 | grep text >file2
```

podría fallar por falta de permisos aun cuando el usuario que lo ejecuta aparentemente poseyera los permisos necesarios. Notar que decimos *podría* y no simplemente que fallará pues, precisamente, hay situaciones en las que fallará y otras en que no. Por la misma esencia de los modelos MLS (al menos aquellos semejantes a BLP), la posibilidad de realizar ciertas acciones sobre ciertos recursos no está dada únicamente por los atributos de seguridad asociados a esos recursos sino también por la relación que existe entre los atributos de seguridad de esos recursos y de otros recursos utilizados indirectamente (cf. confinamiento, sección 1.6.3). Volviendo al ejemplo del comando anterior vemos que intervienen las clases de acceso de los siguientes recursos:

— El programa `sort`

— El archivo `file1`

— El tubo que interconecta `sort` con `grep`

— El programa `grep`

— La terminal en la que se ejecuta el comando

— La cuenta de usuario que lo ejecuta

— El directorio donde (posiblemente) se crea `file2`

Es decir que no sólo intervienen los atributos de los recursos más evidentes sino también los de aquellos que en Linux normalmente no tienen incidencia en el control de acceso. Por ejemplo, el comando puede fallar si

— La clase de acceso del usuario domina estrictamente a la de la terminal y esta domina a la de `file1`, en cuyo caso el comando fallará pues la interconexión hereda la clase de acceso del usuario y se abre en modo read-write lo que viola el confinamiento respecto de la terminal.

En nuestra opinión no es razonable que el comando falle por ese motivo pues no hay nada, excepto la implementación del sistema, que impida a ese usuario ver esa información en esa terminal. Claramente, esto se debe a la decisión de asignar a los objetos recientemente creados la clase de acceso del usuario que los crea. Sin embargo, implementado los controles de acceso en tiempo de `open` no quedan muchas más opciones que en otras situaciones no presenten problemas similares.

Existen otras decisiones que presentan problemas semejantes pero de menor impacto en la usabilidad del sistema (por ejemplo, la regla que impide consultar los atributos DAC de un archivo o directorio a sus dueños).

Futuras versiones de Lisex implementarán modelos de seguridad resistentes a ataques llevados a cabo con caballos de Troya pero que no reduzcan innecesariamente las acciones de los usuarios.

Bibliografía

[AJP95] Marshall D. Abrams, Sushil Jajodia, and Harold J. Podell. *Information Security: an integrated collections of essays.* IEEE Computer Society press, 1995.

[BL73a] D. Elliot Bell and Leonard LaPadula. Secure computer systems: Mathematical foundations. *Technical Report MTR-2547*, I-III, December 1973.

[BL73b] D. Elliot Bell and Leonard LaPadula. Secure computer systems: Mathematical model. *Technical Report ESD-TR-73-278*, II, November 1973.

[CDG+02] Maximiliano Cristiá, Martín Degrati, Pablo Garralda, Gisela Giusti, and Felipe Manzano. Lifia secure linux (lisex): un sistema operativo seguro basado en linux. September 2002.

[Com91] Commission of the European Communities. *Information Technology Security Evaluation Criteria (ITSEC), Provisional Harmonized Criteria: Version 1.2.* Office for Official Publications of the European Communities, Luxemburgo, June 1991.

[Coq] coq.inria.fr.

[Cou96] Council of Europe. Draft recommendation r(96): On the protection of medical data. April 1996.

[Cri02] Maximiliano Cristiá. Formal verification of an extension of a secure, compatible unix file system, 2002.

[CW96] Edmun Clarke and Jannet Wing. Formal methods: state of the art and future directions. *ACM Computing Surveys*, 18(4):626–643, December 1996.

[Gas88] Morrie Gasser. *Building a Secure Computer System.* Van Nostrand Reinhold, 1988.

[GM02] Gisela Giusti and Felipe Manzano. Manual de referencia para el programador. lisex 0.0. *Publicación interna del Lifia*, 2002.

[McL90] John McLean. The specification and modeling of computer security. *IEEE Computer*, 23(1):9–16, January 1990.