L I S E X

GIDIS

Introducción a la seguridad de Lisex 0.0

Maximiliano Cristiá

GIDIS

Grupo de Investigación y Desarrollo en Ingeniería de Software

Resumen

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].

Lisex es un sistema operativo pensado para resistir ataques a la *confidencialidad* (1) efectuados por medio de *caballos de Troya* (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, 6) en el núcleo del sistema, y

— un desarrollo del sistema que utilice *métodos formales* (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 (7) tradicional de Linux y otras características de seguridad menores.

Índice

[1] Los números entre paréntesis, como (1) o (2), indican secciones de este manual donde se puede encontrar más información sobre el tema que está siendo tratado.

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 (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]).

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.

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].

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

[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.

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, 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.

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.

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

$$(SECRET, \{NATO, NUCLEAR\}) \tag{1}$$

$$(TOPSECRET, \{F14, NUCLEAR, CIA\}) \tag{2}$$

donde la clase de acceso 1 tiene nivel $SECRET$ *y conjunto de categorías* $\{NATO, NUCLEAR\}$*, mientras que la clase 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.*

4.1. Orden parcial de las clases de acceso

Definición 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 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

$$CA_1 \succeq CA_2 \quad (3)$$
$$CA_1 \succeq CA_4 \quad (4)$$
$$CA_1 \succeq CA_5 \quad (5)$$
$$CA_1 \succeq CA_6 \quad (6)$$
$$CA_1 \succeq CA_7 \quad (7)$$
$$CA_1 \not\succeq CA_3 \quad (8)$$
$$CA_3 \not\succeq CA_1 \quad (9)$$

$$
\begin{aligned}
CA_2 &\succeq CA_6 && (10)\\
CA_2 &\succeq CA_7 && (11)\\
CA_3 &\succeq CA_6 && (12)\\
CA_3 &\succeq CA_7 && (13)\\
CA_4 &\not\succeq CA_3 && (14)\\
CA_3 &\not\succeq CA_4 && (15)
\end{aligned}
$$

Donde 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 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, 5 es verdadera. Por otro lado, 8 y 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 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.

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 (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 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.

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) por medio de caballos de Troya (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. (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. (6.3)

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

6.1. Comercio electrónico en sistemas MLS

Aquí revisaremos el ejemplo introducido en la sección 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.

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 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 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 (6.3) que, en el caso general, impide ciertas solicitudes de apertura.

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: 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 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 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 15

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 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

6.3.1. 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.

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.

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).

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.

Referencias

[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.

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