Protección de componentes y dispositivos de seguridad mediante un control de acceso basado en kernel

Joaquin Garcia-Alfaro[1], Sergio Castillo[1], Jordi Castellà-Roca[2], Guillermo Navarro[1]

[1] Departament d'Enginyeria de la Informació i les Comunicacions,
ETSE, Universitat Autònoma de Barcelona,
Edifici Q, 08193 Bellaterra, Barcelona,
Email: {jgarcia,scastillo,gnavarro}@deic.uab.es

[2] Departament d'Enginyeria Informàtica i Matemàtiques,
ETSE, Universitat Rovira i Virgili,
Av. Països Catalans, 26, E-43007 Tarragona
Email: jordi.castella@urv.net

Resumen En este artículo presentamos el uso de un mecanismo para la protección de componentes y dispositivos de seguridad en red. Nuestra solución consiste en un control de acceso que intercepta y cancela llamadas al sistema que ponen en peligro la seguridad del componente. De esta manera, si un atacante compromete la seguridad del componente, o de alguno de sus elementos, no será capaz de tomar el control absoluto del sistema – aún habiendo obtenido privilegios de administración. Para resolver las restricciones que ocasiona nuestra propuesta a las tareas normales de un administrador legítimo del sistema, proponemos también en este artículo un esquema de autenticación basado en el uso de tarjetas smart-card, para poder garantizar la identidad del administrador. A través del uso de un protocolo criptográfico, el mecanismo de protección verifica las acciones de administración antes de otorgar los privilegios necesarios para manipular un componente. De lo contrario, el mecanismo de control de acceso entrará en vigor para evitar esta manipulación ilegal. Por último, mostramos también en este artículo, un primer prototipo de nuestra propuesta, desarrollado para sistemas de GNU/Linux, sobre el entorno de trabajo *Linux Security Modules* (LSM).

Palabras clave: Seguridad en Redes, Control de Acceso, Protección de Componentes

1. Introducción

La mayor parte de la investigación actual en el ámbito de componentes de seguridad de red, tales como sistemas cortafuegos y sistemas de detección de intrusos, está principalmente enfocada a mejorar sus mecanismos de clasificación, detección y/o reacción, sin tener en cuenta su propia seguridad. La protección de estos componentes es un serio e

importante problema a resolver ya que, si un atacante consigue manipular la seguridad de estos componentes, será capaz de obtener un control global sobre el propio sistema.

Contrariamente a otros elementos de la red, los componentes de seguridad de un sistema funcionan, por lo general, con privilegios especiales de administración. Por ello, un ataque de intrusión en dichos componentes puede llevar a un atacante a adquirir dichos privilegios de una manera desautorizada. La existencia de errores de programación en el código principal de estos componentes, así como la manipulación ilícita de sus recursos, tales como procesos secundarios o sus ficheros de configuración, son solo un par de ejemplos de cómo un atacante podría evadir la seguridad de estos componentes, y obtener sus privilegios de administración.

En este artículo presentamos SMARTCOP (*Smart Card Enhanced Linux Security Module for Component Protection*), un módulo de protección de componentes a nivel de sistema operativo. SMARTCOP intercepta y cancela llamadas al sistema consideradas ilegales y que atentan contra los componentes a ejecutar. El mecanismo de la protección integrado en SMARTCOP consiste en un control de acceso integrado en el *kernel* del sistema operativo bajo el que se ejecuta el componente.

Esta estrategia introduce, sin embargo, una serie de restricciones de administración, ya que intercepta y cancela posibles acciones ejecutadas por un administrador legítimo del sistema. Para resolver éste problema, SMARTCOP incorpora un mecanismo de autenticación basado en tarjetas smart-card. El objetivo de este mecanismo de autenticación es doble. En primer lugar, ofrece al administrador privilegios indispensables para llevar a cabo sus actividades de administración siempre y cuando acredite satisfactoriamente su identidad. En segundo lugar, refuerza la seguridad de nuestra propuesta contra ataques de escalada de privilegios, tales como robo de contraseñas, ataques a diccionario, o explotación de desbordamientos de buffer.

El resto del documento está organizado de la siguiente manera. Los antecedentes y trabajos relacionados con nuestra propuesta son presentados en la sección 2. La sección 3 ofrece una visión general de nuestro esquema de protección. La sección 4 presenta una breve descripción del desarrollo actual de SMARTCOP en un primer prototipo software para sistemas GNU/Linux. La sección 5 presenta un protocolo criptográfico basado en tarjetas smart-card y que da respuesta a las restricciones de administración introducidas por el mecanismo de protección. Por último, la sección 6 muestra una evaluación del rendimiento de SMARTCOP, y la sección 7 cierra el artículo con una serie de conclusiones a raíz del trabajo presentado.

2. Antecedentes

Existen diversos mecanismos para garantizar una ejecución de procesos segura en sistemas operativos actuales. Una primera aproximación nos lleva a la creación de entornos restringidos, donde procesos con privilegios especiales de administración son ejecutados y controlados fuera del espacio del sistema de confianza. Una segunda aproxima-

ción, más cercana a la que se presenta en este artículo, es la aplicación de un control de acceso sobre las llamadas al sistema por parte del kernel del sistema operativo.

En [4], por ejemplo, encontramos una propuesta para la creación de entornos restringidos dentro de sistemas Unix (o variantes). En esta primera propuesta, los autores presentan el uso de una llamada al sistema que permite restringir a un proceso el área del sistema de ficheros sobre el que tiene acceso. De este modo, el proceso permanece encerrado en un espacio seguro, del cual no puede escapar – aun cuando el proceso se vea comprometido por un fallo de seguridad, el resto del sistema permanecerá a salvo de las actividades ilícitas que se pudieran originar dentro de dicho entorno cerrado.

Esta propuesta requiere, sin embargo, una réplica del sistema de ficheros original para cada entorno que se vaya a crear. El administrador deberá reproducir todas aquellas partes del sistema de ficheros necesarias por el proceso a ejecutar. Así pues, será necesario replicar, por ejemplo, librerías, ficheros de configuración, etcétera. Otra desventaja de esta propuesta es que no garantiza el correcto flujo de ejecución del proceso, es decir, la conducta del proceso puede ser modificada a partir de, por ejemplo, una explotación de desbordamiento de buffer. De este modo, el atacante podría tratar de modificar la configuración o archivos de registro (*logs*) de tal proceso o, en términos generales, cualquier archivo contenido en el sistema de ficheros restringido.

Una extensión de la propuesta anterior es el modelo sandbox del lenguaje Java [3,12,6]. Al igual que en la propuesta anterior, en el modelo sandbox también se ejecutan procesos dentro de entornos controlados. Este modelo, a diferencia de la propuesta anterior, permite realizar un control de acceso a recursos (incluyendo el sistema de ficheros al completo, API de dispositivos de E/S, *threads*, *sockets*, etcétera), así como una garantía en la integridad de los recursos asociados al proceso – dichos recursos no pueden ser manipulados mediante inyección de código maligno.

No obstante, esta segunda propuesta sigue sin dar respuesta a un posible uso mailintencionado de vulnerabilidades existentes fuera del entorno de confianza. Un simple error de programación en un proceso con privilegios especiales, o el uso de contraseñas robadas, puede llevar al atacante, desde el exterior del entorno protegido, a un compromiso de los procesos y/o recursos protegidos.

De acuerdo con la segunda aproximación, es decir, protección mediante un control de acceso sobre llamadas al sistema, encontramos en [11] y [9] dos propuestas. El principal objetivo de éstas es reforzar la seguridad del sistema, controlando las acciones solicitadas por procesos y/o usuarios, así como los recursos contra los que se van a realizar dichas acciones. La habilidad de controlar el acceso a los recursos permite una protección eficiente, asegurando que nada o nadie en el sistema (incluyendo a un posible atacante con privilegios de administrador) puede violar dicha seguridad.

Es por ello que ambas aportaciones han sido utilizadas como punto de partida para nuestra propuesta, tratando de extender y adaptar su diseño general, a una propuesta más específica y acorde a nuestras necesidades, tal y como apuntamos en las próximas secciones.

3. Nuestra propuesta

Como introducíamos al inicio de la sección 1, nuestra principal motivación es la protección de componentes en dispositivos de seguridad de red, tales como sistemas cortafuegos (*Firewalls*), sistemas de detección de intrusos (*Intrusion Detection Systems, IDSs*), o sistemas de prevención de intrusos (*Intrusion Prevention Systems, IPSs*) que, en caso de ser atacados con éxito, pueden proporcionar a un atacante los privilegios necesarios para hacerse con el control del sistema al completo.

Esta situación nos lleva a la necesidad de incorporar mecanismos de protección sobre los distintos elementos de cada componente, tratando de mitigar o eliminar cualquier tipo de acción que atente contra tales elementos y su correcto funcionamiento. De esta forma, aunque el atacante logre comprometer la seguridad del componente, no logrará hacerse con el control del sistema.

De acuerdo con [8], podemos considerar la protección de los elementos a nivel del *kernel* (núcleo) del sistema operativo que los alberga. Por un lado, este tipo de protección garantiza que cualquier llamada al sistema (*system call*) potencialmente hostil contra el componente o parte de sus elementos (por ejemplo, una llamada al sistema *kill* para detener la ejecución de un proceso) podría producirse para detener o hacerse con el control de dicho elemento. Esta protección se traduce en la incorporación de un sistema de control de acceso a las llamadas del kernel del sistema operativo, de manera que pueda autorizar o denegar una llamada en función de diversos criterios, tales como el identificador del proceso que la realiza, parámetros de la llamada, etcétera.

El control de acceso del kernel permite, por tanto, eliminar el concepto de confianza asociado a usuarios privilegiados, relegando la autorización de ejecución de una llamada al control de acceso interno. Adicionalmente, y a diferencia de otros métodos de protección, proporciona un método unificado, evitando la implementación de distintas técnicas específicas para cada componente.

Por otro lado, esta metodología nos permite aplicar el principio de compartimentalización [10]. Este principio de seguridad intenta minimizar el daño que puede ser ocasionado a un sistema, segmentándolo en varios elementos que pueden ser protegidos independientemente. De esta manera, aunque uno de los elementos del componente sea comprometido, podríamos seguir confiando en el correcto funcionamiento del resto.

En nuestro caso particular, distintos elementos de cada componente son ejecutados como procesos. Definiendo los permisos adecuados en el control de acceso, basándonos en el identificador de proceso, por ejemplo, podemos limitar el entorno de interacción de cada elemento en relación al resto de elementos del componente. Con esto conseguimos que si un atacante se hace con el control de un proceso asociado a un componente a través de un desbordamiento de buffer, por ejemplo, estará limitado a realizar aquellas llamadas definidas para éste.

Sin embargo, no siempre es posible obtener una independencia total entre elementos. Es necesario definir de forma específica cuales deben ser las llamadas al sistema conside-

radas como una amenaza cuando son lanzadas hacia un elemento de cada componente. Esto requiere un estudio meticuloso sobre cada una de las llamadas que proporciona el kernel del sistema operativo y de que manera particular pueden ser aprovechadas de forma maliciosa. Y, en segundo lugar, es preciso definir el control de acceso para cada una de estas llamadas. Para nuestro trabajo, proponemos considerar los siguientes tres niveles de protección para dicho proceso de clasificación de llamadas del kernel:

- **Protección de procesos críticos**: engloba toda aquella acción que pueda atentar contra la correcta ejecución de los procesos asociados al componente, ya sea por la interacción sobre estos a través de señales, o la manipulación de su espacio de memoria. Algunos ejemplos serían: ejecución de nuevos programas ya en memoria, cancelación, manipulación del espacio de direcciones y trazado de procesos, etcétera.

- **Protección de mecanismos de comunicación**: recoge todo aquel proceso por el cual un atacante consigue alterar, generar o eliminar cualquier tipo de mensaje intercambiado entre los elementos de un componente.

- **Protección de archivos asociados a los componentes**: considera cualquier acción malintencionada dirigida hacia los archivos que utilizan o manipulan los elementos de un componente, tales como archivos ejecutables, de configuración o de registro.

4. Implementación de un primer prototipo

En esta sección presentamos brevemente el desarrollo actual de SMARTCOP. De acuerdo con el esquema de protección propuesto en la sección anterior, SMARTCOP consiste en un mecanismo de control de acceso integrado en el kernel del sistema operativo, y ha sido desarrollado mediante el marco de trabajo de módulos seguros de Linux (*Linux Security Modules*, LSM) [11], integrado de serie en las versiones del kernel 2.6.x del sistema operativo GNU/Linux.

LSM no proporciona un mecanismo de control de acceso específico, sino que provee un marco de trabajo a través de un conjunto de puntos de intercepción (*hooks*) a lo largo del kernel del sistema operativo. De esta manera, es posible implementar distintas estrategias para realizar un control de acceso. Básicamente distingue los siguientes puntos de intercepción: *Task hooks*, *Program Loading Hooks*, *Filesystems Hooks* y *Network hooks*.

Estos puntos permitirán establecer protección en los tres niveles que enunciábamos en la sección 3. Adicionalmente, LSM otorga un conjunto de beneficios a nuestra implementación que afianzan nuestros objetivos. En primer lugar, introduce una carga mínima en el sistema en comparación a un kernel sin soporte para LSM, sin influir considerablemente en los tiempos de los procesos del componente (ver sección 6). En segundo lugar, el mecanismo de control de acceso puede ser integrado en el sistema como un

módulo, sin necesidad de recompilar el kernel del sistema operativo, reforzando así la modularidad e independencia entre componentes. Por último, y a diferencia de otras propuestas como [8] [9], cuyas implementaciones requieren la modificación de algunas funcionalidades de la versión original 2.6.x del kernel Linux, proporciona un alto grado de flexibilidad y portabilidad a nuestra implementación a través de la concepción de un marco genérico para el control de acceso.

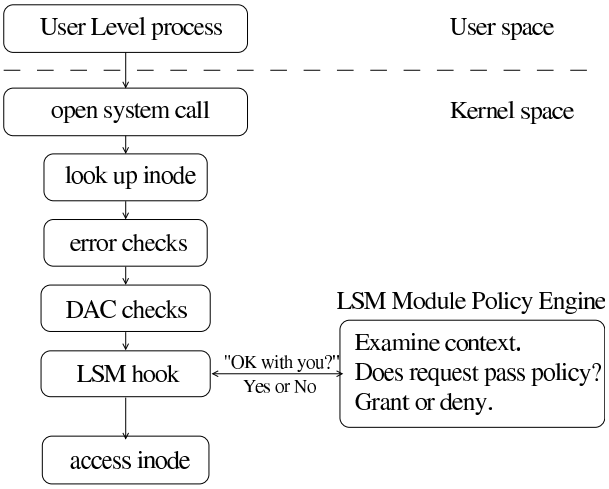

Figura 1. Puntos de intercepción de LSM.

La abstracción que proporciona LSM a través de su interfaz permite que los módulos medien entre los usuarios y los objetos internos del kernel del sistema operativo. Para esto, previo acceso al objeto interno, el punto de intercepción llama a una función que el módulo debe proveer y que será la responsable de conceder o denegar el acceso. En la figura 1 podemos observar esta idea, en la que un módulo registra una función para realizar un control sobre los *inodos* del sistema de ficheros.

A su vez, LSM permite conservar el control de acceso discrecional (*Discretionary Access Control*, DAC) que el kernel Linux proporciona, al interponerse entre dicho DAC y el objeto en si. De esta manera, si un usuario no tiene permisos en relación a un determinado archivo, por ejemplo, el DAC del sistema operativo le denegará el acceso sin llamar a la función que registró el módulo LSM. Esta arquitectura reduce la carga que se introduciría en el sistema en comparación a un control de acceso para la protección de los componentes centralizado en la interfaz de llamadas del sistema operativo. En este último caso, las correspondientes comprobaciones asociadas a la protección siempre se realizarían y, posteriormente, se cedería el control al mecanismo DAC del sistema operativo, lo que supondría realizar, en la mayoría de veces, un trabajo innecesario.

Los elementos de un componente de seguridad deben poder realizar operaciones solo permitidas para el administrador del sistema (como, por ejemplo, filtrado de paquetes,

cancelación de procesos, etcétera). Esto implica que los procesos de sistema asociados a los componentes se deben ejecutar con privilegios de administración. De lo contrario, el mecanismo DAC del kernel del sistema operativo no permitirá la ejecución de determinadas llamadas.

El control de acceso integrado en el kernel se basará en el identificador de proceso (*Process Identifier*, PID) que realiza la llamada y que estará asociado a un elemento específico. Cada función registrada por un módulo LSM, determinará qué componente realiza la llamada a partir de su PID y, a partir de esta información, aplicará el control de acceso basándose también en los parámetros de la llamada. Así, por ejemplo, un componente podrá acceder a sus archivos de configuración, pero no a los de otro.

Un aspecto a tener en cuenta desde el punto de vista de la implementación, es la propia administración de los controles de acceso y la gestión de los componentes. Como exponíamos en la sección 3, un usuario con privilegios de administración no será capaz de lanzar ninguna llamada del sistema que atente contra la seguridad del componente, o de alguno de sus elementos, evitando así que un posible atacante con privilegios de administrador consiga obtener el control del componente. Este hecho se contrapone con la gestión y administración de componentes ya que un administrador no podrá interactuar ni realizar ninguna operación considerada potencialmente peligrosa.

Para solventar esta problemática, proponemos en la siguiente sección el uso de un mecanismo de autenticación basado en tarjetas smart-card que, en caso de validar la identidad de un administrador legal, otorgará los privilegios necesarios para la manipulación y gestión de un componente. De lo contrario, el módulo LSM conservará las restricciones especificadas para asegurar la seguridad de los elementos de dicho componente.

5. Mecanismo de autenticación basado en tarjetas smart-card

Con el objetivo de asegurar correctamente la identidad de un administrador legítimo de un componente o dispositivo protegido mediante SMARTCOP, proponemos en esta sección el uso de un mecanismo de autenticación de dos factores basado en las funciones criptográficas de una tarjeta smart-card. Este mecanismo garantiza los siguientes requisitos:

- La ejecución de las acciones controladas por SMARTCOP deben ser previamente autorizadas por el uso de una tarjeta smart-card;
- La tarjeta únicamente autorizará una acción si el PIN introducido es el correcto;
- El módulo LSM únicamente autorizará la ejecución de dicha acción si la evaluación del protocolo de autenticación por parte de la tarjeta smart-card es válido.

A continuación, se describe en la siguiente sección la estructura y los elementos necesarios para el mecanismo de autenticación propuesto.

5.1. Arquitectura

Definimos la arquitectura del mecanismo de autenticación como una estructura jerárquica formada por varias unidades organizativas, donde la red está a su vez divida en dominios jerárquicos con componentes que deben ser protegidos. Estos componentes, o nodos, son llamados *SMARTCOP Node* (SCM). Por último, cada dominio tiene un *SMARTCOP Server* (SCS), y cada administrador está en posesión de una *smart-card*: *SMARTCOP CARD* (SCC). Describimos a continuación estos componentes.

SMARTCOP Server (SCS) Cada SCS posee un par de llaves criptográficas a las que llamamos *master-key* y su correspondiente certificado. Este certificado es emitido por un SCS superior en la jerarquía e identifica al SCS como válido (o de confianza). El certificado es un certificado de atributo X.509 [13], donde el emisor es la *master-key* del SCS superior, y el sujeto es la *master-key* del SCS certificado.

El SCS del dominio B puede emitir certificados autorizando a una SCC como administrador del dominio B. Generalmente el SCS será gestionado por el administrador a cargo de la red del dominio en cuestión – o unidad organizativa. Es decir, la persona que tiene más conocimiento sobre el dominio de red y sus potenciales administradores, y al mismo tiempo la que tiene el mayor interés en llevar a cabo una buena administración. Este es un punto clave del sistema SMARTCOP, que permite distribuir la gestión de la administración entre dominios o unidades organizativas.

SMARTCOP Node (SCN) Un SCN es un componente que tiene el módulo LSM SMARTCOP. Los parámetros de seguridad del módulo LSM son debidamente inicializados durante su instalación. El principal parámetro es el *Source-of-Authority* (SoA), que se representa mediante una *master-key*. De manera más precisa, la *master-key* del SCS superior. Cuando un administrador realiza una acción protegida en un SCN determinado, utilizando el protocolo 1, el SCN verifica el certificado de la SCC. Después, si viene de un camino de certificación cuya raíz es la *master-key* del SoA, la operación se acepta.

SMARTCOP Card (SCC) Un administrador potencial posee una SCC. Para poder llevar a cabo tareas de administración en un dominio determinado, la SCC debe ser autorizada (o certificada) por el SCS del dominio u otro de un dominio superior en la jerarquía.

Cada SCC tiene un par de llaves criptográficas, que tienen que estar certificadas por una *master-key* (es decir, por un SCS). El motor criptográfico de una *smart-card* es capaz de procesar varias funciones criptográficas, como generación de llaves asimétricas, ejecución de algoritmos de cifrado asimétrico, etcétera. La SCC tiene un *PIN* de operaciones y un *password* administrativo (ver sección 5.2, protocolo 1). El PIN tiene un mínimo de

seis dígitos y se utiliza para autorizar acciones protegidas. De otra parte, el password se utiliza para poder cambiar el PIN y realizar otras tareas de gestión. El administrador del sistema tiene tres intentos consecutivos para introducir el PIN. En el tercer intento, si la *smart-card* recibe un PIN incorrecto, se auto-bloquea. La *smart-card* solo puede ser desbloqueada mediante el password administrativo. De la misma manera, hay tres intentos para introducir el password. Si estos son incorrectos, la tarjeta se bloquea y queda inutilizable.

5.2. Descripción del protocolo

En esta sección se detalla el protocolo criptográfico en el que se basa nuestro mecanismo de autenticación mediante en una tarjeta smart-card.

El administrador del sistema en el paso 1 del protocolo solicita al módulo LSM que se ejecute una acción. A continuación el módulo LSM bloquea el canal de comunicación entre éste y el lector de tarjetas smart-card (ver el paso 2*a*). La información que viaja por este canal es confidencial, es decir, sólo puede ser accedida por el módulo LSM y el lector de tarjetas smart-card. El protocolo no permite que la tarjeta esté introducida en el lector cuando no es necesario.

En el paso 2*c*, el módulo LSM espera hasta que la tarjeta smart-card no sea introducida en el lector, y en el paso 4*e* el módulo LSM no realiza ninguna acción hasta que ha sido extraída del lector.

En el paso 3 el PIN de operación viaja de forma segura desde el teclado hacia el módulo LSM. La seguridad del canal está garantizada por el módulo LSM que ha bloqueado el canal de comunicación entre el teclado y el propio módulo. El módulo LSM envía un valor pseudo-aleatorio, NONCE, y el PIN en el paso 4*c*. La tarjeta smart-card devuelve como respuesta la firma digital de NONCE calculada con su llave privada. Finalmente, el módulo LSM verifica que la firma digital haya sido calculada de forma correcta y que el certificado digital correspondiente es válido (ver el paso 4*g*).

Protocolo 1

1. *El administrador del sistema abre una consola de comandos y pide que se realice la acción* X[3].
2. *LSM recibe la petición de la consola y realiza los pasos siguientes:*
   a) *Establecer una conexión con el lector de tarjetas smart-card. A partir de este momento el canal de comunicación entre el lector de tarjetas smart-card y LSM no puede ser accedido por ningún otro proceso. La comunicación es confidencial entre el lector de tarjetas smart-card y LSM.*
   b) *Mostrar un mensaje en la consola indicando que la tarjeta smart-card debe ser introducida en el lector.*

[3] Suponemos que X sólo se puede llevar a cabo mediante la autorización de la tarjeta smart-card.

c) *Mientras la tarjeta smart-card no sea introducida hacer:*
   1) *Detectar la inserción de la tarjeta smart-card.*

d) *Mostrar un mensaje en la consola indicando que se debe introducir el PIN de operación.*

3. *El administrador del sistema introduce el PIN de operación mediante el teclado.*[4]
4. *LSM realiza los pasos siguientes:*
   a) *Obtener el PIN de operación.*
   b) *Obtener un valor pseudo-aleatorio NONCE.*
   c) *Ejecutar el Procedimiento 1 en el interior de la tarjeta smart-card utilizando el PIN de operación y NONCE, y obteniendo μ como respuesta.*
   d) *Mostrar un mensaje en la consola indicando que la tarjeta smart-card se debe extraer del lector.*
   e) *Mientras la tarjeta smart-card no ha sido extraída hacer:*
      1) *Detectar la extracción de la tarjeta smart-card.*
   f) *si μ es* ERROR *LSM no autoriza la acción X.*
   g) *sino hacer:*
      1) *Verificar que el certificado de la tarjeta smart-card haya sido emitido por una CA que pertenece a la cadena de certificación que tiene como certificado raíz el certificado del par de* llaves maestras *(SoA).*
      2) *Verificar el certificado de la tarjeta smart-card contra una CRL válida.*
      3) *Verificar la firma digital μ con la llave pública P_K obtenida del certificado de la tarjeta smart-card, $P_K(\mu) \stackrel{?}{=} H(NONCE)$.*
      4) *si la verificación es correcta LSM autoriza la acción X.*
      5) *si la verificación no es correcta LSM no autoriza la acción X.*

El Procedimiento 1 se ejecuta en el interior de la tarjeta smart-card. La tarjeta smart-card valida el PIN de operación, y si es correcto calcula la firma digital de NONCE con la llave privada almacenada en su interior.

Procedimiento 1 $[PIN, NONCE]$

1. *Validar el PIN de operación;*
2. *Si el PIN de operación es correcto hacer:*
   a) *Calcular la firma digital de NONCE con la llave privada S_K, $\mu = S_K(NONCE)$;*
   b) *enviar μ como respuesta;*
3. *Si el PIN de operación no es correcto enviar* ERROR *como respuesta;*

[4] Recordamos que el canal de comunicación entre el teclado y LSM no puede ser accedido por ningún proceso.

5.3. Consideraciones de protección

Para asegurar una correcta ejecución del mecanismo de autenticación y protección presentado en esta sección, es necesario garantizar la protección de las entidades y canales de comunicación involucrados en tales procesos (para evitar, por ejemplo, un ataque de impersonalización o una manipulación de la información intercambiada). En el supuesto de no poder asegurar dicha protección, los mecanismos propuestos carecerían de sentido. Por ello, enunciamos a continuación una serie de consideraciones de seguridad necesarias para garantizar la viabilidad de nuestra propuesta.

En primer lugar, es importante recordar que un posible ataque de manipulación contra los binarios (ejecutables) de la consola de ejecución, y demás aplicaciones asociadas, puede suponer una inyección, y en consecuencia, ejecución de código maligno en tiempo de ejecución. A través de esta inyección de código maligno, el atacante podría desviar el flujo de ejecución del proceso de autenticación y/o protección, desviando el resultado hacia una comprobación válida, y tomar el control del sistema – o, incluso, obtener el código PIN de la smart-card.

Para eliminar esta posibilidad, el módulo LSM garantiza que tanto los binarios, como la consola de ejecución que interviene en el proceso de autenticación, no pueden ser sobreescritos por ningún proceso o usuario (incluso con privilegios de administración), permaneciendo en todo momento los permisos de sólo lectura (*read-only*).

En segundo lugar, el binario de la consola de administración y el resto de aplicaciones asociadas a nuestro esquema de autenticación y protección deben ser compilados de forma estática. Esto nos permite, en primer lugar, reducir la complejidad del proceso de protección, ya que nos permite descartar la posibilidad de inyección de código maligno en la carga de librerías compartidas, y demás recursos o ficheros asociados. Al mismo tiempo, nos permite centralizar y reducir errores de programación que podrían ser utilizados por el atacante para manipular el proceso de autenticación y protección. De este modo, el módulo LSM podrá controlar que ninguna llamada al sistema lanzada por procesos del sistema pone en peligro el esquema de autenticación correcto, tales como llamadas al sistema para capturar o manipular información del teclado, llamadas al sistema para cancelación de procesos, o llamadas al sistema para tareas de depuración.

Para evitar una posible manipulación del canal de comunicación entre las distintas entidades del proceso de autenticación, el módulo LSM se encarga de nuevo de proteger las llamadas necesarias para el proceso (es decir, las llamadas entre el proceso asociado a la consola de administración y la tarjeta smart-card). Por ello, podemos descartar tal manipulación por parte de un atacante. Por último, y tal y como se indica en [2], el módulo LSM no necesita ser protegido, ya que debemos asumir el entorno de ejecución intra-kernel como un área de ejecución de confianza – de hecho, es un requisito imprescindible en el modelo de seguridad de cualquier sistema operativo actual.

6. Evaluación de rendimiento

En esta sección se presenta una evaluación del rendimiento que ofrece nuestra implementación de SMARTCOP para sistemas GNU/Linux, desarrollada mediante el marco de trabajo de módulos seguros de Linux (*Linux Security Modules*, LSM).

Para ello, se muestran los resultados de una serie de tests realizados para medir la penalización que supone la instalación de SMARTCOP, en forma de módulo LSM, sobre el funcionamiento normal del sistema. No tendremos en cuenta en esta evaluación, por lo tanto, la penalización en el rendimiento durante las tareas de administración, es decir, en aquellas operaciones realizadas por el administrador del sistema, haciendo uso del esquema de autenticación presentado en la sección anterior.

El conjunto de tests realizados se ha basado en la utilización de la herramienta Strace [1] y del paquete de herramientas LMbench [7]. Strace es una herramienta de depuración, que nos permite rastrear las llamadas al sistema realizadas tras la ejecución de un proceso para, por ejemplo, analizar y evaluar el tiempo de duración de estas llamadas. Por otro lado, LMbench es un paquete de herramientas de análisis para la realización de *microbenchmarks*, a través de los cuales se podrán tomar medidas más completas como tiempos de acceso a ficheros, accesos a memoria, etcétera.

Test Type	2.6.15	2.6.15 + smartcop	% Overhead with smartcop
null call	0.255	0.255	0%
kill	231.10	241.65	4.6%
stat	1.99	2.03	2%
open/close	2.96	3.02	1.9%
select TCP	18.63	18.86	1.2%
sig inst	0.9	0.9	0%
sig handl	1.85	1.88	0.1%
fork proc	95.61	96.52	0.9%
exec proc	100.50	103.86	3.3%
sh proc	2227	2302	3.3%

Process tests, time in μseconds

Test Type	2.6.15	2.6.15 + smartcop	% Overhead with smartcop
pipe	1342	1338	0.2%
AF Unix	1334	1320	1%
TCP	1088	1078	0.9%
file read	1330	1308	1.6%
mmap read	1480	1425	3.8%
mem bcopy	5278	5277	0.01%
mem bzero	4548	4548	0%
mem read	25600	25590	0.03%
mem write	24888	24869	0.07%

Local communication bandwidth in MB/s

Test Type	2.6.15	2.6.15 + smartcop	% Overhead with smartcop
0K file create	193	193	0%
0K file delete	489	489	0%
10K file create	175	176	0.5%
10K file delete	658	668	1.5%
mmap latency	2348	2348	0%
par mem	1.26	1.26	0%
page fault	0.974	0.981	0.8%

File and VM sytem latencies, time in μseconds

Figura 2. Resultados de la evaluación de rendimiento de SMARTCOP.

La evalución fue llevada a cabo en un único equipo con procesador Intel-Pentium M 1.4 GHz, con 512 MB de memoria RAM, disco duro IDE de 5400 rpm, funcionando sobre un sistema operativo Debian GNU/Linux. El objetivo de estos tests es comparar el rendimiento del equipo cuando es utilizado con un kernel Linux 2.6.15 sin soporte para LSM, contra el rendimiento del mismo equipo cuando es utilizado con un kernel Linux 2.6.15 con soporte para LSM y el módulo SMARTCOP cargado sobre él.

Los resultados de estos tests son mostrados en la figura 2, organizados en tres tablas, según los tres niveles de protección que enunciábamos en la sección 3. Como vemos en estos resultados, la penalización introducida por SMARTCOP en el rendimiento de un sistema GNU/Linux 2.6.15 estándar revela que el impacto es mínimo.

Los tests de la primera tabla (*Process tests*) ofrecen la latencia de respuesta en microsegundos de una serie de operaciones relacionadas con aspectos de ejecución de procesos y llamadas al sistema como, por ejemplo, creación de procesos a través de *fork()*, *fork()+exec()* y *sh()*, cancelación de procesos a través de *kill()*, espera a descriptores a través de *select()*, apertura y cierre de ficheros a través de *open()*/*close()*, instalación y manejo de señales, etcétera.

Los resultados de esta primera categoría de tests muestra que más del cincuenta por ciento de las pruebas apuntan a una penalización en el rendimiento inferior al 2 %. La creación de procesos a través de *fork()*, por ejemplo, se ve mínimamente penalizada en un 0.9 %, al igual que la creación de procesos a través de *fork()+exec()* y *sh()* que se ven penalizadas en un 3.3 % (aproximadamente). La cancelación de procesos mediante la llamada al sistema *kill()*, en cambio, es la que resulta más penalizada, con un 4.6 %. Esta penalización es introducida por las validaciones realizadas desde el control de acceso de SMARTCOP, a nivel de kernel, durante las comprobaciones de identificación del proceso, de los parámetros de la llamada al sistema, etcétera.

El segundo conjunto de tests mostrados en la segunda tabla de la figura 2, ofrece como resultado el ancho de banda de una serie de operaciones relacionadas con aspectos de comunicación como, por ejemplo, lecturas, escrituras y copia a zonas de memoria mediante *read()* y *mmap()*, comunicaciones intra-procesos (*InterProcess Communications*, *IPC*) utilizando TCP, tuberías (*pipes*) y sockets de la familia de direcciones Unix (*AF Unix sockets*), etcétera. De nuevo, los resultados de esta segunda categoría muestran que la penalización en el rendimiento es mínima, siendo el proceso de lectura y resumen de ficheros mediante el interfaz de acceso a memoria *mmap()* la operación que presenta una penalización mayor en el rendimiento (un 3.8 %).

Por último, el tercer conjunto de tests mostrados en la tercera tabla de la figura 2 ofrece como resultado la latencia en operaciones relacionadas con la manipulación de ficheros y de memoria. Una vez más, la penalización en el rendimiento del sistema es mínima, siendo la eliminación de ficheros la que mayor penalización presenta (debida a las comprobaciones realizadas por SMARTCOP durante las llamadas asociadas a la eliminación de ficheros).

7. Conclusiones

En este artículo hemos presentado un mecanismo de control de acceso especialmente adecuado para la protección de componentes y dispositivos de seguridad de red como, por ejemplo, sistemas cortafuego, sistemas de detección de intrusos (*Intrusion Detection Systems, IDS*), o sistemas de prevención de intrusos (*Intrusion Detection Systems, IPS*). Si uno de estos componentes, o alguno de sus elementos, es comprometido por un atacante, es muy probable que dicho atacante pueda hacerse con el control absoluto del sistema o de la propia red.

La protección de estos componentes no es una tarea sencilla, especialmente cuando se trata de entornos distribuidos, formados por distintos componentes y/o elementos distribuidos sobre una red compleja – como, por ejemplo, la plataforma de gestión de ataques presentada en [5].

El mecanismo de protección propuesto en este artículo ha sido de desarrollado en un primer prototipo software llamado (*Smart Card Enhanced Linux Security Module for Component Protection*). SMARTCOP ha sido desarrollado para sistemas GNU/Linux a través del marco de trabajo *Linux Security Modules* (*LSM*), e implementa un control de acceso integrado en el kernel Linux para comprobar y cancelar llamadas al sistema que comprometen la seguridad del componente.

La utilización de LSM garantiza que nuestro mecanismo de protección pueda ser proyectado sobre nuevos componentes y elementos, simplemente considerando su propio entorno, y las interacciones que hay en él (en términos de control de acceso), reforzando la modularidad del sistema y proporcionando una forma fácil y genérica de incorporar nuevos elementos sin necesidad de realizar consideraciones particulares sobre como proteger cada uno de los nuevos elementos. Por lo tanto, consideramos que nuestra propuesta ofrece un alto grado de escabilidad.

En esa misma línea, la inclusión de nuevos componentes garantiza una mínima degradación del rendimiento, ya que el marco de trabajo LSM y el esquema de control de acceso implementado no incorpora una complejidad excesiva en términos de computación. Para ello, se ha medido la penalización que supone la instalación de SMARTCOP sobre el rendimiento normal del sistema. El resultado de esta primera evaluación indica que el impacto de SMARTCOP es mínimo.

Además, con el objetivo de reforzar la seguridad del propio mecanismo de protección, SMARTCOP implementa un mecanismo de autenticación basado en tarjetas smart-card que actúa como complemento al control de acceso basado en kernel. Gracias a este mecanismo adicional, será necesario conocer un secreto y disponer de un dispositivo físico (la tarjeta smart-card) para realizar aquellas acciones controladas y protegidas por SMARTCOP. Esta idea nos permite resolver ataques lógicos contra el mecanismo de protección, tales como el robo de contraseñas o la explotación de desbordamientos de buffer.

Por todos estos motivos, podemos concluir que el mecanismo de control de acceso propuesto en este artículo, e implementado en un primer prototipo como módulo LSM para sistemas GNU/Linux, brinda una buena solución en la difícil tarea de proteger componentes y dispositivos de seguridad en red, ofreciendo un alto grado de transparencia respecto al funcionamiento tradicional de tales componentes, y sin necesidad de interferir directamente con procesos del espacio de usuario.

Agradecimientos

J. Garcia-Alfaro, S. Castillo y G. Navarro han sido parcialmente financiados por el Ministerio de Ciencia y Tecnología, a través del proyecto TIC2003-02041, y el departamento DURSI de la Generalitat de Catalunya, a través de las becas 2003FI126 y 2005BE77.

J. Castellà-Roca ha sido parcialmente subvencionado por el departamento DURSI de la Generalitat de Catalunya a través del proyecto 2005-SGR-00446, y del Ministerio de Ciencia y Tecnología, a través del proyecto SEG2004-04352-C04-01 "PROPRIETAS".

Referencias

1. W. Akkerman. Strace, `http://liacs.nl/~wichert/strace/`, 2003.
2. P. Biondi Kernel Level Security Brussels, FOSDEM 2003, September 2003.
3. A. Herzog and N. Shahmehri. Using the Java Sandbox for Resource Control. In *7th Nordic Workshop on Secure IT Systems (NORDSEC 2002)*, Sweden, 2002.
4. P. Hope. Using Jails in FreeBSD for Fun and Profit. *Login;*, 27(3):48–55, June 2002.
5. J. Garcia-Alfaro, F. Autrel, J. Borrell, S. Castillo, F. Cuppens, and G. Navarro. Decentralized publish/subscribe system to prevent coordinated attacks via alert correlation. In *6th Int. Conf. on Information and Communications Security*, 223–235, Spain, 2004. Springer-Verlag.
6. G. McGraw and E. W. Felten. *Java Security: Hostile applets, Holes, and antidotes.* Wiley Computer Publishing, 1997
7. L. McVoy. LMbench, Portable Tools for Performance Analysis. USENIX, 1996.
8. T. Onabuta, T. Inoue, and M. Asaka. A Protection Mechanism for an Intrusion Detection System Based on Mandatory Access Control. In *13th Annual Computer Security Incident Handling Conference*), Toulouse, France, June 2001.
9. A. Ott. The Role Compatibility Security Model. In *7th Nordic Workshop on Secure IT Systems*, Karlstad, Sweden, November 2002.
10. J. Viega, and G. McGraw. *Building Secure Software - How to Avoid Security Problems the Right Way*. Addison-Wesley, September 2002.
11. C. Wright, C. Cowan, S. Smalley, J. Morris, and G. Kroah-Hartman. Linux Security Modules: General Security Support for the Linux Kernel. In *11th USENIX Security Symposium*, San Francisco, California, August 2002.
12. Q. Zhong and N. Edwards. Security in the Large: Is Java's Sandbox Scalable? Technical report, Extended Enterprise Lab, Hewlett Packard Laboratories, Bristol Filton Road,Stoke Gifford Bristol BS34 8QZ, 1998.
13. ITU-T. The Directory: Public-key and attribute certificate frameworks. ITU-T Recommendation X.509, 2000.