APUNTES DE

Introducción a los Sistemas Operativos

2º DE GRADO EN INGENIERÍA INFORMÁTICA

TEMA 3. SEGURIDAD Y PROTECCIÓN

CURSO 2021/2022

ÍNDICE GENERAL

CAPÍTULO 3

SEGURIDAD Y PROTECCIÓN

3.1 INTRODUCCIÓN

Aunque son términos que podemos confundir, la seguridad y la protección son dos cosas distintas en el ámbito de los sistemas operativos. La *protección* tiene carácter interno, ya que es una tarea encargada al sistema operativo. En esta tarea se incluye el impedir accesos no autorizados a los recursos, evitar pérdidas de datos, etc. La *seguridad*, en cambio, tiene un carácter más general en el que se incluyen, además de la protección, otros aspectos como la política de copias de seguridad, la localización física del sistema, las autorizaciones de acceso a la sala o salas donde se ubica el sistema, etc.

3.2 SEGURIDAD

En primer lugar, analizaremos de forma muy genérica en qué consiste la seguridad en un sistema de computación para, posteriormente, centrarnos en la protección.

3.2.1 Amenazas a la seguridad

Para entender los diferentes tipos de amenazas que pueden comprometer la seguridad de un sistema, hace falta disponer de una definición clara de los *requisitos de seguridad* que se deben cumplir en dicho sistema. En el caso de la seguridad de ordenadores y redes, se definen los siguientes requisitos:

- *Confidencialidad*: exige que la información de un sistema de computación pueda ser leída solo por usuarios autorizados. Este tipo de acceso incluye la impresión, la visualización y otras formas de obtención de información, incluyendo el simple conocimiento de la existencia de un objeto.
- *Integridad*: exige que los elementos de un sistema de computación puedan ser modificados solo por usuarios autorizados. La modificación incluye tanto el eliminar datos como el añadir información falsa.
- *Disponibilidad*: exige que los elementos de un sistema de computación estén disponibles solamente para usuarios autorizados.

Nótese que la disponibilidad es diferente de la confidencialidad y la integridad. Así, si un usuario no autorizado consigue tener a su disposición un objeto que, sin embargo, no puede leer ni modificar, podría destruirlo o hacer que otros usuarios no pudieran acceder a él, realizando así un ataque a la disponibilidad del mismo.

3.2.2 Tipos de amenazas

Una vez definidos los requisitos de seguridad, podemos estudiar los tipos de amenazas que pueden poner en peligro la seguridad de un sistema o de una red. Estas amenazas se caracterizan mejor contemplando la función que del sistema como suministrador de información. En general, se produce un flujo de información desde un origen, como un fichero o una región de memoria principal, hacia un destino, como otro fichero o un usuario. El origen y el destino pueden estar en un mismo computador o en computadores distintos. Este flujo normal está representado en la figura 3.1(a). El resto de la figura muestra cuatro categorías generales de amenazas:

- *Interrupción*: se destruye un elemento del sistema, o se hace inaccesible o inútil. Esta es una amenaza a la disponibilidad. Ejemplos: destrucción de un disco duro, corte de una línea de comunicaciones o inutilización del sistema de ficheros.

- *Intercepción*: una parte no autorizada (persona, programa o computador) consigue leer un elemento. Esta es una amenaza a la confidencialidad. Ejemplo: copia ilícita de programas (piratería) o ficheros.

- *Modificación*: una parte no autorizada no solo consigue acceder, sino que también es capaz de falsificar un elemento; constituye un tipo de amenaza a la integridad. Ejemplos: cambio de valores en un fichero de datos, alteración de un programa para que se comporte de manera diferente (virus) o modificación de mensajes transmitidos en una red.

- *Invención*: una parte no autorizada inserta objetos falsos en el sistema. Esta es también una amenaza a la integridad. Ejemplos: inserción de mensajes falsos en una red o la adición de registros a un fichero.

3.2.3 Amenazas según los elementos de un sistema de computación

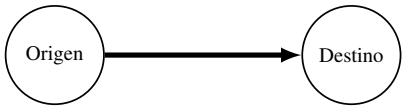
Los elementos de un sistema de computación pueden clasificarse en cuatro categorías: hardware, software, datos y líneas de comunicaciones (redes). La tabla 3.1 indica la naturaleza de las amenazas a las que se enfrenta cada tipo de elemento.

3.2.4 Ataques genéricos a la seguridad

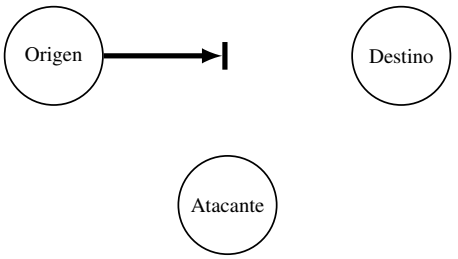
La forma más habitual de probar la seguridad de un sistema es contratar a un grupo de expertos para ver si son capaces de penetrar en él de alguna forma. A este grupo se le conoce como *equipo tigre* o *equipo de penetración*. A continuación, mostramos una lista de algunos de los ataques más comunes que puede intentar el equipo de penetración por tener éxito frecuentemente. Al diseñar un sistema, conviene asegurarse de que puede resistir los siguientes ataques:

1. Reserve espacio de memoria, disco o cinta, y solo léalo. Muchos sistemas no borran el espacio antes de asignarlo y podría contener información interesante escrita por el usuario anterior.

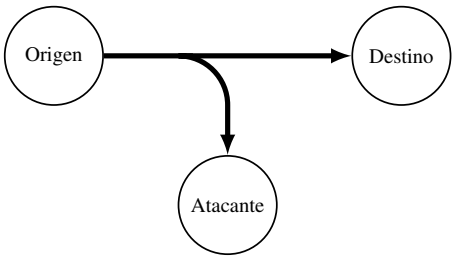
2. Intente llamadas al sistema inválidas, o bien llamadas válidas con parámetros inválidos, o incluso llamadas válidas con parámetros válidos, pero no razonables.



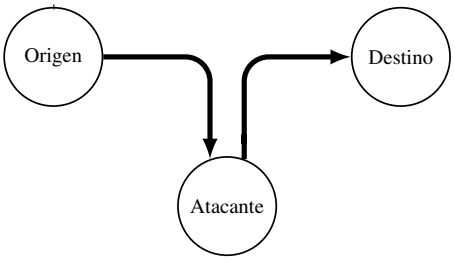
(a) Flujo normal



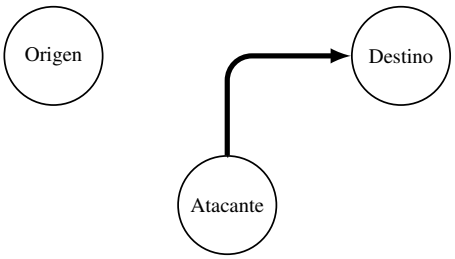
(b) Interrupción (amenaza a la disponibilidad).



(c) Intercepción (amenaza a la confidencialidad).



(d) Modificación (amenaza a la integridad).



(e) Invención (amenaza a la integridad).

Figura 3.1. Tipos de amenazas a la seguridad. La figura (a) muestra un flujo normal de información entre un origen y un destino legítimos. El resto de casos muestran los distintos tipos de amenazas que se pueden dar y cómo afectarían al flujo normal de información.

3. Conéctese a un sistema y oprima DEL, CTRL+C o CTRL+PAUSA a la mitad de la secuencia de acceso. En determinados sistemas, el programa de verificación de contraseñas quedará invalidado y el acceso se considerará exitoso.

4. Engañe al usuario escribiendo un programa que muestre el mensaje `login:` en la pantalla y que después desaparezca. Muchos usuarios irán a la terminal y le indicarán su nombre y contraseña de acceso, los cuales registrará el programa.

5. Busque manuales que digan «no lleve a cabo X» e intente tantas variaciones de X como sea posible.

6. Convenza al administrador del sistema para que modifique el sistema con el fin de que evite ciertas verificaciones vitales de seguridad para algunos usuarios concretos.

7. Si todo lo anterior falla, el atacante debe encontrar al personal de administración del centro de cálculo y engañarlo o sobornarlo. No hay que subestimar los problemas que puedan causar los propios trabajadores.

Elemento	Disponibilidad	Confidencialidad	Integridad
Hardware	Robo o inutilización de equipos, eliminando el servicio. Consumo excesivo de recursos (CPU, disco, etc.).		
Software	Eliminación de programas, denegando el acceso a los usuarios.	Realización de copias no autorizadas del software.	Alteración de un programa en funcionamiento haciéndolo fallar durante su ejecución o haciendo que realice alguna tarea no pretendida.
Datos	Eliminación de ficheros, denegando el acceso a los usuarios.	Lecturas de datos no autorizadas. Un análisis de datos estadísticos revela datos ocultos.	Modificación de ficheros existentes o invención de nuevos ficheros.
Líneas de comunicaciones	Destrucción o eliminación de mensajes. Corte de las líneas de comunicaciones o redes. Consumo excesivo de ancho de banda.	Lectura de mensajes. Observación del tráfico de mensajes.	Mensajes modificados, retardados, reordenados o duplicados. Invención de mensajes falsos.

Tabla 3.1. Elementos de un sistema de computación y posibles amenazas a las que se enfrentan.

3.2.5 Ataques específicos a la seguridad

Además de los ataques genéricos que acabamos de describir, a lo largo de los años han aparecido también diversos ataques específicos que han amenazado la seguridad de los sistemas de computación. Algunos de ellos son los siguientes:

- *Bombas lógicas*: «estallan» en determinados instantes de tiempo (por ejemplo, un viernes 13). En muchas ocasiones, son creadas por el propio desarrollador del programa que las incluye. Trata así de «protegerse» ante determinadas situaciones, como un despido. De este modo, mientras el desarrollador mantenga su puesto de trabajo, tomará las acciones necesarias para evitar que la bomba estalle. Solo si es despedido, la bomba estallará pasado un tiempo.

- *Puertas traseras* (*backdoors*): son programas que muchas veces realizan una tarea correcta, pero a través de los cuales es posible el acceso al sistema desde el exterior o como administrador. En aquellos programas que requieren autenticación, una puerta trasera se puede implementar como una cierta combinación usuario+contraseña que, al ser introducida, proporciona un acceso completo al programa y, a través de él, al sistema.

- *Desbordamiento de buffer*: consiste en aprovechar los fallos de programación de un programa para darle más información de la que es capaz de albergar en el buffer que ha reservado para ella, desbordando así el buffer y llegando a ejecutar código malicioso. En general, un desbordamiento de buffer sobrescribe la pila de una función para que, cuando la función termine, no regrese al punto desde el que se la invocó, sino a un código cuidadosamente preparado. Este código suele cambiar el comportamiento del programa y generalmente se utiliza para convertirse en administrador del sistema y así poder hacer cualquier cambio en el mismo.

- *Caballos de Troya*: sustituyen una orden interna por otra con el mismo nombre, pero que realiza labores ilegales (por ejemplo, enviar información del sistema a computadores externos).

- *Virus* y *gusanos*: programas cuya principal característica es su habilidad para extenderse dentro de un mismo ordenador o entre diferentes ordenadores por medio de sistemas de almacenamiento (discos, memorias USB, etc.) y redes. La diferencia entre un virus y un gusano es que el primero se encuentra dentro de un programa y se copia de ahí a otros programas, mientras que el segundo es un programa en sí que, a través de la red, se va copiando a sí mismo de un ordenador a otro.

- *Spyware*: programas que se instalan sin que el usuario sea consciente de ello ni lo autorice; se ejecutan en segundo plano para recopilar información del sistema y de la actividad del usuario. La información recopilada puede ser de lo más variada, incluyendo información bancaria y de contraseñas. Esta información es enviada posteriormente por estos mismos programas a computadores externos.

- *Rootkits*: programas que permiten un acceso privilegiado continuo a un computador a la vez que mantienen oculta de forma activa su presencia. En otras palabras, son programas que han corrompido el sistema de tal forma que no pueden ser fácilmente detectados y permiten a un atacante externo acceder al sistema con privilegios de administrador, pudiendo realizar sobre el sistema cualquier tipo de operación. Lo habitual es que el acceso se proporcione a través de una puerta trasera (*backdoor*) instalada por el propio *rootkit*.

  Generalmente, un *rootkit* es instalado por un atacante que ha conseguido previamente acceder como administrador al sistema. En algunos casos, llega a modificar incluso el propio núcleo del sistema operativo con nuevos manejadores (*drivers*) para ser indetectable. En estos casos, la eliminación del *rootkit* requiere reinstalar por completo el sistema operativo.

3.2.6 Principios de diseño para la seguridad

Saltzer y Schroeder (1975) identificaron varios principios generales que se pueden utilizar como guía para el diseño de sistemas seguros. Estos principios se pueden resumir en:

1. El diseño de un sistema debe ser público. Los diseñadores se engañarán a sí mismos si piensan que los intrusos nunca llegarán a conocer el funcionamiento interno del sistema.

2. El estado predefinido debe ser el de «no acceso».

3. Se debe verificar la autorización actual. El sistema no debe verificar el permiso, determinar que el acceso está permitido y después no volver a hacer ningún tipo de comprobación. Muchos sistemas verifican el permiso al abrir un fichero y no después de abrirlo. Esto significa que un usuario que abra un fichero y lo tenga abierto durante varias semanas seguirá teniendo acceso a él, incluso en el caso de que el propietario haya cambiado la protección del fichero.

4. Los procesos deben tener el mínimo privilegio posible que les permita seguir realizando su trabajo.

5. El mecanismo de protección debe ser simple, uniforme e integrado hasta las capas más bajas del sistema. Intentar dotar de seguridad a un sistema inseguro ya existente es casi imposible. La seguridad, al igual que lo correcto de un sistema, no

es una característica que se pueda añadir, sino que debe formar parte del diseño desde el primer momento.

6. El esquema elegido debe ser psicológicamente aceptable. Si los usuarios sienten que la protección de sus ficheros implica demasiado trabajo o es molesta (por ejemplo, un pinchazo para analizar una gota de sangre y así saber si eres tú), simplemente no los protegerán.

3.3 PROTECCIÓN

Tal y como vimos en el primer tema, el hardware proporciona, al menos, tres mecanismos que ayudan al sistema operativo a proteger el sistema de los procesos que se ejecutan en él:

- Hardware de direccionamiento de memoria: asegura que un proceso únicamente se puede ejecutar dentro de su espacio de direcciones.
- Cronómetro: garantiza que ningún proceso podrá obtener el control de la CPU sin renunciar al mismo en algún momento.
- Modo dual: asegura que solo el sistema operativo puede realizar las operaciones privilegiadas, como las de E/S.

Pero, además de la protección entre procesos y del propio sistema operativo, existen otros muchos elementos que proteger en un sistema, tanto físicos (discos, impresoras, ...), como lógicos (ficheros, procesos, ...). Por ello, será necesario que el sistema operativo disponga de un mecanismo que le permita controlar el acceso de los procesos a los recursos físicos y lógicos definidos en el sistema de computación. Este mecanismo deberá ofrecer un medio para especificar los controles que se impondrán, así como la manera de aplicarlos. Denominaremos *protección* a este mecanismo. Recordemos que la protección es la parte de la seguridad que es responsabilidad del propio sistema operativo.

3.3.1 Política y mecanismo

En los siguientes apartados veremos algunas técnicas utilizadas por los sistemas operativos para la protección de ficheros y otros elementos del sistema. Todas estas técnicas distinguen claramente entre *política* (que indica qué operaciones será posible realizar sobre qué elementos) y *mecanismo* (que especifica cómo el sistema hará cumplir la política). En otras palabras, el mecanismo es el encargado de llevar a cabo la política.

Las políticas determinan qué se va a proteger, qué procesos (y, a través de ellos, qué usuarios) van a poder acceder a qué recursos, etc. Unas políticas vendrán determinadas por el propio diseño del sistema, otras serán especificadas por los administradores del sistema, mientras que algunas serán definidas por los usuarios para proteger sus propios ficheros y programas.

La separación entre política y mecanismo es importante, ya que las políticas pueden cambiar de un sistema a otro o con el transcurso del tiempo. Por tanto, es deseable contar con mecanismos generales capaces de llevar a cabo distintas políticas (por ejemplo, cambiando la configuración del sistema).

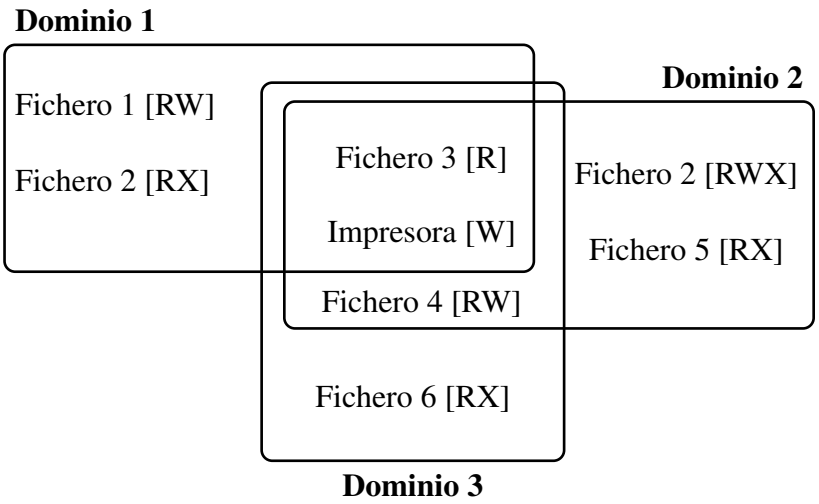


Figura 3.2. Tres dominios de protección.

3.3.2 Dominios de protección

Como hemos indicado, un sistema de cómputo contiene muchos objetos que necesitan protección. Estos objetos pueden pertenecer al hardware (CPU, segmentos de memoria, terminales, unidades de disco, impresoras, ...) o al software (procesos, ficheros, bases de datos, semáforos, ...).

Cada objeto tiene un nombre único mediante el que se referencia, así como un conjunto de operaciones que se pueden realizar sobre él. Por ejemplo, `read` y `write` son operaciones adecuadas para un fichero.

Teniendo en cuenta los objetos existentes en un sistema y los procesos que pueden acceder a ellos, es necesario disponer de un mecanismo que impida que los procesos accedan a aquellos objetos sobre los que no tienen ningún permiso. Este mecanismo también debe posibilitar que los procesos puedan acceder a los objetos sobre los que tienen permitido el acceso, pero solo para realizar aquel subconjunto de operaciones que les hayan sido autorizadas (por ejemplo, se puede permitir que un proceso lea un fichero, pero no escribir en él, o que pueda leerlo y escribirlo, pero no ejecutarlo). En nuestro caso, este mecanismo se llama *dominio de protección* o, simplemente, *dominio*. Un dominio es un conjunto de parejas (objeto, derechos). En este contexto, un derecho representa el permiso para realizar alguna de las operaciones sobre un objeto concreto. En la figura 3.2, donde se muestran tres dominios de protección, los derechos son lectura (R), escritura (W) y ejecución (X). Como se puede apreciar, un mismo objeto puede encontrarse en varios dominios con los mismos derechos (como es el caso de Fichero 3, Fichero 4 e Impresora) o con derechos distintos en cada dominio (como sucede con Fichero 2).

Cada proceso existente en el sistema se tiene que ejecutar en, al menos, uno de los dominios de protección que haya definidos, lo que hará que exista un conjunto de objetos a los que pueda acceder y sobre los que podrá realizar ciertas operaciones. Durante su ejecución los procesos también pueden cambiar de dominio, accediendo así a nuevos objetos y/o a un conjunto diferente de operaciones sobre los objetos para los que ya tienen acceso, aunque esta posibilidad dependerá del sistema.

Objetos

Dominios	Fichero 1	Fichero 2	Fichero 3	Fichero 4	Fichero 5	Fichero 6	Impresora
1	Leer Escribir	Leer Ejecutar	Leer				Escribir
2		Leer Escribir Ejecutar	Leer	Leer Escribir	Leer Ejecutar		Escribir
3			Leer	Leer Escribir		Leer Ejecutar	Escribir

(a)

Objetos

Dominios	Fichero 1	Fichero 2	Fichero 3	Fichero 4	Fichero 5	Fichero 6	Impresora	Dominio 1	Dominio 2	Dominio 3
1	Leer Escribir	Leer Ejecutar	Leer				Escribir			Entrar
2		Leer Escribir Ejecutar	Leer	Leer Escribir	Leer Ejecutar		Escribir			
3			Leer	Leer Escribir		Leer Ejecutar	Escribir	Entrar		

(b)

Figura 3.3. (a) Matriz de protección correspondiente al ejemplo de la figura 3.2. (b) La misma matriz, pero donde los dominios son también objetos.

3.3.3 Matriz de acceso

Un aspecto importante a tener en cuenta es la forma en la que el sistema lleva un registro de las parejas (objeto, derechos) que pertenecen a un dominio dado. Teóricamente, uno puede imaginar una enorme matriz en la que las filas corresponden a los dominios y las columnas a los objetos. Esta matriz se conoce como *matriz de protección* o *matriz de acceso* (ver figura 3.3(a)). Cada celda contendrá los derechos correspondientes a un objeto en un dominio. Con esta matriz, y sabiendo el dominio al que pertenece un proceso, el sistema operativo será capaz de determinar si el proceso puede acceder de alguna forma a un objeto.

En el caso de que los procesos puedan cambiar de un dominio a otro, el propio cambio de dominio se puede incluir con facilidad en el modelo matricial si se observa que un dominio también es un objeto con una operación de tipo «Entrar», que indica si se puede entrar o no en el dominio correspondiente (ver figura 3.3(b)). Incluso la matriz de acceso se puede ver como un objeto que se puede modificar a través de una serie de operaciones. En función del dominio en el que nos encontremos, podremos aplicar unas operaciones u otras.

En la práctica, rara vez se almacena tal cual la matriz de protección que acabamos de describir; como la mayor parte de los dominios solo tiene acceso a un número reducido de objetos, la matriz estará casi vacía en relación con su tamaño. No obstante, existen dos métodos que guardan solo los elementos no vacíos de la matriz, lo que nos permite reducir considerablemente el tamaño de las estructuras de datos que deben almacenar la información de protección. Estos métodos son las listas de control de acceso y las listas de posibilidades. A continuación se explica en detalle en qué consiste cada uno de ellos.

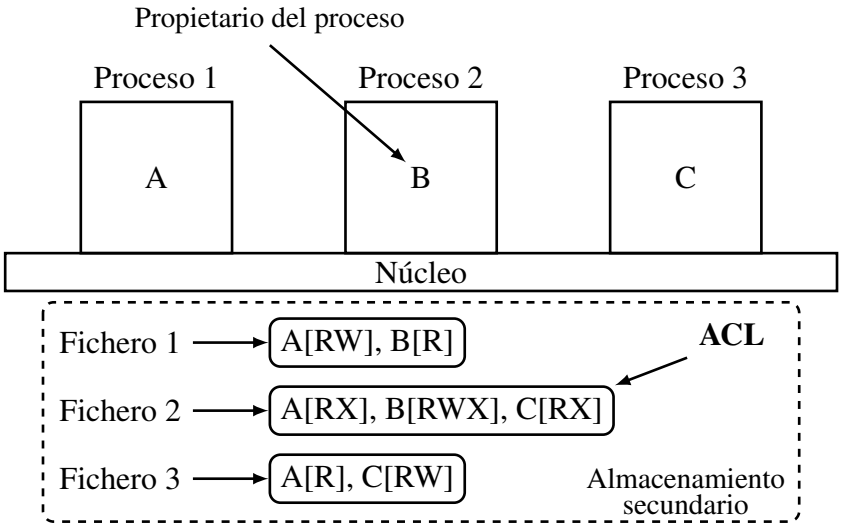


Figura 3.4. Listas de control de acceso para tres ficheros.

3.3.4 Listas de control de acceso

En las *listas de control de acceso* (o ACL, *Access Control List*), la matriz de protección se almacena por columnas, es decir, se asocia a cada objeto una lista con todos los dominios que pueden acceder al objeto y los derechos concretos de cada dominio para dicho acceso.

La figura 3.4 muestra las listas de control de acceso asociadas a tres ficheros, donde estamos suponiendo que los dominios quedan definidos por los usuarios, es decir, todos los procesos de un usuario pertenecen al mismo dominio[1]. De esta forma, por ejemplo, la ACL asociada al fichero F1 indica que los procesos pertenecientes al dominio del usuario «A» pueden leerlo y modificarlo, mientras que los procesos pertenecientes al dominio del usuario «B» solo pueden leerlo. Los procesos pertenecientes al resto de dominios o usuarios, en cambio, no pueden realizar ninguna operación sobre el fichero.

Si, como este ejemplo, las listas de control de acceso se asocian a ficheros, lo más razonable es que cada ACL se almacene en disco junto con su fichero. Sin embargo, como se muestra en la figura 3.4, las ACL están protegidas por el núcleo del sistema operativo, por lo que no pueden ser directamente manipuladas por los procesos de usuario. Dicho de otra manera, si un proceso quiere modificar una ACL, tendrá que enviar la solicitud al sistema operativo, que la validará antes de hacer la modificación correspondiente.

Además de los típicos permisos sobre ficheros (lectura, R; escritura, W; ejecución, X), un sistema puede definir permisos adicionales para realizar otro tipo de operaciones sobre los objetos, como copiar, borrar, ordenar, etc.

Si los usuarios se pueden agrupar, las listas de control de acceso también se pueden aplicar a grupos, por lo que cualquier proceso de un usuario perteneciente a un grupo concreto podrá realizar sobre un objeto lo que la ACL indique para ese grupo. En el caso de que una ACL contenga tanto una entrada para un usuario como entradas para algunos de los grupos a los que pertenece ese mismo usuario, habrá que establecer algún criterio que indique qué derechos prevalecen para los procesos de ese usuario sobre el objeto: los del usuario, los del grupo, los que establecen el acceso más restrictivo, los que permiten un acceso más amplio, los primeros que aparecen en la lista, etc.

Un aspecto a tener en cuenta es que si la ACL de un fichero solo se comprueba

[1] Cada sistema operativo debe definir lo que es o no un dominio. En una sección posterior de este tema veremos cómo define Unix los dominios.

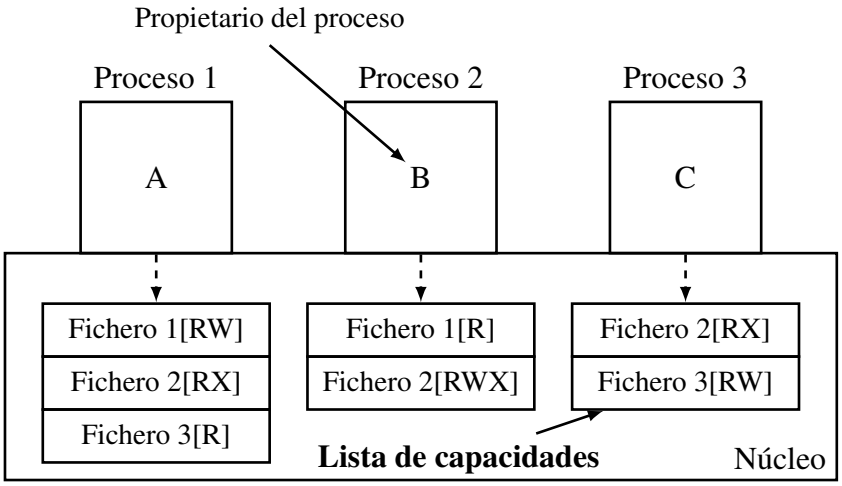


Figura 3.5. Listas de posibilidades asociadas a tres procesos.

cuando se abre este, entonces un proceso podrá seguir accediendo al fichero hasta que lo cierre, aunque se le haya revocado el permiso. Esto, por ejemplo, es lo que sucede en Unix.

3.3.5 Posibilidades o capacidades

Las *listas de posibilidades* o *listas de capacidades* almacenan una matriz de protección por filas, por lo que, en teoría, se trataría de asociar a cada dominio una lista de los objetos a los cuales puede acceder junto con una indicación de las operaciones permitidas sobre cada uno. Sin embargo, ya que son los procesos, repartidos entre los diferentes dominios, los que realmente acceden a los objetos, las listas de posibilidades se suelen asociar a procesos, es decir, cada proceso tendrá su propia lista, en la que cada elemento recibe el nombre de *capacidad* o *posibilidad*.

La figura 3.5 muestra las listas de posibilidades asociadas a tres procesos; es equivalente a la figura 3.4 que almacena la matriz de protección usando una ACL por objeto.

Generalmente, cada posibilidad (entrada de una lista) contiene varios campos que indican el tipo de objeto al que hace referencia la entrada, los derechos que el proceso tiene sobre el objeto y un apuntador al propio objeto (por ejemplo, el número de nodo-i[2] en el caso de un fichero en Unix). De esta forma, al realizar una operación sobre un objeto, el proceso hará referencia al objeto sobre el que quiere realizar la operación indicando la posición de la capacidad correspondiente en la lista de capacidades. Por ejemplo, un proceso podría realizar una operación diciendo lo siguiente: «Lee 1024 bytes del fichero al que apunta la posibilidad 2». De hecho, esto es lo que ocurre en Unix cuando se hace una operación sobre un fichero a través de su descriptor de fichero, pues este número indica la posición de la posibilidad correspondiente al fichero sobre el que se va a hacer la operación.

Aunque las listas de posibilidades se asocian a procesos, deben ser protegidas del manejo indebido por parte de los propios procesos, pues el sistema operativo confía en la información almacenada en ellas para determinar si el acceso a un objeto por parte de un proceso está o no permitido. Una forma de conseguir esto es manteniendo la lista de posibilidades en el núcleo del sistema operativo y hacer que los procesos de usuario hagan referencia a las posibilidades mediante su número, como acabamos de ver.

[2]Los nodos-i son una técnica de implementación de ficheros que estudiaremos en el siguiente tema.

Además de los derechos dependientes del objeto (como la lectura, la escritura y la ejecución), las posibilidades suelen tener también derechos genéricos que son aplicables a todas las posibilidades y, por tanto, a cualquier objeto referenciado por una posibilidad. Por ejemplo:

- Copiar posibilidad: crea una nueva posibilidad para el mismo objeto.
- Eliminar posibilidad: elimina un dato de la lista.
- Destruir objeto: elimina de forma permanente un objeto y una posibilidad.

Una característica interesante de las listas de posibilidades es que un proceso que posee una capacidad puede pasársela a otros procesos, generar una versión más restrictiva de la misma, etc.

Nos queda por describir cómo se construye una posibilidad para un objeto. Al igual que los procesos se crean y destruyen continuamente, con sus listas de posibilidades ocurre lo mismo. La solución de Unix, que comentaremos en detalle más adelante, consiste en combinar las listas de posibilidades con las listas de control de acceso. De esta forma, por ejemplo, la primera vez que un proceso accede a un fichero (con una operación `open()` o similar), el sistema operativo consulta la ACL asociada al fichero para saber si el proceso tiene permisos para realizar la operación que desea. Si no tiene permiso, se produce un error. En caso contrario, se crea una posibilidad para dicho fichero en la lista de posibilidades del proceso y se devuelve la posición ocupada por la posibilidad (esta posición es el descriptor de fichero). Los accesos posteriores al fichero utilizarán la posibilidad para demostrar rápidamente que está permitido el acceso. Cuando el proceso haya terminado de acceder al fichero, la posibilidad se destruirá (tal como hace, por ejemplo, la operación `close()`).

3.3.6 Comparación

Las listas de control de acceso se corresponden directamente con las necesidades de los usuarios del sistema: cuando un usuario crea un objeto, especifica los dominios que pueden acceder al objeto así como las operaciones permitidas. Sin embargo, como la información para un dominio no está localizada, es difícil determinar el conjunto de objetos y derechos a los que tiene acceso cada dominio. Además, hay que comprobar cada acceso al objeto, lo que requiere realizar una búsqueda en la ACL, la cual puede consumir mucho tiempo en un sistema de gran envergadura con largas listas de acceso.

Las listas de posibilidades, en cambio, no se corresponden directamente con las necesidades de los usuarios, lo que las hace algo más complejas. No obstante, son útiles para saber a qué objetos puede acceder un proceso y qué operaciones puede realizar sobre ellos. Por tanto, el proceso que intenta acceder a un objeto debe presentar una posibilidad para ese acceso. Después, el sistema de protección será el encargado de verificar que dicha posibilidad es válida.

3.3.7 Cancelación (o revocación)

En un sistema de protección dinámico es necesario a veces cancelar derechos de acceso a objetos que son compartidos por usuarios. Pueden surgir varias preguntas relacionadas con la cancelación:

- *Inmediata* o *demorada*. ¿La cancelación ocurre inmediatamente o se demora? Si se demora, es decir, si no tiene un efecto inmediato, ¿podemos averiguar cuándo será efectiva?

- *Selectiva* o *general*. Cuando se cancela un derecho de acceso a un objeto, ¿afecta a todos los usuarios con derecho de acceso a ese objeto, o podemos seleccionar un subconjunto de usuarios?

- *Parcial* o *total*. ¿Puede cancelarse un subconjunto de los derechos de un usuario sobre un objeto, o tenemos que cancelar todos los derechos?

- *Temporal* o *permanente*. ¿Puede cancelarse permanentemente un acceso (es decir, ese derecho de acceso nunca más estará disponible) o puede cancelarse y recuperarse posteriormente?

Con las listas de control de acceso, la cancelación es bastante sencilla. Dado un objeto, se busca en su ACL los derechos que serán cancelados y simplemente se eliminan. La cancelación es inmediata y puede ser general o selectiva, total o parcial, o permanente o temporal.

Las posibilidades, sin embargo, presentan un problema de cancelación más difícil. Dado que las posibilidades de un objeto están distribuidas por el sistema, es necesario localizarlas antes de poder cancelarlas. Es decir, al sistema le lleva tiempo determinar todas las posibilidades existentes para un cierto objeto y eliminarla, ya que pueden estar almacenadas en listas de posibilidades de muchos procesos. Evidentemente, para cada objeto se podría llevar un registro de todas las posibilidades asociadas y no tener que hacer una búsqueda por todo el sistema, pero este registro podría consumir una cantidad importante de memoria. Además, en algunos sistemas, los procesos pueden transferir posibilidades a otros procesos sin conocimiento del sistema operativo, por lo que el registro anterior no sería útil. No obstante, existen varias técnicas que permiten implementar la cancelación de posibilidades de una forma eficiente, aunque presentan algunos inconvenientes.

Una primera técnica consiste en hacer que cada posibilidad apunte hacia un objeto indirecto, en vez de apuntar al objeto en sí. Si el objeto indirecto apunta hacia el objeto real, el sistema siempre puede romper esta conexión, invalidando así todas las posibilidades.

Otra técnica es asignar a cada objeto un número aleatorio (que tendría que ser enorme para evitar posibles colisiones) y almacenarlo también en la posibilidad. De esta forma, cuando se intenta acceder a un objeto a través de su posibilidad, se comparan los dos números (el de la posibilidad y el del objeto) y se permite la operación si ambos números coinciden. El propietario de un objeto puede solicitar que se cambie el número aleatorio en dicho objeto, lo cual invalidaría también todas las posibilidades existentes.

Como vemos, ninguna de estas dos técnicas permite ni una revocación selectiva ni parcial, es decir, no es posible retirar el permiso a un usuario, pero sí mantener el permiso a los demás usuarios, o eliminar solo unos permisos manteniendo el resto.

3.4 AUTENTICACIÓN DE USUARIOS

La protección, tal como la hemos descrito, es estrictamente un problema interno del sistema operativo: ¿cómo proporcionamos acceso controlado a los programas y datos almacenados en un sistema de computación? La seguridad, por otra parte, no solo requiere un adecuado sistema de protección, sino también tener en cuenta el entorno externo donde opera el sistema. Por ello, la protección interna no es útil si, por ejemplo, los discos donde se almacenan los ficheros pueden extraerse del sistema de cómputo y transportarse a otro donde no haya protección. Estos problemas de seguridad son esencialmente problemas administrativos, no del sistema operativo.

El principal problema de seguridad para los sistemas operativos es el de la validación. El sistema de protección depende de la capacidad del sistema operativo para conocer a qué dominios pertenecen los procesos en ejecución. Si los dominios se definen (total o parcialmente) en función de los usuarios, esta habilidad depende en última instancia, a su vez, de nuestra capacidad para identificar a cada usuario del sistema. El problema de la identificación de los usuarios cuando se conectan al sistema se conoce como *autenticación de usuarios*.

Aunque existe una gran diversidad de métodos (tarjeta+PIN, huella dactilar, etc.), el método más general para realizar la autenticación de usuarios consiste en usar un identificador más una contraseña. Cuando un usuario quiere iniciar una sesión, no solo proporciona un nombre o identificador (ID), sino también una contraseña. La contraseña permite autenticar el ID del individuo que se conecta al sistema. Por su parte, el ID introduce seguridad en dos sentidos, ya que determina:

1. Si el usuario está autorizado para acceder al sistema. Lo habitual es que los sistemas solo permitan el acceso a aquellos usuarios que ya tienen un ID registrado.

2. Los privilegios acordados con el usuario, es decir, lo que el usuario puede y no puede hacer en el sistema. Habitualmente, un usuario (o un grupo reducido de usuarios) tendrá un estatus de supervisor, administrador o «superusuario» que le permitirá llevar a cabo funciones protegidas especialmente por el sistema operativo. Algunos sistemas también disponen de cuentas anónimas cuyos usuarios tienen privilegios más restringidos que los demás.

3.4.1 Contraseñas en Unix

Como ejemplo de autenticación de usuarios, vamos a analizar el esquema de contraseñas de Unix, donde las contraseñas nunca se almacenan en claro (es decir, sin cifrar). La figura 3.6 muestra tanto el procedimiento para almacenar una nueva contraseña como los pasos a seguir para verificar si una contraseña es correcta o no.

A la hora de almacenar una nueva contraseña para un usuario, se pasan como parámetros a una rutina de cifrado conocida como `crypt()` la propia la contraseña, un valor «base» (en inglés, *salt*) y el algoritmo de cifrado a utilizar. Esta rutina devuelve la contraseña cifrada, la cual se almacena en el fichero de contraseñas[3] en la línea asociada al ID del usuario (ver figura 3.6(a)). En concreto, el formato del campo que guarda la contraseña es: `$tipo$base$contraseña_cifrada`, donde el tipo es un número que indica el algoritmo empleado (1 para MD5, 5 para SHA256 y 6 para SHA512) y la base es la original sin cifrar. El tipo, la base y la contraseña cifrada se separan mediante el carácter «$».

El valor de la base normalmente está relacionado con el momento en que se asigna la contraseña a un usuario y sirve para impedir que contraseñas iguales se cifren de la misma manera. Es decir, si dos usuarios eligen la misma contraseña, esta será asignada en momentos diferentes a cada usuario, por lo que las bases usadas en cada caso serán distintas y los resultados del cifrado también. Si no se usara la base, dos contraseñas iguales producirían el mismo resultado de cifrado. Por tanto, conociendo la contraseña de un usuario y viendo que su contraseña cifrada es igual a la de otro usuario, se sabría inmediatamente la contraseña sin cifrar del otro.

[3] Originalmente, las contraseñas cifradas se guardaban en el fichero `/etc/passwd` en Unix, pero, como ese fichero necesita tener permisos de lectura para todos los usuarios, en la actualidad se almacenan en el fichero `/etc/shadow`, al que no pueden acceder los usuarios que no son administradores del sistema.

Cuando un usuario intenta conectarse a un sistema Unix, debe proporcionar un ID y una contraseña. El sistema operativo utiliza el ID como índice en el fichero de contraseñas para recuperar el tipo (que indica el algoritmo de cifrado), la base (sin cifrar) y la contraseña cifrada. El tipo y la base, junto con la contraseña introducida por el usuario, se pasan como parámetros a la función `crypt()`, cuyo resultado se compara con la contraseña cifrada almacenada en el fichero de contraseñas. Si el resultado obtenido es igual al valor almacenado, la contraseña es aceptada (ver figura 3.6(b)). En caso contrario, se produce un error de autenticación.

A continuación se muestran dos ejemplos donde se usa SHA512 como algoritmo de cifrado y donde se ve cómo una misma contraseña produce resultados distintos al cambiar la base:

- `crypt("hola","$6$u4C5efNyL$")` →
  `$6$u4C5efNyL$YGJJEAuTE91a.FVZqyOmiuddXyXp/.aRzxdmWp`
  `ERK04cI6Zp1gDy04Juq..N0e5BB2Bp53rqx4fA8dOq6c5aI0`

- `crypt("hola","$6$ZWPRzGQbgFDCp$")` →
  `$6$ZWPRzGQbgFDCp$oJY9bSmyrboMVZvV95k7Mf6ZfygTS4bM2V`
  `KHi7nBIgvuzjgQfPbyRW06w76JoG69/LKdAfXE4Qmw3vjic.yWN1`

Se puede ver que la base que se pasa a `crypt()` como segundo parámetro aparece tal cual al principio de la cadena resultante.

3.4.2 Estrategias de elección de contraseñas

Las funciones *hash* utilizadas para el cifrado de contraseñas están diseñadas de tal forma que no existe la correspondiente función inversa, es decir, no existe una función conocida que, dada una contraseña cifrada, nos la devuelva sin cifrar. Además, el diseño de estas funciones evita los ataques por adivinación (probar continuamente posibles contraseñas con la esperanza de que alguna coincida con la original), incluso usando supercomputadores que son capaces de probar millones de contraseñas por segundo.

Por desgracia, la naturaleza humana ofrece al atacante una alternativa práctica. Algunos usuarios, cuando pueden elegir su propia contraseña, eligen una fácilmente adivinable, como una contraseña de solo dos o tres caracteres. La solución a este problema es hacer que el sistema rechace cualquier contraseña que tenga menos de, por ejemplo, 6 caracteres.

La longitud de las contraseñas es solo una parte del problema. Aun obligando a los usuarios a elegir contraseñas con una longitud mínima, muchos escogerán contraseñas igualmente adivinables, como su propio nombre, el nombre de su calle, una palabra común del diccionario u otras similares. Esto, sin duda, facilita al atacante la tarea de adivinar contraseñas.

La lectura que se puede extraer de todo esto es que, si se permite que los usuarios puedan elegir cualquier contraseña, muchos elegirán una demasiado corta o demasiado fácil de adivinar. En el otro extremo, si se les asignan contraseñas a los usuarios que consten de, por ejemplo, 10 caracteres imprimibles seleccionados aleatoriamente, la averiguación de las contraseñas será prácticamente imposible, pero también será casi imposible que la mayoría de los usuarios recuerden su contraseña[4].

[4]Estas contraseñas aleatorias no son tan raras hoy en día. Muchos navegadores nos sugieren una contraseña formada por caracteres aleatorios cuando nos registramos en un sitio web. Para evitar recordar todas estas contraseñas, es posible utilizar una *cartera de contraseñas* en la que se almacenan las contraseñas y de donde se recuperan cuando es necesario. Esta cartera de contraseñas está, a su vez, cifrada y se desbloquea con una contraseña que, ahora sí, el usuario debe conocer.

Afortunadamente, aunque limitemos el universo de contraseñas a cadenas de caracteres que se puedan recordar fácilmente, el tamaño de dicho universo sigue siendo demasiado grande como para que la averiguación de contraseñas sea algo práctico. El objetivo es, entonces, eliminar las contraseñas adivinables y a la misma vez permitir que los usuarios elijan contraseñas que les resulten fáciles de recordar. Existen varias técnicas básicas, aunque aquí solo vamos a describir dos: la instrucción del usuario y la inspección proactiva de contraseñas.

En la *instrucción del usuario*, se explica a los usuarios la importancia de seleccionar contraseñas difíciles de adivinar y se les enseña cómo escoger contraseñas seguras. También se recomiendan algunos buenos hábitos para trabajar con contraseñas: cambiarlas con frecuencia, no almacenarlas/anotarlas en lugares visibles, no comunicárselas a nadie, etc. La figura 3.7 muestra varias técnicas efectivas para elegir contraseñas. En la actualidad, una buena contraseña está formada por letras mayúsculas y minúsculas, números y signos de puntuación (u otros caracteres no alfanuméricos). Muchos sistemas solo admiten contraseñas que cumplen estos criterios.

En la *inspección proactiva de contraseñas*, un usuario puede elegir su propia contraseña, pero, si en el momento de la selección el sistema estima (mediante diversos algoritmos y heurísticas) que la contraseña es fácilmente adivinable, la rechaza. Por lo tanto, con esta aproximación los usuarios pueden elegir contraseñas recordables de un espacio bastante grande de contraseñas que no es probable que sean adivinadas en un ataque. Incluso el administrador del sistema puede comprobar la robustez de las contraseñas de los usuarios ejecutando él mismo programas de averiguación de contraseñas (también llamados *cracks*).

Como se puede ver, las dos técnicas no son incompatibles, es más, se complementan perfectamente, ya que podemos enseñar a los usuarios a elegir contraseñas que no sean rechazadas por el sistema (a veces, es bastante frustrante ver cómo el sistema rechaza una y otra vez las contraseñas que le vamos introduciendo).

3.5 PROTECCIÓN EN UNIX

Esta sección describe cómo Unix pone en práctica muchos de los conceptos que hemos visto en este capítulo.

3.5.1 Dominios en Unix

En Unix, cada usuario se identifica mediante un número llamado *identificador de usuario* o UID (*User IDentifier*). Este número se asocia al *login* del usuario a través de la entrada correspondiente del fichero `/etc/passwd`. Además, existen *grupos de usuarios* en torno a los cuales se organizan los usuarios. Estos grupos también se identifican mediante un número llamado *identificador de grupo* o GID (*Group IDentifier*). El fichero `/etc/group` es el que asocia el nombre del grupo con su número; también almacena la lista de usuarios pertenecientes a cada grupo.

Un aspecto a tener en cuenta es que puede haber varios usuarios definidos con el mismo UID, o grupos con el mismo GID. Por tanto, para el sistema operativo se tratará siempre del mismo usuario o del mismo grupo.

Dentro del sistema, un usuario puede hacer lo que sus procesos tengan permitido, lo cual viene determinado, a su vez, por el UID y el GID asociados a cada proceso. Por lo tanto, cada par de valores (UID, GID) define un *dominio en Unix*, ya que, como veremos a continuación cuando describamos las ACL restringidas (apartado 3.5.2), ese par determina qué ficheros o dispositivos puede usar un proceso y con qué permisos.

En otras palabras, dos procesos con el mismo par (UID, GID) tendrán acceso al mismo conjunto de objetos por pertenecer al mismo dominio, mientras que procesos con diferentes valores (UID, GID) tendrán acceso a un conjunto distinto de objetos.

Entre todos los UID, existe uno especial que es el 0. Los procesos pertenecientes a este UID tienen todos los privilegios, es decir, puede hacer cualquier cosa en el sistema. Generalmente, este UID está asociado al usuario con nombre *root*, aunque también puede estar asociado a otros usuarios, pues, tal como hemos indicado anteriormente, Unix permite que varios usuarios compartan un mismo UID y/o GID.

Dado que el usuario *root* tiene todos los privilegios, será importante elegir una buena contraseña para esta cuenta y proteger dicha contraseña para impedir cualquier acceso no autorizado al sistema, especialmente con este usuario. Como medida de protección adicional, algunas distribuciones de Linux no permiten iniciar una sesión como *root*. Este es el caso de Ubuntu, donde cualquier tarea administrativa se hace a través de la orden `sudo`.

3.5.2 Listas de control de acceso restringidas

En Unix, cada fichero y cada dispositivo (siempre que este tenga una entrada de fichero especial asociada en `/dev`) pertenece a un usuario y a un grupo. Estos ficheros tienen asociadas listas de control de acceso que se califican como *restringidas* por estar limitadas a tres conjuntos de usuarios:

- Propietario: el usuario propietario del fichero.
- Grupo: grupo de usuarios al que pertenece el fichero.
- Otros: resto de usuarios.

Cada conjunto de usuarios tiene asociados tres posibles permisos: lectura («r»), escritura («w») y ejecución («x»). Estos permisos se suelen mostrar mediante una cadena de 9 letras, 3 para cada conjunto de usuarios. Un ejemplo de dicha cadena sería `rwxrwxr-x`, que indica que tanto el usuario propietario del fichero como el grupo al que pertenece tienen permisos de lectura, escritura y ejecución, y el resto de usuarios solo tiene permisos de lectura y ejecución.

Internamente, estos permisos se almacenan mediante 9 bits, uno por permiso. Así, la cadena de permisos anterior se almacenaría internamente como `111111101`, que también se puede escribir en octal como 775. La tabla 3.2 muestra algunos ejemplos de permisos (unos con más sentido que otros).

Hay sistemas Unix, como Linux, que además de estas ACL restringidas también implementan listas de control de acceso completas para aquellos sistemas de ficheros que las admiten. Gracias a estas listas, es posible especificar con gran detalle los usuarios y grupos concretos que podrán acceder a un fichero y con qué permisos. Aquí, sin embargo, nos centraremos en las ACL restringidas.

Todo lo que un proceso puede hacer sobre un determinado fichero (incluyendo directorios, que, como veremos en el siguiente tema, también son ficheros), depende del UID y GID a los que pertenece el proceso (es decir, de su usuario y grupo propietarios), del UID y GID a los que pertenece el fichero, y de la ACL restringida del fichero. A la hora de comprobar el acceso, se llevan a cabo los siguientes pasos:

1. Si el UID del proceso coincide con el UID del fichero, se consultan los permisos del fichero asociados al usuario propietario del fichero, ignorando el resto de permisos. El proceso podrá hacer entonces lo que indiquen dichos permisos.

Binario	Simbólico	Accesos permitidos
`111000000`	`rwx------`	El propietario puede leer, escribir y ejecutar
`111111000`	`rwxrwx---`	El propietario y el grupo pueden leer, escribir y ejecutar
`110100000`	`rw-r-----`	El propietario puede leer y escribir; el grupo puede leer
`110100100`	`rw-r--r--`	El propietario puede leer y escribir; todos los demás pueden leer
`111101101`	`rwxr-xr-x`	El propietario puede hacerlo todo, el resto pueden leer y ejecutar
`000000000`	`---------`	Nadie tiene acceso (salvo el root)
`000000111`	`------rwx`	Solo los demás tienen acceso (extraño, pero válido)

Tabla 3.2. Algunos ejemplos de ACL restringidas en Unix para ficheros.

2. Si lo anterior no se cumple, entonces se comprueba si el GID del proceso coincide con el GID del fichero. Si coinciden, entonces se consultan los permisos del fichero asociados al grupo propietario del fichero, ignorando el resto de permisos. De nuevo, el proceso podrá hacer lo que indiquen dichos permisos.

   Nótese que si el proceso es lanzado por un usuario que pertenece a varios grupos, entonces se comprobará si alguno de los GID de esos grupos coincide con el GID del fichero. En tal caso, se aplicará lo indicado en el párrafo anterior.

3. Si lo anterior tampoco se cumple, entonces el proceso podrá hacer lo que indiquen los permisos asociados al resto de usuarios.

En el caso de los directorios, los permisos «r», «w» y «x» cambian ligeramente de significado, el cual se entiende mejor si tenemos en cuenta que los directorios son ficheros que, básicamente, guardan los nombres de los ficheros (incluidos subdirectorios) que existen en el directorio. El permiso «r» en un directorio indica que se puede listar su contenido (por ejemplo, mediante la orden `ls`). El permiso «w» indica que se puede modificar su contenido, es decir, se pueden crear, borrar y renombrar (cambiar de nombre) ficheros y subdirectorios. Finalmente, el permiso «x» indica que podemos acceder a los nodos-i de los ficheros que hay en el directorio, lo que se traduce en que podemos consultar los atributos de los ficheros del directorio (por ejemplo, usando la orden `ls -l`), hacer que el directorio sea nuestro directorio actual o atravesar un directorio para acceder a sus subdirectorios (siempre que tengamos permiso para ello en los subdirectorios). Como la creación y eliminación de ficheros de cualquier tipo supone acceder a los nodos-i de los mismos, el permiso «w» en un directorio será efectivo solo si se acompaña del permiso «x». La tabla 3.3 muestra algunos ejemplos de permisos para directorios.

3.5.3 Dominio de un proceso y cambio de dominio

Por omisión, cada proceso se ejecuta siempre con el dominio (UID, GID) del usuario que lo crea. No obstante, de forma excepcional, los ejecutables pueden poseer en sus atributos los bits SETUID y SETGID, que pueden cambiar el dominio final al que pertenece un proceso.

En Unix, cuando se crea un proceso con `fork()`, el nuevo proceso hereda del padre su dominio. Sin embargo, si un proceso cualquiera realiza la llamada al sistema

Binario	Simbólico	Accesos permitidos
`111000000`	`rwx------`	Solo el propietario puede acceder al directorio y hacer cualquier operación en él
`111101101`	`rwxr-xr-x`	Cualquier usuario puede acceder al directorio, pero solo el propietario puede cambiar su contenido
`011011011`	`-wx-wx-wx`	Cualquiera puede copiar ficheros en el directorio y también borrar (siempre que conozca el nombre de aquello que quiere borrar)
`000000000`	`---------`	Nadie tiene acceso (salvo el root)
`001001001`	`--x--x--x`	Cualquier usuario puede acceder a los subdirectorios, pero ninguno (salvo el root) puede acceder al directorio en sí

Tabla 3.3. Algunos ejemplos de ACL restringidas en Unix para directorios.

`execve()` y el ejecutable que se pasa como parámetro tiene activo el bit SETUID, entonces el proceso no solo cambiará de programa, sino que también cambiará su UID por el del usuario propietario del fichero ejecutable, cambiando así de dominio de forma efectiva (observa que un usuario puede ejecutar los programas pertenecientes a otros usuarios siempre que tenga permiso para ello). Si dicho bit no está activo, el proceso conservará su UID (posiblemente heredado del padre) y, por tanto, su dominio.

Algo similar ocurre con el bit SETGID. Si este bit está activo, en una llamada al sistema `execve()`, el proceso cambiará de programa y también de GID, que pasará a ser el del grupo propietario del fichero ejecutable. Esto, al igual que sucede con el bit SETUID, también produce un cambio de dominio. En cambio, si el bit no está activo, el GID no cambiará y el proceso seguirá perteneciendo al mismo dominio.

¿Qué pasa si, por ejemplo, un programa con el bit SETUID y/o el bit SETGID activo se ejecuta desde un intérprete de órdenes? Pues exactamente lo que acabamos de describir. Cuando un usuario interactúa con un intérprete de órdenes, realmente lo hace con el proceso correspondiente. Este proceso tendrá como dominio el del usuario. Cuando el usuario ejecuta una orden, el intérprete de órdenes creará un hijo que ejecutará la orden con `execve`, pudiendo producirse un cambio de dominio en el proceso hijo si el fichero ejecutable tiene activo alguno de los bits mencionados.

Un posible cambio de dominio producido por los bits SETUID y SETGID hace que, en realidad, un proceso tenga dos parejas de identificadores: el usuario/grupo real (UID, GID) y el usuario/grupo efectivo (EUID, EGID). La primera pareja se corresponde con el par (UID, GID) del proceso padre que crea el proceso (que, como hemos explicado, puede ser un intérprete de órdenes), mientras que la segunda se corresponde con el par (UID, GID) con el que el proceso accede a los ficheros de forma efectiva. Por lo tanto, el par (EUID, EGID) es el que realmente determina qué puede hacer un proceso y qué no. Normalmente, estas dos parejas de identificadores son iguales, salvo cuando están activos los bits SETUID y SETGID, como acabamos de ver. Un programa en C puede usar las funciones `getuid()`/`geteuid()` y `getgid()`/`getegid()` para conocer su usuario/grupo real/efectivo.

Como ejemplo práctico de uso de estos bits, supongamos que el usuario «pepe» del grupo «usuarios» ejecuta el siguiente programa (nótese que tiene permiso para realizar esta ejecución):

```
-rwsr-xr-x root root 20 Abr 12:00 passwd
```

Al ejecutarlo, el proceso `passwd` tendrá los siguientes dominios:

- UID: pepe
- GID: usuarios
- EUID: root
- EGID: usuarios

Por lo tanto, tiene root como usuario efectivo, lo que le permitirá leer y escribir el contenido del fichero `/etc/shadow` para cambiar su contraseña, a pesar de que este fichero suele tener los siguientes propietario, grupo y permisos [5]:

```
---------- 1 root shadow 1880 abr 20 12:36 shadow
```

Con estos permisos, es evidente que ningún usuario «normal» puede acceder al fichero `/etc/shadow`. Por ejemplo, la ejecución de la orden `cat /etc/shadow` fallará para cualquier usuario excepto el root[6], ya que el ejecutable `cat` suele tener los siguientes propietario, grupo y permisos:

```
-rwxr-xr-x root root 50416 abr 20  12:00 cat
```

Nótese que la orden `passwd` permite a un usuario cambiar su contraseña, pero no la de los demás usuarios a pesar de tener permiso de escritura en todo el fichero `/etc/shadow` gracias al bit SETUID. Sin embargo, el usuario root puede cambiar la contraseña de cualquier usuario. Para saber qué tipo de usuario está cambiando una contraseña, `passwd` utiliza en su implementación la función `getuid()` que, como hemos indicado, devuelve el UID real del proceso, es decir, el UID del usuario que ejecuta el programa. De esta manera, sabe qué contraseñas puede cambiar. Así, si la función devuelve 0, entonces el usuario root es el que está ejecutando el programa y, por tanto, puede cambiar la contraseña de cualquier usuario. En cambio, si la función devuelve un valor distinto de 0, entonces el programa lo está ejecutando un usuario normal que solo podrá cambiar su contraseña.

3.5.4 Combinación de listas de control de acceso y de posibilidades

Como hemos comentado en la sección 3.3.5, Unix implementa un sistema híbrido entre listas de control de acceso y listas de posibilidades que funciona de la siguiente forma. Como hemos visto, los ficheros tienen asociada una ACL restringida. Cuando un proceso quiere abrir un fichero, se comprueba la ACL para ver si tiene acceso. En caso afirmativo, se crea una entrada en la *tabla de ficheros abiertos* de ese proceso y se devuelve un índice a la misma denominado *descriptor del fichero*. Entre otras cosas, la entrada contiene los permisos para ese proceso sobre ese fichero (leer, escribir, etc.). De este modo, la información de permisos de la tabla de ficheros abiertos es en sí una lista de capacidades del proceso. Cuando el proceso quiera realizar una operación sobre el fichero, solo tendrá que pasar el descriptor del fichero como parámetro para tener acceso al mismo. Cuando se cierra el fichero, se elimina la entrada de la tabla.

Como el sistema operativo mantiene la tabla de ficheros abiertos, esta no puede ser corrompida por el programa del usuario ejecutándose en modo usuario en el proceso;

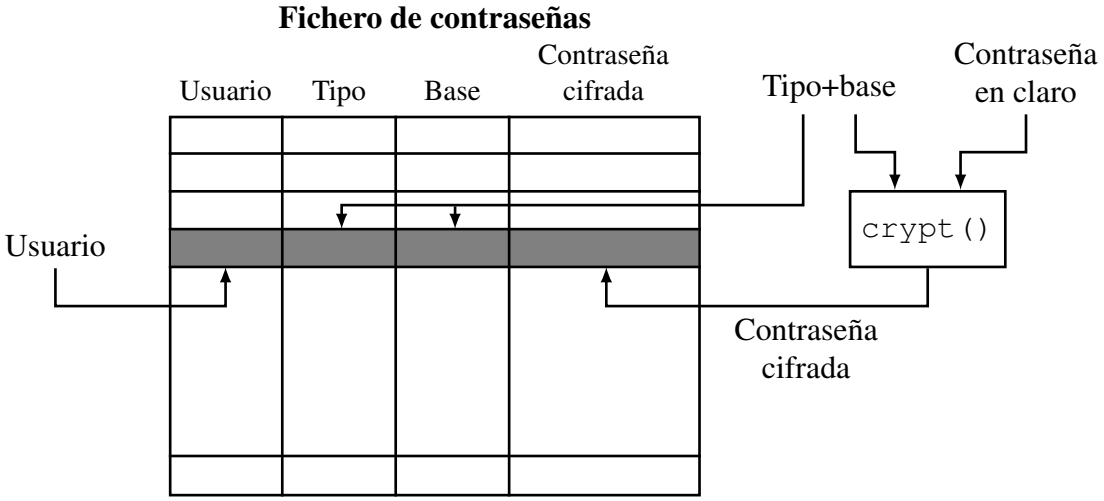
[5] Los permisos exactos del fichero `/etc/shadow` pueden cambiar de una distribución a otra de Linux.

[6] Según los permisos mostrados, aparentemente, ni siquiera el usuario root podría acceder al fichero `/etc/shadow`; sin embargo, el root es un usuario especial que no tiene restricciones en el sistema.

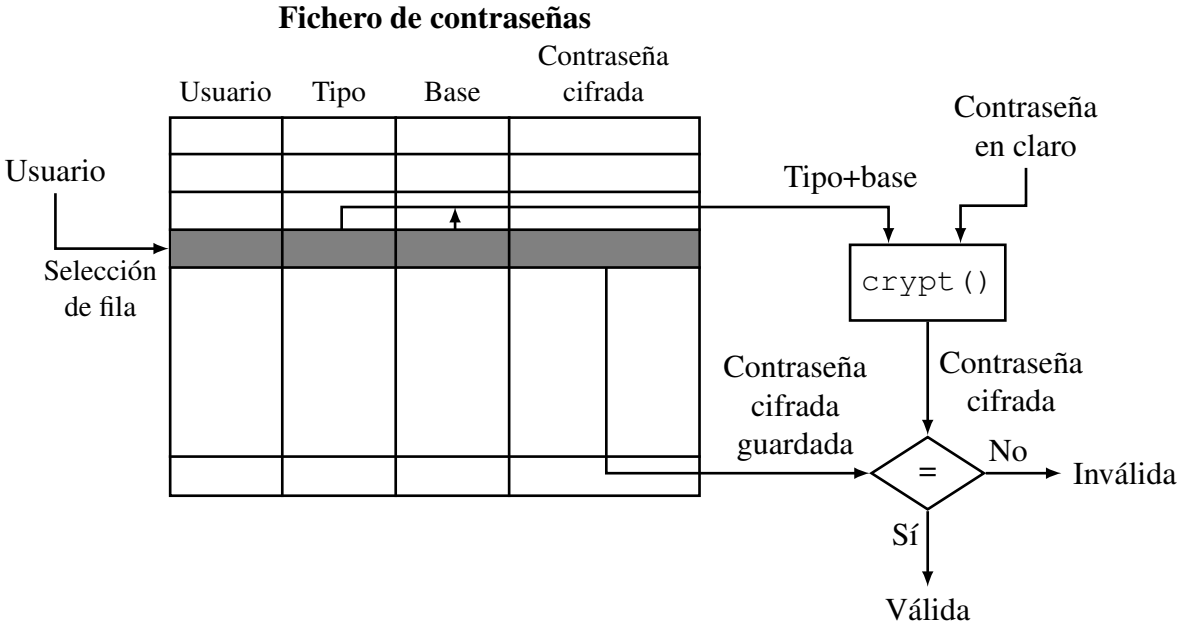
por tanto, el programa del usuario solo puede acceder a los ficheros que están abiertos y, puesto que el acceso se verifica al abrir el fichero, se asegura la protección.

Nótese que aún debe verificarse el derecho en cada acceso, y que la entrada de la tabla de ficheros tiene una posibilidad solo para las operaciones permitidas. De esta forma, si se abre un fichero solo para lectura (aunque los permisos también permitan abrirlo para escritura), entonces se colocará una posibilidad de acceso de lectura en la entrada de la tabla. Si después se intenta efectuar una escritura en el fichero, comparando la operación solicitada con la posibilidad en la entrada de la tabla de ficheros se determinará la violación de la protección.

Un problema de esta combinación de listas de control de acceso y listas de posibilidades es que la ACL de un fichero solo se comprueba al abrirlo. Si un proceso abre un fichero y su ACL cambia después, el proceso podrá seguir accediendo al fichero mientras lo tenga abierto, aunque la nueva ACL no le permitiría el acceso si lo abriera otra vez. Esto incumple el principio de diseño que establece que la autorización se tiene que comprobar continuamente (ver sección 3.2.6). No obstante, este tipo de seguridad un poco más laxa permite a cambio mejorar el rendimiento al trabajar con ficheros.



(a) Almacenamiento de una nueva contraseña para un usuario.



(b) Verificación de la contraseña introducida por un usuario.

Figura 3.6. Esquema de contraseñas de Unix.

Verso	Contraseña
Con cien cañones por banda	100cbanda
viento en popa a toda vela	Vpvela
no corta el mal sino vuela	Ncmvuela
un velero bergantín	1vbergantín

(a) Líneas de unos versos escogidos

Ciudad	Expresión intermedia	Contraseña
Sevilla	Sevilla tiene un color especial	STUCE
París	París bien vale una misa	PBVUM
Granada	Granada, tierra soñada por mí	GTSPM
Valencia	Valencia es la tierra de las flores	VELTDLF

(b) Expresiones inspiradas por nombres de ciudades.

Comida	Contraseña
Cacahuetes con chocolate	caconchoco
Caramelos de piña y coco	carpicoco
Berenjenas fritas	berenfrit

(c) Comidas que no gustan.

Transformación	Expresión ilustrativa	Contraseña
Transliteración	gibraltar	jivraltar
	berenjena	verengena
Entretejer caracteres de palabras sucesivas	canciller, hierro	canchierllerro
Traducción	extranjeros	etraniere
Sustitución de letras por dígitos decimales (índice de la letra en orden alfabético módulo 10)	repollo	9576226
Sustitución de números decimales por letras (de la posición correspondiente en orden alfabético)	12/10/1492	abaadrb
Desplazamiento desde una posición de partida en un teclado	calabacín	vsñsnsvom
Sustitución por sinónimos	quemarropa	ardevestido
Sustitución por antónimos	malnacido	bienmuerto
Activación de las mayúsculas del teclado	6-6-1944	&_&_!)$$
Sustitución por abreviaturas	humedad relativa	humrel
Sustitución por acrónimos	Madres Contra la Droga Organización Nacional de Mujeres	mcdonm
Repeticiones	pan	panpan
Manipulación de imágenes de letras (rotar las letras 180 grados)	cambembo	cawpewpo

(d) Técnicas de transformación

Figura 3.7. Estrategias efectivas para la selección de contraseñas fáciles de recordar por parte del usuario y difícilmente adivinables para un atacante.