Seguridad en la administración de usuarios, módulos PAM y funciones HASH para el cifrado de contraseñas.

Nicolás García Martín

Departamento Ciencias de la Computación y de la Automatización, Universidad de Salamanca

Plaza de la Merced, s/n, 37008, Salamanca, España

nikomartn@usal.es

Resumen: Gestión de los usuarios en Debian 9, sus privilegios y restricciones, autentificación de estos mediante una arquitectura abierta y modular, y cifrado de sus claves de acceso.

Palabras clave: Debian, Stretch, Usuarios, PAM, Hash, passwd, sudoers, security, UID, adduser.

Nota: Cada vez que se hace referencia a Debian en este documento, se refiere a la distribución GNU/Linux. La comunidad de Debian distribuye varias versiones con otros núcleos de sistema operativo.

1. Administración de usuarios en Debian

En las distribuciones Unix, y como tal Linux, los permisos que se le conceden a los usuarios son manejados por sus cuentas de usuario. La cuenta de usuario más importante de un sistema Unix es la denominada “root” o superusuario, dado que es la cuenta con todos los permisos accesibles, es la administradora del sistema operativo [1]. Es tan importante que en FHS dispone de directorio especial para ella apartado de los directorios para usuarios regulares, y se localiza en /root. En Debian existen varios modos de acceder a derechos de superusuario, dependiendo de cómo se haya configurado el equipo.

Se puede realizar un cambio de usuario mediante el comando “su” (substitute user) [2]. Es recomendable utilizar el comando seguido de “- “, “-l” o “—login” puesto que esto reinicia las variables de entorno. Si vamos a iniciar sesión, es

conveniente no traernos las variables de la sesión anterior e igual de cierto es para cuando se inicia sesión como root. Además, esto iniciará una login Shell.

Para iniciar sesión como root podemos llamar al comando su sin argumentos “su –“. El sistema nos pedirá la contraseña de administrador y accederemos a la cuenta. [2]

Otra opción, es que un usuario regular tenga permisos de superusuario, para ello, el usuario debe de pertenecer al grupo de usuarios **sudo** (se explica más adelante). Cuando un usuario pertenece a dicho grupo, si a las llamadas a comandos se las antecede con “sudo”, dicho comando se ejecutará con derechos de superusuario.
El comando “sudo” solo solicita la contraseña del usuario. Debian recomienda esta técnica frente a utilizar la cuenta de administrador directamente, por las ventajas de llevar un registro de quién realizó qué comando, no necesitar compartir una cuenta de administración entre varias personas (incluida contraseña), y minimizar los riesgos de ser el usuario más poderoso del sistema en todo momento. [3].

“sudo” permite realizar una acción “haciéndose pasar” por otro usuario, no es una mera herramienta para hacerse pasar por administrador, sino para ejecutar comandos sin necesidad de cambiar de usuario.

Puede instalarse Debian 9 de forma que no “exista” cuenta root, o, mejor dicho, que no sea directamente accesible, para ello, no debe escribirse contraseña de root en la instalación. Debian 9 concederá entonces permisos de superusuario a la cuenta con la que se instale el sistema. [1] En dicho caso, puede simularse el acceso a cuenta de administrador mediante “sudo su [-]” o “sudo -i”.

Para cerrar la sesión de un usuario, pueden utilizarse los comandos “exit [n]” (Cntrl+D) o “logout [n]” para cerrar una login Shell.

El fichero de configuración de los permisos de los usuarios en Debian se encuentra en “/etc/sudoers”, una configuración podría ser la siguiente:

```
# User privilege specification
root    ALL=(ALL:ALL) ALL

# Allow members of group sudo to execute any command
%sudo   ALL=(ALL:ALL) ALL
```

Como se ve, el usuario root y cualquier usuario perteneciente al grupo “sudo” tiene todos los permisos posibles.

Los UID son los identificadores de usuario y los GID identificadores de grupo. En Debian, los usuarios y grupos tienen asignado un ID entre 1000 y 59999, salvo

que se indique lo contrario, por ejemplo, "—system" asignará un ID entre 100 y 999. Este comportamiento está definido en "/etc/adduser.conf". [5]
Los grupos son conjuntos de usuarios, y como tal, se les puede conceder permisos en "/etc/sudoers".

El comando para visualizar información para un usuario es "id" [4], por ejemplo; uid=1000(nikomartn) gid=1000(nikomartn), groups=1000(nikomartn), 4(adm), 24(cdrom), 27(sudo), 30(dip), 46(plugdev)
Muestra que mi identificador de usuario es 1000, y pertenezco a diversos grupos, entre ellos "sudo". El usuario root siempre tiene uid=0. [1]
El comando "groups" permite conocer los grupos a los que está asociado un usuario.

La información sobre los usuarios está almacenada en "/etc/passwd" y la de los grupos en "/etc/group". La información sobre las contraseñas se encuentra en "/etc/shadow". Los administradores del sistema Debian pueden utilizar los comandos "adduser", que permiten añadir usuarios y grupos. Por ejemplo "adduser <usuario>" permite crear un usuario normal con contraseña, directorio en home, variables de entorno para cuando haga login entre otras cosas. "deluser" permite eliminar usuarios, grupos y asociaciones. Se recomienda leer la página man de adduser para ver todas las opciones que ofrece. [6]
El comando adduser también relaciona usuarios con grupos "adduser <usuario> <grupo>.

El comando "passwd" permite modificar las contraseñas de los usuarios, como es de esperar, los usuarios regulares pueden modificar solo sus propias contraseñas, pero los administradores pueden modificar cualquier contraseña. [7] Entre otras cosas también puede hacer expirar las contraseñas o impedir su modificación.

Un administrador puede también iniciar sesión a una cuenta con el comando "login". Cuando se inicia el sistema Unix, el proceso init es el encargado de llamar a este comando para permitir el acceso del usuario. En Debian 9, las responsabilidades de init las tiene systemd, y login está establecido como un servicio de este. [8] El servicio se denomina "systemd-logind.service", y puede ser controlado con el comando "loginctl" [9]. Puede ser encontrado en "/usr/lib/systemd/systemd-logind".

2. Seguridad en la administración de usuarios en Debian

Ahora que ya se comprende como funciona la administración de usuarios, podemos estudiar a fondo cómo se proporciona de seguridad al sistema.

Un fichero muy importante en la seguridad de Debian es "/etc/sudoers" o el directorio "/etc/sudoers. d". Aquí se almacena la política de seguridad en Debian. Para configurar directamente este fichero, es obligatorio [3] utilizar el comando "visudo" [10]. Este comando permite editar el fichero como sección crítica

(evitando su manipulación por diferentes usuarios al mismo tiempo) y comprobar errores. Debian permite seleccionar el editor de texto preferido el enlace simbólico en "/etc/alternatives/editor", el cual será llamado por visudo.

Si bien es cierto que Unix permite establecer permisos de acceso a un fichero según dueño y grupo, las directivas de sudo permiten solicitar identificación y manipular los privilegios de usuarios y grupos. "sudoers" permite establecer quien puede llamar a qué comandos, como qué usuarios y desde qué computadoras.

Extended Backus-Naur Form es la gramática del fichero sudoers, Cada definición se compone de un conjunto de reglas, símbolo:: = definición | opcional |opcional…
Y contiene los siguientes operadores, '?' indica opcional, '*' indica 0 o más veces y '+' indica 1 o más veces. [11]

El fichero sudoers permite 4 tipos de alias, los alias de usuario especifican usuarios, grupos de usuarios (antecedidos por %) y grupos en red (antecedidos por +). Los comentarios se anteceden de '#'. Sigamos el siguiente ejemplo:

Un hospital quiere permitir el acceso a un programa que maneja información de los pacientes solo a médicos jefes de sección y el personal administrativo, no se desea que los residentes puedan acceder. El sistema contiene los grupos de usuarios "administrativo" y "médico".

```
#Especifiquemos los grupos para mayor claridad
User_Alias MEDICOS = %medico
User_Alias ADMINISTRATIVOS = %administrativo

#Los jefes de sección son limitados, podemos especificarlos directamente en el fichero
User_Alias JEFES_SECCION = "aperez", "susanagf", "ariasmn", "carlosrg"

#Ahora podemos crear un grupo para concederle permisos específicos
User_Alias ACCESO_PACIENTES = ADMINISTRATIVOS, JEFES_SECCION

#O podemos crear un grupo de médicos que no son jefes de sección y excluirlo
User_Alias NO_ACCESO_PACIENTES = MEDICOS, !JEFES_SECCION

Las runas permiten especificar el usuario por su UID,
Runas_Alias ROOT = #0
Runas_Alias ADMINS = %admin, root

Los alias de anfitrión permiten especificar redes
Host_Alias SERVER = 192.168.0.1
```

Los alias de comando permiten especificar conjuntos de comandos, imaginemos que, del ejemplo anterior, los programas de administración de los pacientes son "/usr/local/bin/paciente-editar",
"/usr/local/bin/paciente-recetas",
"/usr/local/bin/ingresos".
Y dichos ejecutables tienen como grupo "staff", pero queremos que los jefes de sección puedan acceder también a ellos si se identifican.

Cmnd_Alias PROGRAMAS_PACIENTES = /usr/local/bin/paciente-*, /usr/local/bin/ingresos

Y ahora podemos especificar permisos exclusivos a ACCESO_PACIENTES sobre estos programas, las reglas de sudo se componen de lo siguiente:

Usuario desde = (como usuario) contraseña? : comandos

Así puede entenderse como root puede hacer cualquier cosa:
root ALL = (ALL:ALL) ALL

Si solo queremos que los médicos con acceso puedan acceder desde el servidor del hospital, la directiva sería la siguiente:
ACCESO_PACIENTES SERVER = (ALL) PROGRAMAS_PACIENTES

Ahora, un jefe de sección, que pertenece al grupo "medico" puede acceder también a estos ejecutables si los antecede de sudo siempre y cuando se identifique mediante su contraseña. Pero si un médico que no es jefe de sección intenta acceder, sudo no se lo permitirá.

El sistema Debian permite otorgar de modularidad el fichero sudoers, para ello "/etc/sudoers" ha de terminar con la línea "#includedir /etc/sudoers.d" que permite añadir ficheros independientes como módulos de políticas de seguridad. [11]

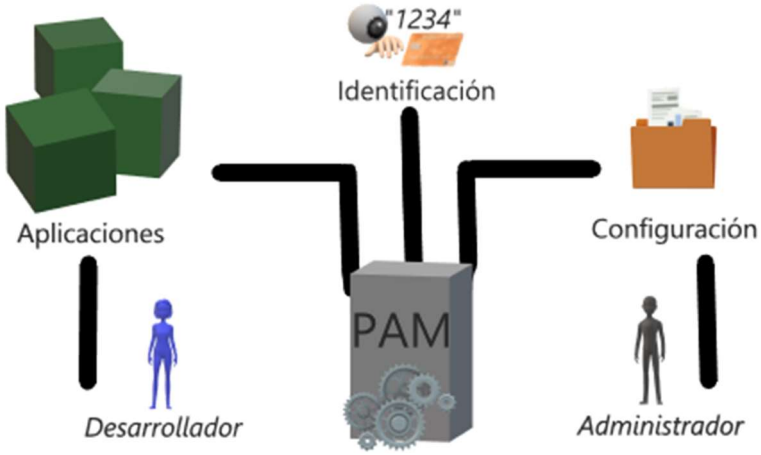
Para las definiciones anteriores del archivo "/etc/sudoers" se ha utilizado la documentación [12] y [13] para Ubuntu por ser más sencilla, pero no se recomienda seguir en su totalidad estas guías para Debian puesto que muchas veces estas distribuciones toman decisiones de diseño distintas y podrían no ser compatibles.

Todo lo que se ha explicado para con los usuarios, puede ser aplicado a programas y en particular servicios(daemons), los cuales nunca deberían de ejecutarse con todos los permisos sobre la máquina. A estos puede otorgárseles un UID y restringirles los permisos. [14]. Como ya se ha mencionado, podemos comprobar los usuarios registrados en el sistema en "/etc/passwd".

3. Los módulos de autentificación conectables (PAM)

En lugar de hacer que cada programa se encargue de la autentificación de los usuarios, en Debian puede recurrirse a los PAM, los cuales son una interfaz que permite integrar las utilidades de autentificación con nuevos sistemas sin la necesidad de reescribir el código. [15]

Un ejemplo de esto podría ser un sistema de bibliotecas, el cual solo permite retirar libros a usuarios que poseen una tarjeta de identificación, podemos utilizar módulos PAM para integrar los certificados de estas tarjetas.



Se puede abordar PAM mediante 3 ópticas, la de un administrador de sistemas, un desarrollador de módulo o un desarrollador de aplicaciones que utiliza PAM. [16]

[16] Como administrador, PAM hace de intermediario entre la identificación del sistema y la de las aplicaciones, y su comportamiento puede ser configurado con "/etc/pam.conf" o "/etc/pam.d" si se desea añadir comportamientos como módulos. De esta manera, si se desea cambiar el comportamiento de una aplicación en lo referente a comprobar el usuario, basta con modificar estos archivos.

La sintaxis de la configuración es la siguiente:

#Marca un comentario
Servicio tipo control modulo-ruta modulo-argumentos

El **servicio** no es necesario especificarlo si se está configurando un módulo, el cual debe de tener el servicio como nombre de fichero. El **tipo** es el grupo al que

hace referencia la regla, son “account” para establecer reglas de acceso a servicio sin autentificación, “auth” para la autentificación, “password” para actualizar el token asociado a la autentificación del usuario y “session” para reglas antes y después de que el servicio le sea otorgado.

Anteceder al tipo con ‘-‘, indica opcionalidad si el módulo está cargado en el sistema.

“**control**” especifica el comportamiento de la regla. Los valores admitidos son los siguientes:

“required” indica que un fallo será devuelto después de que la pila de módulos haya sido ejecutada.
“requisite” funciona como “required” pero el control será devuelto inmediatamente a la aplicación o a una pila PAM superior.
“sufficient” indica que ante el éxito de un módulo, y sin que ninguno haya fallado anteriormente en la pila, se devuelve éxito sin ejecutar posteriores módulos.
“optional” especifica que el éxito o fracaso del módulo es solo importante si es el único módulo asociado con ese servicio y tipo.
“include” incluye un módulo de configuración pasado como argumento.
“substack” funciona como “include” pero el éxito o fracaso de dicho módulo será considerado como una unidad, es decir, aunque en el módulo exista una regla con “sufficient” con éxito, el control será devuelto al módulo superior, no a la aplicación.

Se puede perfilar con más detalle el comportamiento de control con valores y diferentes acciones especificados en “/usr/include/security/_pam_types.h”. Las acciones disponibles son “ignore”, “bad” (fallo), “die” (fallo, terminar y volver a la aplicación), “ok” (el valor de retorno del módulo implica el retorno del conjunto de la pila), “done” (ok, terminar y volver a la aplicación).

Las reglas control se especifican como [valor=acción valor’=acción’ …]
Así por ejemplo la regla sufficient, se podría expresar también como:
[success=done new_authtok_reqd=done default=ignore]

Por ejemplo, el módulo “/etc/pam.d/su” tiene la siguiente regla:
“auth sufficient pam_rootok.so”

El servicio es “su” por ser éste el nombre del fichero, el tipo es “auth” porque es sobre la autentificación, “sufficient” porque si el módulo tiene éxito, no es necesario llamar a más módulos

El módulo [17] pam_rootok.so comprueba que el UID real del usuario es 0.

Por lo que, al llamar a “su”, se comprobará que el UID sea 0, de serlo, se permitirá el acceso sin más comprobaciones.

Lógicamente es más seguro comenzar con un sistema muy restringido e ir administrando permisos que hacerlo con uno sin ninguna seguridad e imponer todas las restricciones que se nos ocurran. Para ello, se puede configurar el servicio “other”, para todos aquellos que no han sido configurados particularmente.

Otra configuración interesante es “/etc/pam.d/common-password” que define las políticas sobre contraseñas. Es recomendable editar este fichero mediante “pam-auth-update”[23]. En un fichero típico podemos encontrar “password [success=1 default=ignore] pam_usnix.so obscure sha512”, el cual indica que las contraseñas han de ocultarse mediante la función hash “sha512”.

Se pueden aplicar políticas predefinidas con la directiva “@include”. En Debian 9 se incluyen las políticas estándar de Unix, en los módulos “/etc/pam.d/common-*”

Es importante hablar del directorio “/etc/security” donde se pueden incluir políticas de seguridad, que serán interpretadas por módulos PAM. Así por ejemplo en “/etc/pam.d/su” se tiene la regla “sesión required pam_limits.so”. Antes de otorgar el servicio se comprobarán las reglas en “/etc/security/limits.conf” y si el módulo no permite el acceso, lo denegará después de que el resto de posibles módulos terminen su ejecución.

En “limits.conf” podemos limitar a los usuarios, por ejemplo, en el tamaño de disco concedido, el número de procesadores…
La sintaxis de este fichero es la siguiente:
<identificación> <tipo> <recurso> <valor>
El símbolo % permite indicar grupos. Los tipos de restricciones pueden ser hard o soft, y los recursos limitables son “core, data, fsize, memlock, nofile(ficheros abiertos a la vez), rss. stack,cpu (tiempo cpu), nproc, as (espacio de direcciones), maxlogins, maxsyslogins, priority, locks (bloqueos de ficheros), sigpending, msgqueue, nice, rtprio (prioridad en tiempo real), chroot (específico de Debian)”.

Así, por ejemplo, si en el ejemplo del hospital queremos limitar a los médicos a tener una única sesión abierta a la vez:
“medico hard maxlogins 1”

El módulo ha de estar activado para login en “/etc/pam.d/login”
“session required pam_limits.so”

Entonces cuando un médico trate de hacer login o su teniendo ya abierta una sesión, se devolverá que hay demasiados inicios de sesión y se denegará el permiso. (**Cuidado con este ejemplo, es posible que, si se utiliza un entorno de escritorio, se requiera varias sesiones iniciadas para el mismo usuario**).

Como otras veces, Debian dispone de un directorio "/etc/security/limits.d", para implementar estas políticas de manera modular.

Así, todas las aplicaciones que se compilen utilizando el api PAM, utilizarán las reglas y configuración establecidas por el administrador del sistema. Lo que incluye la capacidad de añadir nuevos sistemas de autentificación. Podemos modificar de una sola vez la política de seguridad de muchas aplicaciones sin necesidad de recompilarlas.

En "/etc/security", en una instalación típica de Debian 9 pueden encontrarse los siguientes ficheros de configuración:

Fichero	**Descripción**
"access.conf"	Define las combinaciones con las que serán permitidos los accesos, entre usuarios/grupos/punto de acceso
"limits.conf"	Permite limitar los recursos que serán concedidos a los usuarios en el momento de hacer login
"pam_env.conf"	Permite gestionar las variables de entorno que se cargarán o descargarán cuando un usuario hace login.
"time.conf"	Restringe el acceso a un usuario al sistema o a programas dependiendo de la hora del día.
"capability.conf"	Restringe las capacidades de los usuarios del sistema.
"pwquality.conf"	Permite configurar los requisitos de calidad de las contraseñas.
"groups.conf"	Otorga membresía de grupo a un usuario para un servicio.
"namespace.conf"	Configura la ejecución del script "/etc/security/namespace.init"
"sepermit.conf"	Configura el módulo "pam_sepermit" para la concesión o rechazo de acceso según la configuración de SELinux.

Como desarrolladores del sistema, es posible que deseemos desarrollar PAM para casos particulares, en cuyo caso, en C se ha de incluir la biblioteca

<security/pam__appl.h> y compilar con -lpam en el caso de gcc. En el caso de Linux se incluye también una biblioteca con funciones que facilitan la tarea de crear aplicaciones conscientes de PAM, (<security/pam_misc.h>, compilar con "-lpam__misc").

Los servicios de PAM se inicializan en una aplicación mediante la llamada a " int pam_start (service_name, user, pam_conversation, pamh);" El nombre del servicio es una cadena que especifica el servicio a aplicar, el usuario también una cadena al objetivo, que será almacenado como un ítem "PAM_USER", pam_conversation es una estructura que describirá el protocolo, y el pamh es una estructura abstracta que mantiene el contexto de las sucesivas llamadas a las funciones de PAM. Esta estructura no debería de ser comprobada directamente, para ello se provee las funciones "int pam_get_item(const pam_handle_t *pamh, int item_type, const void **item);" y su opuesto "pam_set_item()". Los valores de retorno son constantes simbólicas "PAM_ABORT, PAM_BUF_ERR, PAM_SUCCESS_, PAM_SYSTEM_ERR".

La estructura pamh es el contexto de la interacción entre la aplicación y PAM, por lo que no es recomendable modificarla sin las adecuadas funciones aportadas por la biblioteca.

Se puede terminar el uso de PAM mediante la función "int pam_end (pamh, pam_status); Esta función libera la memoria utilizada por la biblioteca en el contexto pamh.

Por ejemplo, el proceso de autentificación se realiza con la llamada a la función "int pam_authenticate(pam_handle_t *pamh, int flags);" Pero la biblioteca PAM ha de funcionar de manera abstracta, dado que se desconoce si el usuario se va a identificar remotamente, o mediante una interfaz gráfica o cualquier otro caso. Para esto, PAM provee de una función de conversación, que le permite adaptarse a un protocolo que el cliente pueda comprender.

Los ítems más importantes que pueden manejarse del contexto son PAM_SERVICE, PAM_USER, PAM_USER_PROMPT (cadena "de saludo" que se le mostrará al usuario cuando vaya a hacer login), PAM_TTY, PAM_RUSER (nombre de usuario remoto si accede remotamente), PAM_RHOST, PAM_AUTHTOK. Se recomienda profundizar el uso de las anteriores funciones [18].

La conversación con los módulos PAM se maneja mediante las siguientes estructuras:

```
struct pam_message {
   int msg_style;
   const char *msg;
};
```

```
struct pam_response {
   char *resp;
   int resp_retcode;
};

struct pam_conv {
   int (*conv)(int num_msg, const struct pam_message **msg,
           struct pam_response **resp, void *appdata_ptr);
   void *appdata_ptr;
};
```

*conv es la función de callback que provee la aplicación para que PAM pueda comunicarse.
Las variables num_msg y pam_message **msg funcionan de igual manera que los argumentos de main, se ha de informar del tamaño del vector de mensajes. Por cada llamada se ha de liberar la memoria de los mensajes bajo la responsabilidad del que inicia la conversación. El array permite pasar varios mensajes de una sola llamada, pero dado que algunas implementaciones de PAM tratan el array de manera distinta, la única solución para mantener la compatibilidad es perder esta característica y solo mandar los mensajes de uno en uno [20]. Si exclusivamente se desarrolla para Debian 9, no existe tal inconveniente.

La constante PAM_SILENT indica a PAM que no envíe mensajes a la aplicación. Todas las funciones de PAM aceptan este token.

Para facilitar el desarrollo en Linux, se ofrece en la biblioteca "<security/pam_misc.h>" la siguiente función:

```
int misc_conv( num_msg,
                msgm,
                response,
                appdata_ptr);
int num_msg;
const struct pam_message **msgm;
struct pam_response **response;
void *appdata_ptr;
```

Su funcionalidad se explica en [19].

El manejo de la sesión se provee mediante las funciones "int pam_open_session(pam_handle_t *pamh, int flags);" e "int pam_close_session(pam_handle_t *pamh, int flags);" cuya funcionalidad puede ser especificada por el administrador del sistema.

La gestión de cuentas en una aplicación debería comprobarse mediante la función "int pam_acct_mgmt(pam_handle_t *pamh, int flags);" El cual comprueba que la cuenta del usuario sea correcta. PAM también puede configurar y borrar credenciales de usuario mediante "int pam_setcred(pam_handle_t *pamh, int flags);".

Claro está que la biblioteca solo garantiza la autentificación, es responsabilidad de la aplicación modificar su funcionamiento en función de las respuestas de las funciones.

Aquí puede encontrarse un ejemplo de la implementación del uso de la interfaz PAM en una aplicación. [22]

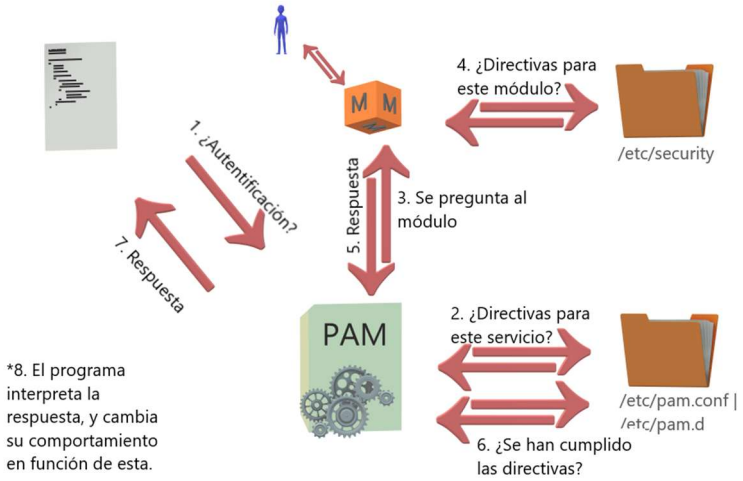
También existe la posibilidad de desarrollar un módulo PAM, por ejemplo, para incorporar nuevos sistemas de identificación. En dicho caso, ha de utilizarse la biblioteca "<security/pam_modules.h>", y enlazar el código objeto como una biblioteca dinámica con -shared y -lpam. La interfaz es la misma que para realizar aplicaciones que hacen uso de los PAM.

Se pueden considerar las seis funciones fundamentales de un módulo pam en 4 familias. Las seis funciones han de estar implementadas o devolver "PAM_SERVICE_ERR" y guardar un mensaje en el log del sistema; si en cambio la acción es considerada inapropiada, ha de devolverse "PAM_IGNORE". En caso contrario, una configuración inapropiada podría hacer fallar al sistema. Además, todas las funciones han de admitir la opción "PAM_SILENT".

Familia de funciones	***Función***	***Descripción***
Gestión de autentificación	"pam_sm_authenticate()"	Realiza la tarea de autentificar al usuario.
	"pam_sm_setcred()"	Altera las credenciales del usuario respecto al correspondiente esquema de autorización.
Gestión de cuentas	"pam_sm_acct_mgnt()"	Establece si el usuario tiene permitido ganar acceso en este momento.
Gestión de sesión	"pam_sm_open_session()"	Sirve para comenzar una sesión.
	"pam_sm_close_session()"	Sirve para cerrar una sesión.
Gestión del token	"pam_sm_chauthtok()"	Permite reconfigurar el token de autentificación de un usuario.

Tabla de familias de funciones de la interfaz PAM [21].

En definitiva, puede resumirse el esquema de funcionamiento de PAM con el siguiente diagrama:



El programa preguntará por autentificación, se comprobarán las directivas para dicho servicio en "/etc/pam.conf" o "/etc/pam.d", se llamarán a los módulos especificados para el servicio, los cuales comprobarán sus directivas particulares. Una vez se han obtenido los resultados de los módulos, según la configuración de las directivas del servicio, se le responde al programa, que tendrá que interpretar la respuesta.

4. Login. Obtención de Hash y contraseñas.

La información sobre las cuentas de usuario está almacenada en el fichero "/etc/passwd", cada línea representa a un usuario y cada campo está separado por el símbolo ':'.
NombreDeLogin:TipoContraseña:UID:GID:Nombre:DirectorioHome:InterpreteDeComandos

El tipo de contraseña puede estar vacío, indicando que la cuenta no tiene contraseña, una 'x' representa una encriptación de esta en "/etc/shadow", un '*' o '!' impide el acceso login al usuario. [24]

Las contraseñas son ocultadas en Debian mediante cifrado por una función hash. Estas son funciones, por las cuales, según los valores de entrada, devuelven una salida de un conjunto finito imagen [25][26]. Almacenando la salida y no la contraseña, puede repetirse la operación y comparar los resultados, mejorando así la seguridad.

Las funciones hash para la encriptación deben tener las siguientes propiedades para considerarse óptimas[27]:

Sea h una función hash para la cual $x \rightarrow h(x)$ *en* $H:U \rightarrow M$

Compresión: *h para cualquier entrada de x de arbitraria longitud, retorna h(x) de longitud fija.*
Facilidad de computación: *dados h y una entrada x, la salida h(x) es fácil de computar.*
Resistencia a preimagen: *Para cualquier* $y \in M$ *es computacionalmente inconcebible encontrar una preimagen x tal que* $h(x) = y$ *si su correspondiente entrada es desconocida.*
Resistencia a segunda preimagen: *Para cualquier h(x) es inconcebible computacionalmente encontrar otro* $x' \neq x$ *por el cual* $h(x') = h(x)$*, el caso negativo se denomina colisión.*

Nótese el uso de la palabra inconcebible, dado el progreso de la computación, cálculos que se creen imposibles de realizar en un periodo de tiempo "lucrativo" con la tecnología actual, pueden ser resueltos rápidamente en un futuro. Es el caso de anteriores funciones criptográficas. [28] La naturaleza de la función es reducir para un número infinito de entradas uno finito de salidas, por lo que no puede crearse una función inyectiva.

Los algoritmos de cálculo hash utilizados en Debian, vienen de una rama familiar que se inicia con el algoritmo MD4 creado (1990) por Ronald Rivest, profesor del MIT, Bert den Boer y Antoon Bosselaers encontraron en 1991 colisiones. El algoritmo fue revisado por Rivers como MD5, para el cual también se encontraron colisiones por los mismos [29], dichas colisiones fueron aprovechadas por el programa maligno "flame" suplantando una firma digital en Microsoft Windows [31].

A pesar de sus fallos como función criptográfica, resulta útil para comprobar, por ejemplo, la integridad de dos ficheros. Por lo que se incluye con Debian 9 con el comando "md5sum". A este podemos pasarle como argumento un fichero y devolverá una cadena casi-única de 128 bits.

Inspirados en MD5, se crearon las "Secure Hash Algorithm" o "SHA" por el NIST, de los cuales los SHA-0 y SHA-1 con salida de 160 bits ya tienen colisiones encontradas. [32]

El resto de la familia SHA, por encima de la salida de 200 bits, resulta segura por el momento. Como se explicó anteriormente, por defecto Debian 9 utiliza sha512 para ocultar las contraseñas.

Referencias

1. https://wiki.debian.org/Root
2. https://linux.die.net/man/1/su
3. https://wiki.debian.org/es/sudo
4. https://wiki.debian.org/ShellCommands
5. https://www.debian.org/doc/debian-policy/ch-opersys.html
6. https://packages.debian.org/es/stretch/adduser
7. http://man7.org/linux/man-pages/man1/passwd.1.html
8. https://www.freedesktop.org/software/systemd/man/systemd-logind.service.html
9. https://www.freedesktop.org/software/systemd/man/loginctl.html#
10. https://manpages.debian.org/stretch/sudo-ldap/visudo.8.en.html
11. https://linux.die.net/man/5/sudoers
12. https://ubuntuforums.org/showthread.php?t=1132821
13. https://help.ubuntu.com/community/Sudoers
14. Administración de sistemas Linux Dee-Ann LeBlanc Anaya
15. www.linux-pam.org/whatispam.html
16. http://www.linux-pam.org/Linux-PAM-html/
17. https://linux.die.net/man/8/pam_rootok
18. http://www.linux-pam.org/Linux-PAM-html/adg-interface-by-app-expected.html
19. http://www.linux-pam.org/Linux-PAM-html/adg-libpam-functions.html
20. https://linux.die.net/man/3/pam_conv
21. http://www.linux-pam.org/Linux-PAM-html/mwg-expected-of-module-overview.html
22. http://www.linux-pam.org/Linux-PAM-html/adg-example.html
23. https://manpag.es/debian-stretch/8+pam-auth-update
24. https://www.debian.org/doc/manuals/debian-reference/ch04.en.html
25. https://es.wikipedia.org/wiki/Función_hash
26. https://md5hashing.net/hash
27. http://cacr.uwaterloo.ca/hac/about/chap9.pdf
28. https://www.schneier.com/blog/archives/2005/02/cryptanalysis_o.html
29. https://eprint.iacr.org/2004/199.pdf
30. https://www.researchgate.net/publication/221355081_An_Attack_on_the_Last_Two_Rounds_of_MD4
31. https://blogs.technet.microsoft.com/srd/2012/06/06/flame-malware-collision-attack-explained/
32. https://security.googleblog.com/2017/02/announcing-first-sha1-collision.html