Módulos de Autenticación Cargables

Resumen

Este artículo describe los principios y mecanismos subyacentes de la librería Pluggable Authentication Modules (PAM), y explica cómo configurar PAM, cómo integrar PAM en aplicaciones y cómo escribir módulos PAM.

Tabla de contenidos

1. Introducción . . . 1
2. Términos y Convenciones . . . 2
3. Aspectos Fundamentales de PAM . . . 5
4. Configuración de PAM . . . 8
5. Módulos PAM de FreeBSD . . . 11
6. Programación de aplicaciones PAM . . . 14
7. Programación del módulo PAM . . . 14
Apéndice A: Ejemplo de aplicación PAM . . . 14
Apéndice B: Ejemplo de módulo PAM . . . 18
Apéndice C: Ejemplo de función de conversación PAM . . . 22
Lecturas adicionales . . . 24

1. Introducción

La librería Pluggable Authentication Modules (PAM) es una API para servicios relacionados con la autenticación que permite al administrador del sistema añadir nuevos métodos de autenticación simplemente instalando nuevos módulos PAM y modificando las políticas de autenticación editando el archivo de configuración.

PAM fue definido y desarrollado en 1995 por Vipin Samar y Charlie Lai de Sun Microsystems, y no ha cambiado mucho desde entonces. En 1997, el Open Group publicó la especificación preliminar X/Open Single Sign-on (XSSO), que estandarizó la API de PAM y añadió las extensiones para el single sign-on. En el momento de escribir este artículo, esta especificación aún no se ha adoptado como estándar.

Aunque este artículo se centra principalmente en FreeBSD 5.x, que usa OpenPAM, debería ser aplicable de igual manera a FreeBSD 4.x, que usa Linux-PAM, y a otros sistemas operativos como Linux y Solaris™.

2. Términos y Convenciones

2.1. Definiciones

La terminología que rodea PAM es bastante confusa. Ni el documento original de Samar y Lai, ni la especificación XSSO hicieron ningún intento de definir formalmente los términos para los diversos actores y entidades involucradas en PAM, y los términos que usan (pero no definen) a veces son engañosos y ambiguos. El primer intento de establecer una terminología coherente e inequívoca fue un documento técnico escrito por Andrew G. Morgan (autor de Linux-PAM) en 1999. Si bien la terminología elegida por parte de Morgan fue un gran avance, en opinión de este autor, no es de ninguna manera perfecta. Las definiciones que se muestran son un intento, fuertemente inspiradas por Morgan, de definir de forma precisa y sin ambigüedades los términos para todos los actores y entidades involucradas en PAM.

cuenta

El conjunto de credenciales que el solicitante solicita al árbitro.

solicitante

El usuario o entidad que solicita la autenticación.

árbitro

El usuario o entidad que tiene los privilegios necesarios para verificar las credenciales del solicitante y la autoridad para otorgar o denegar la solicitud.

cadena

Una secuencia de módulos que se invocarán en respuesta a una solicitud PAM. La cadena incluye información sobre el orden en el que invocar los módulos, qué argumentos pasar y cómo interpretar los resultados.

cliente

La aplicación responsable de iniciar una solicitud de autenticación en nombre del solicitante y de obtener la información de autenticación necesaria de él.

funcionalidad

Uno de los cuatro grupos básicos de funcionalidad proporcionados por PAM: autenticación, gestión de cuentas, gestión de sesiones y actualización del token de autenticación.

módulo

Una colección de una o más funciones relacionadas que implementan una funcionalidad de autenticación particular, recogida en un único archivo binario (normalmente cargable dinámicamente) e identificado por un solo nombre.

política

El conjunto completo de instrucciones de configuración que describen cómo manejar las solicitudes PAM para un servicio en particular. Una política normalmente consta de cuatro cadenas, una para cada funcionalidad, aunque algunos servicios no utilizan las cuatro facilidades.

servidor

La aplicación que actúa en nombre del árbitro para conversar con el cliente, recuperar información de autenticación, verificar las credenciales del solicitante y otorgar o rechazar solicitudes.

servicio

Una clase de servidores que proporcionan una funcionalidad similar o relacionada y que requieren una autenticación similar. Las políticas de PAM se definen por cada servicio, por lo que todos los servidores que reclaman el mismo nombre de servicio estarán sujetos a la misma política.

sesión

El contexto dentro del cual el servidor presta el servicio al solicitante. Una de las cuatro funcionalidades de PAM, la gestión de sesiones, se ocupa exclusivamente de establecer y destruir este contexto.

token

Un trozo de información asociado con la cuenta, como una contraseña o frase, que el solicitante debe proporcionar para probar su identidad.

transacción

Una secuencia de solicitudes del mismo solicitante a la misma instancia del mismo servidor, comenzando con la autenticación y la configuración de la sesión y terminando con el desmantelamiento de la sesión.

2.2. Ejemplos de Uso

Esta sección tiene como objetivo ilustrar los significados de algunos de los términos definidos anteriormente mediante un puñado de ejemplos simples.

2.2.1. El Cliente y el Servidor Son Uno

Este sencillo ejemplo muestra a `alice` haciendo su(1) a `root`.

```
% whoami
alice

% ls -l `which su`
-r-sr-xr-x  1 root  wheel  10744 Dec  6 19:06 /usr/bin/su

% su -
Password: xi3kiune
# whoami
root
```

- El solicitante es `alice`.
- La cuenta es `root`.

- El proceso su(1) es a la vez cliente y servidor.
- El token de autenticación es `xi3kiune`.
- El árbitro es `root`, que es el motivo por el que su(1) tiene establecido el setuid a `root`.

2.2.2. El Cliente y el Servidor Están Separados

El ejemplo de abajo muestra a `eve` intentando iniciar una conexión ssh(1) contra `login.example.com`, pide iniciar sesión como `bob` y lo consigue. ¡Bob debería haber escogido una contraseña mejor!

```
% whoami
eve

% ssh bob@login.example.com
bob@login.example.com's password:
% god
Last login: Thu Oct 11 09:52:57 2001 from 192.168.0.1
Copyright (c) 1980, 1983, 1986, 1988, 1990, 1991, 1993, 1994
    The Regents of the University of California.  All rights reserved.
FreeBSD 4.4-STABLE (LOGIN) 4: Tue Nov 27 18:10:34 PST 2001

Welcome to FreeBSD!
%
```

- El solicitante es `eve`.
- El cliente es el proceso ssh(1) de Eve.
- El servidor es el proceso sshd(8) en `login.example.com`
- La cuenta es `bob`.
- El token de autenticación es `god`.
- Aunque no se muestra en el ejemplo, el árbitro es `root`.

2.2.3. Política de Ejemplo

Lo siguiente es la política por defecto de FreeBSD para `sshd`:

```
sshd    auth        required    pam_nologin.so  no_warn
sshd    auth        required    pam_unix.so no_warn try_first_pass
sshd    account     required    pam_login_access.so
sshd    account     required    pam_unix.so
sshd    session     required    pam_lastlog.so  no_fail
sshd    password    required    pam_permit.so
```

- Esta política se aplica al servicio `sshd` (que no está necesariamente restringida al servidor sshd(8).)
- `auth`, `account`, `session` y `password` son funcionalidades.

- pam_nologin.so, pam_unix.so, pam_login_access.so, pam_lastlog.so and pam_permit.so son módulos. En el ejemplo se ve claramente que pam_unix.so proporciona al menos dos funcionalidades (autenticación y gestión de cuentas.)

3. Aspectos Fundamentales de PAM

3.1. Funcionalidades y Primitivas

La API de PAM ofrece seis primitivas de autenticación diferentes agrupadas en cuatro funcionalidades, que se describen a continuación.

`auth`

Autenticación. Esta funcionalidad tiene que ver con la autenticación de un solicitante y el establecimiento de las credenciales de la cuenta. Proporciona dos primitivas:

- pam_authenticate(3) autentica al solicitante, normalmente solicitando un token de autenticación y comparándolo con un valor almacenado en una base de datos u obtenido de un servidor de autenticación.
- pam_setcred(3) establece las credenciales de la cuenta tales como el ID de usuario, la pertenencia a grupos y los límites de recursos.

`account`

Gestión de cuentas. Esta funcionalidad se encarga de los problemas de disponibilidad con las cuentas, que no están relacionados con la autenticación, como restricciones de acceso según la hora del día o la carga del servidor de trabajo. Proporciona una única primitiva:

- pam_acct_mgmt(3) verifica que la cuenta solicitada está disponible.

`session`

Gestión de sesiones. Esta funcionalidad gestiona tareas asociadas con el establecimiento y desmantelamiento de sesiones, tales como la contabilidad de login. Proporciona dos primitivas:

- pam_open_session(3) realiza tareas asociadas con el establecimiento de sesión: añade una entrada en las bases de datos utmp y wtmp, arranca un agente SSH, etc.
- pam_close_session(3) realiza tareas asociadas con el desmantelamiento de la sesión: añade una entrada en las bases de datos utmp y wtmp, parar un agente SSH, etc.

`password`

Gestión de contraseñas. Esta funcionalidad se usa para cambiar el token de autenticación asociado a una cuenta, ya sea porque ha caducado o porque el usuario desea cambiarla. Proporciona una única primitiva:

- pam_chauthtok(3) cambia el token de autenticación, opcionalmente verifica que es difícil de adivinar, que no ha sido usada previamente, etc.

3.2. Módulos

Los módulos son un concepto central en PAM; después de todo, son la "M" en "PAM". Un módulo PAM es un trozo de código auto-contenido que implementa las primitivas de una o más funcionalidades para un mecanismo en particular; posibles mecanismos para la funcionalidad de autenticación, por ejemplo, incluye la base de datos de contraseñas de UNIX®, NIS, LDAP and Radius.

3.2.1. Nomenclatura de Módulos

FreeBSD implementa cada mecanismo en un módulo separado, llamado `pam_mechanism.so` (por ejemplo, `pam_unix.so` para el mecanismo de UNIX®.) Otras implementaciones a veces tienen módulos separados para funcionalidades separadas e incluyen en el nombre del módulo el nombre de la funcionalidad así como el nombre del mecanismo. Por mencionar un ejemplo, Solaris™ tiene un módulo `pam_dial_auth.so.1` que se usa de forma habitual para autenticar usuarios conectados mediante "dialup".

3.2.2. Versionado de Módulos

La implementación original de PAM en FreeBSD, basada en Linux-PAM, no usaba números de versión para los módulos PAM. Esto normalmente causaría problemas con las aplicaciones heredadas (legacy), que podrían estar vinculadas con versiones anteriores de las bibliotecas del sistema, ya que no había forma de cargar una versión correspondiente de los módulos requeridos.

OpenPAM, por otro lado, busca módulos que tengan el mismo número de versión que la biblioteca PAM (actualmente 2), y solo recurre a un módulo sin versión si no se puede cargar un módulo que tenga versión. Por lo tanto, se pueden proporcionar módulos heredados (legacy) para aplicaciones heredadas (legacy), lo cual permite que las aplicaciones nuevas (o compiladas de nuevo) aprovechen los módulos más recientes.

Aunque los módulos PAM de Solaris™ normalmente tienen un número de versión, no están versionados realmente, porque el número es una parte del nombre del módulo y debe incluirse en la configuración.

3.3. Cadenas y Políticas

Cuando un servidor inicia una transacción PAM, la librería PAM intenta cargar una política para el servicio especificada en la llamada pam_start(3). La política especifica cómo se deberían procesar las solicitudes de autenticación, y está definida en un fichero de configuración. Este es otro concepto central en PAM: la posibilidad que tiene el administrador de afinar la política de seguridad del sistema (en el sentido más amplio de la palabra) simplemente editando un fichero de texto.

Una política consta de cuatro cadenas, una para cada una de las cuatro funcionalidades de PAM. Cada cadena es una secuencia de instalaciones de configuración, cada una especifica un módulo a invocar, algunos parámetros (opcionales) para pasar al módulo y un flag de control que describe cómo interpretar el código de retorno del módulo.

Comprender los flags de control es esencial para comprender los archivos de configuración de PAM. Hay cuatro flags de control diferentes:

`binding`

Si el módulo tiene éxito y ningún módulo anterior en la cadena ha fallado, la cadena se terminará de inmediato y se otorgará la solicitud. Si el módulo falla, el resto de la cadena se ejecuta, pero la solicitud finalmente se deniega.

Este flag de control fue introducido por Sun en Solaris™ 9 (SunOS™ 5.9), y también se soporta en OpenPAM.

`required`

Si el módulo tiene éxito, el resto de la cadena se ejecutará, y la solicitud se otorgará a menos que otro módulo falle. Si el módulo falla, el resto de la cadena también se ejecutará, pero la solicitud será denegada al final.

`requisite`

Si el módulo tiene éxito, el resto de la cadena se ejecutará y la solicitud se aceptará a menos que falle algún otro módulo. Si el módulo falla, la cadena terminará inmediatamente y la solicitud será denegada.

`sufficient`

Si el módulo tiene éxito y ningún módulo anterior de la cadena ha fallado, la cadena terminará de inmediato y se aceptará la solicitud. Si el módulo falla, se ignorará y se ejecutará el resto de la cadena.

Como la semántica de este flag puede ser algo confusa, especialmente cuando se usa para el último módulo de una cadena, se recomienda usar el flag de control del `binding` si la implementación lo admite.

`optional`

El módulo se ejecuta, pero el resultado es ignorado. Si todos los módulos en una cadena están marcados como `optional`, todas las solicitudes serán concedidas.

Cuando un servidor invoca una de las seis primitivas de PAM, PAM recupera la cadena para la funcionalidad a la que pertenece la primitiva e invoca cada uno de los módulos enumerados en la cadena, en el orden en el que se enumeran, hasta que llega al final, o determina que no es necesario ningún procesamiento adicional (porque un módulo `binding` o `sufficient` tuvo éxito, o porque falló un módulo `requisite`). La solicitud es aceptada si y solo si se invocó al menos un módulo, y todos los módulos que no sean opcionales tuvieron éxito.

Ten en cuenta que es posible, aunque no muy común, tener el mismo módulo listado varias veces en la misma cadena. Por ejemplo, un módulo que busca nombres de usuario y contraseñas en un servidor de directorio podría invocarse varias veces con diferentes parámetros que especifican diferentes servidores de directorio para contactar. PAM trata diferentes ocurrencias del mismo módulo en la misma cadena como módulos diferentes no relacionados.

3.4. Transacciones

A continuación se describe el ciclo de vida de una transacción PAM típica. Ten en cuenta que si alguno de estos pasos falla, el servidor debe informar al cliente con un mensaje de error adecuado y cancelar la transacción.

1. Si es necesario, el servidor obtiene credenciales de árbitro mediante un mecanismo independiente de PAM - normalmente al haber sido iniciado como `root`, o haber sido establecido el setuid a `root`.
2. El servidor invoca pam_start(3) para inicializar la librería PAM y especifica su nombre de servicio y la cuenta objetivo, y registra una función de conversación adecuada.
3. El servidor obtiene información relacionada con la transacción (como el nombre de usuario del solicitante y el nombre del host en el que se ejecuta el cliente) y la envía a PAM utilizando pam_set_imte(3).
4. El servidor invoca pam_authenticate(3) para autenticar al solicitante.
5. El servidor llama a pam_acct_mgmt(3) para verificar que la cuenta solicitada está disponible y es válida. Si la contraseña es correcta pero ha expirado, pam_acct_mgmt(3) devolverá `PAM_NEW_AUTHTOK_REQD` en lugar de `PAM_SUCCESS`.
6. Si el paso anterior devolvió `PAM_NEW_AUTHTOK_REQD`, el servidor llama a pam_chauthtok(3) para forzar al cliente a cambiar el token de autenticación para la cuenta solicitada.
7. Ahora que el solicitante se ha antenticado correctamente, el servidor invoca pam_setcre(3) para establecer las credenciales de la cuenta solicitada. Puede hacer esto porque actúa en nombre del árbitro, y posee las credenciales del mismo.
8. Una vez que se han establecido las credenciales correctas, el servidor llama a pam_open_session(3) para establecer la sesión.
9. El servidor ahora realiza cualquier servicio que solicite el cliente—por ejemplo, proporciona un shell al solicitante.
10. Una vez que el servidor ha terminado con el cliente, llama a pam_close_session(3) para desmantelar la sesión.
11. Finalmente, el servidor llama a pam_end(3) para notificar a la librería PAM que ha terminado y que puede liberar los recursos que se hubieran adquirido durante el curso de la transacción.

4. Configuración de PAM

4.1. Archivos de Políticas de PAM

4.1.1. El fichero /etc/pam.conf

El archivo de configuración tradicional de PAM es /etc/pam.conf. Este archivo contiene todas las configuraciones de PAM para tu sistema. Cada línea del archivo describe un paso es una cadena, como se muestra a continuación:

```
login   auth    required        pam_nologin.so  no_warn
```

Los campos son, en orden de aparición: nombre del servicio, nombre de la funcionalidad, flag de control, nombre del módulo y argumentos del módulo. Cualquier campo adicional se interpreta como un argumento adicional del módulo.

Se construye una cadena separada para cada par servicio / funcionalidad, de forma que aunque el orden en que aparecen lineas para el mismo servicio y funcionalidad es significativo, el orden que el que aparecen listados los servicios y las funcionalidades individuales no lo es. Los ejemplos en el documento original de PAM agrupaba la configuración por funcionalidades y el pam.conf por defecto de Solaris™ todavía lo hace, pero la configuración por defecto de FreeBSD agrupa las lineas de configuración por servicio. Cualquiera de las dos formas está bien; cualquiera de las dos tiene sentido.

4.1.2. El directorio /etc/pam.d

OpenPAM y Linux-PAM soportan un mecanismo de configuración alternativo, que es el mecanismo preferido en FreeBSD. En este esquema, cada política está contenida en un archivo separado con el nombre del servicio al que se aplica. Estos archivos se almacenan en /etc/pam.d/.

Estos archivos de política por servicio tienen solo cuatro campos en lugar de los cinco de pam.conf: el campo del nombre del servicio se omite. Por lo tanto, en lugar de la línea de ejemplo pam.conf de la sección anterior, debería tener la siguiente línea en /etc/pam.d/login:

```
auth    required        pam_nologin.so  no_warn
```

Como resultado de esta sintaxis simplificada, es posible utilizar la misma política para múltiples servicios vinculando cada nombre de servicio al mismo archivo de política. Por ejemplo, para utilizar la misma política para los servicios `su` y `sudo`, uno podría hacer lo siguiente:

```
# cd /etc/pam.d
# ln -s su sudo
```

Esto funciona porque el nombre del servicio se determina a partir del nombre del archivo en lugar de especificarse en el archivo de política, por lo que el mismo archivo se puede usar para múltiples servicios con nombres diferentes.

Como la política de cada servicio se almacena en un archivo separado, el mecanismo pam.d también facilita la instalación de políticas adicionales para paquetes de software de terceros.

4.1.3. El Orden de Búsqueda de La Política

Como hemos visto anteriormente, las políticas de PAM se pueden encontrar en varios sitios. ¿Qué sucede si existen políticas para el mismo servicio en varios sitios?

Es esencial comprender que el sistema de configuración de PAM se centra en las cadenas.

4.2. Desglose de una Línea de Configuración

Como se explicó en Archivos de Políticas de PAM, cada línea de /etc/pam.conf consiste en cuatro o más campos: el nombre del servicio, el nombre de la funcionalidad, el flag de control, el nombre del módulo, y cero o más argumentos para el módulo.

El nombre del servicio suele ser (aunque no siempre) el nombre de la aplicación a la que se aplica la declaración. Si no estás seguro, consulta la documentación de la aplicación para determinar qué nombre de servicio utiliza.

Ten en cuenta que si usas /etc/pam.d/ en lugar de /etc/pam.conf, el nombre del servicio se especifica mediante el nombre del archivo de política y se omite de las líneas de configuración actuales, que luego empiezan con el nombre de la funcionalidad.

La funcionalidad es una de las cuatro palabras claves de funcionalidad descritas en Funcionalidades y Primitivas.

Del mismo modo, el flag de control es uno de los cuatro descritos en Cadenas y Políticas, que describen cómo interpretar el código de retorno de un módulo. Linux-PAM soporta una sintaxis alternativa que te permite especificar la acción a asociar con cada código de retorno posible, pero se debería evitar ya que no es estándar y está muy ligado al modo en el que Linux-PAM despacha las llamadas de servicio (que difiere enormemente del modo en que lo hacen Solaris™ y OpenPAM). No es sorprendente que OpenPAM no soporte esta sintaxis.

4.3. Políticas

Para configurar PAM correctamente, es esencial comprender cómo se interpretan las políticas.

Cuando una aplicación invoca pam_start(3), la librería PAM carga la política para el servicio especificado y construye cuatro cadenas de módulos (uno para cada funcionalidad). Si una o más de estas cadenas está vacía, las cadenas correspondientes de la política para el `otro` servicio son sustituidas.

Cuando la aplicación llama más tarde a una de los seis primitivas de PAM, la biblioteca PAM recupera la cadena para la funcionalidad correspondiente y llama a la función apropiada del servicio en cada módulo del listado en la cadena, en el orden en el que fueron listados en la configuración. Después de cada llamada a una función del servicio, el tipo de módulo y el código de error devuelto por la función del servicio se utilizan para determinar qué sucede a continuación. Con algunas excepciones, que se analizaran a continuación, se aplica la siguiente tabla:

Tabla 1. Resumen de la ejecución del chain en PAM

	PAM_SUCCESS	**PAM_IGNORE**	**other**
binding	if (!fail) break;	-	fail = true;
required	-	-	fail = true;
requisite	-	-	fail = true; break;
sufficient	if (!fail) break;	-	-

	PAM_SUCCESS	PAM_IGNORE	other
optional	-	-	-

Si `fail` es verdadero al final de una cadena, o cuando se alcanza un "break", el repartidor (dispatcher) devuelve el código de error devuelto por el primer módulo que falló. De lo contrario, devuelve `PAM_SUCCESS`.

La primera excepción es que el código de error `PAM_NEW_AUTHTOK_REQD` se trata como un éxito, si ningún módulo falla y al menos un módulo devuelve `PAM_NEW_AUTHTOK_REQD`, el repartidor devolverá `PAM_NEW_AUTHTOK_REQD`.

La segunda excepción es que pam_setcred(3) trata los módulos `binding` y `sufficient` como si fueran `required`.

La tercera y última excepción es que pam_chauthtok(3) ejecuta la cadena entera dos veces (una para las comprobaciones preliminares y una para establecer realmente la contraseña), y en la fase preliminar trata los módulos `binding` y `sufficient` como si fueran `required`.

5. Módulos PAM de FreeBSD

5.1. pam_deny(8)

El módulo pam_deny(8) es uno de los módulos más sencillos que están disponibles; responde a cualquier petición con `PAM_AUTH_ERR`. Es útil para deshabilitar un servicio rápidamente (añádelo al comienzo de cada cadena), o para terminar cadenas de módulos `sufficient`.

5.2. pam_echo(8)

El módulo pam_echo(8) simplemente pasa sus argumentos a la función de conversación como un mensaje `PAM_TEXT_INFO`. Es más útil para depurar, pero también puede servir para mostrar mensajes como "El uso no autorizado será perseguido" antes de iniciar el procedimiento de autenticación.

5.3. pam_exec(8)

El módulo pam_exec(8) interpreta su primer argumento como el nombre de un programa a ejecutar, y el resto de argumentos son pasados a ese programa como argumentos de línea de comandos. Una posible aplicación es usarlo para ejecutar un programa que monte, en el momento de iniciar sesión, el directorio home del usuario.

5.4. pam_ftpusers(8)

El módulo pam_ftpusers(8)

5.5. pam_group(8)

El módulo pam_group(8) acepta o rechaza solicitantes basándose en su pertenencia a un grupo concreto (normalmente `wheel` para su(1)). Está pensado principalmente para mantener el comportamiento tradicional del su(1) de BSD, pero tiene muchos otros usos, como excluir a ciertos grupos de usuarios de un determinado servicio.

5.6. pam_guest(8)

El módulo pam_guest(8) permite inicios de sesión de invitados utilizando nombres de inicio de sesión fijos. Se pueden establecer varios requisitos para la contraseña, pero el comportamiento por defecto es permitir cualquiera mientras que el nombre de inicio de sesión sea uno asociado a una cuenta de invitado. Se puede usar el módulo pam_guest(8) de forma sencilla para implementar inicios de sesión anónimos en FTP.

5.7. pam_krb5(8)

El módulo pam_krb(8)

5.8. pam_ksu(8)

El módulo pam_ksu(8)

5.9. pam_lastlog(8)

El módulo pam_lastlog(8)

5.10. pam_login_access(8)

El módulo pam_login_acces(8) proporciona una implementación de la primitiva de gestión de cuentas que aplica las restricciones de inicio de sesión especificadas en la tabla login.acces(5).

5.11. pam_nologin(8)

El módulo pam_nologin(8) rechaza los inicios de usuarios que no sean root cuando existe el fichero /var/run/nologin. Este fichero normalmente es creado por shutdown(8) cuando quedan menos de cinco minutos para el tiempo de apagado programado.

5.12. pam_opie(8)

El módulo pam_opie(8) implementa el método de autenticación opie(4). El sistema opie(4) es un mecanismo de reto-respuesta donde la respuesta a cada reto es una función directa del reto y de una clave, de forma que la respuesta se puede computar fácilmente "en el momento" por cualquiera que posea la clave, eliminando la necesidad de listas de contraseñas. Además, como opie(4) nunca reutiliza un reto que ha sido contestado correctamente, no es vulnerable a ataques de

repetición.

5.13. pam_opieaccess(8)

El módulo pam_opieaccess(8) acompaña el módulo pam_opie(8). Su propósito es aplicar las restricciones codificadas en opieaccess(5), que regulan las condiciones bajo las que un usuario que se autenticara normalmente con opie(4) se pueda autenticar usando métodos alternativos. Esto se usa principalmente para prohibir el uso de autenticación mediante contraseña desde máquinas en las que no se confía.

Para ser efectivo, el módulo pam_opieaccess(8) se debe listar como `requisite` justo después de una entrada `sufficient` para pam_opie(8), y antes que otros módulos, en la cadena `auth`.

5.14. pam_passwdqc(8)

El módulo pam_passwdqc(8)

5.15. pam_permit(8)

El módulo pam_permit(8) es uno de los módulos disponibles más simples; responde a cualquier petición con `PAM_SUCCESS`. Es útil como parámetro de sustitución para servicios donde una o más cadenas de otro modo estarían vacías.

5.16. pam_radius(8)

El módulo pam_radius(8)

5.17. pam_rhosts(8)

El módulo pam_rhosts(8)

5.18. pam_rootok(8)

El módulo pam_rootok(8) reporta éxito si y sólo si el id del usuario real del proceso llamante (que se asume que es ejecutado por el solicitante) es 0. Esto es útil para servicios que no sean de red como su(1) o passwd(1), a los que `root` debería tener acceso automático.

5.19. pam_securetty(8)

El módulo pam_securetty(8)

5.20. pam_self(8)

El módulo pam_self(8) reporta éxito si y sólo si los nombres de los solicitantes concuerdan con el de la cuenta objetivo. Es útil para servicios no de red como su(1), donde la identidad del solicitante se

puede verificar fácilmente.

5.21. pam_ssh(8)

El módulo pam_ssh(8) proporciona tanto autenticación como servicios de sesión. El servicio de autenticación permite a los usuarios que tienen claves secretas de SSH protegidas por contraseña en su directorio ~/.ssh autenticarse ellos mismos tecleando la contraseña. El servicio de sesión arranca ssh-agent(1) y lo precarga con las claves que se desencriptaron en la fase de autenticación. Esta característica es particularmente útil para inicios de sesión locales, ya sea en X (usando xdm(8) u otro gestor de sesiones de X que sea compatible con PAM) o en la consola.

5.22. pam_tacplus(8)

El módulo pam_tacplus(8)

5.23. pam_unix(8)

El módulo pam_unix(8) implementa la autenticación mediante contraseña tradicional de UNIX®, usando getpwnam(3) para obtener la contraseña de la cuenta objetivo y comparándola con la proporcionada por el solicitante. También proporciona servicios de gestión de cuentas (forzando tiempos de expiración de cuentas y contraseñas) y servicios de cambio de contraseñas. Este es probablemente el módulo más útil ya que la mayoría de administradores querrán mantener este comportamiento histórico al menos para algunos servicios.

6. Programación de aplicaciones PAM

Esta sección aún no se ha escrito.

7. Programación del módulo PAM

Esta sección aún no se ha escrito.

Apéndice A: Ejemplo de aplicación PAM

Lo que sigue es una implementación mínima de su(1) utilizando PAM. Date cuenta de que usa la función de conversación openpam_ttyconv(3) específica de OpenPAM, que tiene su prototipo en security/openpam.h. Si quieres compilar esta aplicación en un sistema con una librería PAM diferente, tendrás que proporcionar tu propia función de conversación. Una función de conversación robusta es sorprendentemente difícil de implementar; la que se presenta en Ejemplo de función de conversación PAM es un buen punto de partida, pero no debería utilizarse en aplicaciones en el mundo real.

```
/*-
 * Copyright (c) 2002,2003 Networks Associates Technology, Inc.
 * All rights reserved.
```

```
 *
 * This software was developed for the FreeBSD Project by ThinkSec AS and
 * Network Associates Laboratories, the Security Research Division of
 * Network Associates, Inc.  under DARPA/SPAWAR contract N66001-01-C-8035
 * ("CBOSS"), as part of the DARPA CHATS research program.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in the
 *    documentation and/or other materials provided with the distribution.
 * 3. The name of the author may not be used to endorse or promote
 *    products derived from this software without specific prior written
 *    permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE AUTHOR AND CONTRIBUTORS ``AS IS'' AND
 * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED.  IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE LIABLE
 * FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
 * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
 * OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
 * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
 * OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
 * SUCH DAMAGE.
 *
 * $P4: //depot/projects/openpam/bin/su/su.c#10 $
 * $FreeBSD: head/en_US.ISO8859-1/articles/pam/su.c 38826 2012-05-17 19:12:14Z hrs $
 */

#include <sys/param.h>
#include <sys/wait.h>

#include <err.h>
#include <pwd.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <syslog.h>
#include <unistd.h>

#include <security/pam_appl.h>
#include <security/openpam.h> /* for openpam_ttyconv() */

extern char **environ;

static pam_handle_t *pamh;
```

```
static struct pam_conv pamc;

static void
usage(void)
{

    fprintf(stderr, "Usage: su [login [args]]\n");
    exit(1);
}

int
main(int argc, char *argv[])
{
    char hostname[MAXHOSTNAMELEN];
    const char *user, *tty;
    char **args, **pam_envlist, **pam_env;
    struct passwd *pwd;
    int o, pam_err, status;
    pid_t pid;

    while ((o = getopt(argc, argv, "h")) != -1)
        switch (o) {
        case 'h':
        default:
            usage();
        }

    argc -= optind;
    argv += optind;

    if (argc > 0) {
        user = *argv;
        --argc;
        ++argv;
    } else {
        user = "root";
    }

    /* initialize PAM */
    pamc.conv = &openpam_ttyconv;
    pam_start("su", user, &pamc, &pamh);

    /* set some items */
    gethostname(hostname, sizeof(hostname));
    if ((pam_err = pam_set_item(pamh, PAM_RHOST, hostname)) != PAM_SUCCESS)
        goto pamerr;
    user = getlogin();
    if ((pam_err = pam_set_item(pamh, PAM_RUSER, user)) != PAM_SUCCESS)
        goto pamerr;
    tty = ttyname(STDERR_FILENO);
    if ((pam_err = pam_set_item(pamh, PAM_TTY, tty)) != PAM_SUCCESS)
```

```
        goto pamerr;

    /* authenticate the applicant */
    if ((pam_err = pam_authenticate(pamh, 0)) != PAM_SUCCESS)
        goto pamerr;
    if ((pam_err = pam_acct_mgmt(pamh, 0)) == PAM_NEW_AUTHTOK_REQD)
        pam_err = pam_chauthtok(pamh, PAM_CHANGE_EXPIRED_AUTHTOK);
    if (pam_err != PAM_SUCCESS)
        goto pamerr;

    /* establish the requested credentials */
    if ((pam_err = pam_setcred(pamh, PAM_ESTABLISH_CRED)) != PAM_SUCCESS)
        goto pamerr;

    /* authentication succeeded; open a session */
    if ((pam_err = pam_open_session(pamh, 0)) != PAM_SUCCESS)
        goto pamerr;

    /* get mapped user name; PAM may have changed it */
    pam_err = pam_get_item(pamh, PAM_USER, (const void **)&user);
    if (pam_err != PAM_SUCCESS || (pwd = getpwnam(user)) == NULL)
        goto pamerr;

    /* export PAM environment */
    if ((pam_envlist = pam_getenvlist(pamh)) != NULL) {
        for (pam_env = pam_envlist; *pam_env != NULL; ++pam_env) {
            putenv(*pam_env);
            free(*pam_env);
        }
        free(pam_envlist);
    }

    /* build argument list */
    if ((args = calloc(argc + 2, sizeof *args)) == NULL) {
        warn("calloc()");
        goto err;
    }
    *args = pwd->pw_shell;
    memcpy(args + 1, argv, argc * sizeof *args);

    /* fork and exec */
    switch ((pid = fork())) {
    case -1:
        warn("fork()");
        goto err;
    case 0:
        /* child: give up privs and start a shell */

        /* set uid and groups */
        if (initgroups(pwd->pw_name, pwd->pw_gid) == -1) {
            warn("initgroups()");
```

```
            _exit(1);
        }
        if (setgid(pwd->pw_gid) == -1) {
            warn("setgid()");
            _exit(1);
        }
        if (setuid(pwd->pw_uid) == -1) {
            warn("setuid()");
            _exit(1);
        }
        execve(*args, args, environ);
        warn("execve()");
        _exit(1);
    default:
        /* parent: wait for child to exit */
        waitpid(pid, &status, 0);

        /* close the session and release PAM resources */
        pam_err = pam_close_session(pamh, 0);
        pam_end(pamh, pam_err);

        exit(WEXITSTATUS(status));
    }

pamerr:
    fprintf(stderr, "Sorry\n");
err:
    pam_end(pamh, pam_err);
    exit(1);
}
```

Apéndice B: Ejemplo de módulo PAM

Lo siguiente es una implementación mínima de pam_unix(8), que sólo ofrece servicios de autenticación. Debería compilar con la mayoría de implementaciones de PAM, pero aprovecha las extensiones de OpenPAM si están disponibles: fíjate en el uso de pam_get_authtok(3), que simplifica enormemente preguntar por la contraseña de usuario.

```
/*-
 * Copyright (c) 2002 Networks Associates Technology, Inc.
 * All rights reserved.
 *
 * This software was developed for the FreeBSD Project by ThinkSec AS and
 * Network Associates Laboratories, the Security Research Division of
 * Network Associates, Inc.  under DARPA/SPAWAR contract N66001-01-C-8035
 * ("CBOSS"), as part of the DARPA CHATS research program.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
```

```
 * are met:
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in the
 *    documentation and/or other materials provided with the distribution.
 * 3. The name of the author may not be used to endorse or promote
 *    products derived from this software without specific prior written
 *    permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE AUTHOR AND CONTRIBUTORS ``AS IS'' AND
 * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED.  IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE LIABLE
 * FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
 * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
 * OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
 * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
 * OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
 * SUCH DAMAGE.
 *
 * $P4: //depot/projects/openpam/modules/pam_unix/pam_unix.c#3 $
 * $FreeBSD: head/en_US.ISO8859-1/articles/pam/pam_unix.c 38826 2012-05-17 19:12:14Z
hrs $
 */

#include <sys/param.h>

#include <pwd.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <unistd.h>

#include <security/pam_modules.h>
#include <security/pam_appl.h>

#ifndef _OPENPAM
static char password_prompt[] = "Password:";
#endif

#ifndef PAM_EXTERN
#define PAM_EXTERN
#endif

PAM_EXTERN int
pam_sm_authenticate(pam_handle_t *pamh, int flags,
    int argc, const char *argv[])
{
#ifndef _OPENPAM
```

```
    struct pam_conv *conv;
    struct pam_message msg;
    const struct pam_message *msgp;
    struct pam_response *resp;
#endif
    struct passwd *pwd;
    const char *user;
    char *crypt_password, *password;
    int pam_err, retry;

    /* identify user */
    if ((pam_err = pam_get_user(pamh, &user, NULL)) != PAM_SUCCESS)
        return (pam_err);
    if ((pwd = getpwnam(user)) == NULL)
        return (PAM_USER_UNKNOWN);

    /* get password */
#ifndef _OPENPAM
    pam_err = pam_get_item(pamh, PAM_CONV, (const void **)&conv);
    if (pam_err != PAM_SUCCESS)
        return (PAM_SYSTEM_ERR);
    msg.msg_style = PAM_PROMPT_ECHO_OFF;
    msg.msg = password_prompt;
    msgp = &msg;
#endif
    for (retry = 0; retry < 3; ++retry) {
#ifdef _OPENPAM
        pam_err = pam_get_authtok(pamh, PAM_AUTHTOK,
            (const char **)&password, NULL);
#else
        resp = NULL;
        pam_err = (*conv->conv)(1, &msgp, &resp, conv->appdata_ptr);
        if (resp != NULL) {
            if (pam_err == PAM_SUCCESS)
                password = resp->resp;
            else
                free(resp->resp);
            free(resp);
        }
#endif
        if (pam_err == PAM_SUCCESS)
            break;
    }
    if (pam_err == PAM_CONV_ERR)
        return (pam_err);
    if (pam_err != PAM_SUCCESS)
        return (PAM_AUTH_ERR);

    /* compare passwords */
    if ((!pwd->pw_passwd[0] && (flags & PAM_DISALLOW_NULL_AUTHTOK)) ||
        (crypt_password = crypt(password, pwd->pw_passwd)) == NULL ||
```

```
        strcmp(crypt_password, pwd->pw_passwd) != 0)
        pam_err = PAM_AUTH_ERR;
    else
        pam_err = PAM_SUCCESS;
#ifndef _OPENPAM
    free(password);
#endif
    return (pam_err);
}

PAM_EXTERN int
pam_sm_setcred(pam_handle_t *pamh, int flags,
    int argc, const char *argv[])
{

    return (PAM_SUCCESS);
}

PAM_EXTERN int
pam_sm_acct_mgmt(pam_handle_t *pamh, int flags,
    int argc, const char *argv[])
{

    return (PAM_SUCCESS);
}

PAM_EXTERN int
pam_sm_open_session(pam_handle_t *pamh, int flags,
    int argc, const char *argv[])
{

    return (PAM_SUCCESS);
}

PAM_EXTERN int
pam_sm_close_session(pam_handle_t *pamh, int flags,
    int argc, const char *argv[])
{

    return (PAM_SUCCESS);
}

PAM_EXTERN int
pam_sm_chauthtok(pam_handle_t *pamh, int flags,
    int argc, const char *argv[])
{

    return (PAM_SERVICE_ERR);
}

#ifdef PAM_MODULE_ENTRY
```

```
PAM_MODULE_ENTRY("pam_unix");
#endif
```

Apéndice C: Ejemplo de función de conversación PAM

La función de conversación que se presenta abajo es una versión muy simplificada de la función openpam_ttyconv(3) de OpenPAM. Es completamente funcional y debería darle al lector una buena idea de cómo se debería de comportar una función de conversación, pero es de lejos demasiado simple como para usarla en el mundo real. Incluso si no usas OpenPAM, siéntete libre para descargarte el código fuente y adaptar openpam_ttyconv(3) a tus necesidades; creemos que es todo lo robusta que puede llegar a ser una función de conversación basada en tty.

```
/*-
 * Copyright (c) 2002 Networks Associates Technology, Inc.
 * All rights reserved.
 *
 * This software was developed for the FreeBSD Project by ThinkSec AS and
 * Network Associates Laboratories, the Security Research Division of
 * Network Associates, Inc.  under DARPA/SPAWAR contract N66001-01-C-8035
 * ("CBOSS"), as part of the DARPA CHATS research program.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in the
 *    documentation and/or other materials provided with the distribution.
 * 3. The name of the author may not be used to endorse or promote
 *    products derived from this software without specific prior written
 *    permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE AUTHOR AND CONTRIBUTORS ``AS IS'' AND
 * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED.  IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE LIABLE
 * FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
 * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
 * OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
 * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
 * OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
 * SUCH DAMAGE.
 *
 * $FreeBSD: head/en_US.ISO8859-1/articles/pam/converse.c 38826 2012-05-17 19:12:14Z
```

```
hrs $
 */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>

#include <security/pam_appl.h>

int
converse(int n, const struct pam_message **msg,
    struct pam_response **resp, void *data)
{
    struct pam_response *aresp;
    char buf[PAM_MAX_RESP_SIZE];
    int i;

    data = data;
    if (n <= 0 || n > PAM_MAX_NUM_MSG)
        return (PAM_CONV_ERR);
    if ((aresp = calloc(n, sizeof *aresp)) == NULL)
        return (PAM_BUF_ERR);
    for (i = 0; i < n; ++i) {
        aresp[i].resp_retcode = 0;
        aresp[i].resp = NULL;
        switch (msg[i]->msg_style) {
        case PAM_PROMPT_ECHO_OFF:
            aresp[i].resp = strdup(getpass(msg[i]->msg));
            if (aresp[i].resp == NULL)
                goto fail;
            break;
        case PAM_PROMPT_ECHO_ON:
            fputs(msg[i]->msg, stderr);
            if (fgets(buf, sizeof buf, stdin) == NULL)
                goto fail;
            aresp[i].resp = strdup(buf);
            if (aresp[i].resp == NULL)
                goto fail;
            break;
        case PAM_ERROR_MSG:
            fputs(msg[i]->msg, stderr);
            if (strlen(msg[i]->msg) > 0 &&
                msg[i]->msg[strlen(msg[i]->msg) - 1] != '\n')
                fputc('\n', stderr);
            break;
        case PAM_TEXT_INFO:
            fputs(msg[i]->msg, stdout);
            if (strlen(msg[i]->msg) > 0 &&
                msg[i]->msg[strlen(msg[i]->msg) - 1] != '\n')
                fputc('\n', stdout);
```

```
            break;
        default:
            goto fail;
        }
    }
    *resp = aresp;
    return (PAM_SUCCESS);
 fail:
        for (i = 0; i < n; ++i) {
                if (aresp[i].resp != NULL) {
                        memset(aresp[i].resp, 0, strlen(aresp[i].resp));
                        free(aresp[i].resp);
                }
        }
        memset(aresp, 0, n * sizeof *aresp);
    *resp = NULL;
    return (PAM_CONV_ERR);
}
```

Lecturas adicionales

Artículos

Making Login Services Independent of Authentication Technologies Vipin Samar. Charlie Lai. Sun Microsystems.

X/Open Single Sign-on Preliminary Specification. The Open Group. 1-85912-144-6. June 1997.

Pluggable Authentication Modules. Andrew G. Morgan. 1999-10-06.

Manuales de usuario

PAM Administration. Sun Microsystems.

Páginas web relacionadas

OpenPAM homepage Dag-Erling Smørgrav. ThinkSec AS.

Linux-PAM homepage Andrew Morgan.

Página de PAM de Solaris. Sun Microsystems.