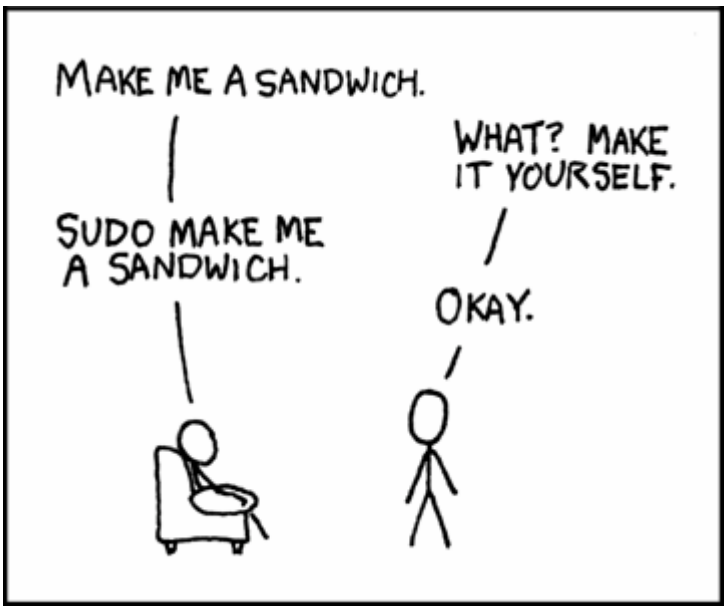
PAM

Pluggable authentication modules



Pau Conejero Alberola
Ingeniería Técnica en Informática de Sistemas – II55

ÍNDICE

Introducción……3

Arquitectura……4

Funcionamiento……7

Configuración……9

Desarrollo……11

Ejemplos configuración……13

Ejemplos desarrollo……17

Bibliografía……20

1 - Introducción PAM

¿Qué es PAM exactamente? PAM ofrece una capa intermedia entre el usuario y la aplicación en el momento de la autenticación. La capa intermedia está basada en un sistema modular para ofrecer diferentes funcionalidades. PAM es básicamente un mecanismo flexible para la autenticación de usuarios en sistemas UNIX. Posiblemente la idea principal del mecanismo es aportar flexibilidad en las tareas de autenticación. PAM no es una aplicación o protocolo, sino una colección de bibliotecas que se pueden utilizar en aplicaciones compiladas. Es decir PAM ofrece la funcionalidad de enlazar bibliotecas dinámicas a las aplicaciones para dotarlas de mecanismos de autenticación y políticas de seguridad de una manera fácil, robusta y flexible.

Para poder entender el funcionamiento de PAM tenemos que recordar que una librería dinámica no se copia en nuestro programa al compilarlo. Su funcionamiento es el siguiente; en el momento de ejecución del ejecutable cada vez que el código necesite algo de librería irá a buscarla.

Históricamente los sistemas UNIX ofrecen autenticación basada en un password. El usuario escribe un password que es validado contra el fichero /etc/shadow, sin embargo actualmente existen diferentes métodos de autenticación para integrar en un sistema UNIX o en aplicaciones donde es necesario autenticar al usuario. Estos problemas se pueden solucionar de una manera elegante y segura mediante directrices PAM.

PAM ofrece la funcionalidad de gestionar políticas de seguridad en nuestro sistema y al mismo tiempo separa el proceso de autenticación en el desarrollo de aplicaciones, liberando al desarrollador de implementar sistemas de autenticación. Si una aplicación está habilitada para PAM, la directiva de seguridad de esta aplicación puede ser configurada por un administrador de sistemas sin modificar ni actualizar la aplicación. En resumen podrían sintetizarse las ventajas más importantes de PAM en los siguientes puntos:

- Ofrece un sistema de autenticación común y centralizada
- Permite a los desarrolladores abstraerse de las laboras de autenticación
- Facilita el mantenimiento de aplicaciones
- Ofrece flexibilidad y control tanto para el desarrollador como para el administrador
- Fácilmente migrable entre sistemas UNIX

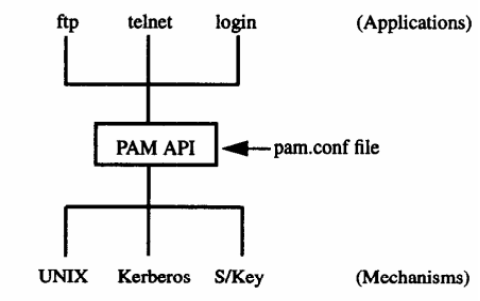
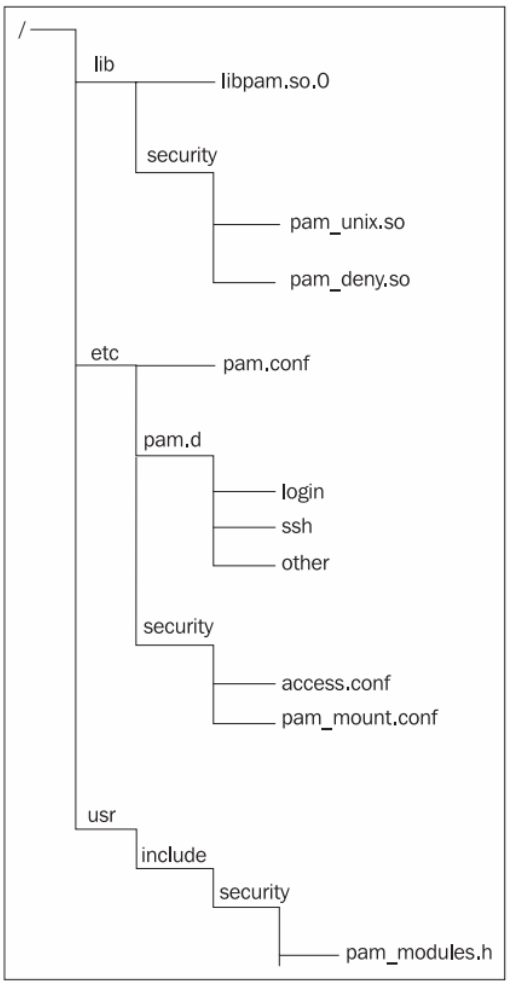


Figure 1: The Basic PAM Architecture

2 – Arquitectura PAM

Como hemos comentado anteriormente PAM está basado en un conjunto de librerías y su arquitectura funcional es modular. El núcleo central de PAM reside en la biblioteca (libpam) y una colección de módulos que se vinculan directamente con las bibliotecas. Cada módulo realiza una tarea específica, pudiendo unir diferentes módulos en una pila. En la siguiente imagen podemos observar la estructura de los ficheros que forman PAM.

- **/lib:** Directorio donde reside la librería principal
- **/lib/security:** Directorio donde residen los módulos dinámicos
- **/etc:** Directorio donde residen los archivos de configuración
- **/etc/security:** Directorio donde residen los archivos de configuración de funcionalidades extra
- **/usr/incluye/security:** Directorio donde residen los headers para el desarrollo de aplicaciones



El concepto de PAM consta de:

Módulos PAM: Conjunto de librerías dinámicas para autenticaciones específicas

Pila de módulos: Pila con diferentes módulos, para poder enlazar funcionalidades en serie

PAM-AWARE: Servicio que necesita autenticación utilizando la pila de módulos

Configuración: Archivos de configuración de las aplicaciones que utilizan PAM

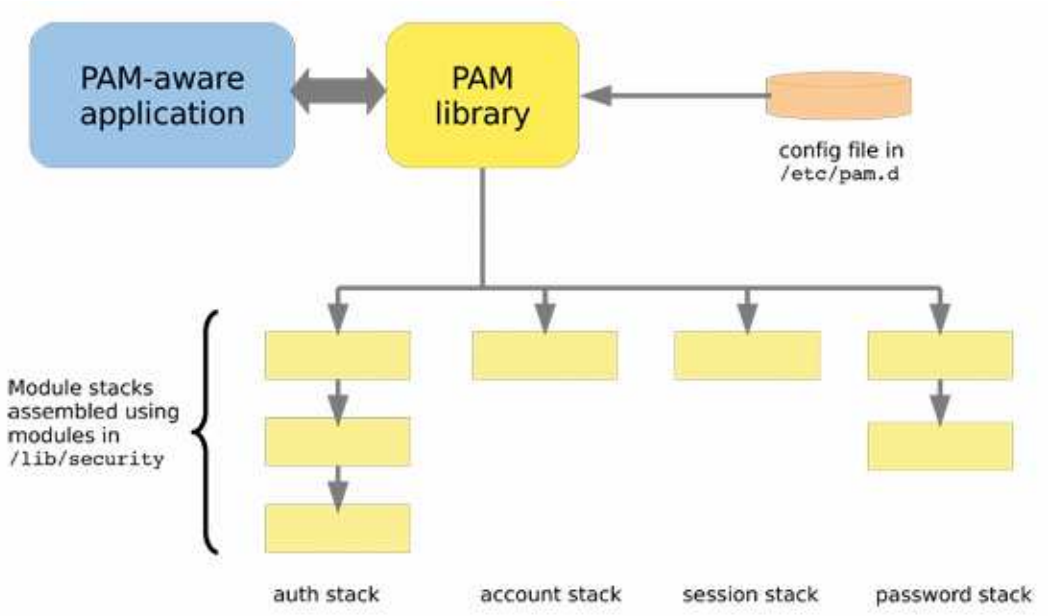
Argumentos: Parámetros de configuración de los módulos

Evaluación: Mecanismo para evaluar las salidas de los diferentes módulos

La misión de PAM no es únicamente comprobar que un usuario pueda autenticarse en un sistema UNIX. Su alcance es mucho mayor y pueden dividirse sus tareas en cuatro grupos independientes de gestión, cada uno de los cuales se encarga de un aspecto diferente de los servicios restringidos

El tipo de módulo, o tarea, es el grupo de gestión que la norma se corresponde con él. Cualquier módulo PAM puede soportar los cuatro tipos diferentes:

- **Auth stack**
- **Account stack**
- **Session stack**
- **Password stack**



- **Authentication:** Autentica el usuario, y luego otorga los privilegios correspondientes

Utilizar diferentes métodos de autenticación (Clauer - Pass - RSA ...)

- **Account:** Provee verificación de tipos de servicio para la cuenta

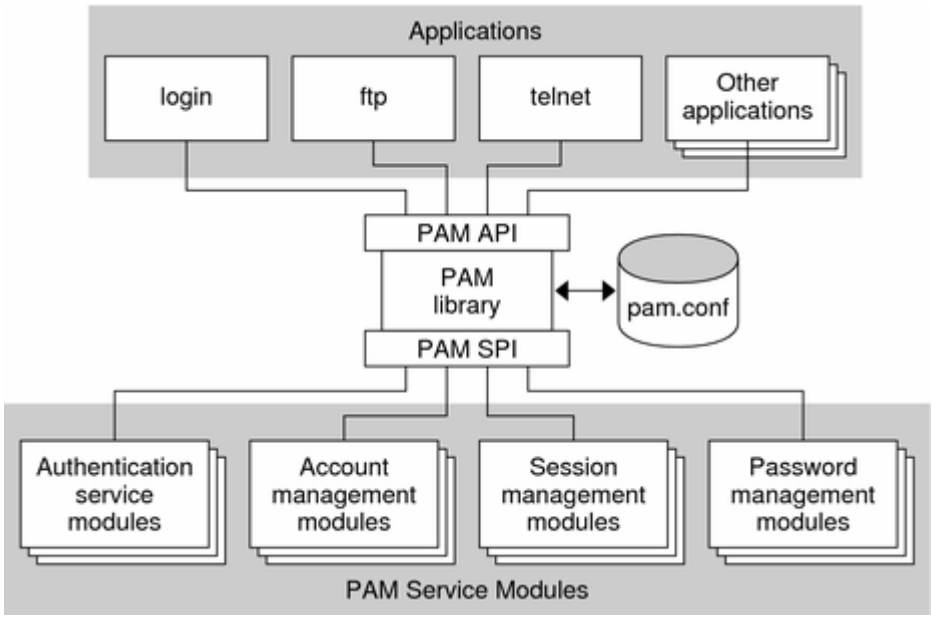
Permitir acceso en función tiempo/locacización - Acceso recursos - Verificación - Expiración pass

- **Password:** Responsable de actualizar los mecanismos de autenticación

Fortaleza password - Longitudad Min/Max

- **Session:** Acciones que se deben hacer antes de que se otorgue un servicio y luego de que el servicio haya sido retirado.

Registro accesos - Montaje directorios

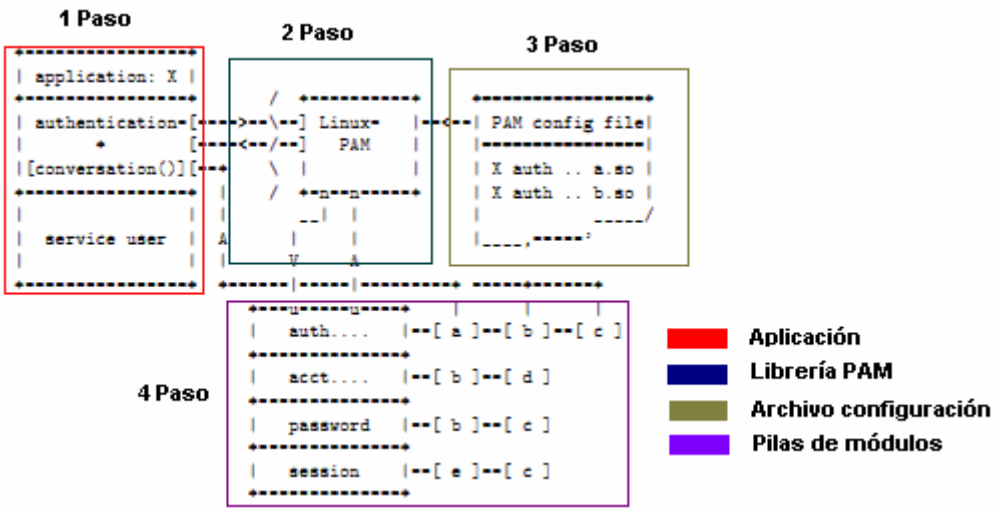


Una vez hemos identificado los objetos necesarios para el funcionamiento de PAM, podemos empezar a preguntarnos cómo funciona. En el siguiente apartado estudiaremos el comportamiento de una aplicación con soporte para PAM. Como hemos comentado anteriormente tenemos que recordar que las librerías PAM no están compiladas en la aplicación, ella misma es la encargada de buscar las librerías necesarias para poder funcionar.

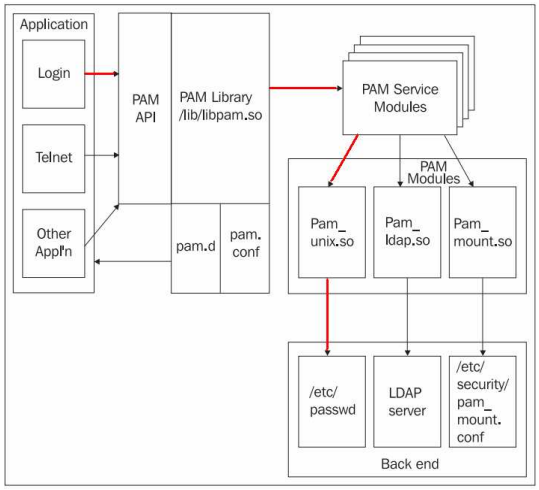
3 – Funcionamiento PAM

En el siguiente ejemplo pondremos en práctica todos los objetos que intervienen en el proceso que PAM realiza para poder gestionar las funcionalidades de seguridad de la aplicación. La pregunta que nos realizamos es la siguiente; ¿Cómo se organizan estas ideas?

Una aplicación X quiere utilizar las funcionalidades PAM, para ello interactúa con las librerías sin tener que conocer ningún detalle de cómo está configurado el sistema para la aplicación. La biblioteca será la que se encargue de leer la configuración de PAM para conocer la política de autenticación que hay que aplicar. Los módulos se colocan en una pila según el grupo de gestión y el orden en el que aparecen en la configuración. Finalmente PAM ofrece a la aplicación una serie de funciones para llevar a cabo las diferentes tareas de cada grupo, mientras que la aplicación brinda a PAM una función de conversación destinada a intercambiar información textual con el usuario. Mediante la función de conversación PAM, se libera de tener que preocuparse de cómo enviar/recibir información del cliente.

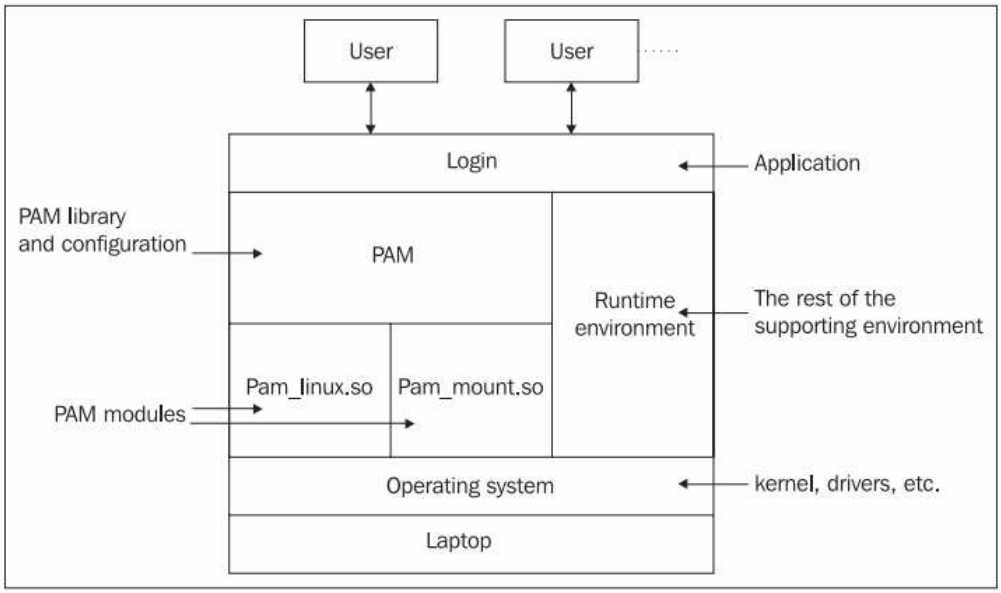


En el anterior ejemplo hemos observado el funcionamiento global de una aplicación mediante PAM. Una vez tenemos una idea global de cómo interactúan todos los elementos podemos profundizar con la aplicación login que ofrecen todos los sistemas UNIX.



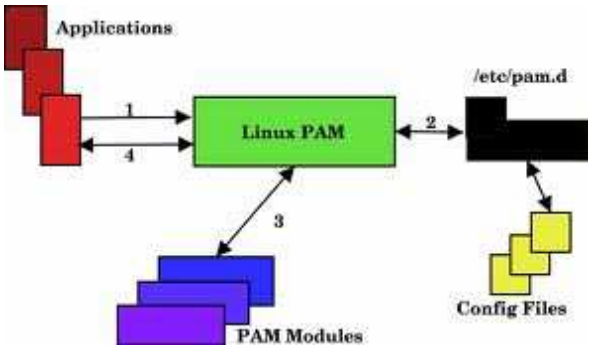
La imagen ilustra el proceso de autenticación de la aplicación login. En primer lugar, la aplicación interactúa con la API de PAM que nos proporciona el interfaz necesario para poder comunicarnos con la librería principal (libpam), y ésta lee la configuración PAM para conocer la política de autenticación. En segundo lugar, utilizaremos el servicio de AUTHENTICATION que ofrece PAM para poder validar los usuarios en el sistema. El módulo genérico para la autenticación de usuarios recibe el nombre de pam_unix.so. Este módulo es el encargado de validar la información suministrada por el usuario contra el fichero /etc/passwd, en caso de existir /etc/shadow el módulo atacará este fichero.

En la siguiente imagen podemos observar el mismo proceso salvo la diferencia de que ahora los módulos que se utilizarán son diferentes, aportando nuevas funcionalidades.



Para poder obtener información sobre si una aplicación soporta PAM, podemos observar las librerías dinámicas que tiene enlazadas. En este caso podemos observar que efectivamente tiene dependencias con PAM.

```
$ ldd /bin/login | grep libpam
        libpam.so.0 => /lib/libpam.so.0 (0xb7f47000)
        libpam_misc.so.0 => /lib/libpam_misc.so.0 (0xb7f43000)
```



Comunicación entre los objetos que forman PAM

4 – Configuración PAM

Los ficheros de configuración de PAM son utilizados para configurar módulos de servicio PAM para ofrecer servicios del sistema. Podemos dividir los archivos en dos categorías:

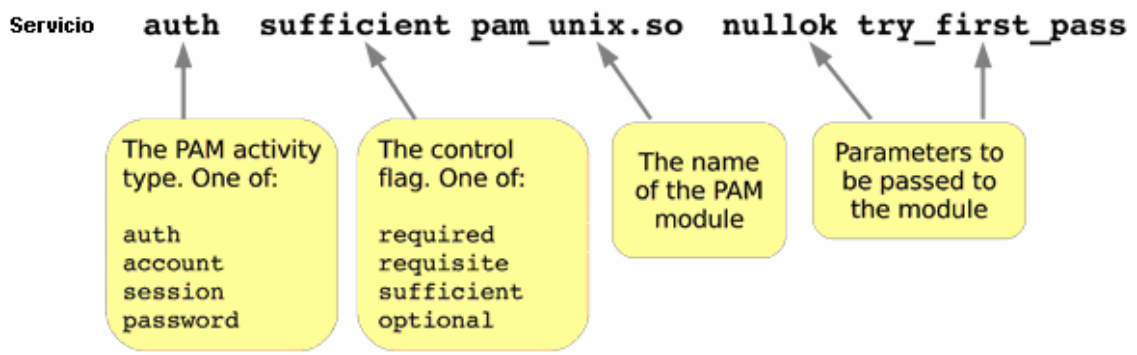
- Fichero genérico
- Fichero específico de un servicio

Actualmente la política a seguir es la separación de cada servicio en un fichero de configuración propio, aplicamos la técnica “divide y vencerás”.

Los ficheros de configuración PAM están compuestos por una serie de reglas a aplicar. Todas las reglas tienen una sintaxis común.

```
service-type    module-type     control         module-path     arguments
```

- **Servicio:** Nombre del servicio que utilizará políticas PAM
- **Tipo:** Grupo de administración donde se aplica la regla
- **Control:** Comportamiento que PAM debe seguir en caso de que el módulo asociado tengo éxito o falle. En este campo las reglas se puede apilar
- **Módulo:** Nombre del módulo específico
- **Argumentos:** Argumentos que se envían al módulo



En el apartado de control podemos diferenciar dos alternativas a seguir:

- Notación simple

Binding	Include	Optional
Required	Requisite	Sufficient

- Notación extendida

Ignore	Bad	Die
Ok	Done	Reset

La sintaxis más simple consiste en una sola palabra clave, mientras que la otra (más precisa y compleja) implica el uso de corchetes y parejas valor-acción. En este apartado estudiaremos la notación simple. Este campo puede tomar únicamente seis valores diferentes:

- **Required:** Indica que es necesario que el módulo tenga éxito para que la pila también lo tenga. Si se produce un fallo, no se notifica hasta que se procesa el resto de la pila.

- **Requisite:** Indica que es necesario que el módulo tenga éxito para que la pila también lo tenga. Si se produce un fallo, el control se devuelve inmediatamente a la aplicación.

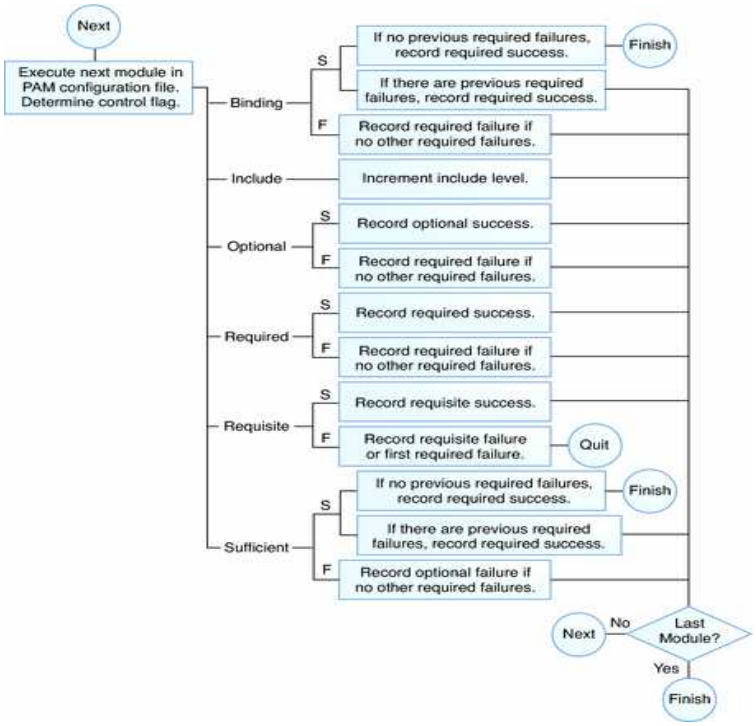
- **Sufficient:** El éxito en este módulo, si no se ha producido un fallo en los procesados anteriormente en la pila es "suficiente". El procesamiento se detiene, incluso ignorando posibles required.

- **Optional:** PAM ignora los módulos con este indicador. Su valor será tenido en cuenta sólo en caso de que no se haya llegado a ningún valor concreto de éxito o fracaso.

- **Include**: Agrega líneas de un archivo de configuración de PAM independiente que se utilizará en este momento en la pila PAM. Este indicador no controla el comportamiento de éxito o error.

- **Binding**: El éxito en el cumplimiento de los requisitos devuelve inmediatamente un valor de éxito a la aplicación si no ha fallado ningún módulo required anterior.

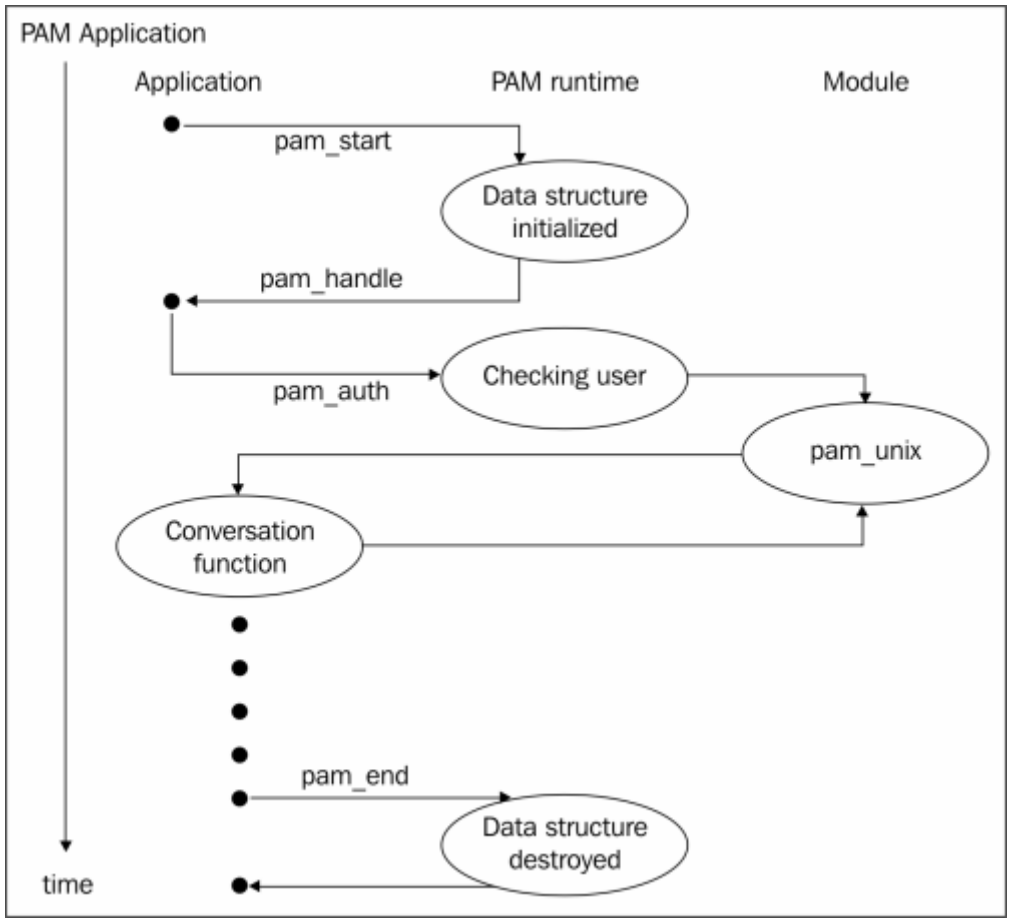
En el siguiente ejemplo se puede observar el funcionamiento de la pila de módulos que incorpora PAM. El primer diagrama indica cómo se registra el éxito o error para cada tipo de indicador de control.



5 – Desarrollo PAM

Llegados a este punto ya tenemos una idea general de las funcionalidades que ofrece PAM. En este apartado trataremos la integración de PAM en aplicaciones diseñadas por el desarrollador, como hemos comentando anteriormente, una de las funcionalidades que ofrece PAM es liberar al programador de realizar un sistema de autenticación. En la siguiente imagen podemos observar los pasos genéricos que son necesarios para que la aplicación pueda disponer de las funcionalidades que queramos integrar.

1. Iniciación PAM mediante la directiva ***"pam_start"***
2. Creación de la estructura PAM para albergar información ***"pam_handle"***
3. Llamada inicial autenticación PAM mediante la directiva ***"pam_auth"***
4. Función de conversación entre usuario y PAM
5. Directivas de seguridad
6. Finalización PAM mediante la directiva ***"pam_end"***




```
PAM_EXTERN int pam_sm_FUNC(pam_handle_t *pamh, int flags,
int argc, const char **argv)
```

En el siguiente ejemplo podemos observar el funcionamiento del comportamiento anteriormente descrito en una aplicación. En este caso hemos utilizado la aplicación "su" para comprobar si un usuario puede autenticarse para cambiarse de usuario.

```c
#include <security/pam_appl.h>
#include <security/pam_misc.h>
#include <stdio.h>

int main ()
{
  pam_handle_t* pamh;
  struct pam_conv pamc;

  /* Set up the PAM conversation.  */
  pamc.conv = &misc_conv;
  pamc.appdata_ptr = NULL;
  /* Start a new authentication session.  */
  pam_start ("su", getenv ("USER"), &pamc, &pamh);
  /* Authenticate the user.  */
  if (pam_authenticate (pamh, 0) != PAM_SUCCESS)
    fprintf (stderr, "Authentication failed!\n");
  else
    fprintf (stderr, "Authentication OK.\n");
  /* All done.  */
  pam_end (pamh, 0);
  return 0;
}
```

Librerías necesarias PAM

Estructuras necesarias PAM

Función interactuar usuario

Iniciar PAM

Aplicación "su"

Usuario defecto

Autenticación

Finalizar PAM

pam_start - initiate a PAM transaction

```c
#include <security/pam_appl.h>

int pam_start (
     const char                 *service,
     const har                  *user,
     const struct pam_conv      *pam_conv,
     pam_handle_t              **pamh
);
```

pam_authenticate - perform authentication within the PAM framework

```c
#include <security/pam_appl.h>

int pam_authenticate (
     pam_handle_t     *pamh,
     int               flags
);
```

pam_end - terminates the PAM transaction

```c
#include <security/pam_appl.h>

int pam_end (
     pam_handle_t     *pamh,
     int               status,
);
```

6 – Ejemplos Configuración

- **Comando "su"**

Permitir la obtención de privilegios root sin introducir ningún password. Para poder aplicar la siguiente regla es necesario modificar el fichero de configuración /etc/pam.d/su.

```
auth    sufficient    pam_permit.so
```

- **Deshabilitar comando "su"**

Acotar la entrada del root mediante la restricción de utilización del comando "su". Para poder aplicar la siguiente regla es necesario modificar el fichero de configuración /etc/pam.d/su.

```
auth     requisite    pam_deny.so
```

- Apagado sistema

Restringir el apagado del sistema. Las siguientes reglas restringen las operaciones en el funcionamiento del sistema, por una parte el usuario no tiene sufrientes privilegios y el root puede realizar las tareas sin realizar una autenticación previa. Para poder aplicar las siguientes reglas es necesario modificar los ficheros de configuración /etc/pam.d/halt y /etc/pam.d /shutdown.

```
auth       sufficient   pam_rootok.so
auth       required     pam_deny.so
```

Una configuración alternativa podría ser la necesidad de realizar una autenticación en el momento de apagar la máquina por parte del administrador. Podríamos pensar en sistemas críticos en los que el apagado y el reinicio de las maquinas requieran una confirmación interactiva. Para poder exportar la siguiente configuración al reiniciar es necesario modificar el fichero /etc/pam.d/reboot.

- Acceso root

Forzar la entrada root mediante el comando "su". La siguiente regla restringe el acceso directo del administrador. Para poder obtener privilegios previamente tenemos que autenticarnos en el sistema mediante un rol de usuario. Para poder aplicar las siguientes reglas es necesario modificar el fichero de configuración /etc/pam.d/su.

```
auth    required    pam_securetty.so
```

Eliminar la configuración de terminales seguras para el acceso root

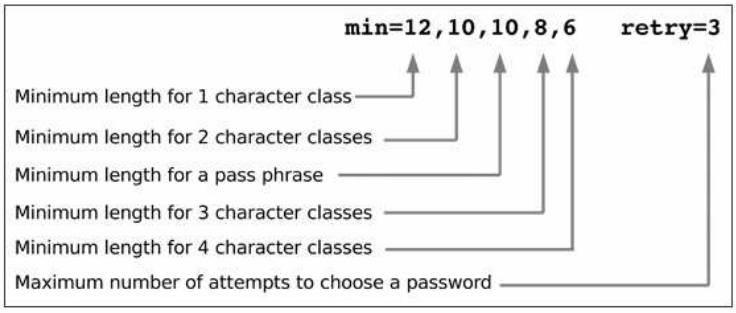
```
# cp /etc/securetty /etc/securetty.saved
# echo "" > /etc/securetty
```

- **Fortaleza password**

Módulos PAM:	libpam-cracklib - libpam-passwdqc

Posiblemente el talón de Aquiles de un sistema seguro es la interacción con el usuario final. La mayoría de sistemas informáticos utilizan el método login-password para realizar el proceso de autenticación. Surge la necesidad de establecer una política de contraseñas para limitar el impacto del usuario. Para poder aplicar la siguiente regla es necesario modificar el fichero /etc/pam.d/passwd. Podemos especificar los caracteres mínimos de cada clase. Existen 4 tipos de clases: Minúsculas – Mayúsculas – Dígitos – Símbolos. Por ejemplo, la clase 4 requiere un mínimo de 6 caracteres formados por (Minúsculas – Mayúsculas – Dígitos - Símbolos).

```
password    requisite    pam_passwdqc.so  min=12,10,10,8,6 retry=3
```



Podemos aumentar la seguridad de la entrada al sistema añadiendo una regla para limitar el tiempo transcurrido en cada intento de autenticacion. Para ello podemos utilizar la librería pam_cracklib.

```
# vi /etc/common-auth
auth required pam_tally.so onerr=fail deny=3 unlock_time=3600
```

Una vez tengamos la configuración adecuada el siguiente paso es informar al usuario de la política de contraseñas del sistema para poder generar passwords que cumplan los restricciones marcadas. Para ello utilizaremos el módulo “pam_echo.so” que viene incorporado por defecto en PAM. Editando el fichero /etc/pam.d/passwd podemos incorporar la siguiente línea para mostrar información al usuario en el momento de actualizar el password.

```
password optional pam_echo.so file=/usr/share/doc/good-password.txt
```

- **Autenticación doble factor password**

Módulos PAM:	libpam-cracklib - libpam-passwdqc

Habilitar un proceso de autenticación basado en doble factor mediante la introducción de 2 contraseñas diferentes. Podemos pensar en sistemas que necesitan una doble confirmación. Para poder aplicar la siguiente regla es necesario editar el fichero /etc/pam.d/login y crear un segundo archivo para contener las contraseñas de la segunda autenticación.

- Fichero configuración PAM

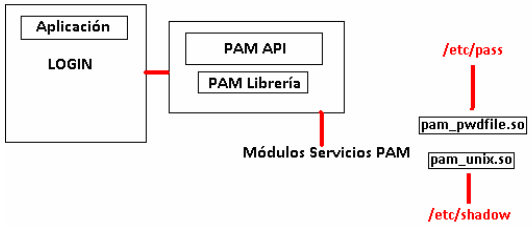
auth required pam_unix.so
auth required pam_pwdfile.so pwdfile /etc/pass

- Fichero 2 contraseña

```
$ openssl passwd -1 <password>
$ openssl passwd -crypt <password>
$ htpasswd -nbd <username> <password>
```

```
maato:$1$yII45doY$ZSH/biluHy12zp9KOuEQ9/
frozencow:SiN2d/00c/ubI
```

El resultado de la operación se almacenera en el fichero /etc/pass, por lo tanto la entrada al sistema quedara de la siguiente manera:



En esta configuración es necesaria la validación de las 2 contraseñas para autenticar la entrada

- **Autenticación doble factor token físico**

Módulos PAM:	libpam-usb- libpam-blue

Habilitar un proceso de autenticación basado en doble factor mediante la aportación de 2 tokens físicos. Podemos pensar en sistemas que necesitan una doble confirmación basada en dispositivos físicos. Para poder aplicar la siguiente regla es necesario editar el fichero /etc/pam.d/login y crear 2 archivos de configuración. El primer archivo contiene información relativa al dispositivo usb y el segundo archivo contiene información relativa al dispositivo bluetooth.

- Fichero configuración PAM

auth required pam_blue.so
auth required pam_unix.so

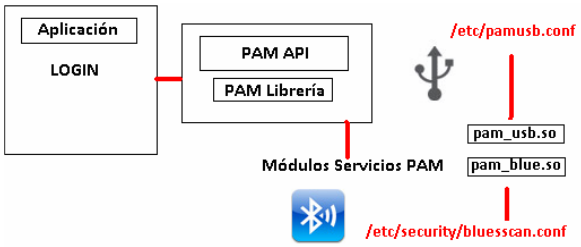
- Fichero de configuración Bluetooth - /etc/security/bluesscan.conf

```
USUARIO =
{
   # bluetooth device name
   name  = NombreDispositivo;

   # bluetooth mac address
   bluemac = XX:XX:XX:XX:XX:XX;

   # a seaparate timeout
   timeout = 10;
}
```

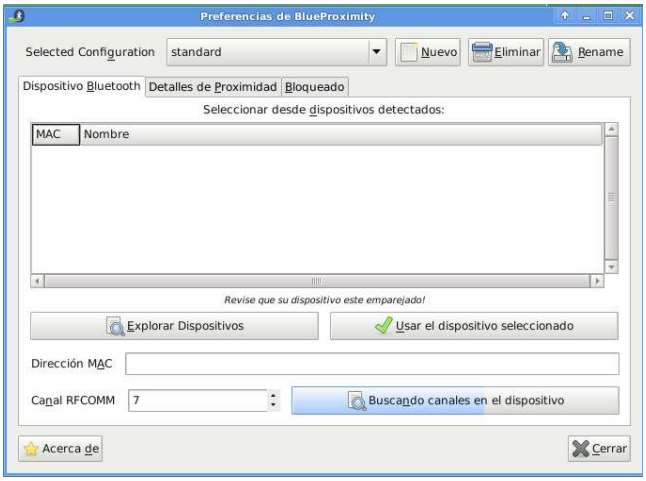
- Fichero de configuración dispositivo usb - /etc/pamusb.conf

```
<device id="USUARIO">
<vendor>
        Kingston
</vendor>
<model>
        DataTraveler 2.0
</model>
<serial>
        Kingston_DataTraveler_XX_XXXXXXXXXXXXXXXX
</serial>
<volume_uuid>
        XXX-XXX
</volume_uuid>
</device>
```

En la siguiente imagen podemos observar el funcionamiento de la doble autenticación mediante los dispositivos físicos. En este caso la autenticación en el sistema requiere que los 2 dispositivos tengan éxito para poder autenticarnos correctamente.



Blueproximity: Aplicación para detectar la proximidad de dispositivos Bluetooth. Podemos configurar diversas acciones según la proximidad de un dispositivo, detectar que el usuario se ha desplazado de su lugar de trabajo y bloquear la pantalla. Una vez el usuario regresa al lugar de trabajo automáticamente la aplicación detecta el cambio y desbloquea la pantalla.



7 – Ejemplos Desarrollo

En el siguiente ejemplo podemos observar la integración de PAM en una aplicación. El siguiente código realiza una serie de operaciones de seguridad antes de ejecutar la función encargada de la ejecución del código desarrollado para un propósito específico. Es decir antes de posicionarnos en la función "Run_My_Big_Application()" hemos tenido que chequear una serie de directrices. El éxito de la ejecución radica en el paso satisfactorio de las políticas de seguridad que incorpora la aplicación.

```
#include <security/pam_appl.h>
#include <security/pam_misc.h>
#include <pwd.h>
#include <sys/types.h>
#include <stdio.h>
#define MY_CONFIG "su"
static struct pam_conv conv = { misc_conv, NULL };

main( )
{
  pam_handle_t *pamh;
  int result;
  struct passwd *pw;
  if ((pw = getpwuid(getuid( ))) == NULL)
    perror("getpwuid");
  else if ((result = pam_start(MY_CONFIG, pw->pw_name, &conv, &pamh)) != PAM_SUCCESS)
    fprintf(stderr, "start failed: %d\n", result);
  else if ((result = pam_authenticate(pamh, 0)) != PAM_SUCCESS)
    fprintf(stderr, "authenticate failed: %d\n", result);
  else if ((result = pam_acct_mgmt(pamh, 0)) != PAM_SUCCESS)
    fprintf(stderr, "acct_mgmt failed: %d\n", result);
  else if ((result = pam_end(pamh, result)) != PAM_SUCCESS)
    fprintf(stderr, "end failed: %d\n", result);
  else
    Run_My_Big_Application( );            /* Run your application code */
}
```

```
struct passwd {
    char   *pw_name;      /* user name */
    char   *pw_passwd;    /* user password */
    uid_t  pw_uid;        /* user id */
    gid_t  pw_gid;        /* group id */
    char   *pw_gecos;     /* real name */
    char   *pw_dir;       /* home directory */
    char   *pw_shell;     /* shell program */
};
```

API PAM	
getpwuid()	Retornar un puntero a la estructura passwd
pam_start()	Iniciar estructura y transacciones PAM
pam_authenticate()	Autenticar al usuario
pam_acct_mgmt()	Determinar cuentas validas usuario
pam_end())	Terminar transacción PAM

El ejemplo anterior tiene como finalidad:

- Autenticar al usuario que ejecutará la aplicación
- Chequear permisos de utilización de la aplicación
- Chequear la validez de la cuenta del usuario autenticado
- Ejecutar el código específico de la aplicación

Por ejemplo con la función pam_acct_mgmt() podríamos limitar el usuario a utilizar la aplicación solamente los domingos por la mañana.

En el siguiente código podemos observar la estructura de un módulo desarrollado para PAM

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <security/pam_appl.h>
#include <security/pam_modules.h>

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)
{
        printf("Acct mgmt\n");
        return PAM_SUCCESS;
}

PAM_EXTERN int pam_sm_authenticate( pam_handle_t *pamh, int flags,int argc, const char **argv )
 {
        int retval;

        const char* pUsername;
        retval = pam_get_user(pamh, &pUsername, "Username: ");

        printf("Welcome %s\n", pUsername);

        if (retval != PAM_SUCCESS) {
                return retval;
        }

        if (strcmp(pUsername, "Konejero") != 0) {
                return PAM_AUTH_ERR;
        }

        return PAM_SUCCESS;
}
```

Estructuras Utilizadas

```
int pam_get_user (
    pam_handle_t    *pamh,
    char           **user,
    const char      *prompt
);
```

```
int pam_sm_acct_mgmt (
    pam_handle_t    *pamh,
    int              flags,
    int              argc,
    const char     **argv
);
```

```
int pam_sm_setcred(
    pam_handle_t    *pamh,
    int              flags,
    int              argc,
    const char     **argv
);
```

```
int pam_authenticate (
    pam_handle_t    *pamh,
    int              flags
);
```

La función del módulo PAM es incorporar un mecanismo para permitir el acceso al sistema de un usuario introduciendo simplemente el nombre de usuario. La función principal recibe el nombre de pam_sm_authenticate() que es la encargada de contrastar el usuario introducido con el valor del código. Un ejemplo real del módulo lo podríamos aplicar a la aplicación ssh. Configurando los parámetros necesarios en el fichero de configuración podemos incorporar este mecanismo.

Para poder utilizar el módulo en una aplicación PAM hay que seguir los siguientes pasos:

- Compilar módulo PAM

```
$ gcc -fPIC -c pam_example.c
$ ld -x --shared -o pam_example.so pam_example.o
$ sudo cp  pam_example.o  /lib/security
```

- Configurar fichero configuración PAM ssh

Editar el fichero de configuración /etc/pam.d/sshd introduciendo la siguiente línea

```
auth sufficient mypam.so
```

- Alternativamente podemos especificar el módulo de manera estática

```
struct pam_module _pam_deny_modstruct = {
    "pam_deny",
    NULL,
    NULL,
    NULL,
    pam_sm_open_session,
    pam_sm_close_session,
    NULL
};
```

8 – Bibliografía

- **Packt.Publishing.Pluggable.Authentication.Modules.Dec.2006** (Kenneth Geisshirt)
- **The Linux-PAM Module Writers' Guide** (Andrew G. Morgan, Thorsten Kukuk)
- **The Linux-PAM System Administrators' Guide** (Andrew G. Morgan, Thorsten Kukuk)
- **Advanced Linux Programming** (Mark Mitchell, Jeffrey Oldham, Alex Samuel)
- **RFC PAM - http://www.kernel.org/pub/linux/libs/pam/pre/doc/rfc86.0.txt.gz**
- **Documentación MAN**
- **Documentación Módulos PAM**