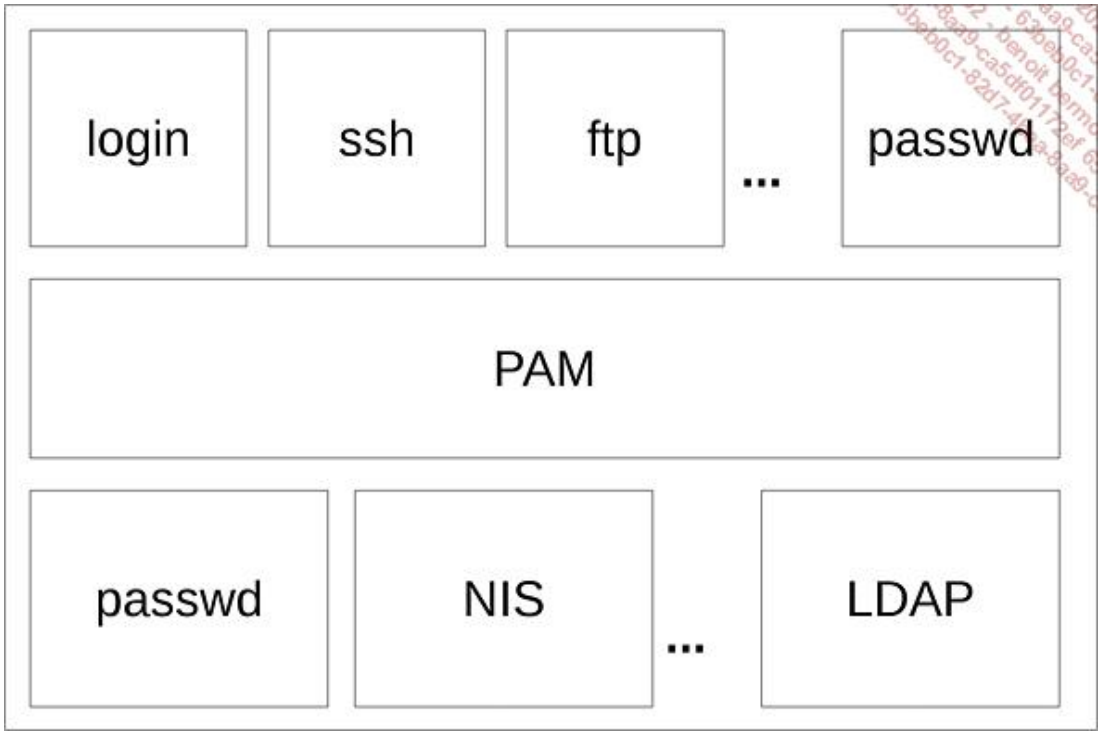
PAM

1. Le principe

PAM se positionne en interface entre les applications et les méthodes d'authentification.



Le principal objectif de PAM est de proposer une couche d'abstraction entre les applications et les méthodes d'authentification. Ainsi, une application qui se veut souple et évolutive quant aux méthodes d'authentification qu'elle emploie n'aura d'autre besoin que d'être compatible avec PAM. Cela signifie qu'elle devra être capable de s'adresser à la couche d'authentification PAM, et le reste ne la regarde pas. En parallèle, les procédés d'authentification quels qu'ils soient, doivent être exploitables par la mécanique PAM.

Une application demande à PAM si un utilisateur peut se connecter. PAM en fonction de sa configuration, appelle des modules fonctionnels qui vont exploiter une méthode d'authentification. Si le résultat est positif (l'utilisateur a fourni les bons éléments d'authentification), PAM renvoie l'autorisation de connexion à l'application.

PAM a un autre avantage. Nous venons de voir que la demande d'authentification entrainait le chargement de modules. Il se trouve que le nombre de ces modules n'est pas limité et qu'ils peuvent être cumulés. Il est donc tout à fait possible de demander une double authentification selon deux méthodes différentes. De plus, on peut profiter de la séquence d'authentification sous PAM pour provoquer le chargement de bibliothèques sans rapport avec l'authentification. De nombreuses actions peuvent donc être gérées dès l'authentification réussie.

En résumé : lors de la demande d'authentification, des modules PAM sont chargés en fonction d'un fichier de configuration, et ces modules provoquent certaines actions, relevant de l'authentification proprement dite ou d'autres actions.

2. Les modules PAM

a. Les principaux modules PAM

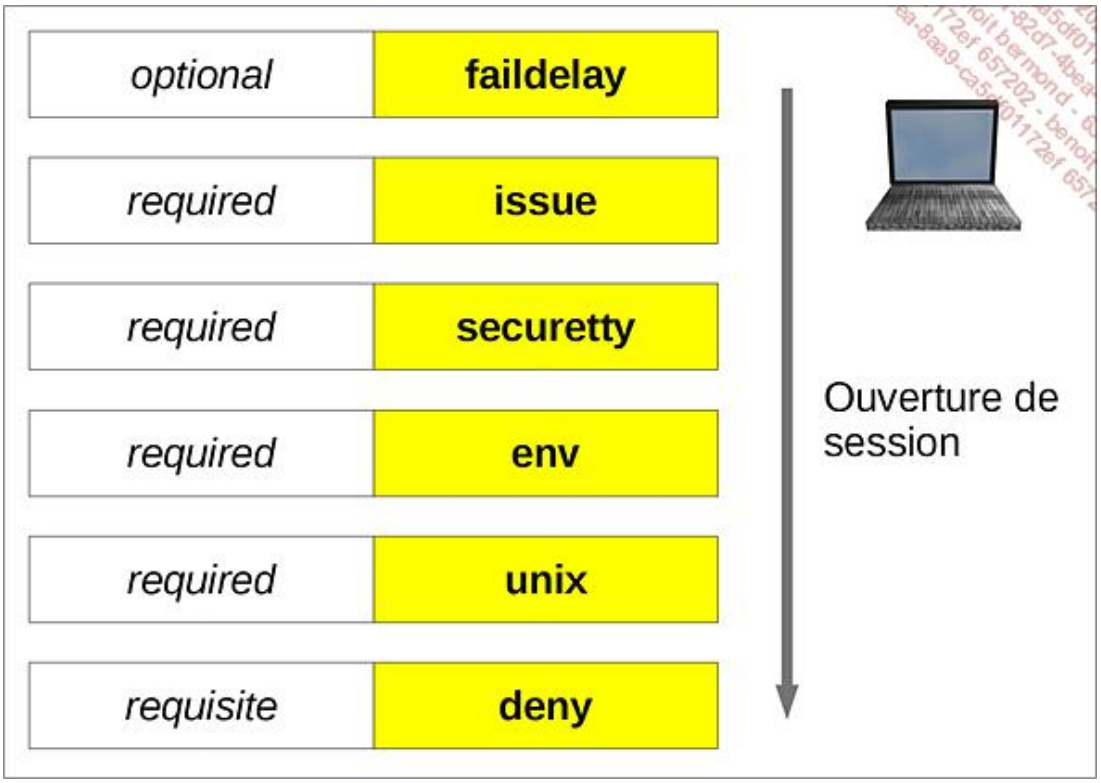
Les modules PAM, appelés lors des opérations d'authentification sont nombreux et d'usages variés. Certains d'entre eux sont néanmoins rencontrés très fréquemment et leur existence est à connaître. D'autres sont plus ou moins fréquents selon les distributions, mais connaître leur fonctionnement et leurs objectifs permet de mieux comprendre la mécanique et la philosophie de PAM.

Ces modules sont dans des fichiers dont l'emplacement normalisé est **/lib/security**.

Principaux modules PAM

pam_securetty.so	Interdit le login par le compte root excepté sur les terminaux listés dans /etc/securetty.
pam_nologin.so	Si le fichier /etc/nologin existe, affiche son contenu à toute tentative d'ouverture de session et interdit le login à tout autre que root.
pam_env.so	Déclare des variables d'environnement lues dans /etc/environnement ou dans le fichier donné en référence par le paramètre « envfile= ».
pam_unix.so	Permet l'authentification par la méthode traditionnelle des fichiers /etc/passwd et /etc/shadow.
pam_deny.so	Voie de garage. Est généralement exécuté si aucun autre module n'est exécuté avec succès.
pam_permit.so	Renvoie un retour positif inconditionnellement.
pam_limits.so	Affecte certaines limitations fonctionnelles à des utilisateurs ou des groupes en fonction des données du fichier /etc/security/limits.conf.
pam_cracklib.so	S'assure que le mot de passe employé présente un niveau de sécurité suffisant.
pam_selinux.so	Si selinux est activé sur le système, ce module va s'assurer que le shell sera bien exécuté dans le contexte de sécurité adéquat.
pam_lastlog.so	Affiche les informations sur la dernière ouverture de session réussie.
pam_mail.so	Vérifie la présence de nouveaux mails pour un utilisateur (messagerie interne).

b. Fonctionnement en piles de modules



Pour une action donnée, l'authentification par exemple, plusieurs modules PAM peuvent être appelés. On dit alors qu'on a affaire à une pile de module PAM. Le fonctionnement en pile est un des apports majeurs des services PAM.

3. Configuration de PAM

a. Structure des fichiers de configuration

Les premières versions de PAM trouvaient leur configuration dans un fichier **/etc/pam.conf**. La grande complexité de PAM a rapidement rendu nécessaire une structure plus modulaire pour les éléments de configuration. La quasi-totalité des implémentations actuelles exploite donc un répertoire **/etc/pam.d** contenant autant de fichiers que d'applications exploitant PAM. Si le répertoire **/etc/pam.d** existe, le fichier **/etc/pam.conf** n'est pas consulté.

Chaque application s'appuyant sur PAM aura besoin d'un fichier (en général du même nom que l'application) qui contiendra sa configuration PAM.

Format d'un fichier de /etc/pam.d

Le fichier contiendra autant de lignes qu'on souhaite appeler de modules avec pour chaque ligne la structure suivante :

type contrôle module arguments

Fichier de pam.d : format standard	
type	Représente le type d'action qui nécessite le recours à PAM. Les quatre valeurs possibles sont : **auth**, **account**, **password** et **session**.
contrôle	Indique comment le module doit réagir au succès ou à l'échec de son exécution. Les valeurs courantes sont **required**, **requisite**, **sufficient** et **optional**.
module	Le nom du module appelé. Le format normalisé est : **pam_*service*.so**. Où *service* représente le nom courant du module.
arguments	Paramètres optionnels envoyés au module pour modifier son fonctionnement.

Les valeurs possibles de type et de contrôle seront expliquées plus loin, mais nous avons déjà la possibilité de comprendre la structure du fichier de configuration. Dans l'extrait ci-dessous, on voit que la ligne ne concerne que les opérations d'authentification (auth), que l'exécution du module est obligatoire (required), que le module exploite la méthode d'authentification traditionnelle unix, c'est-à-dire les fichiers passwd et shadow (pam_unix.so), et enfin que ce module doit accepter une authentification faite avec un mot de passe vide (nullok). Notez que le paramètre nullok est spécifique au module, et que chaque module supportera tous les paramètres voulus par son développeur.

Extrait d'un fichier de configuration pam pour l'application login

Dans cet exemple, il est question d'authentification (auth), l'exécution du module est obligatoire (required), le module exploite le fichier des mots de passe historique (pam_unix.so). Enfin, l'utilisation d'un mot de passe vide est autorisée comme indiqué par l'argument (nullok).

```
auth required pam_unix.so  nullok
```

b. Les types d'action de PAM

Chaque ligne d'un fichier de configuration PAM doit commencer par l'un des quatre mots-clés qui détermine dans quel type d'action le module est compétent.

- auth : l'activité d'authentification proprement dite. Les modules appelés avec l'action auth sont exécutés pour ou pendant l'authentification.
- account : accès à des informations des comptes autres que les éléments d'authentifications proprement dits.
- session : actions à réaliser avant ou après l'ouverture de session.
- password : gestion des mots de passe.

Extrait du fichier de configuration pam pour l'application login

Cet exemple, extrait très allégé d'un fichier login standard illustre bien le concept de pile aussi bien que la nature modulaire de PAM.

On y trouve d'abord deux modules appelés pour l'authentification : pam_securetty profite de l'authentification pour vérifier que le compte n'est pas celui du superutilisateur, et pam_unix, qui réalise l'authentification proprement dite à partir du fichier /etc/passwd.

Le même module pam_unix est aussi déclaré sous le type account. Si des applications compatibles PAM ont besoin d'informations sur des comptes d'utilisateurs, elles auront besoin du module pam_unix sous le type account.

Le module pam_env est appelé sous le type session, cela assure son exécution (et donc la déclaration de variables) au sein de la session utilisateur.

Le module pam_cracklib est appelé sous le type password. Si une application de gestion de mots de passe compatible PAM souhaite modifier un mot de passe, elle devra en passer par le contrôle de complexité effectué par le module cracklib.

```
auth      required    pam_securetty.so
auth      required    pam_unix.so
account   required    pam_unix.so
session   required    pam_env.so readenv=1 envfile=/etc/default/locale
password  required    pam_cracklib.so retry=3 minlen=6
```

c. Les comportements des modules

Les modules vont être appelés avec un « control_flag » (marqueur de contrôle) qui va déterminer le comportement sur échec ou réussite du module.

Cet élément obligatoire est le deuxième champ sur la ligne de configuration.

- required : le module doit obligatoirement renvoyer un succès. Si un module d'authentification est required, son échec empêche l'ouverture de session. Les autres modules de la pile sont néanmoins exécutés.

- requisite : le module doit obligatoirement renvoyer un succès. Si un module d'authentification est requisite, son échec empêche le l'ouverture de session. Les autres modules de la pile ne sont pas exécutés.

- sufficient : si le module est exécuté avec succès et si aucun module required ou requisite n'a échoué, les autres modules de la pile sont ignorés.

- optional : le module peut réussir ou échouer sans influencer le reste de la pile. C'est-à-dire que si un module optional échoue, et qu'un module required de la même pile réussit, alors le résultat global de l'exécution de la pile est positif.

Exemples de fichiers de configuration PAM

Observons ici deux fichiers PAM, l'un gdm gérant l'ouverture de session graphique sous environnement Gnome, et l'autre gdm-autologin assurant l'ouverture automatique sans mot de passe de la session graphique. Les différences entre ces deux modes de fonctionnement portant sur l'authentification de l'utilisateur, nous ne nous intéresserons dans cet exemple qu'aux modules déclarés sous le type auth.

Les premiers modules chargés, pam_nologin et pam_env sont communs aux deux fichiers. Pour mémoire, pam_nologin interdit la connexion des utilisateurs ordinaires si le fichier /etc/nologin existe et a été renseigné par l'administrateur, et pam_env définit diverses variables au moment de l'authentification.

Le fichier gdm inclut ensuite le sous-fichier common-auth qui va appeler les éléments d'authentification voulus sur ce système (au minimum pam_unix pour l'authentification traditionnelle), puis charge le module pam_gnome_keyring qui permettra à des utilisateurs dûment authentifiés sous Gnome d'accéder à certaines fonctionnalités qui nécessiteraient normalement une réauthentification.

Le fichier gdm-autologin en revanche ne charge plus qu'un module : pam_permit qui renvoie un résultat positif dans tous les cas, dont l'exécution est obligatoire (le module est required), et qui va donc autoriser l'ouverture de session inconditionnellement.

Le fichier de configuration pam pour l'ouverture de session manuelle Gnome : gdm

```
auth    requisite       pam_nologin.so
auth    required        pam_env.so readenv=1
auth    required        pam_env.so readenv=1 envfile=/etc/default/locale
@include common-auth
auth    optional        pam_gnome_keyring.so
@include common-account
session [success=ok ignore=ignore module_unknown=ignore default=bad] pam_selinux.so close
session required        pam_limits.so
@include common-session
session [success=ok ignore=ignore module_unknown=ignore default=bad] pam_selinux.so open
session optional        pam_gnome_keyring.so auto_start
@include common-password
```

Le fichier de configuration pam pour l'ouverture de session automatique Gnome : gdm-autologin

```
auth    requisite       pam_nologin.so
auth    required        pam_env.so readenv=1
auth    required        pam_env.so readenv=1 envfile=/etc/default/locale
auth    required        pam_permit.so
@include common-account
session [success=ok ignore=ignore module_unknown=ignore default=bad] pam_selinux.so close
session required        pam_limits.so
@include common-session
session [success=ok ignore=ignore module_unknown=ignore default=bad] pam_selinux.so open
@include common-password
```