1117.4pt

Pluggable Authentication Modules

Dag-Erling Smørgrav

$FreeBSD: doc/fr_FR.ISO8859-1/articles/pam/article.sgml,v 1.4 2007/01/20 12:25:44 blackend Exp $

Cet article a été écrit pour le Projet FreeBSD par ThinkSec AS et les laboratoires de Networks Associates, la division de recherche en sécurité de Networks Associates, Inc. sous le contrat DARPA/SPAWAR N66001-01-C-8035 (“CBOSS”), en tant que partie du programme de recherche DARPA CHATS.

Cet article décrit les principes sous-jacent et les mécanismes de la bibliothèque PAM, il explique comment configurer PAM, l’intégrer dans les applications, et écrire ses propres modules PAM.

Version française de Clément Mathieu <*cykl@mAdchAt.org*>.

Table of Contents

1. Introduction **1**

2. Termes et conventions **2**

3. Les bases de PAM **5**

4. Configuration de PAM **9**

5. Les modules PAM de FreeBSD **11**

6. Programmation d’applications PAM **14**

7. Programmation de modules PAM **14**

A. Exemples d’application PAM **14**

B. Exemple d’un module PAM **17**

C. Exemple d’une fonction de conversation PAM **20**

Lectures complémentaires **21**

1. Introduction

La bibliothèque PAM est une API généralisée pour les services relevant de l'authentification permettant à un administrateur système d'ajouter une nouvelle méthode d'authentification en ajoutant simplement un nouveau module PAM, ainsi que de modifier les règles d'authentification en éditant les fichiers de configuration.

PAM a été conçu et développé en 1995 par Vipin Samar et Charlie Lai de Sun Microsystems, et n'a pas beaucoup évolué depuis. En 1997 l'Open Group publie les premières spécifications XSSO qui standardisent l'API PAM et ajoute des extensions pour un simple (ou plutot intégré) "sign-on". Lors de l'écriture de cet article, la spécification n'a toujours pas été adoptée comme standard.

Bien que cet article se concentre principalement sur FreeBSD 5.x, qui utilise OpenPAM, il devrait également être applicable à FreeBSD 4.x qui utilise Linux-PAM, ainsi qu'à d'autres systèmes d'exploitations tels que Linux ou Solaris.

1.1. Marques déposées

2. Termes et conventions

2.1. Définitions

La terminologie de PAM est plutôt confuse. Ni la publication originale de Samar et Lai, ni la spécification XSSO n'ont essayé de définir formellement des termes pour les acteurs et les entités intervenant dans PAM, les termes qu'ils utilisent (mais ne définissent pas) sont parfois trompeurs et ambigus. Le premier essai d'établir une terminologie consistante et non ambiguë fut un papier écrit par Andrew G. Morgan (l'auteur de Linux-PAM) en 1999. Bien que les choix de Morgan furent un énorme pas en avant, ils ne sont pas parfait d'après l'auteur de ce document. Ce qui suit, largement inspiré par Morgan, est un essai de définir précisément et sans ambiguïté des termes pour chaque acteur ou entité utilisé dans PAM.

compte

L'ensemble de permissions que le demandeur demande a l'arbitre.

demandeur

L'utilisateur ou l'entité demandant authentification.

arbitre

L'utilisateur ou l'entité possédant les privilèges nécessaires pour vérifier la requête du demandeur ainsi que l'autorité d'accorder ou de rejeter la requête.

chaîne

Une séquence de modules qui sera invoquée pour répondre à une requête PAM. La chaîne comprend les informations concernant l'ordre dans lequel invoquer les modules, les arguments à leur passer et la façon d'interpréter les résultats.

client

L'application responsable de la requête d'authentification au nom du demandeur et de recueillir l'information d'authentification nécessaire.

mécanisme

Il s'agit de l'un des quatre groupes basiques de fonctionnalités fournit par PAM : authentification, gestion de compte, gestion de session et mise à jour du jeton d'authentification.

module

Une collection d'une ou plusieurs fonctions implémentant un service d'authentification particulier, rassemblées dans un fichier binaire (normalement chargeable dynamiquement) et identifié par un nom unique.

règles

Le jeu complet de configuration des règles décrivant comment traiter les requêtes PAM pour un service particulier. Une règle consiste normalement en quatre chaînes, une pour chaque mécanisme, bien que quelques services n'utilisent pas les quatre mécanismes.

serveur

L'application agissant au nom de l'arbitre pour converser avec le client, récupérer les informations d'authentification, vérifier les droits du demandeur et autoriser ou rejeter la requête.

service

Un ensemble de serveurs fournissant des fonctionnalités similaires ou liées et nécessitant une authentification similaire. Les règles de PAM sont définies sur un le principe de par-service; ainsi tous les serveurs qui demandent le même nom de service seront soumis aux mêmes règles.

session

Le contexte dans lequel le service est délivré au demandeur par le serveur. L'un des quatre mécanismes de PAM, la gestion de session, s'en occupe exclusivement par la mise en place et le relâchement de ce contexte.

jeton

Un morceau d'information associé avec un compte tel qu'un mot de passe ou une passphrase que le demandeur doit fournir pour prouver son identité.

transaction

Une séquence de requêtes depuis le même demandeur vers la même instance du même serveur, commençant avec l'authentification et la mise en place de la session et se terminant avec le démontage de la session.

2.2. Exemples d'utilisation

Cette section a pour but d'illustrer quelques-uns des termes définis précédemment à l'aide d'exemples basiques.

2.2.1. Client et serveurs ne font qu'un

Cet exemple simple montre `alice` utilisant su(1) pour devenir `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
```

- Le demandeur est `alice`.
- Le compte est `root`.
- Le processus su(1) est à la fois client et serveur.
- Le jeton d'authentification est `xi3kiune`.
- L'arbitre est `root`, ce qui explique pourquoi su(1) est setuid `root`.

2.2.2. Client et serveur sont distincts.

L'exemple suivant montre `eve` essayant d'initier une connexion ssh(1) vers `login.exemple.com`, en demandant à se logguer en tant que `bob`. La connexion réussit. Bob aurait du choisir un meilleur mot de passe !

```
% 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!
%
```

- Le demandeur est `eve`.
- Le client d'`eve` est représenté par les processus ssh(1)
- Le serveur est le processus sshd(8) sur `login.example.com`
- Le compte est `bob`.
- Le jeton d'identification est `god`.
- Bien que cela ne soit pas montré dans cet exemple, l'arbitre est `root`.

2.2.3. Exemple de règles

Les lignes qui suivent sont les règles par défaut de `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
```

- Cette politique s'applique au service `sshd` (qui n'est pas nécessairement restreint au serveur sshd(8))
- `auth`, `account`, `session` et `password` sont des mécanismes.
- `pam_nologin.so`, `pam_unix.so`, `pam_login_access.so`, `pam_lastlog.so` et `pam_permit.so` sont des modules. Il est clair dans cet exemple que `pam_unix.so` fournit au moins deux mécanismes (authentification et gestion de compte).

2.3. Conventions

Cette section n'a pas encore été écrite.

3. Les bases de PAM

3.1. Mécanismes et primitives

L'API PAM fournit six primitives d'authentification différentes regroupées dans quatre mécanismes qui seront décrits dans la partie suivante.

`auth`

Authentification. Ce mécanisme concerne l'authentification du demandeur et établit les droits du compte. Il fournit deux primitives :

- pam_authenticate(3) authentifie le demandeur, généralement en demandant un jeton d'identification et en le comparant a une valeur stockée dans une base de données ou obtenue par le biais d'un serveur d'authentification.
- pam_setcred(3) établi les paramètres du compte tel que l'uid, les groupes dont le compte fait parti ou les limites sur l'utilisation des ressources.

`account`

Gestion de compte. Ce mécanisme concerne la disponibilité du compte pour des raisons autres que l'authentification. Par exemple les restrictions basées sur l'heure courante ou la charge du serveur. Il fournit une seule primitive:

- pam_acct_mgmt(3) vérifie que le compte demandé est disponible.

`session`

Gestion de session. Ce mécanisme concerne la mise en place de la session et sa terminaison, par exemple l'enregistrement de la session dans les journaux. Il fournit deux primitives:

- pam_open_session(3) accomplie les tâches associées à la mise en place d'une session : ajouter une entrée dans les bases `utmp` et `wtmp`, démarrer un agent SSH...
- pam_close_session(3) accomplie les tâches associées à la terminaison d'une session : ajouter une entrée dans les bases `utmp` et `wtmp`, arrêter l'agent SSH...

`password`

Gestion des mots de passe. Ce mécanisme est utilisé pour modifier le jeton d'authentification associé à un compte, soit parce qu'il a expiré, soit parce que l'utilisateur désire le changer. Il fournit une seule primitive:

- pam_chauthtok(3) modifie le jeton d'authentification, et éventuellement vérifie que celui-ci est assez robuste pour ne pas être deviné facilement ou qu'il n'a pas déjà utilisé.

3.2. Modules

Les modules sont le concept clef de PAM; après tout ils constituent le "M" de PAM. Un module PAM est lui-même un morceau de code qui implémente les primitives d'un ou plusieurs mécanismes pour une forme particulière d'authentification; par exemple, les bases de mots de passe UNIX que sont NIS, LDAP et Radius.

3.2.1. Nom des modules

FreeBSD implémente chaque mécanismes dans un module distinct nommé `pam_`*`mécanisme`*`.so` (par exemple `pam_unix.so` pour le mécanisme Unix .) Les autres implementations possèdent parfois des modules séparés pour des mécanismes séparés et incluent aussi bien le nom du service que celui du mécanisme dans le nom du module. Un exemple est le module `pam_dial_auth.so.1` de Solaris qui est utilisé pour authentifier les utilisateurs dialup.

3.2.2. Gestion des versions de module

L'implémentation originale de PAM par FreeBSD, basée sur Linux-PAM, n'utilisait pas de numéro de version pour les modules PAM. Ceci peut poser des problèmes avec les applications tiers qui peuvent être liées avec d'anciennes bibliothèques systèmes, puisqu'il n'y a pas possibilité de charger la version correspondante du module désiré.

Pour sa part, OpenPAM cherche les modules qui ont la même version que la bibliothèque PAM (pour le moment 2) et se rabat sur un module sans version si aucun module avec version n'a put être chargé. Ainsi les anciens modules peuvent être fournis pour les anciennes applications, tout en permettant aux nouvelles applications (ou bien nouvellement compilées) de tirer parti des modules les plus récents.

Bien que les modules PAM de Solaris possèdent généralement un numéro de version, ils ne sont pas réellement versionnés car le numéro correspond à une partie du nom du module et doit être inclus dans la configuration.

3.3. Chaînes et politiques

Lorsqu'un serveur initie une transaction PAM, la bibliothèque PAM essaie de charger une politique pour le service spécifié dans l'appel a pam_start(3) . La politique indique comment la requête d'authentification doit être traitée et est définie dans un fichier de configuration. Il s'agit de l'autre concept clef de PAM : la possibilité pour l'administrateur de configurer la politique de sécurité d'un système en éditant simplement une fichier texte.

Une politique consiste en quatre chaînes, une pour chacune des quatre mécanismes de PAM. Chaque chaîne est une suite de règles de configuration, chacune spécifiant un module à invoquer, des paramètres, options, à passer au module et un drapeau de contrôle qui décrit comment interpréter le code de retour du module.

Comprendre le drapeau de contrôle est essentiel pour comprendre les fichiers de configuration de PAM. Il existe quatre drapeaux de contrôle différents :

`binding`

Si le module réussit et qu'aucun module précédent de la chaîne n'a échoué, la chaîne s'interrompt immédiatement et la requête est autorisée. Si le module échoue le reste de la chaîne est exécuté, mais la requête est rejetée au final.

Ce drapeau de contrôle a été introduit par Sun Solaris dans la version 9 (SunOS 5.9); il est aussi supporté par OpenPAM.

`required`

Si le module réussit, le reste de la chaîne est exécuté, et la requête est autorisée si aucun des autres modules n'échoue. Si le module échoue, le reste de la chaîne est exécuté, mais au final la requête est rejetée.

`requisite`

Si le module réussit le reste de la chaîne est exécuté, et la requête est autorisée sauf si d'autres modules échoués. Si le module échoue la chaîne est immédiatement terminée et la requête est rejetée.

`sufficient`

Si le module réussit et qu'aucun des modules précédent n'a échoué la chaîne est immédiatement terminée et la requête est allouée. Si le module échoue il est ignore et le reste de la chaîne est exécuté.

Puisque la sémantique de ce drapeau peut être un peu confuse, spécialement lorsqu'il s'agit de celui du dernier module de la chaîne, il est recommandé d'utiliser le drapeau `binding` à la place de celui-ci sous la condition que l'implémentation le supporte.

`optional`

Le module est exécuté mais le résultat est ignoré. Si tout les modules de la chaîne sont marqués `optional`, toutes les requêtes seront toujours acceptées.

Lorsqu'un serveur invoque l'une des six primitives PAM, PAM récupère la chaîne du mécanisme à laquelle la requête correspond et invoque chaque module de la chaîne dans l'ordre indiqué, jusqu'à ce que la fin soit atteinte ou qu'aucune exécution supplémentaire ne soit nécessaire (soit à cause du succès d'un module en `binding` ou `sufficient`, soit à cause de l'échec d'un module `requisite`). La requête est acceptée si et seulement si au moins un module a été invoqué, et que tout les modules non optionnels ont réussi.

Notez qu'il est possible, bien que peu courant, d'avoir le même module listé plusieurs fois dans la même chaîne. Par exemple un module qui détermine le nom utilisateur et le mot de passe à l'aide d'un serveur directory peut être invoqué plusieurs fois avec des paramètres spécifiant différents serveurs a contacter. PAM considère les différentes occurrences d'un même module dans une même chaîne comme des modules différents et non liés.

3.4. Transactions

Le cycle de vie d'une transaction PAM typique est décrit ci-dessous. Notez que si l'une de ces étapes échoue, le serveur devrait reporter un message d'erreur au client et arrêter la transaction.

1. Si nécessaire, le serveur obtient les privilèges de l'arbitre par le biais d'un mécanisme indépendant de PAM — généralement en ayant été démarré par `root` ou en étant setuid `root`.
2. Le serveur appel pam_start(3) afin d'initialiser la bibliothèque PAM et indique le service et le compte cible, et enregistre une fonction de conversation appropriée.

3. Le serveur obtient diverses informations concernant la transaction (tel que le nom d'utilisateur du demandeur et le nom d'hôte de la machine sur lequel le client tourne) et les soumet à PAM en utilisant la fonction pam_set_item(3).
4. Le serveur appel pam_authenticate(3) pour authentifier le demandeur.
5. Le serveur appel la fonction pam_acct_mgmt(3) qui vérifie que le compte est valide et disponible. Si le mot de passe est correct mais a expiré, pam_acct_mgmt(3) retournera `PAM_NEW_AUTHTOK_REQD` à la place de `PAM_SUCCESS`.
6. Si l'étape précédente a retourné `PAM_NEW_AUTHTOK_REQD`, le serveur appel maintenant pam_chauthtok(3) pour obliger l'utilisateur à changer le jeton d'authentification du compte désiré.
7. Maintenant que le demandeur a été correctement authentifié, le serveur appelle pam_setcred(3) pour obtenir les privilèges du compte désiré. Il lui est possible de faire ceci parce qu'il agit au nom de l'arbitre dont il possède les privilèges.
8. Lorsque les privilèges corrects ont été établi le serveur appelle pam_open_session(3) pour mettre en place la session.
9. Maintenant le serveur effectue les services demandés par le client — par exemple fournir un shell au demandeur.
10. Lorsque le serveur a fini de servir le client, il appelle pam_close_session(3) afin de terminer la session.
11. Pour finir, le serveur appelle pam_end(3) afin signaler à la bibliothèque PAM que la transaction se termine et qu'elle peut libérer les ressources qu'elle a alloué au cours de la transaction.

4. Configuration de PAM

4.1. Emplacement des fichiers de configuration

Le fichier de configuration de PAM est traditionnellement `/etc/pam.conf`. Ce fichier contient toutes les politiques de PAM pour votre système. Chaque ligne du fichier décrit une étape dans une chaîne, tel que nous allons le voir ci-dessous:

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

Les champs sont respectivement, le service, le nom du mécanisme, le drapeau de contrôle, le nom du module et les arguments du module. Tout champ additionnel est considéré comme argument du module.

Une chaîne différente est construite pour chaque couple service/mécanisme; ainsi, alors que l'ordre des lignes est important lorsqu'il s'agit des mêmes services ou mécanismes, l'ordre dans lequel les différents services et mécanismes apparaissent ne l'est pas — excepté l'entrée pour le service `other`, qui sert de référence par défaut et doit être placé à la fin. L'exemple du papier original sur PAM regroupait les lignes de configurations par mécanisme et le fichier `pam.conf` de Solaris le fait toujours, mais FreeBSD groupe les lignes de configuration par service. Toutefois il ne s'agit pas de la seule possibilité et les autres possèdent aussi un sens.

OpenPAM et Linux-PAM offrent un mécanisme de configuration alternatif où les politiques sont placées dans des fichiers séparés portant le nom du service auquel ils s'appliquent. Ces fichiers sont situés dans `/etc/pam.d/` et ne contiennent que quatre champs à la place de cinq — le champ contenant le nom du service est omis. Il s'agit du mode par défaut dans FreeBSD 4.x. Notez que si le fichier `/etc/pam.conf` existe et contient des informations de

configuration pour des services qui n'ont pas de politique spécifiée dans `/etc/pam.d`, ils seront utilisés pour ces services.

Le gros avantage de `/etc/pam.d/` sur `/etc/pam.conf` est qu'il est possible d'utiliser la même politique pour plusieurs services en liant chaque nom de service à un fichier de configuration. Par exemple pour utiliser la même politique pour les services `su` et `sudo`, nous pouvons faire comme ceci :

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

Ceci fonctionne car le nom de service est déterminé a partir du nom de fichier plutôt qu'indiqué à l'intérieur du fichier de configuration, ainsi le même fichier peut être utilisé pour des services nommés différemment.

Un autre avantage est qu'un logiciel tiers peu facilement installer les politiques pour ses services sans avoir besoin d'éditer `/etc/pam.conf`. Pour continuer la tradition de FreeBSD, OpenPAM regardera dans `/usr/local/etc/pam.d` pour trouver les fichiers de configurations; puis si aucun n'est trouvé pour le service demandé, il cherchera dans `/etc/pam.d/` ou `/etc/pam.conf`.

Finalement, quelque soit le mécanisme que vous choisissiez, la politique "magique" `other` est utilisée par défaut pour tous les services qui n'ont pas leur propre politique.

4.2. Breakdown of a configuration line

Comme expliqué dans la section *Emplacement des fichiers de configuration*, chaque ligne de `pam.conf` consiste en quatre champs ou plus: le nom de service, le nom du mécanisme, le drapeau de contrôle, le nom du module et la présence ou non d'arguments pour le module.

Le nom du service est généralement, mais pas toujours, le nom de l'application auquelle les règles s'appliquent. Si vous n'êtes pas sûr, référez vous à la documentation de l'application pour déterminer quel nom de service elle utilise.

Notez que si vous utilisez `/etc/pam.d/` à la place de `/etc/pam.conf`, le nom du service est spécifié par le nom du fichier de configuration et n'est pas indiqué dans les lignes de configuration qui, dès lors, commencent par le nom du mécanisme.

Le mécanisme est l'un des quatre mots clef décrit dans la section *Mécanismes et primitives*

De même, le drapeau de contrôle est l'un des quatre mots clef décrits dans la section *Chaînes et politiques* et décrit comment le module doit interpréter le code de retour du module. Linux-PAM supporte une syntaxe alternative qui vous laisse spécifier l'action à associer à chaque code de retour possible; mais ceci devrait être évité puisque ce n'est pas standard et étroitement lié à la façon dont Linux-PAM appelle les services (qui diffère grandement de la façon de Solaris et OpenPAM). C'est sans étonnement que l'on apprend qu'OpenPAM ne supporte pas cette syntaxe.

4.3. Politiques

Pour configurer PAM correctement, il est essentiel de comprendre comment les politiques sont interprétées.

Lorsqu'une application appelle pam_start(3) la bibliothèque PAM charge la politique du service spécifié et construit les quatre chaînes de module (une pour chaque mécanisme). Si une ou plusieurs chaînes sont vides, les chaînes de la politique du service `other` sont utilisées.

Plus tard, lorsque l'application appelle l'une des six primitives PAM, la bibliothèque PAM récupère la chaîne du mécanisme correspondant et appelle la fonction appropriée avec chaque module listé dans la chaîne. Après chaque

appel d'une fonction de service, le type du module et le code d'erreur sont retournés par celle-ci pour déterminer quoi faire. À quelques exceptions près, dont nous parlerons plus tard, la table suivante s'applique:

Table 1. Résumé de la chaîne d'exécution PAM

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

Si `fail` est vrai à la fin de la chaîne, ou lorsqu'un "break" est atteint, le dispatcheur retourne le code d'erreur renvoyé par le premier module qui a échoué. Autrement `PAM_SUCCESS` est retourné.

La première exception est que le code d'erreur `PAM_NEW_AUTHOK_REQD` soit considéré comme un succès, sauf si aucun module n'échoue et qu'au moins un module retourne `PAM_NEW_AUTHOK_REQD` le dispatcheur retournera `PAM_NEW_AUTHOK_REQD`.

La seconde exception est que pam_setcred(3) considère les modules `binding` et `sufficient` comme s'ils étaient `required`.

La troisième et dernière exception est que pam_chauthtok(3) exécute la totalité de la chaîne deux fois (la première pour des vérifications préliminaires et la deuxième pour mettre le mot de passe) et lors de la première exécution il considère les modules `binding` et `sufficient` comme s'ils étaient `required`.

5. Les modules PAM de FreeBSD

5.1. pam_deny(8)

Le module pam_deny(8) est l'un des modules disponibles les plus simples; il répond à n'importe qu'elle requête par `PAM_AUTH_ERR`. Il est utile pour désactiver rapidement un service (ajoutez-le au début de chaque chaîne), ou pour terminer les chaînes de modules `sufficient`.

5.2. pam_echo(8)

Le module pam_echo(8) passe simplement ses arguments à la fonction de conversation comme un message `PAM_TEXT_INFO`. Il est principalement utilisé pour le debogage mais il peut aussi servir à afficher un message tel que "Les accès illégaux seront poursuivits" avant de commencer la procédure d'authentification.

5.3. pam_exec(8)

Le module pam_exec(8) prend comme premier argument le nom du programme à exécuter, les arguments restant étant utilisés comme arguments pour ce programme. L'une des applications possibles est d'utiliser un programme qui monte le répertoire de l'utilisateur lors du login.

5.4. pam_ftp(8)

Le module pam_ftp(8)

5.5. pam_ftpusers(8)

Le module pam_ftpusers(8)

5.6. pam_group(8)

Le module pam_group(8) accepte ou rejette le demandeur à partir de son appartenance à un groupe particulier (généralement `wheel` pour su(1)). Il a pour but premier de conserver le comportement traditionnel de su(1) mais possède d'autres applications comme par exemple exclure un certain groupe d'utilisateurs d'un service particulier.

5.7. pam_krb5(8)

Le module pam_krb5(8)

5.8. pam_ksu(8)

Le module pam_ksu(8)

5.9. pam_lastlog(8)

Le module pam_lastlog(8)

5.10. pam_login_access(8)

Le module pam_login_access(8)

5.11. pam_nologin(8)

Le module pam_nologin(8)

5.12. pam_opie(8)

Le module pam_opie(8) implémente la méthode d'authentification opie(4). Le système opie(4) est un mécanisme de challenge-response où la réponse à chaque challenge est une fonction directe du challenge et une phrase de passe, ainsi la réponse peut facilement être calculée "en temps voulu" par n'importe qui possédant la phrase de passe ce qui élimine le besoin d'une liste de mots de passe. De plus, puisque opie(4) ne réutilise jamais un mot de passe qui a reçu une réponse correcte, il n'est pas vulnérable aux attaques basée sur le rejouage.

5.13. pam_opieaccess(8)

Le module pam_opieaccess(8) est un compagnon du module pam_opie(8). Son but est de renforcer les restrictions codifiées dans opieaccess(5), il régule les conditions sous lesquelles un utilisateur qui normalement devrait s'authentifier par opie(4) est amené à utiliser d'autres méthodes. Ceci est généralement utilisé pour interdire l'authentification par mot de passe depuis des hôtes non digne de confiance.

Pour être réellement effectif, le module pam_opieaccess(8) doit être listé comme `requisite` immédiatement après une entrée `sufficient` pour pam_opie(8) et avant tout autre module, dans la chaîne `auth`.

5.14. pam_passwdqc(8)

Le module pam_passwdqc(8)

5.15. pam_permit(8)

Le module pam_permit(8) est l'un des modules disponibles les plus simples; il répond à n'importe quelle requête par `PAM_SUCCESS`. Il est utile pour les services où une ou plusieurs chaînes auraient autrement été vides.

5.16. pam_radius(8)

Le module pam_radius(8)

5.17. pam_rhosts(8)

Le module pam_rhosts(8)

5.18. pam_rootok(8)

Le module pam_rootok(8) retourne un succès si et seulement si l'identifiant d'utilisateur réel du processus appelant est 0. Ceci est utile pour les services non basés sur le réseau tel que su(1) ou passwd(1) où l'utilisateur `root` doit avoir un accès automatique.

5.19. pam_securetty(8)

Le module pam_securetty(8)

5.20. pam_self(8)

Le module pam_self(8) retourne un succès si et seulement si le nom du demandeur correspond au nom du compte désiré. Il est utile pour les services non basés sur le réseau tel que su(1) où l'identité du demandeur peut être vérifiée facilement .

5.21. pam_ssh(8)

Le module pam_ssh(8)

5.22. pam_tacplus(8)

Le module pam_tacplus(8)

5.23. pam_unix(8)

Le module pam_unix(8) implémente l'authentification Unix traditionnelle par mot de passe, il utilise getpwnam(3) pour obtenir le mot de passe du compte visé et le compare avec celui fournit par le demandeur. Il fournit aussi des services de gestion de compte (désactivation du compte et date d'expiration) ainsi que des services pour le changement de mot de passe. Il s'agit certainement du module le plus utile car la plupart des administrateurs désirent garder le comportement historique pour quelques services.

6. Programmation d'applications PAM

Cette section n'a pas encore été écrite.

7. Programmation de modules PAM

Cette section n'a pas été encore écrite.

A. Exemples d'application PAM

Ce qui suit est une implémentation minimale de su(1) en utilisant PAM. Notez qu'elle utilise la fonction de conversation openpam_ttyconv(3) spécifique à OpenPAM qui est prototypée dans `security/openpam.h`. Si vous désirez construire cette application sur un système utilisant une bibliothèque PAM différente vous devrez fournir votre propre fonction de conversation. Une fonction de conversation robuste est étonnamment difficile à implémenter; celle présentée dans l'appendice*Exemple d'une fonction de conversation PAM* est un bon point de départ, mais ne devrait pas être utilisée dans des applications réelles.

```
#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);
}
```

B. Exemple d'un module PAM

Ce qui suit est une implémentation minimale de pam_unix(8) offrant uniquement les services d'authentification. Elle devrait compiler et tourner avec la plupart des implémentations PAM, mais tire parti des extensions d'OpenPAM si elles sont disponibles : notez l'utilisation de pam_get_authtok(3) qui simplifie énormément l'affichage de l'invite pour demander le mot de passe à l'utilisateur.

```
#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
```

C. Exemple d'une fonction de conversation PAM

La fonction de conversation présentée ci-dessous est une version grandement simplifiée de la fonction openpam_ttyconv(3) d'OpenPAM. Elle est pleinement fonctionnelle et devrait donner au lecteur une bonne idée de comment doit se comporter une fonction de conversation, mais elle est trop simple pour une utilisation réelle. Même si vous n'utilisez pas OpenPAM, N'hésitez pas à télécharger le code source et d'adapter openpam_ttyconv(3) à vos besoins, nous pensons qu'elle est raisonnablement aussi robuste qu'une fonction de conversation orientée tty peut l'être.

```
#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);
}
```

Lectures complémentaires

Ceci est une liste de documents concernant PAM et les domaines gravitant autours. Elle n'a pas la prétention d'être complète.

Publications

Rendre les services de connexion indépendants des technologies d'authentification (http://www.sun.com/software/solaris/pam/pam.external.pdf), Vipin Samar and Charlie Lai, Sun Microsystems.

X/Open Single Sign-on Preliminary Specification (http://www.opengroup.org/pubs/catalog/p702.htm), The Open Group, 1-85912-144-6, June 1997.

Pluggable Authentication Modules (http://www.kernel.org/pub/linux/libs/pam/pre/doc/current-draft.txt), Andrew G. Morgan, October 6, 1999.

Guides utilisateur

Administration de PAM (http://www.sun.com/software/solaris/pam/pam.admin.pdf), Sun Microsystems.

Page internet liées

La page d'OpenPAM (http://openpam.sourceforge.net/), Dag-Erling Smørgrav, ThinkSec AS.

La page de Linux-PAM (http://www.kernel.org/pub/linux/libs/pam/), Andrew G. Morgan.

La page de Solaris PAM (http://wwws.sun.com/software/solaris/pam/), Sun Microsystems.