Módulos de Autenticação Plugáveis

Dag-Erling Smørgrav
Revisão: 374a3c75a7

This article was written 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.

Resumo

Esse artigo descreve princípios subjacentes e mecanismos da biblioteca de Módulos de Autenticação Plugáveis (PAM), e explica como configurar, como integrar com outras aplicações e como escrever novos módulos.

Índice

1. Introdução 1
2. Termos e convenções 2
3. PAM Essencial 4
4. Configuração do PAM 7
5. Módulos PAM do FreeBSD 9
6. Programação de Aplicação PAM 11
7. Programação de Módulos PAM 11
A. Exemplo de Aplicação PAM 11
B. Exemplo do módulo PAM 14
C. Exemplo de função de conversação PAM 17
Leitura Adicional 18

1. Introdução

A biblioteca dos Módulos de Autenticação Plugáveis (PAM) é uma API generalizada para serviços relacionados com autenticações as quais permitem ao administrador de sistema adicionar um novo método de autenticação simplesmente pela instalação de um novo módulo PAM, e modificar a política de autenticação pela edição do arquivo de configuração.

O PAM foi definido e desenvolvido em 1995 por Vipin Samar e Charlie Lai da Sun Microsystems, e não teve muitas mudanças até hoje. Em 1997, o Open Group publicou a especificação preliminar do X/Open Single Sign-on (XSSO), a qual padroniza a API do PAM e adiciona extensões para autenticação única ou integrada. No momento da redação deste documento, esta especificação ainda não tinha sido adotada como padrão.

Apesar deste artigo focar primariamente no sistema FreeBSD 5.x, o qual usa o OpenPAM, ele poderá ser igualmente aplicado ao FreeBSD 4.x, o qual usa o Linux-PAM, e a outros systemas operacionais, tais como o Linux e o Solaris™.

2. Termos e convenções

2.1. Definições

A terminologia em torno do PAM é bastante confusa. Nem o artigo original de Neither Samar e Lai nem a especificação original do XSSO fizeram algum esforço para definir formalmente os termos de vários atores e entidades envolvidas no PAM, e os termos que eles usam (mas não definem) são algumas vezes duvidosos ou ambíguos. A primeira tentativa de estabelecer uma terminologia consistente e não ambígua foi feita no artigo escrito por Andrew G. Morgan (autor do Linux-PAM) em 1999. A escolha da terminologia de Morgan foi um grande avanço, mas na opinião deste autor, não é perfeita. O que segue é uma tentativa, fortemente inspirada por Morgan, de definir termos precisos e não ambíguos para todos os atores e entidades envolvidas no PAM.

conta (account)	Um conjunto de credenciais que o requerente está solicitando ao mediador.
requerente (applicant)	Usuário ou entidade que solicita autenticação
Mediador (arbitrator)	Usuário ou entidade a qual tem privilégios necessários para verificar as credenciais do requerente e autorizar ou não a solicitação
chain	Uma sequência de módulos que irá ser chamada em resposta a uma solicitação do PAM. A chain inclui informações sobre a ordem a qual invocar os módulos, quais argumentos foram passados e como interpretar os resultados.
client	A aplicação responsável por inicializar uma solicitação de autenticação em nome do requerente para obter as informações necessárias da autenticação dele.
recursos	Um dos quatro grupos básicos de funcionalidades fornecidos pelo PAM: autenticação, gerenciamento de conta, gerenciamento de sessão e atualização de token de autenticação.
módulo	Uma coleção de uma ou mais funções relacionadas implementando um recurso de autenticação específico, reunidas em um único arquivo binário (normalmente carregável dinamicamente) e identificadas por um único nome.
política	O conjunto completo de instruções de configuração que descrevem como lidar com solicitações do PAM para um serviço específico. Uma política normalmente consiste em quatro chains, uma para cada recurso, embora alguns serviços não utilizem os quatro recursos.
servidor	O aplicativo agindo em nome do mediador para conversar com o cliente, recuperar informações de autenticação, verificar as credenciais do requerente e conceder ou negar solicitações.
serviço	Classe de servidores que provêm recursos similares ou relacionados e requerem autenticação similar. As políticas do PAM são definidas por serviço, portanto, todos os servidores que reivindicam o mesmo nome de serviço estarão sujeitos à mesma política.

sessão	O contexto com o qual serviços são apresentados para o requerente pelo servidor. Um dos quatro recursos do PAM, gerenciamento de sessão, é concedido exclusivamente configurando e derrubando esse contexto.
token	Um pedaço de informação associada à conta, como uma senha sendo uma palavra ou uma frase, que o solicitante deve fornecer para provar sua identidade.
transação	Uma sequência de solicitações do mesmo requerente para a mesma instância do mesmo servidor, começando com a configuração de autenticação e sessão e terminando com a desmontagem da sessão.

2.2. Exemplos de uso

Esta seção tem como objetivo ilustrar os significados de alguns dos termos definidos acima por meio de alguns exemplos simples.

2.2.1. O cliente e o servidor são um

Este exemplo simples mostra `alice` usando su(1) para se tornar `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
```

- O requerente é `alice`.
- A conta é `root`.
- O processo su(1) é ao mesmo tempo, o cliente e o servidor.
- O token de autenticação é `xi3kiune`.
- O mediador é `root`, e é por isso que su(1) possui setuid para `root`.

2.2.2. O cliente e o servidor são separados

O exemplo abaixo mostra `eve` tentar iniciar uma conexão ssh(1) com `login.example.com` , solicitar para efetuar login como `bob` e ter exito. Bob deveria ter escolhido uma senha melhor!

```
% 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!
%
```

- O requerente é `eve`.
- O cliente é o processo ssh(1) de Eve.
- O servidor é o processo sshd(8) em `login.example.com`

- A conta é `bob`.
- O token de autenticação é `god`.
- Embora isso não seja mostrado neste exemplo, o mediador é `root`.

2.2.3. Exemplo de política

A seguir, a política padrão do 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 ao serviço `sshd` (que não é necessariamente restrito ao servidor sshd(8)).
- `auth`, `account`, `session` e `password` são recursos.
- `pam_nologin.so`, `pam_unix.so`, `pam_login_access.so`, `pam_lastlog.so` e `pam_permit.so` são módulos. Fica claro neste exemplo que o `pam_unix.so` fornece pelo menos dois recursos (autenticação e gerenciamento de conta).

3. PAM Essencial

3.1. Recursos e Primitivas

A API do PAM oferece seis primitivas de autenticação diferentes agrupadas em quatro recursos, descritos abaixo.

`auth`
: *Autenticação.* Este recurso se preocupa em autenticar o requerente e estabelecer as credenciais da conta. Ele fornece duas primitivas:

  - pam_authenticate(3) autentica o requerente, geralmente solicitando um token de autenticação e comparando-o com um valor armazenado em um banco de dados ou obtido de um servidor de autenticação.
  - pam_setcred(3) estabelece credenciais de conta, como ID de usuário, associação de grupo e limites de recursos.

`account`
: *Gerenciamento de contas.* Esse recurso lida com problemas de disponibilidade de conta não relacionados à autenticação, como restrições de acesso com base na hora do dia ou na carga de trabalho do servidor. Ele fornece uma única primitiva:

  - pam_acct_mgmt(3) verifica se a conta solicitada está disponível.

`session`
: *Gerenciamento de sessão.* Esse recurso lida com tarefas associadas à configuração e desmontagem da sessão, como a contabilização de login. Ele fornece duas primitivas:

  - pam_open_session(3) executa tarefas associadas à configuração da sessão: adiciona uma entrada nos bancos de dados `utmp` e `wtmp`, inicia um agente SSH, etc.
  - pam_close_session(3) executa tarefas associadas à desmontagem da sessão: adiciona uma entrada nos bancos de dados `utmp` e `wtmp`, pare o agente SSH, etc.

`password`
: *Gerenciamento de senhas.* Esse recurso é usado para alterar o token de autenticação associado a uma conta, porque expirou ou porque o usuário deseja alterá-lo. Ele fornece uma única primitiva:

  - pam_chauthtok(3) altera o token de autenticação, opcionalmente, verificando se é suficientemente difícil de adivinhar, se não foi usado anteriormente etc.

3.2. Módulos

Módulos são um conceito muito central no PAM; afinal, eles são os "M" no "PAM". Um módulo PAM é um código de programa autocontido que implementa as primitivas em uma ou mais instalações para um mecanismo específico; possíveis mecanismos para o recurso de autenticação, por exemplo, incluem os bancos de dados de senhas UNIX®, NIS, LDAP e Radius.

3.2.1. Nomeação de Módulos

O FreeBSD implementa cada mecanismo em um único módulo, chamado `pam_`*`mechanism`*`.so` (por exemplo, `pam_unix.so` para o mecanismo UNIX®. Outras implementações às vezes possuem módulos separados para instalações separadas e incluem o nome do recurso, bem como o nome do mecanismo no nome do módulo. Para citar um exemplo, Solaris™ tem um módulo `pam_dial_auth.so.1` que é comumente usado para autenticar usuários de conexões discadas.

3.2.2. Versionando Módulos

A implementação original do PAM no FreeBSD, baseada no Linux-PAM, não utilizou números de versão para os módulos PAM. Isso normalmente causaria problemas com aplicativos legados, que poderiam estar vinculados a versões mais antigas das bibliotecas do sistema, pois não havia como carregar uma versão correspondente dos módulos necessários.

O OpenPAM, por outro lado, procura por módulos que possuam o mesmo número de versão que a biblioteca PAM (atualmente 2), e só retorna a um módulo não versionado se nenhum módulo versionado puder ser carregado. Assim, os módulos legados podem ser fornecidos para aplicativos legados, permitindo que novos aplicativos (ou recém-construídos) aproveitem os módulos mais recentes.

Embora os módulos PAM do Solaris™ normalmente tenham um número de versão, eles não são realmente versionados, porque o número é uma parte do nome do módulo e deve ser incluído na configuração.

3.3. Cadeias e Políticas

Quando um servidor inicia uma transação PAM, a biblioteca PAM tenta carregar uma política para o serviço especificado na chamada pam_start(3). A política especifica como as solicitações de autenticação devem ser processadas e definidas em um arquivo de configuração. Este é o outro conceito central no PAM: a possibilidade de o administrador ajustar a política de segurança do sistema (no sentido mais amplo da palavra) simplesmente editando um arquivo de texto.

Uma política consiste em quatro cadeias, uma para cada uma dos quatro recursos do PAM. Cada chain é uma sequência de instruções de configuração, cada uma especificando um módulo para invocar, alguns parâmetros (opcionais) para passar para o módulo e um sinalizador de controle que descreve como interpretar o código de retorno do módulo.

Entender os sinalizadores de controle é essencial para entender os arquivos de configuração do PAM. Existem quatro diferentes flags de controle:

`binding`
: Se o módulo tiver exito e nenhum módulo anterior na chain tiver falhado, a chain será encerrada imediatamente e a solicitação será concedida. Se o módulo falhar, o resto da chain é executado, mas a solicitação é negada no final.

Esta flag de controle foi introduzida pela Sun no Solaris™9 (SunOS™ 5.9), e também é suportado pelo OpenPAM.

`required`
Se o módulo tiver exito, o restante da chain será executada e a solicitação será concedida, a menos que algum outro módulo falhe. Se o módulo falhar, o restante da chain também será executado, mas a solicitação será negada no final.

`requisite`
Se o módulo for tiver exito, o restante da chain será executado e a solicitação será concedida, a menos que algum outro módulo falhe. Se o módulo falhar, a chain será encerrada imediatamente e a solicitação será negada.

`sufficient`
Se o módulo tiver exito e nenhum módulo anterior na chain tiver falhado, a chain será encerrada imediatamente e a solicitação será concedida. Se o módulo falhar, o módulo será ignorado e o resto da chain será executado.

Como a semântica dessa flag pode ser um pouco confusa, especialmente quando ela é usada para o último módulo em uma chain, é recomendado que a flag de controle `binding` seja usada em seu lugar, se a implementação o suportar.

`optional`
O módulo é executado, mas seu resultado é ignorado. Se todos os módulos em uma chain estiverem marcados como `optional` , todas as solicitações serão sempre concedidas.

Quando um servidor invoca uma das seis primitivas PAM, o PAM recupera a chain para o recurso ao qual a primitiva pertence, e invoca cada um dos módulos listados na chain, na ordem em que estão listados, até chegar ao fim ou determina que nenhum processamento adicional é necessário (porque um módulo `binding` ou `sufficient` teve exito, ou porque um módulo `requisite` falhou.) O pedido é concedido se e somente se pelo menos um módulo foi chamado e todos os módulos não opcionais tiveram exito.

Note que é possível, embora não muito comum, ter o mesmo módulo listado várias vezes na mesma chain. Por exemplo, um módulo que procura nomes de usuário e senhas em um servidor de diretório pode ser chamado várias vezes com parâmetros diferentes, especificando diferentes servidores de diretórios para contato. O PAM trata diferentes ocorrências do mesmo módulo na mesma chain de módulos diferentes e não relacionados.

3.4. Transações

O ciclo de vida de uma transação típica do PAM é descrito abaixo. Observe que, se qualquer uma dessas etapas falhar, o servidor deverá informar uma mensagem de erro adequada ao cliente e anular a transação.

1. Se necessário, o servidor obtém as credenciais do mediador por meio de um mecanismo independente do PAM - mais comumente em virtude de ter sido iniciado por `root` ou de ser setuid `root`.

2. O servidor chama pam_start(3) para inicializar a biblioteca PAM, especificar seu nome de serviço e a conta de destino e registrar uma função de conversação adequada.

3. O servidor obtém várias informações relacionadas à transação (como o nome de usuário do requerente e o nome do host no qual o cliente é executado) e o envia ao PAM usando pam_set_item(3).

4. O servidor chama pam_authenticate(3) para autenticar o requerente.

5. O servidor chama pam_acct_mgmt(3) para verificar se a conta solicitada está disponível e é válida. Se a senha estiver correta mas expirar, pam_acct_mgmt(3) retornará `PAM_NEW_AUTHTOK_REQD` em vez de `PAM_SUCCESS` .

6. Se a etapa anterior retornasse `PAM_NEW_AUTHTOK_REQD` , o servidor agora chamaria pam_chauthtok(3) para forçar o cliente a alterar o token de autenticação para a conta solicitada.

7. Agora que o requerente foi devidamente autenticado, o servidor chama pam_setcred(3) para estabelecer as credenciais da conta solicitada. É capaz de fazer isso porque age em nome do mediador e possui as credenciais do madiador.

8. Depois que as credenciais corretas forem estabelecidas, o servidor chamará pam_open_session(3) para configurar a sessão.

9. O servidor agora executa qualquer serviço solicitado pelo cliente - por exemplo, fornecer ao requerente um shell.

10. Quando o servidor terminar de atender ao cliente, ele chamará pam_close_session(3) para derrubar a sessão.

11. Finalmente, o servidor chama pam_end(3) para notificar a biblioteca PAM que ela esta pronta e que pode liberar quaisquer recursos alocados no curso da transação.

4. Configuração do PAM

4.1. Arquivos de política do PAM

4.1.1. O arquivo `/etc/pam.conf`

O arquivo de política tradicional do PAM é `/etc/pam.conf` . Este arquivo contém todas as políticas do PAM para o seu sistema. Cada linha do arquivo descreve uma etapa em uma chain, conforme mostrado abaixo:

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

Os campos estão, na ordem: nome do serviço, nome do recurso, flag de controle, nome do módulo e argumentos do módulo. Quaisquer campos adicionais são interpretados como argumentos adicionais do módulo.

Uma chain separada é construída para cada par de serviço/recurso, portanto, embora a ordem na qual as linhas para o mesmo serviço e recurso apareçam seja significativa, a ordem na qual os serviços e recursos individuais são listados não é. Os exemplos no artigo original do PAM agruparam as linhas de configuração por recurso, e o suporte do Solaris™ ao `pam.conf` ainda faz isso, mas a configuração de ações do FreeBSD configura as linhas por serviço. De qualquer maneira está bem; De qualquer forma, faz o mesmo sentido.

4.1.2. O diretório `/etc/pam.d`

O OpenPAM e o Linux-PAM suportam um mecanismo de configuração alternativo, que é o mecanismo preferido no FreeBSD. Neste esquema, cada política está contida em um arquivo separado com o nome do serviço ao qual se aplica. Esses arquivos são armazenados em `/etc/pam.d/` .

Esses arquivos de políticas por serviço possuem apenas quatro campos, em vez de cinco no `pam.conf` : o campo nome do serviço é omitido. Assim, em vez da linha de exemplo no `pam.conf` da seção anterior, a seguinte linha deve estar em `/etc/pam.d/login` :

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

Como consequência dessa sintaxe simplificada, é possível usar a mesma política para vários serviços vinculando cada nome de serviço a um mesmo arquivo de política. Por exemplo, para usar a mesma política para os serviços `su` e `sudo`, pode-se fazer o seguinte:

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

Isso funciona porque o nome do serviço é determinado a partir do nome do arquivo em vez de ser especificado no arquivo de políticas, portanto, o mesmo arquivo pode ser usado para vários serviços com nomes diferentes.

Como a política de cada serviço é armazenada em um arquivo separado, o mecanismo `pam.d` também facilita a instalação de políticas adicionais para pacotes de software de terceiros.

4.1.3. A ordem de pesquisa da política

Como vimos acima, as políticas do PAM podem ser encontradas em vários lugares. O que acontece se as políticas para o mesmo serviço existirem em vários lugares?

É essencial entender que o sistema de configuração do PAM está centrado em chains.

4.2. Quebra de uma linha de configuração

Como explicado em Seção 4.1, "Arquivos de política do PAM", cada linha em `/etc/pam.conf` consiste em quatro ou mais campos: o nome do serviço, o nome do recurso, a flag de controle, o nome do módulo e nenhum ou mais argumentos do módulo.

O nome do serviço é geralmente (embora nem sempre) o nome do aplicativo ao qual a instrução se aplica. Se não tiver certeza, consulte a documentação do aplicativo individual para determinar qual nome de serviço ele usa.

Note que se você usar `/etc/pam.d/` em vez de `/etc/pam.conf` , o nome do serviço é especificado pelo nome do arquivo de política e omitido a partir das linhas de configuração atuais, que então começam com o nome da instalação.

O recurso é uma das quatro palavras-chave do recurso descritas em Seção 3.1, "Recursos e Primitivas".

Da mesma forma, a flag de controle é uma das quatro palavras-chave descritas em Seção 3.3, "Cadeias e Políticas", descrevendo como interpretar o código de retorno do módulo. O Linux-PAM suporta uma sintaxe alternativa que permite especificar a ação para associar com cada código de retorno possível, mas isso deve ser evitado, pois não é padrão e está intimamente ligado à forma como o Linux-PAM envia chamadas de serviço (que difere muito da maneira que Solaris™ e OpenPAM fazem isso). Não surpreendentemente, o OpenPAM não suporta esta sintaxe.

4.3. Políticas

Para configurar o PAM corretamente, é essencial entender como as políticas são interpretadas.

Quando um aplicativo chama pam_start(3), a biblioteca PAM carrega a diretiva do serviço especificado e constrói quatro chains de módulos (uma para cada recurso). Se uma ou mais dessas chains estiverem vazias, as chains correspondentes da política para o `outro` serviço são substituídas.

Quando o aplicativo chama mais tarde uma das seis primitivas PAM, a biblioteca PAM recupera a chain para o recurso correspondente e chama a função de serviço apropriado em cada módulo listado na chain, na ordem em que foram listadas na configuração. Após cada chamada para uma função de serviço, o tipo de módulo e o código de erro retornado pela função de serviço são usados para determinar o que acontece a seguir. Com algumas exceções, discutidas abaixo, a tabela a seguir se aplica:

Tabela 1. Resumo de execução da cadeia PAM

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

Se `fail` for true no final de uma chain, ou quando um "break" for atingido, o dispatcher retornará o código de erro retornado pelo primeiro módulo que falhou. Caso contrário, retorna `PAM_SUCCESS` .

A primeira exceção é que o código de erro `PAM_NEW_AUTHTOK_REQD` é tratado como um sucesso, exceto que se nenhum módulo falhar e pelo menos um módulo retornar `PAM_NEW_AUTHTOK_REQD` , o dispatcher retornará `PAM_NEW_AUTHTOK_REQD` .

A segunda exceção é que pam_setcred(3) trata os módulos `binding` e `sufficient` como se eles fossem `required` .

A terceira e última exceção é que pam_chauthtok(3) executa a chain inteira duas vezes (uma vez para verificações preliminares e uma vez para definir a senha), e na fase preliminar, ele trata os módulos `binding` e `sufficient` como se fossem `required`.

5. Módulos PAM do FreeBSD

5.1. pam_deny(8)

O módulo pam_deny(8) é um dos módulos mais simples disponíveis; responde a qualquer pedido com `PAM_AUTH_ERR`. É útil para desabilitar rapidamente um serviço (adicioná-lo ao topo de cada chain) ou para encerrar chains de módulos `sufficient`.

5.2. pam_echo(8)

O módulo pam_echo(8) simplesmente passa seus argumentos para a função de conversação como uma mensagem `PAM_TEXT_INFO`. É principalmente útil para depuração, mas também pode servir para exibir mensagens como "O acesso não autorizado será processado" antes de iniciar o procedimento de autenticação.

5.3. pam_exec(8)

O módulo pam_exec(8) leva seu primeiro argumento a ser o nome de um programa a ser executado, e os argumentos restantes são passados para esse programa como argumentos de linha de comando. Uma aplicação possível é usá-lo para executar um programa no momento do login, que monta o diretório pessoal do usuário.

5.4. pam_ftpusers(8)

The pam_ftpusers(8) module

5.5. pam_group(8)

O módulo pam_group(8) aceita ou rejeita os requerentes com base em sua participação em um determinado grupo de arquivos (normalmente `wheel` para su(1)). Ele é destinado principalmente para manter o comportamento tradicional do su(1) do BSD, mas tem muitos outros usos, como excluir determinados grupos de usuários de um serviço particular.

5.6. pam_guest(8)

O módulo pam_guest(8) permite logins convidados usando nomes de login fixos. Vários requerimentos podem ser colocados na senha, mas o comportamento padrão é permitir qualquer senha, desde que o nome de login seja o de uma conta de convidado. O módulo pam_guest(8) pode ser facilmente utilizado para implementar logins FTP anônimos.

5.7. pam_krb5(8)

O módulo pam_krb5(8)

5.8. pam_ksu(8)

O módulo pam_ksu(8)

5.9. pam_lastlog(8)

O módulo pam_lastlog(8)

5.10. pam_login_access(8)

O módulo pam_login_access(8) fornece uma implementação da primitiva de gerenciamento de contas que impõe as restrições de login especificadas na tabela login.access(5).

5.11. pam_nologin(8)

O módulo pam_nologin(8) recusa logins não-root quando existe um `/var/run/nologin` . Este arquivo é normalmente criado por shutdown(8) quando restam menos de cinco minutos até o horário de encerramento programado.

5.12. pam_opie(8)

O módulo pam_opie(8) implementa o método de autenticação opie(4). O sistema opie(4) é um mecanismo de desafio-resposta em que a resposta a cada desafio é uma função direta do desafio e uma frase-senha, então a resposta pode ser facilmente computada "just in time" por qualquer pessoa que possua a senha, eliminando a necessidade de listas de senhas. Além disso, como opie(4) nunca reutiliza um desafio que tenha sido respondido corretamente, ele não é vulnerável a ataques de repetição.

5.13. pam_opieaccess(8)

O módulo pam_opieaccess(8) é um módulo complementar para pam_opie(8). Sua finalidade é impor as restrições codificadas em opieaccess(5), que regulam as condições sob as quais um usuário que normalmente se autenticaria usando opie(4) tem permissão para usar métodos alternativos. Isso geralmente é usado para proibir o uso de autenticação de senha de hosts não confiáveis.

Para ser eficaz, o módulo pam_opieaccess(8) deve ser listado como `requisite` imediatamente após uma entrada `sufficient` para pam_opie(8), e antes de qualquer outro módulo, na chain `auth`.

5.14. pam_passwdqc(8)

O módulo pam_passwdqc(8)

5.15. pam_permit(8)

O módulo pam_permit(8) é um dos módulos mais simples disponíveis; responde a qualquer pedido com `PAM_SUCCESS` . É útil como um espaço reservado para serviços onde uma ou mais chains estariam vazias.

5.16. pam_radius(8)

O módulo pam_radius(8)

5.17. pam_rhosts(8)

O módulo pam_rhosts(8)

5.18. pam_rootok(8)

O módulo pam_rootok(8) reporta sucesso se e somente se o ID do usuário real do processo que o chama (que é assumido como sendo executado pelo requerente) é 0. Isso é útil para serviços que não estão em rede, como su(1) ou passwd(1), para o qual o `root` deve ter acesso automático.

5.19. pam_securetty(8)

O módulo pam_securetty(8)

5.20. pam_self(8)

O módulo pam_self(8) reporta sucesso se, e somente se, os nomes do requerente coincidem com os da conta de destino. É mais útil para serviços que não estão em rede, como su(1), onde a identidade do requerente pode ser facilmente verificada.

5.21. pam_ssh(8)

O módulo pam_ssh(8) fornece serviços de autenticação e de sessão. O serviço de autenticação permite que os usuários que tenham chaves secretas SSH protegidas por senha em seu diretório `~/.ssh` se autentiquem digitando sua frase secreta. O serviço de sessão inicia o ssh-agent(1) e o pré-carrega com as chaves que foram descriptografadas na fase de autenticação. Esse recurso é particularmente útil para logins locais, seja em X (usando xdm(1) ou outro gerenciador de login X que reconhece o PAM) ou no console.

5.22. pam_tacplus(8)

O módulo pam_tacplus(8)

5.23. pam_unix(8)

O módulo pam_unix(8) implementa a autenticação de senha UNIX® tradicional, usando getpwnam(3) para obter a senha da conta de destino e compará-la com a fornecida pelo requerente. Ele também fornece serviços de gerenciamento de conta (impondo tempos de expiração de conta e senha) e serviços de alteração de senha. Este é provavelmente o módulo mais útil, já que a grande maioria dos administradores desejará manter um comportamento histórico para pelo menos alguns serviços.

6. Programação de Aplicação PAM

Esta seção ainda não foi escrita.

7. Programação de Módulos PAM

Esta seção ainda não foi escrita.

A. Exemplo de Aplicação PAM

O que vem a seguir é uma implementação mínima de su(1) utilizando o PAM. Observe que ele usa a função de conversa openpam_ttyconv(3) específica do OpenPAM, que é prototipada em `security/openpam.h` . Se você deseja construir este aplicativo em um sistema com uma biblioteca PAM diferente, você terá que fornecer sua própria função de conversação. Uma função de conversa robusta é surpreendentemente difícil de implementar; o apresentado em Apêndice C, *Exemplo de função de conversação PAM* é um bom ponto de partida, mas não deve ser usado em aplicações do 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 $
 * $FreeBSD$
 */

#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. Exemplo do módulo PAM

Segue-se uma implementação mínima de pam_unix(8), oferecendo apenas serviços de autenticação. Ele deve ser compilado e executado com a maioria das implementações do PAM, mas aproveita as extensões do OpenPAM, se disponível: observe o uso de pam_get_authtok(3), que simplifica enormemente solicitando ao usuário uma senha.

```
/*-
 * 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 $
 * $FreeBSD$
 */

#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. Exemplo de função de conversação PAM

A função de conversação apresentada abaixo é uma versão bastante simplificada do openpam_ttyconv() do OpenPAM. Ele é totalmente funcional e deve dar ao leitor uma boa ideia de como uma função de conversa deve se comportar, mas é simples demais para uso no mundo real. Mesmo se você não estiver usando o OpenPAM, sinta-se à vontade para baixar o código-fonte e adaptar openpam_ttyconv(3) aos seus usos; acreditamos que seja tão robusto quanto uma função de conversa orientada para tty pode razoavelmente ser.

```
/*-
 * 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.
 *
 * $FreeBSD$
 */

#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);
}
```

Leitura Adicional

Papers

[1] *Tornando os Serviços de Login Independentes das Tecnologias de Autenticação*. Vipin Samar e Charlie Lai. Sun Microsystems.

[2] *Especificação Preliminar do X/Open Single Sign-on*. O grupo aberto. 1-85912-144-6. Junho de 1997.

[3] *Módulos de Autenticação Plugáveis*. Andrew G. Morgan. 06-10-1999.

Manuais do usuário

[4] *Administração do PAM*. Sun Microsystems.

Páginas Web Relacionadas

[5] *Homepage do OpenPAM*. Dag-Erling Smørgrav. ThinkSec AS.

[6] *Página inicial do Linux-PAM*. Andrew G. Morgan.

[7] *Homepage do Solaris PAM*. Sun Microsystems.