Alisson Gomes Cerqueira

Implementação de Módulos PAM e NSS para Autenticação Segura e Distribuída

Monografia de Pós-Graduação “Lato Sensu” apresentada ao Departamento de Ciência da Computação para obtenção do título de Especialista em “Administração em Redes Linux”

Orientador
Prof. Joaquim Quintero Uchôa

Lavras
Minas Gerais - Brasil
2005

Alisson Gomes Cerqueira

Implementação de Módulos PAM e NSS para Autenticação Segura e Distribuída

Monografia de Pós-Graduação “Lato Sensu” apresentada ao Departamento de Ciência da Computação para obtenção do título de Especialista em “Administração em Redes Linux”

Aprovada em *11 de Setembro de 2005*

Prof. Douglas Machado Tavares

Prof. Samuel Pereira Dias

Prof. Joaquim Quintero Uchôa
(Orientador)

Lavras
Minas Gerais - Brasil

Agradecimentos

A todos que de alguma forma me ajudaram, muito obrigado!

Resumo

Este trabalho tem por objetivo a construção de um módulo NSS para a aplicação PAMRAD, que é um módulo PAM (*Plugable Authentication Modules*) para autenticação criptografada, distribuída e de fácil gerenciamento em estações Linux. Com esse módulo será possível usuários autenticarem remotamente em servidores Linux a partir de estações Linux, sem que haja necessidade de uma cópia local dos arquivos `/etc/passwd`, `/etc/shadow` e `/etc/group`. Normalmente a autenticação de um usuário em Linux pode ser feita localmente, comparando a senha fornecida pelo usuário com uma senha localizada em um arquivo na máquina local, ou por um servidor LDAP ou um servidor NIS, sendo que estes dois últimos possui um modelo centralizado das informações. Como os atuais meios de autenticação ou são inseguros ou há um enorme esforço para configurá-lo de forma segura decidiu-se criar um novo modelo de autenticação completo. Para concluir o processo de autenticação remota, a partir do PAMRAD, será desenvolvido um módulo NSS que possibilitará as várias aplicações existentes em Linux descobrirem as informações dos usuários sem a necessidade de replicar informações ou reescrever as aplicações.

Dedico seguramente a minha autêntica mãe.

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

Atualmente, as redes de computadores estão muito presente no dia-a-dia das pessoas que as utilizam, seja no escritório, nas empresas e até mesmo nas casas. O uso da rede tem por objetivo o compartilhamento de recursos, como internet, sistemas de arquivos, impressoras entre outros. Junto ao uso das redes tornou-se comum ao usar um computador ter primeiro que digitar se identificar como usuário válido através de senha (mais conhecido como *login*). Isso permite que o computador identifique o usuário que está entrando na rede e busque algumas informações extras, como diretório de trabalho do usuário, grupos a que ele pertence etc. Sendo assim, cada estação deve de alguma forma saber quem são seus usuários, suas senhas e as informações adicionais. O problema está justamente em como guardar e administrar os nomes dos usuários, senhas e outras informações de forma segura e de fácil administração. Imagine os cenários:

- As informações seriam guardadas em cada máquina da rede e a validação do usuário/senha poderia ocorrer localmente na estação.
- Num segundo cenário, as informações estariam todas centralizadas num único computador da rede (servidor) e a validação seria realizada neste servidor.

No primeiro cenário há um processo seguro para o *login* do usuário, no entanto a administração dos usuários da rede seria um trabalho muito difícil, pois a cada usuário que fosse fazer parte da rede ou fosse sair seria necessário atualizar os arquivos de usuário e senha em todos os computadores pertencente à rede. Também, caso um usuário trocasse a senha na estação a senha seria trocada apenas naquele computador local, se ele fosse entrar num outro computador teria que usar a senha antiga.

No segundo cenário, a administração das informações da rede (*passwd*, *shadow*, *groups*, etc.) fica mais fácil de gerenciar, pois elas ficariam centralizadas. No entanto as opções para construir este cenário é uma tarefa não trivial nos sistemas GNU/Linux. Desta forma, este trabalho aborda, de forma sucinta, as principais soluções existentes atualmente e apresenta uma nova forma de centralizar as informações dos usuários numa rede de forma segura e de fácil administração.

Este trabalho foi organizado da seguinte forma: o Capítulo 2 apresenta os conceitos de autenticação, autorização e segurança. No Capítulo 3 explica-se como uma comunicação cliente/servidor pode ser realizada por meio de *sockets* e a segurança realizada com *Secure Sockets Layer* (SSL). O Capítulo 4 expõe como o processo de autenticação é realizado em sistemas GNU/Linux, como as informações dos usuários são recuperadas e como escrever um módulo *Name Service Switch* (NSS). No Capítulo 5 é apresentado a implementação do *PAM Remote Authentication Daemon* (PAMRAD) e como pode ser configurado. Por fim, as conclusões são apresentadas no Capítulo 6, com indicação de trabalhos futuros.

Capítulo 2

Autenticações Linux

Para que os usuários tenham acesso a uma estação numa rede é preciso que ele seja autenticado, ou seja é necessário verificar se ele é quem diz ser, para isto o usuário fornece suas credenciais, muitas vezes, composta de nome e senha. E para fazer uso dos recursos do computador e da rede é necessário autorização, ou seja, o sistema determina aquilo que pode ser usado pelo usuário e o que pode ser feito, ler, escrever ou executar.

De acordo (CONECTIVA S.A., 2004) nos sistemas linux, originalmente, a autenticação era realizada por meio de senhas criptografadas no arquivo local chamado `/etc/passwd`. Um programa como o `login` pedia o nome do usuário e a senha, criptografava a senha e comparava o resultado com o armazenado naquele arquivo. Além disso, caso o usuário desejasse realizar uma autenticação remota com o `ssh` seria este o responsável pela autenticação do usuário, ou seja, cada programa continha a forma que a autenticação era realizada dentro do próprio programa. Caso a forma de autenticação mudasse, uma autenticação pelas digitais do usuário, por exemplo, ou precisasse atualizar o algoritmo de criptografia todos os programas de autenticação deveriam ser reescritos para suportar as novas mudanças. Esta dependência implicava numa enorme inflexibilidade ao modelo de autenticação ou dificuldade de manutenção.

Pensando na dificuldade de configuração, manutenção e atualização deste mecanismo de autenticação a Sun®. desenvolveu um novo mecanismo, no qual serviços de entrada no sistema tal como `login`, `rlogin` e `telnet` deveriam ser customizados para incorporar as mudanças (SAMAR; LAI, 1996). O *framework Pluggable Authentication Module* (PAM) permite que várias tecnologias possam ser adicionadas sem mudar qualquer serviço de *login*, desse modo preservando os ambientes existentes. Mais informações sobre PAM serão discutidas no Capítulo 4.

2.1 Autenticações Remotas em Sistemas GNU/Linux

O GNU/linux já possui alguns meios de autenticação remota, entre eles os mais conhecidos são o *Lightweight Directory Access Protocol* (LDAP) e *Network Information Service* (NIS).

2.1.1 Lightweight Directory Access Protocol

De acordo com a definição de (MALèRE, 2004) *Lightweight Directory Access Protocol* (LDAP), é um protocolo cliente/servidor para acessar serviços de Diretório, originalmente foi desenvolvido como um *front-end* para o X.500[1]. LDAP trabalha em cima do protoloco TCP/IP ou outro serviço de transferência orientado a conexão. O protocolo LDAP está definido na *Request for Comments* (RFC) 2251 – (WAHL; KILLE, 1997).

Um Diretório é parecido com um banco de dados, mas tende a ser mais descritivo, as informações são baseadas em atributos e organizadas numa estrutura de árvore. As informações no Diretório se caracterizam por serem lidas muito frequentemente, ou seja, são bastante consultadas, mas dificilmente são alteradas. Serviços de Diretórios são implementados para poderem responder rapidamente, mesmo quando há um grande volume de informações e operações de consulta. Um implementação aberta, *open-source*, do protocolo LDAP é o OpenLDAP[2].

Funcionamento do LDAP

No serviço de Diretório LDAP um ou mais servidores LDAP possuem os dados numa árvore de Diretório LDAP. O cliente conecta no servidor LDAP e "pergunta" alguma coisa, o servidor responde ou então aponta para um outro servidor LDAP para que o cliente possa pegar a resposta.

Autenticação com LDAP

Para acessar um serviço LDAP, primeiro é preciso que o cliente se autentique no serviço, isto é, ele deve dizer ao servidor LDAP quem está acessando os dados, com posse desta informação o servidor pode decidir o que é permito ao cliente visualizar e fazer. Após uma autenticação realizada com sucesso pelo cliente cada nova requisição ao servidor LDAP é verificado se o cliente tem permissão para fazer esta requisição, este processo é chamado de controle de acesso.

[1] X.500 define o *Directory Access Protocol* (DAP), ou Protocolo de Acesso a Diretório, para clientes quando eles entrarem em contato com um servidor de diretório.

[2] http://www.openldap.org/

No LDAP, a autenticação é realizada por meio de uma "ligação" que o cliente faz ao servidor. LDAP v3 suporta três tipos de autenticação: **anônima**, **simples** e ***Simple Authentication and Security Layer* (SASL – RFC 2222 (MYERS, 1997))** , sendo que a RFC 2222 define uma série de mecanismos de autenticação SASL, como Kerberos, GSSAPI e S/Key. O LDAP v2 possui a autenticação **anônima**, **simples** (senha em texto claro) e **Kerberos v4** (Apêndice A.1.1). Portanto, é importante tomar cuidado com a versão LDAP que está sendo utilizada, pois diferentes versões suportam diferentes tipos de autenticação.

Quando um cliente envia uma requisição LDAP sem fazer a "ligação" ele é tratado como anônimo. No protocolo LDAP v2 o cliente inicia a conexão com o servidor LDAP com a operação de "ligação" que contém informações para ser autenticado. A autenticação simples consiste em enviar ao servidor LDAP o DN[3] do cliente (usuário) e a sua senha em texto claro (sem criptografia). Este mecanismo tem problemas de segurança porque a senha pode ser lida por alguém mal intencionado na rede. Para evitar a exposição da senha na autenticação simples é recomendado usar um canal de comunicação encriptado (tal com SSL[4]), o qual é suportado por servidores LDAP.

O SASL – RFC 2222 (MYERS, 1997) – especifíca um protocolo baseado em desafio-resposta (*challenge-response*) no qual dados são trocados entre o cliente e o servidor com o propósito de autenticação do cliente e estabelecer um nível de segurança da comunição com o cliente. Na prática do SASL, LDAP pode suportar qualquer tipo de autenticação desde que esteja de acordo com o cliente e servidor LDAP. O pacote *Cyrus-SASL*, uma implementação bastante usada do SASL em Linux, está disponível no endereço
`http://asg.web.cmu.edu/sasl/sasl-library.html`.

Além da autenticação dos usuários para acessar informações da árvore de Diretório, o servidor LDAP pode autenticar usuários de outros serviços (`Sendmail`, `Login`, `Ftp`, etc.). A migração das informações do usuário para o servidor LDAP pode ser feita usando PAM, detalhes Capítulo 4. O modulo de autenticação PAM para LDAP está disponível no endereço:
`http://www.padl.com/OSS/pam_ldap.html`

Para mais informações de como usar e configurar a autenticação com LDAP usando OpenLdap consulte (CONECTIVA S.A., 2004) e (MALèRE, 2004).

[3] *Domain Name*
[4] Secure Sockets Layer , (Apêndice 3.2).

2.1.2 Network Information Service

O *Network Information Service* (NIS) tem como principal objetivo compartilhar as informações (`passwd`, `shadow`, `groups`, etc.) na rede a partir do servidor de modo transparente. Desta forma os usuários da rede podem fazer o *login* em qualquer estação, que esteja usando o cliente NIS.

NIS foi desenvolvido para ser independente de DNS[5] e tem uma leve diferença. Enquanto o foco do DNS está em fazer simples comunicação usando nome de *hosts* ao invés de endereços IP, o foco do NIS está em fazer com que a administração da rede seja melhor gerenciável por meio de um controle centralizado das informações sobre a rede. NIS não armazena apenas informações sobre nome do *host* e endereços, mas também sobre usuários (`/etc/passwd`), a própria rede (`/etc/hosts`) e outros serviços disponíveis na rede.

Tipos de *Host* NIS

Há três tipo de *host* NIS:

- Servidor mestre.
- Servidor escravo.
- Clientes de servidor NIS.

Qualquer estação na rede pode ser um cliente NIS, mas apenas computadores com capacidade de armazenamento podem ser um servidor NIS, seja mestre ou escravo. Um servidor pode atuar como cliente e servidor, caso seja necessário.

Servidor NIS

Um servidor NIS é um computador que responde as requisições para o serviço NIS. Há duas variedades de servidor NIS, o mestre (*master*) e o escravo (*slave*). O computador designado como servidor mestre possui um conjunto de mapas que o administrador de sistema cria e atualiza quando necessário. Cada domínio NIS deve ter um, e somente um, servidor mestre, que pode propagar atualizações com baixa degradação no desempenho da rede (SUN MICROSYSTEMS, 2003).

É possível adicionar outros servidores NIS no domínio como servidores escravo. Um servidor escravo tem uma cópia completa do conjunto de mapas NIS do servidor mestre. Sempre que o servidor mestre tiver seus mapas atualizados

[5] *Domain Name System* é um sistema que armazena informações sobre nome de *hosts* e domínios, um tipo de banco de dados distribuído na rede.

será propagado entre os servidores escravo. Estes ainda podem assumir qualquer sobrecarga de requisições do servidor mestre, minimizando os erros de "servidor indisponível".

Um recurso interessante do NIS é que o administrador do sistema pode atribuir a cada mapa NIS um servidor mestre a ele, pois cada mapa possui o nome do servidor mestre que responderá àquele mapa. Desta forma é possível designar diferentes servidores para atuar como mestre e escravo para diferentes mapas.

Cliente NIS

Clientes NIS executam processos que requisita as informações dos mapas no servidor. Os clientes não fazem distinção entre um servidor mestre ou escravo, desde que todos servidores NIS tenham a mesma informação.

De acordo com (LIMA, 2003),

> *Se a segurança do sistema for importante não se deve usar o NIS (NEMETH; HEIN, 2001). As informações compartilhadas pelo NIS trafegam livremente pela rede sem criptografia alguma, então qualquer um pode interceptar essas informações, como senha de usuários. Mesmo essas senhas sendo originalmente criptografadas é relativamente fácil descobrir senhas pobres, isto é, senhas mal elaboradas. Outro problema grave do NIS é que qualquer máquina na rede pode se passar como um servidor NIS, podendo passar para os clientes informações administrativas (senhas, por exemplo) falsas.*

Para mais informações sobre NIS consulte (SUN MICROSYSTEMS, 2003) e `http://www.linux-nis.org/nis/`

2.2 Comentários Finais

Se a segurança do sistema for importante não se deve usar o NIS [ENH01]. Um problema do NIS, relacionado a segurança, é que as informações compartilhadas pelo NIS trafegam livremente pela rede sem criptografia alguma, então qualquer um pode interceptar essas informações. Outro problema grave do NIS é que qualquer máquina na rede pode se passar como um servidor NIS, podendo passar para os clientes informações administrativas (senhas, por exemplo) falsas (LIMA, 2003).

Ainda citando (LIMA, 2003), o protocolo LDAP não foi desenvolvido com a finalidade de autenticação. Desta forma, não se sabe se o LDAP vai se desenvolver na direção de ajudar os administradores de sistemas ou não. Por esta

razão, implementar autenticação distribuída e segura utilizando LDAP não é uma tarefa fácil. A documentação para esta tarefa também precisa ser melhorada, pois vários administradores interessados em utilizá-lo reclamam e sofrem para poder ter LDAP funcionando como entidade autenticadora em sua rede.

Capítulo 3

Comunicação Segura

3.1 Sockets

Socket é um mecanismo de comunicação cliente/servidor, podendo ser desenvolvido para comunicar-se localmente, entre os processos[1] de um sistema operacional, ou por meio de uma rede de computadores. Funções Linux como `printing` e aplicações como `rlogin` e `ftp` freqüentemente usam *sockets* para se comunicar (MATTHEW; RICHARD, 2004). O *socket* possui uma clara distinção entre quem é o servidor e quem é o cliente, podendo vários clientes se comunicar com um único servidor.

Para um melhor entendimento sobre *sockets*, é interessante pensar nele como se fosse uma porta de um canal de comunicação que permite a um processo em execução enviar/receber dados para/de outro processo que pode estar sendo executado no mesmo computador ou num outro remotamente, como exemplificado na Figura 3.1.

A Figura 3.2 mostra ações realizadas numa comunicação com *socket* em uma comunicação comum cliente/servidor.

3.2 Secure Sockets Layer

O *Secure Socket Layer* (SSL, atualmente na versão 3) é um protocolo de comunicação que implementa um canal seguro para comunicação entre aplicações, de forma transparente e independente da plataforma. Desta forma ele provê um mecanismo genérico de segurança, na camada de transporte.

[1] A especificação UNIX98 versão 2 define um processo como um espaço de endereçamento com uma ou mais *threads* executando naquele espaço de endereço, e os recursos do sistema são requisitados pelas *threads*.

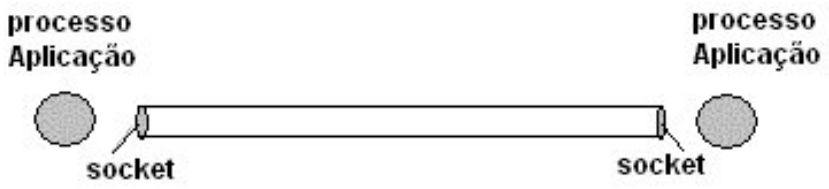

Figura 3.1: Modelo Comunicação Socket.

O SSL foi desenvolvido pela *Netscape Communications* em sua versão inicial em julho de 1994. Sua especificação foi submetida ao grupo de trabalho *World Wide Web Consortium* (W3C[2]) – RFC 2246 (DIERKS; ALLEN, 1999).

Sua proposta é permitir a autenticação de servidores, encriptação de dados, integridade de mensagens e, como opção, a autenticação do cliente, operando nas comunicações entre aplicativos. O SSL visa garantir os seguintes objetivos:

Segurança criptográfica para conexões assegurando a privacidade na conexão, com a utilização de algoritmos simétricos (como o DES – (SCHNEIER, 1996) – ou RC4 que negociam uma chave secreta na primeira fase do *handshaking* (usando chaves públicas assimétricas);

Autenticação do Servidor/Cliente por meio de algoritmos assimétricos como o RSA (Apêndice A.2.3) ou o DES (SCHNEIER, 1996);

Confiabilidade na conexão com o uso de Códigos de Autenticação de Mensagens (MAC).

Este projeto utiliza o OpenSSL como implementação do SSL. O projeto OpenSSL é um esforço colaborativo para desenvolver um *toolkit*[3] robusto e completo de código aberto. Ele implementa o SSL v2 e v3 e o protocolo *Transport Layer Security* (TLS v1). Para mais informações consulte `http://www.openssl.org/`. Ele foi escolhido por ser um *software* livre com uma excelente implementação do SSL, pois foi baseada na implementação da biblioteca `SSLeay` desenvolvida por Eric A. Young e Tim J. Hudson (OPENSSL PROJECT,).

[2] O W3C desenvolve tecnologias padrões para a criação e a interpretação dos conteúdos para Web, informações `http://www.w3.org/`

[3] *Toolkit* é um conjunto de progamas e procedimentos auxiliando no desenvolvimento de outros programas de computadores.

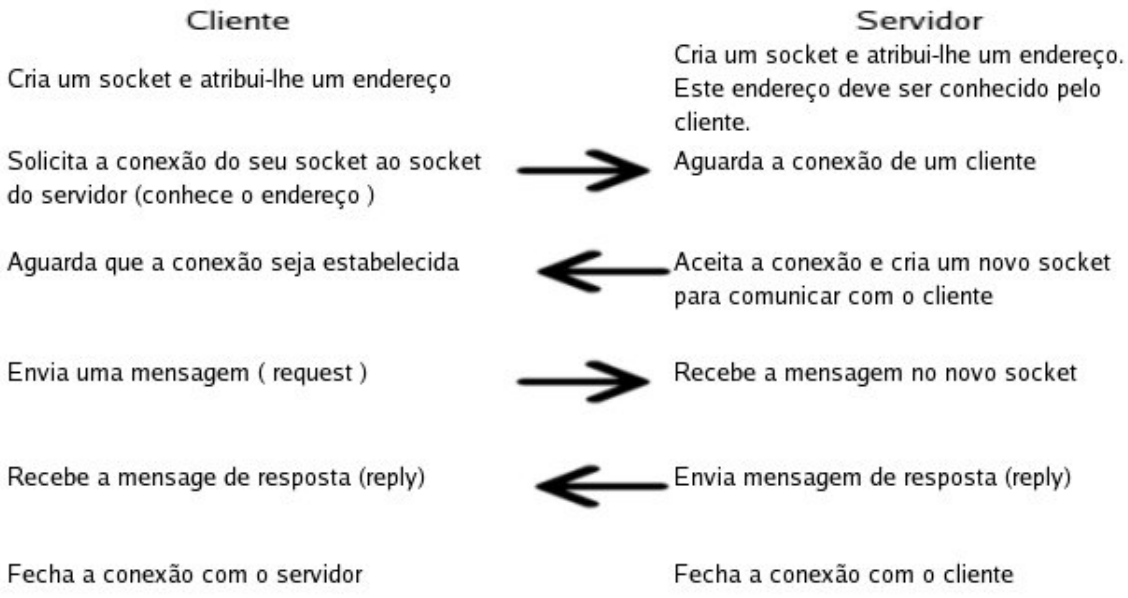

Figura 3.2: Modelo Comunicação Socket.

Capítulo 4

Plugable Authentication Modules e *Name Service Switch*

Com o objetivo de eliminar a depedência entre os aplicativos de autenticação e o modelo de autenticação utilizada, a SUN criou o *Pluggable Authentication Modules* (PAM) em 1995 e divulgou as especificações em um RFC 86.0 – (SAMAR; SCHEMERS, 1995). Com base neste documento, foi realizada a implementação do PAM para Linux, que ficou conhecido como Linux-PAM.

4.1 *Plugable Authentication Modules*

Com o Linux-PAM diversos modelos de autenticação podem ser usados sem ter que mudar qualquer serviço de *login*, pois agora, a autenticação não está codificada dentro dos aplicativos. Linux-PAM pode ser usada para integrar serviços de *login* com diferentes tecnologias de autenticação, tal como RSA, DCE, Kerberos, S/Key, etc.

De acordo com (SAMAR; LAI, 1996) os objetivos da autenticação PAM são:

- O administrador do sistema deverá ser capaz de escolher o mecanismo de autenticação padrão para cada máquina. Isso pode ir de um simples mecanismo baseado em senha até um método biométrico ou um sistema baseado em *smart card*.
- O administrador deverá ser capaz de configurar um mecanismo de autenticação para o usuário com base na aplicação. Por exemplo, um *site* pode requisitar uma autenticação *S/KEY* (SCHNEIER, 1996) para acesso `telnet`, enquanto permite sessões de *login* via console apenas com a autenticação UNIX.

- O *framework*[1] deverá suportar requisitos gráficos de uma aplicação. Por exemplo, para uma sessão de *login* com interface gráfica, o usuário poderá entrar com o nome, e a senha deverá ser digitada numa outra tela. Para serviços de rede, como, `ftp` e `telnet`, o nome e a senha do usuário devem ser transmitidas pela rede até o servidor.

- O administrador poderá ser capaz de configurar vários protocolos de autenticação para cada aplicação. Por exemplo, o administrador pode querer que os usuários sejam a autenticados por `kerberos` e RSA ao mesmo tempo.

- O administrador poderá configurar o sistema que o usuário é autenticado com todos os protocolos de autenticação sem a nececissade do usuário repetir a senha.

- Os serviços de entrada no sistema não tem que ser modificado quando o mecanismo de autenticação em questão mudar. Isto é muito útil pois nem sempre se tem o código fonte destes serviços.

- A arquitetura deverá fornecer um modelo de autenticação *pluggable* (acopláveis), bem como outras tarefas relacionadas, tal como gerenciamento de senha, conta e sessão.

- Por razões de compatibilidade aos sistemas antigos, a arquitetura deverá suportar o modelo de autenticação dos serviços de entrada das antigas aplicações.

- A API[2] deverá ser independente do sistema operacional.

Não são responsabilidades do PAM:

- O problema de como identificar, uma vez estabelecido, comunicação com outras partes interessadas (também conhecido como *single-sign-on*[3]). O foco do projeto PAM está em fornecer um esquema genérico através do qual usuários possam definir sua identidade a uma máquina. O *Generic Security Services Application Programming Interface* (GSS-API) pode ser usado

[1] No desenvolvimento de *software*, *framework* é definido como uma estrutura de suporte no qual outros projetos de *software* podem ser organizados e desenvolvidos. Um *framework* pode ser programas de suporte, bibliotecas ou um outro *software* que ajuda no desenvolvimento e se acopla a diferentes componentes de um projeto de *software*.

[2] *Application Programming Interface* (API) é um conjunto normalizado de rotinas e chamadas de *software* que podem ser referenciadas por um programa aplicativo para acessar serviços.

[3] Single sign-on (SSO) é uma forma especializada de autenticação de sistemas que permite um usuário autenticar uma vez e ganhar acesso a outros sistemas.

por aplicações para comunicação segura e autenticada sem conhecimento do mecanismo base de identificação do usuário.

- O problema de enviar senha pela rede em texto claro.

4.1.1 Funcionamento do PAM

O coração dos componentes do *framework* PAM está na biblioteca de autenticação API (o *front-end*[4]) e no módulo de autenticação (o *back end*[5]), conectados por meio de um *Service Provider Inteface* (SPI). As aplicações são escritas para a API PAM, enquanto aqueles que executam a autenticação são escritas para o SPI PAM e retornam ao módulo que é independentes da aplicação.

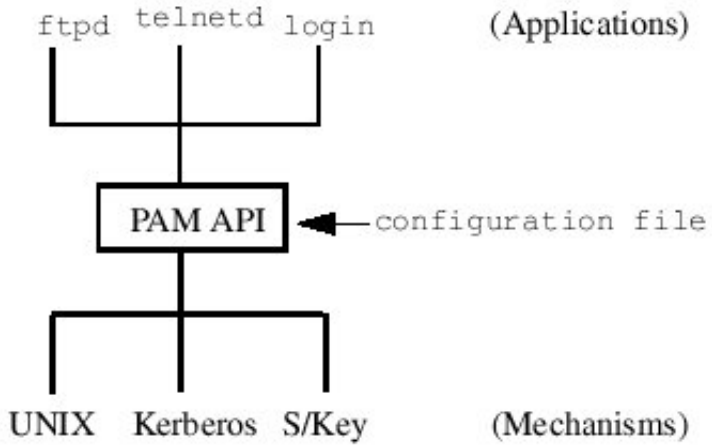

Figura 4.1: Arquitetura Básica do PAM (SAMAR; LAI, 1996).

A Figura 4.1 mostra a relação entre uma aplicação, o módulo PAM e os mecanismo de autenticação. As três aplicações ilustradas, `login`, `telnetd` e `ftpd` usam as interfaces[6] de autenticação PAM. Quando uma aplicação chama a API PAM, ela carrega o módulo apropriado para autenticação, como determinado no arquivo de configuração, então, a requisição é passada para o mecanismo responsável pela autenticação, por exemplo, UNIX *password*, Kerberos, S/Key, para exe-

[4] *Front-end* é o mesmo que interface com o usuário, podendo ser uma interface com o usuário final de uma aplicação ou o usuário de uma API.

[5] *Back end* é a parte que processa a entrada do *front-end*.

[6] No presente texto o termo *interface* se refere as interfaces definidas numa API ou especificação, enquanto o termo *função* é utilizada para uma interface já implementada de acordo sua especificação.

cutar a operação especificada. A camada PAM então devolve a resposta do mecanismo de autenticação para a aplicação.

PAM unifica o sistema de autenticação e o controle de acesso, permitindo acoplar módulos de autenticação através de interfaces bem definida. O acoplamento pode ser definido por vários meios, um destes usa arquivos de configuração, que será apresentado na Seção 4.1.3.

Um módulo PAM possui um conjunto de seis funções que devem ser agrupadas em quatro grupos de gerenciamento de forma independente, são eles: *gerenciamento de autenticação*, *gerenciamento de conta*, *gerenciamento de sessão* e *gerenciamento de senha* (SAMAR; LAI, 1996). Um módulo deve definir todas as funções com pelo menos um destes grupos. Um único módulo deve conter, necessariamente, funções para todos quatro grupos. Por exemplo, o programa `login` pode querer verificar além da senha (*gerenciamento de autenticação e senha*), se a conta do usuário já expirou (*gerenciamento de conta*), também pode guardar informações sobre a sessão do usuário (*gerenciamento de sessão*), desta forma o programa `login` precisa acessar os quatro gerenciamento do PAM. No entanto, um programa como `su` pode apenas acessar o *gerenciamento de autenticação*. Ele permite assim, algumas aplicações usar um gerenciamento de senha e autenticação específico mas compartilhar o gerenciamento de conta e sessão de outros módulos, por esta razão são independentes, podendo ser implementados cada um em separado e *pluggable*.

Segue uma breve descrição dos quatro tipos de gerenciamento PAM:

Gerenciamento de Autenticação possui a interface

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

para verificar a identidade do usuário corrente, e a interface

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

usada para definir as credenciais do processo corrente associado com a autenticação, '`pamh`'. A ação realizada é denotada por '`flag`', podendo definir, iniciar, atualizar, reiniciar ou destruir as credenciais do usuário.

Gerenciamento de Conta possui a interface

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

para verificar se o usuário autenticado deverá ganhar acesso a sua conta. Esta função pode implementar expiração da conta e restrições de acesso com base no horário.

Gerenciamento de Sessão possui as interfaces

```
int
pam_open_session(pam_handle_t *pamh, int flags);
```
e
```
int
pam_close_session(pam_handle_t *pamh, int flags);
```
a primeira deve ser chamada para informar ao módulo que uma nova sessão foi iniciada. Todos programas que usa PAM deverá invocá-la quando iniciar uma nova sessão. A segunda deve ser chamada após o término de uma sessão, ela deverá ser invocada para informar ao principal módulo que a sessão foi terminada.

Gerenciamento de Senha possui a interface

```
int
pam_chauthok(pam_handle_t *pamh, int flags);
```
é chamada para mudar o '*token*' de autenticação associado ao usuário referenciado pelo manipulador da autenticação '`pamh`'.

Mais detalhes sobre a interface pode ser consultada em (SAMAR; LAI, 1996) e (SAMAR; SCHEMERS, 1995).

4.1.2 Interface *Framework* PAM

O *framework* PAM provê um conjunto de interfaces para dar suporte aos módulos já citados e fornecer comunicação entre módulos e aplicações.

Interfaces Administrativas

O conjunto de transações PAM inicia com uma função `pam_start()` e termina com uma função `pam_end()`. As interfaces `pam_get_item()` e `pam_set_item()` são usadas para ler e escrever informações associadas com uma transação PAM. Caso um erro ocorra com qualquer uma das interfaces PAM, a mensagem de erro poderá ser impressa com `pam_strerror()`.

Módulo Comunicação da Aplicação

Durante o início da aplicação algumas informações, tal como o nome do usuário, é salva no *framework* PAM através da função `pam_start()` que pode ser utilizada por módulos base. A aplicação pode passar dados para o módulo e este passa de volta enquanto comunica com o usuário. Módulos PAM podem se comunicar com uma aplicação por meio de variáveis de ambiente, associadas por um nome através

da API `pam_putenv()` para escrever um valor. A aplicação poderá ler a variável associada com as APIs `pam_getenv()` e `pam_getenvlist()`.

Módulo de Comunicação com Usuário

A função `pam_start()` também passa uma chamada de função de volta, conhecida como função de conversação, a ser usada mais tarde pelo módulo base para ler e escrever informações de autenticação. Por exemplo, essas funções podem enviar uma mensagem perguntando qual é a senha, sendo que isto é determinado pela aplicação. Desta forma, aplicações com ou sem interface gráfica podem usar o PAM.

Comunicação entre Módulos

Embora os módulos sejam independentes, eles podem compartilhar informações em comum, sobre a sessão da autenticação, tal com, nome do usuário, nome do serviço, senha, e função de conversação através das interfaces `pam_get_item()` e `pam_set_item()`. A aplicação também pode usar essas funções (APIs) para trocar o estado de uma informação depois de ter chamado a função `pam_start()` pela primeira vez.

Módulo Estado da Informação

O módulo de serviço PAM pode querer manter determinados estados de algumas informações na sessão PAM. O módulo de serviço pode usar as interfaces `pam_get_data()` e `pam_set_data()` para acessar e atualizar informações PAM de um específico módulo. Os módulos PAM são carregados de acordo a demanda, sendo assim não existe uma iniciação dos módulos por parte do *framework* PAM. Se o módulo de serviço PAM tem que iniciar determinadas tarefas isto deve ser feito quando o módulo é chamado pela primeira vez. Contudo, se determinadas tarefas devem ser feitas quando a sessão do usuário é finalizada, o módulo deverá usar `pam_set_data()` para especificar as funções que serão chamadas quando a aplicação chamar `pam_end()`.

4.1.3 Configuração PAM

É comum encontrar configurações do PAM num único arquivo `/etc/pam.conf`, onde todas as aplicações são configuradas. Porém, nas distribuições GNU/Linux mais recentes a configuração é definida no diretório `/etc/pam.d`, sendo que cada serviço possui um arquivo com seu nome. Somente este modelo será explicado, para mais informações consulte o manual do PAM com o comando `man`

pam. Caso exista o diretório /etc/pam.d pode ser que a configuração do arquivo /etc/pam.conf seja ignorada.

Os valores no arquivo de configuração do Linux-PAM não diferenciam entre maiúscula e minúscula, exceto o valor do caminho onde se encontra o módulo. Há dois caracteres especiais que podem ser usados no arquivo de configuração:

- # – toda linha que inicia com este caractere é um comentário.
- \ – indica que a configuração do serviço continua na próxima linha.

Como já foi dito, cada arquivo no diretório /etc/pam.d possui o nome do serviço, por exemplo, para o serviço login haverá um arquivo /etc/pam.d/login. Cada arquivo possui quatro colunas, sendo que a última é opcional , com o seguinte formato:

Tipo módulo — Sinal de controle — caminho do módulo — argumentos

Tipo do módulo Há quatro tipos de módulo:

- auth Este módulo fornece dois aspectos de autenticação do usuário. Primeiro, ele estabelece se o usuário é quem ele diz ser, instruindo a aplicação a pedir uma senha ao usuário ou outra forma de identificação. Segundo, o módulo pode conceder a membros do grupo outros privilégios através de suas credenciais.
- account este módulo é usado basicamente para restringir/permitir acesso ao serviço, podendo ser com base na hora do dia, número máximo de recursos por usuário, etc.
- session primeiramente, este módulo está associado a executar coisas que precisam ser feitas antes e depois que o serviço é liberado. Coisas como, registrar informações (*log*) a respeito de abertura/fechamento de dados trocados com o usuário, montagem de diretórios, etc.
- password por último, este módulo é usado para atualizar o *token* de autenticação associado ao usuário.

Sinal de controle *(Control-flag)* O sinal de controle é utilizada para indicar de que forma a biblioteca do PAM reagirá ao sucesso ou falha do módulo que está associado. Outra função que o sinal de controle pode executar é dar prioridades a cada módulo, visto que eles podem ser empilhados. Existem dois modos de sintaxe para o sinal de controle um mais antigo e o tradicional que divide a sintaxe em: REQUIRED, REQUISITE, SUFFICIENT, OPTIONAL. O modo mais novo, mais elaborado e específico separa da seguinte forma: VALUE=ACTION. A parte ACTION é nomeada como: IGNORE, BAD, DIE, OK, DONE, RESET. Para um melhor entendimento sobre sinal de controle consulte (UCHôA, 2003) e (MORGAN, 2002).

Caminho do Módulo (*Module Path*) Localização do módulo PAM no sistema, geralmente, esses módulos ficam no diretório `/lib/security`, alguns dos módulos disponíveis são: *pam_access, pam_chroot, pam_cracklib, pam_deny, pam_env, pam_filter, pam_ftp.so, pam_group, pam_issue, pam_krb4, pam_lastlog, pam_limits, pam_listfile, pam_mail, pam_mkhomedir, pam_motd, pam_nologin, pam_permit, pam_pwdb, pam_radius, pam_rhosts_auth, pam_rootok, pam_securetty, pam_tally, pam_time, pam_unix, pam_userdb, pam_warn, pam_wheel*. Para saber mais sobre cada módulo consulte (MORGAN, 2002).

argumentos Os argumentos fazem parte de uma lista de símbolos que podem ser colocados ao final de cada módulo, se desejado. Estes argumentos, provavelmente, são entendidos por qualquer módulo. Exemplos:

`debug` utiliza a chamada do `syslog(3)` para guardar informações no arquivo de *log* do sistema.

`no_warn` instrui o módulo a não guardar mensagens de alerta (*warning)* para a aplicação.

Existem outros argumentos (`use_first_pass`, `try_first_pass`, `use_mapped_pass`, `expose_account`), para verificar o que cada um faz consulte (MORGAN, 2002).

A Tabela 4.1 mostra um arquivo de configuração para uma aplicação de autenticação, cada entrada representa uma linha do arquivo.

Tabela 4.1: Exemplo Configuração Módulo PAM

Tipo módulo	Sinal	Módulo e argumentos
auth	required	/lib/security/$ISA/pam_env.so
auth	sufficient	/lib/security/$ISA/pam_unix.so likeauth nullok
auth	required	/lib/security/$ISA/pam_deny.so
account	sufficient	/lib/security/$ISA/pam_succeed_if.so uid < 1000
account	required	/lib/security/$ISA/pam_unix.so
password	requisite	/lib/security/$ISA/pam_cracklib.so retry=3
password	sufficient	/lib/security/$ISA/pam_unix.so nullok use_authtok md5 \ shadow
password	required	/lib/security/$ISA/pam_deny.so
session	required	/lib/security/$ISA/pam_limits.so
session	required	/lib/security/$ISA/pam_unix.so

4.2 Name Service Switch

O *Name Service Switch* (NSS) é configurado num arquivo chamado `nsswitch.conf`, em sistemas GNU/Linux, normalmente, encontra-se no diretório `/etc`. O *name service*

switch controla como uma máquina cliente ou uma aplicação obtem informações da rede. O *name service switch* é usado pelas aplicações clientes que chamam qualquer interface do tipo `getXXXbyYYY()`, onde `XXX` simboliza o nome da base de dados que deseja acessar e `YYY` o tipo de informação que se deseja buscar. Por exemplo, `gethostbyname()`, `getpwuid()`, `getpwnam()`.

A idéia básica é colocar a implementação de diferentes serviços oferecidos para acessar base de dados em módulos separados. Tendo como vantagem:

- Contribuidores podem adicionar novos serviços sem ter que adicioná-lo a biblioteca `GNU C`.
- Os módulos podem ser atualizados separadamente.
- A imagem da biblioteca `C` é pequena.

As base de dados do NSS disponíveis são:

- `aliases` Apelidos dos usuários para conta de *e-mail*.
- `ethers` Número Ethernet.
- `group` Grupos de usuários.
- `hosts` Nome e número de *host*.
- `netgroup` Lista de grupos na rede, composta pelo *host*, usuário e domínio.
- `networks` Nome e número da rede.
- `protocols` Protocolos da rede.
- `passwd` Nome dos usuários, diretório e `shell`.
- `rpc` Nomes e números de chamdas de procedimento remoto (RPC - *Remote procedure Call*).
- `services` Serviços da rede.
- `shadow` Senhas do usuário criptografada.
- `automount`
- `bootparams`
- `netmasks`
- `publickey`

Cada máquina tem seu próprio arquivo `nsswitch.conf`, sendo que em cada linha do arquivo é identificado o nome da base de dados que está sendo configurada, seguido por uma ou mais fontes de localização das informações. As atuais fontes de informação que podem ser configuradas no arquivo `nsswitch.conf` são apresentadas na Tabela 4.2.

O objetivo deste trabalho é adicionar uma nova fonte de informações para o NSS. O qual será chamado de `remote`.

Tabela 4.2: Fontes de Informação do Arquivo *nsswitch.conf*

fonte de informação	Descrição
`files`	um arquivo guardado no diretório `/etc` da máquina cliente. Por exemplo, `/etc/passwd`
`nisplus`	Uma tabela `NIS+`. Por exemplo, a tabela de *hosts*.
`nis`	Um mapa `NIS`. Por exemplo, um mapa de *hosts*
`compat`	`compat` pode ser usado para informações de grupo e senha, seu objetivo é suportar o velho estilo `+` ou `-`.
`dns`	Pode ser usado para especificar qual informação pode ser buscada do *DNS*.
`ldap`	Pode ser usado para especificar entradas a serem obtidas de um diretório LDAP.

4.2.1 Configuração NSS

De algum modo o código do NSS deve dizer o que deseja do usuário. Por esta razão existe o arquivo de configuração `/etc/nsswitch.conf`. Para cada base de dados o arquivo possui uma especificação de como o processo de *lookup*[7] deverá ocorrer. Um exemplo de arquivo `nsswitch.conf` é mostrado na Figura 4.2.

```
#
# /etc/nsswitch.conf
#
passwd:     files
shadow:     files
group:      files

hosts:      files nisplus nis dns
networks:   nisplus [NOTFOUND=return] files

ethers:     nisplus [NOTFOUND=return] db files
protocols:  nisplus [NOTFOUND=return] db files
rpc:        nisplus [NOTFOUND=return] db files
services:   nisplus [NOTFOUND=return] db files
```

Figura 4.2: Exemplo de Arquivo `/etc/nsswitch.conf`

A primeira coluna é a base de dados, o resto da linha especifica como o processo de *lookup* deverá ser feito. É importante ressaltar que cada base de dados é configurada individualmente, tendo assim uma maior flexibilidade.

A configuração para a base de dados pode ter dois itens diferentes:

- o primeiro é a especificação do serviço, `files`, `db` ou `nis`.

[7] *Lookup* busca, pesquisa.

- o segundo é a reação que o resultado da busca (*lookup*) deverá ter. Por exemplo, `[NOTFOUND=RETURN]`.

4.2.2 Como o NSS Encontra suas Funções

Na verdade, os nomes no arquivo NSS são simples *strings*, que são usadas para encontrar de forma implícita funções endereçadas. Para a `GLIBC`, cada serviço dever ter uma biblioteca `/lib/libnss_SERVIÇO.so.X` para todo `SERVIÇO` que estiver sendo usado. Na instalação padrão da GLIBC pode-se usar `files`, `db`, `nis` e `nisplus`. Para *hosts*, usa-se `dns` como serviço extra, para senha, grupo e *shadow* pode-se ainda usar `compat`. Estes serviços não deverão ser usados pelo `libc5` com `NYS`. O número da versão é o `X` onde 1 é para glibc 2.0 e 2 para glibc 2.1. Por exemplo, para o serviço `files` haverá a biblioteca `/lib/libnss_files.so.2`.

4.2.3 Ações do NSS

Há duas formas do processo *lookup* ser executado no NSS:

Única fonte as informações serão buscada por apenas uma fonte, ou seja um único serviço é especificado. Neste caso, se a busca encontrar a informação procurada uma mensagem de *sucesso* (`SUCCESS`) é retornada. Caso a informação não seja encontrada, o processo de busca pára e retorna uma mensagem diferente (`NOTFOUND | UNAVAIL | TRYAGAIN`). O que cada serviço faz com a mensagem de retorno varia de acordo a função que está sendo buscada pelo serviço.

Múltiplas fontes se um determinado tipo de informação possui múltiplas fontes de informação a busca é realizada pela primeira fonte listada. Se a busca encontrar a informação, uma mensagem de *sucesso* (`SUCCESS`) é retornada. Caso a informação não seja encontrada na primeira fonte, a busca é feita utilizando a segunda fonte. O processo de busca usa todas as fontes até que a informação seja encontrada, ou até que um comando obrigando a parar seja dado. Se todas as fontes listadas são usadas e a informação desejada não seja encontrada, o processo de busca pára e retorna uma mensagem diferente de sucesso (`NOTFOUND | UNAVAIL | TRYAGAIN`).

As ações no processo de *lookup* permite um controle bem refinado no NSS. Elas podem ser colocadas entre dois serviços com colchetes. A forma geral é:

```
[ ( !?  status = action )+ ]
```

onde,

```
status => success | notfound | unavail | tryagain

action => return | continue
```

Não há diferença entre maiúscula/minúscula nas palavra-chaves. O valor do *status* é o resultado de uma chamada a função de *lookup* de um serviço em específico.

O significada das ações são mostradas na Tabela 4.3.

Tabela 4.3: Ações de Resposta para a Mensagem de *Status*

Ação	**Significado**
`return`	Pára de procurar pela informação.
`continue`	Continua a buscar a informação na próxima fonte configurada.

Os *status* tem o seguinte significado:

- `success` nenhum erro ocorreu e a informação procurada é retornada. A ação padrão para `success` é `return` (`SUCCESS=return`).
- `notfound` a fonte de informações respondeu que nenhuma entrada foi encontrada, ou seja, o arquivo ou tabela foi acessada em busca da informação mas não encontrou o valor procurado. A ação padrão para `notfound` é `continue` (`NOTFOUND=continue`).
- `unavail` o serviço está permanentemente indisponível ou não responde, em outras palavras, a fonte de informação (arquivo, tabela, DNS) não foram encontrados ou não pode ser acessado. A ação padrão para `unavail` é `continue` (`UNAVAIL=continue`).
- `tryagain` O serviço está temporariamente indisponível, pode estar sendo usado no momento, podendo responder num outro momento. Ou seja, a fonte de informação foi encontrada, mas não pode responder. A ação padrão para `tryagain` é `continue` (`TRYAGAIN=continue`).

Mais informações podem ser encotradas em (FREE SOFTWARE FOUNDATION, 2005) — seção *The NSS Configuration File*.

4.2.4 Interface NSS

Nesta seção será visto como construir as interfaces dos módulos NSS e como as funções contidas no módulo são identificadas por seus nomes. O nome de cada função consiste de duas partes:

`_nss_serviço_função`

naturalmente `serviço` corresponde ao nome do módulo em que a função é encontrada. a parte `função` é derivada das interfaces da própria biblioteca C. Por exemplo, se o usuário chamar a função `gethostbyname` e o serviço que estiver sendo usado é `files` a função NSS usada será:

`_nss_files_gethostbyname_r`

no módulo

`libnss_files.so.2`

Módulos NSS possui apenas versões reentrante[8] das funções de *lookup* (FREE SOFTWARE FOUNDATION, 2005). Isto é, se o usuário chamasse a função `gethostbyname_r` isto também terminaria na função acima – `_nss_files_gethostbyname_r`. Para todas as funções de interface do usuário a biblioteca C mapeia esta chamada a uma chamada à função reentrante. Para funções reentrantes isto é trivial desde que a relação seja (quase) a mesma. Para a versão não-reentrante a biblioteca mantém *buffers* internos que são usados para substituir o *buffer* fornecido pelo usuário.

4.2.5 Interfaces das Funções no módulo NSS

Nesta seção será analisado o protótipo de uma função reentrante no módulo NSS e quais as funções deve-se implementar caso deseja-se disponibilizar um novo serviço ao NSS. As funções reentrantes possuem argumentos adicionais (comparado com a versão padrão, funções não-reentrantes), isto para que as funções sejam usadas seguramente em aplicações *multihread*. O protótipo para as versões reentrante e não-reentrante da função `gethostbyname` são:

```
struct hostent*
gethostbyname (const char *name)

int
gethostbyname_r (const char *name, struct hostent *result_buf,
                 char *buf, size_t buflen, struct hostent **result,
                 int *h_errnop)
```

Neste caso, o protótipo para a função no módulo NSS será:

```
enum nss_status
_nss_files_gethostbyname_r (const char *name,
                            struct hostent *result_buf,
                            char *buf, size_t buflen,
                            int *errnop, int *h_errnop)
```

[8] Uma função reentrante não mantém dados estáticos sobre sucessivas chamadas, nem retorna ponteiro para dados estáticos. Todos os dados devem ser fornecidos pelo chamador da função. Uma função reentrante não deve chamar funções não-reentrantes.

A diferença entre as funções reentrantes é a mudança no valor de retorno, as funções do módulo NSS sempre retornan um valor de `nss_status` — `NSS_STATUS_TRYAGAIN | NSS_STATUS_UNAVAIL | NSS_STATUS_NOTFOUND | NSS_STATUS_SUCCESS` — e o argumento `struct hostent **result` é omitido no módulo NSS.

De acordo com (FREE SOFTWARE FOUNDATION, 2005) existe um quinto valor de retorno, o `NSS_STATUS_RETURN`. Este valor é usado apenas internamente por algumas funções em situações que não podem ser usado os outros valores. É recomendado olhar o código fonte da `GLIBC` para saber de mais detalhes.

Há dois tipos de funções a implementar quando deseja-se construir um serviço NSS. A primeira são as funções `getXXXbyYYY` e um segundo tipo são as funções `ENT`.

Funções do tipo `getXXXbyYYY`

Cada serviço no módulo NSS possui um conjunto de funções que são chamadas para encontrar a informação requisitada, sendo que as funções `getXXXbyYYY` são as mais importantes no módulo NSS, onde `XXX` simboliza o nome da base de dados, ou uma abreviação, que deseja acessar e `YYY` o tipo de informação que se deseja buscar. Por exemplo, `getpwuid()`, buscará o `uid` numa base de dados, se o módulo for `files` buscará no arquivo `/etc/passwd`. A Tabela 4.4 mostra as funções que devem ser implementadas para cada serviço.

Funções do tipo `ENT`

As funções do tipo `ENT` são composta de um grupo de três funções:

- `setXXXent` funções deste tipo devem colocar o ponteiro do arquivo para o início da base de dados, ou seja, a próxima entrada é o primeiro item da base de dados.
- `getXXXent_r` funções deste tipo recuperam as informações associada ao tipo de dados (`struct aliasent`, `struct passwd`, etc.). A primeira vez que uma função desse tipo é chamada recupera o primeiro item da base de dados, sucessivas chamadas recuperarão o próximo item. Quando o último item é alcançado e ela é chamada novamente deve retornar `NSS_STATUS_NOTFOUND`. Funções `getXXXent_r` recebem os seguintes argumentos: `(STRUCTURE *result, char *buffer, size_t buflen, int *errnop)`. Sempre que a informação for encontrada `NSS_STATUS_SUCCESS` é retornado. Quando `buffer` passado for pequeno para guardar informações extras, `NSS_STATUS_TRYAGAIN` deverá ser retornado.
- `endXXXent` funções deste tipo fecha a base de dados.

A Tabela 4.5 mostra as funções que devem ser implementadas para cada serviço. Mais informações consulte (FREE SOFTWARE FOUNDATION, 2005).

Tabela 4.4: Funções GET e GETBY para cada Serviço NSS

Serviço	Funções GET e GETBY
aliases	enum nss_status _nss_SERVIÇO_getaliasbyname_r
group	enum nss_status _nss_SERVIÇO_getgrgid_r enum nss_status _nss_SERVIÇO_getgrnam_r
hosts	enum nss_status _nss_SERVIÇO_gethostbyaddr_r enum nss_status _nss_SERVIÇO_gethostbyname_r enum nss_status _nss_SERVIÇO_gethostbyname2_r
networks	enum nss_status _nss_SERVIÇO_getnetbyname enum nss_status _nss_SERVIÇO_getnetbyaddr
protocols	enum nss_status _nss_SERVIÇO_getprotobyname enum nss_status _nss_SERVIÇO_getprotobynumber
password	enum nss_status _nss_SERVIÇO_getpwnam_r enum nss_status _nss_SERVIÇO_getpwuid_r
rpc	enum nss_status _nss_SERVIÇO_getrpcbyname enum nss_status _nss_SERVIÇO_getrpcbynumber
services	enum nss_status _nss_SERVIÇO_getservbyname enum nss_status _nss_SERVIÇO_getservbyport
shadow	enum nss_status _nss_SERVIÇO_getspnam_r

4.2.6 Estendendo NSS

Uma das vantagens do NSS é que ele pode ser estendido facilmente. Há duas formas de estender: adicionando uma nova base de dados ou adicionando um outro serviço. O presente trabalho visa adicionar uma nova base de dados remota utilizando um canal criptografado. É importante mencionar que adicionar uma nova base de dados é independente de adicionar um outro serviço, porque um novo serviço não precisa suportar todas as bases de dados ou funções do *lookup*.

Os fontes para um novo serviço não precisam e nem devem fazer parte da biblioteca GNU C. Aquele(a) que desenvolve têm controle completo sobre os fontes e seu desenvolvimento.

Cada novo módulo NSS criado deve seguir a especificação para o nome da biblioteca e o número, pois, é a partir do nome que a biblioteca é usada pelo sistema. Para a versão 2.1 da GLIBC a biblioteca deve terminar com o número 2 — libnss_NAME.so.2, a versão 2.0 termina com o número 1. Caso a GLIBC mude para uma versão incompatível à versão 2.1, o número será aumentado.

Desenvolvedores de um novo serviço NSS devem certificar-se de que o módulo está sendo criado usando o número correto. Isto significa que o próprio arquivo deve ter o nome correto nos sistemas ELF o soname (nome compartilhado do objeto) deve também ter este número (FREE SOFTWARE FOUNDATION, 2005). Para construir um módulo a partir de um grupo de arquivos objeto em um sistema ELF usando GNU CC deverá ser feito da seguinte forma:

```
gcc -shared -o libnss_NAME.so.2 -Wl,-soname,libnss_NAME.so.2
ARQUIVOS_OBJETOS
```

Normalmente o diretório de bibliotecas NSS é o `$prefix/lib`, onde `$prefix` corresponde ao valor usado na opção `-prefix` na configuração do sistema.

Tabela 4.5: Funções `ENT` para cada Serviço NSS

Serviço	**Funções `ENT`**

`aliases`	`enum nss_status _nss_SERVIÇO_getaliasent_r`
	`enum nss_status _nss_SERVIÇO_setaliasent(void)`
	`enum nss_status _nss_SERVIÇO_endaliasent(void)`

`group`	`enum nss_status _nss_SERVIÇO_getgrent_r`
	`enum nss_status _nss_SERVIÇO_setgrent(void)`
	`enum nss_status _nss_SERVIÇO_endgrent(void)`

`hosts`	`enum nss_status _nss_SERVIÇO_gethostent_r`
	`enum nss_status _nss_SERVIÇO_sethostent(void)`
	`enum nss_status _nss_SERVIÇO_endhostent(void)`

`networks`	`enum nss_status _nss_SERVIÇO_getnetent_r`
	`enum nss_status _nss_SERVIÇO_setnetent(void)`
	`enum nss_status _nss_SERVIÇO_endnetent(void)`

`protocols`	`enum nss_status _nss_SERVIÇO_getprotoent_r`
	`enum nss_status _nss_SERVIÇO_setprotoent(void)`
	`enum nss_status _nss_SERVIÇO_endprotoent(void)`

`password`	`enum nss_status _nss_SERVIÇO_getpwent_r`
	`enum nss_status _nss_SERVIÇO_setpwent(void)`
	`enum nss_status _nss_SERVIÇO_endpwent(void)`

`rpc`	`enum nss_status _nss_SERVIÇO_getrpcent_r`
	`enum nss_status _nss_SERVIÇO_setrpcent(void)`
	`enum nss_status _nss_SERVIÇO_endrpcent(void)`

`services`	`enum nss_status _nss_SERVIÇO_getservent_r`
	`enum nss_status _nss_SERVIÇO_setservent(void)`
	`enum nss_status _nss_SERVIÇO_endservent(void)`

`shadow`	`enum nss_status _nss_SERVIÇO_getspent_r`
	`enum nss_status _nss_SERVIÇO_setspent(void)`
	`enum nss_status _nss_SERVIÇO_endspent(void)`

`ethers`	`enum nss_status _nss_SERVIÇO_getetherent_r`
	`enum nss_status _nss_SERVIÇO_setetherent(void)`
	`enum nss_status _nss_SERVIÇO_endetherent(void)`

`netgroup`	`enum nss_status _nss_SERVIÇO_getnetgrent_r`
	`enum nss_status _nss_SERVIÇO_setnetgrent(void)`
	`enum nss_status _nss_SERVIÇO_endnetgrent(void)`

Capítulo 5

PAMRAD

O PAMRAD teve como origem o trabalho (LIMA, 2003) no qual foi implementado um módulo PAM capaz de fazer o gerenciamento de autenticação, o gerenciamento de conta, o gerenciamento de sessão e o gerenciamento de senhas num servidor remoto por meio de um canal criptografado. No entanto para que o processo de autenticação/autorização remotamente ocorresse, integralmente, seria preciso que um módulo NSS também funcionasse. O trabalho citado anteriormente implementou alguns métodos para que a autenticação ocorresse de fato – `getgrnam`, `getgrgid`, `getpwnam`, `getpwuid`. Desta forma o presente trabalho vem apresentar uma implementação completa do NSS para os seguintes serviços: `groups`, `passwd` e `shadow`. A implementação do módulo NSS engloba o tratamento *thread safe* para as diversas requisições solicitadas pelos clientes PAMRAD.

O `pamrad` deve estar sendo executado como um *daemon* — serviço — no servidor, aguardando por conexões dos clientes. Seja por conexões para autenticar/autorizar — módulo `pam_remote.so`, seja por conexões para buscar informações do serviço NSS (`groups`, `passwd`, `shadow`[1]) — biblioteca `libnss_remote.so.2`.

Conforme definido em (LIMA, 2003) o módulo PAM e a aplicação PAM utilizam uma arquitetura cliente/servidor, sendo que o módulo é o cliente e a aplicação é o servidor. A comunicação é realizada por meio de *sockets* e o protocolo SSL através da biblioteca OpenSSL, ilustrada na Figura 5.1.

5.1 Implementação

A linguagem C++ foi utilizada em quase todo o projeto, exceto parte do módulo NSS, `libnss_remote.so.2`[2] que fica nas máquinas cliente, que foi usado o modelo de alocação de memória da linguagem `C` (`malloc` e `free`) para as funções reentrantes que deveriam ser "vistas" como funções `C`, para isto utilizou-se a palavra-chave `extern "C"`.

[1] Os serviços `hosts` e `aliases` estão em fase de conclusão

[2] Fontes: `nss_remote.cpp, headers/nss_remote_shadow.h, headers/nss_remote_pwd.h, headers/nss_remote_grp.h, headers/nss_remote_hosts.h, headers/nss_remote_alias.h`

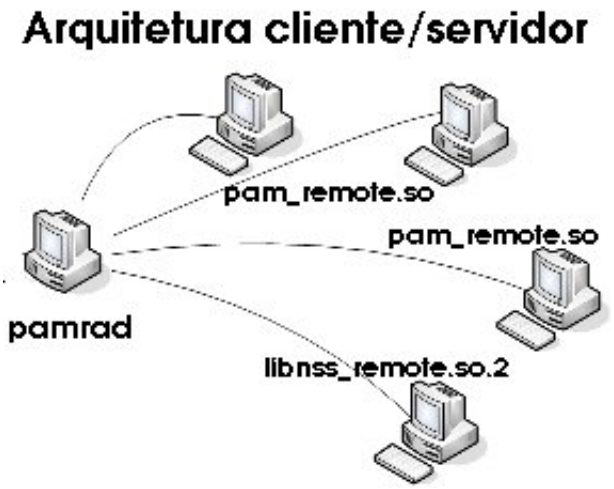

Figura 5.1: Arquitetura Cliente/Servidor PAMRAD

A compilação foi feita utilizando o compilador `GCC` — *GNU Compiler Collection, Thread model: posix - gcc version 3.3.3 20040412* — encontrado no Fedora Core 2.

Implementação `libnss_remote.so.2` – Cliente

A implementação do NSS constitui-se basicamente da implementação das funções da Tabela 4.4 e 4.5, de acordo com os serviços oferecido pelo `PAMRAD`, sendo que cada uma destas funções quando chamada pelo cliente é feita uma solicitação de conexão com o servidor (`pamrad`). Caso a conexão seja aberta com sucesso é enviada uma informação dizendo ao `pamrad` qual tipo de dado deseja buscar, logo após os argumentos são passados, caso a situação exige algum, como nome do usuário, identificador do usuário ou grupo, etc. O servidor com base nesta requisição busca a informação em uma de suas bases de dados — um dos arquivos `/etc/passwd`, `/etc/groups` ou `/etc/shadow`. O diagrama de classe do `libnss_remote.so.2` está ilustrado na Figura 5.2.

Figura 5.2: Diagrama de Classes `libnss_remote.so.2`

Implementação `pamrad` – Servidor

O servidor possui todas as funções implementadas no cliente para que possa atender suas requisições. Atualmente cada serviço trabalha como uma *thread* dentro do `pamrad`, desta forma cada serviço possui seu próprio fluxo de execução, concorrentemente. A API utilizada para criar e gerenciar as *threads* foi a `pthread.h`, cuja interface tem sido especificada pelo comitê de padronização IEEE (*Institute of Electrical and Electronics Engineers*) no padrão 1003.1 POSIX (*Portable Operating System Interface*), edição 2003. Esta API é encapsulada pela classe `Thread.hpp`, conforme diagrama de classes apresentado na Figura 5.3.

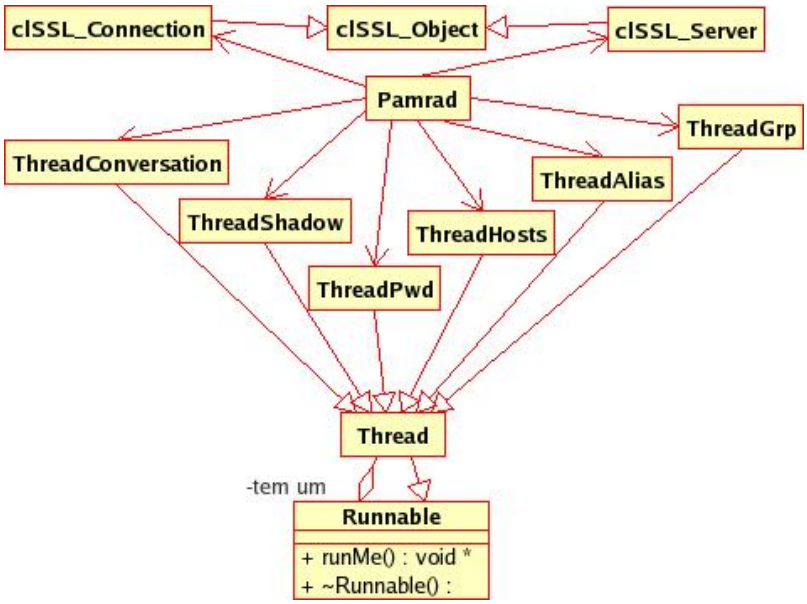

Figura 5.3: Diagrama de Classes PAMRAD

Como pode ser observado pelo diagrama de classes além de possuir uma *thread* para cada serviço implementado, também há uma *thread* para atender as solicitações do módulo PAM, uma *thread* de conversação. A cada nova conexão estabelecida com o cliente um nova *thread* é criada, de acordo o tipo de informação que o cliente deseja buscar. O algoritmo para tratar a concorrência das solicitações ainda deve ser melhorado, visando buscar um melhor desempenho do serviço.

5.2 Configuração

A configuração do PAMRAD, basicamente, é feita em seis passos:

1. Criação da chave privada para o servidor e dos certificados, tanto para o cliente como para o servidor.
2. Compilar o módulo, a aplicação e o módulo NSS.
3. Criar o serviço para o PAMRAD receber as conexões e configurar o Linux-PAM no servidor.
4. Configurar os arquivos do módulo PAM no cliente e instalar o módulo NSS.
5. Configurar o `/etc/nsswitch.conf` para usar o módulo NSS `remote`.
6. Configurar o módulo Linux-PAM para autenticar com o `pam_remote.so`.

5.2.1 Criação dos Certificados e Chave Privada

Conforme exemplificado em (LIMA, 2003), este processo se resume da seguinte forma:

> *O módulo e a aplicação necessitam de um certificado e uma chave privada SSL. Na instalação já são copiados esses arquivos, mas é interessante cada rede ter o seu certificado e chave privada. Para criá-los utilize os seguintes comandos:*
>
> ```
> openssl req -new -text -out cert.req
>
> openssl rsa -in privkey.pem -out
> <nome do arquivo de chave>
>
> openssl req -x509 -in cert.req -text -key
> <nome do arquivo de chave> -out
> <nome do arquivo de certificado>
> ```
>
> *O cliente necessita apenas do arquivo de certificado, e este deve ser informado no arquivo de configuração no cliente. O servidor precisa tanto do certificado quanto da chave privada e esses arquivos devem ser informados no arquivo de configuração no servidor.*

Os arquivos de certificado e a chave privada SSL deverão ficar no diretório de configuração do PAMRAD, este diretório é o `/etc/pam_remote`, tanto no servidor quanto nas máquinas cliente. Para evitar problemas, deve-se colocar o certificado do servidor e sua chave privada no diretório `PAMRAD/conf/server` e o certificado do cliente no diretório `PAMRAD/conf/client`, pois quando os comandos de instalação forem executados todos os arquivos necessários serão copiados. A estrutura dos diretórios é apresentada na Figura 5.4.

5.2.2 Compilação

Para instalar o PAMRAD, deve-se descompactar o arquivo `PAMRAD.tar.gz`, que são os fontes do `PAMRAD`, num diretório qualquer. Com o *shell* dentro do diretório criado **`PAMRAD/`**, deve-se executar o comando

```
make all
```

para compilar todo o programa.

Para compilar cada parte em separado deve-se usar o comando `make pam_remote` para compilar os módulos PAM e NSS, o comando `make pamrad` para compilar a aplicação que ficará no servidor. Os arquivos compilados (`libnss_remote.so.2.0`, `pam_remote.so` e `pamrad`) ficarão no diretório `PAMRAD/bin/`, conforme mostrado na Figura 5.4.

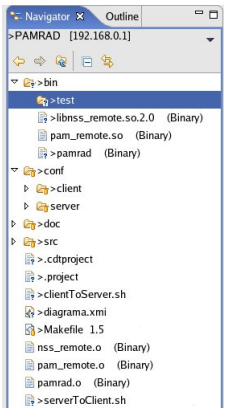

Figura 5.4: Diretório PAMRAD

5.2.3 Configuração e Instalação no Servidor

Após a compilação do programa, criação dos certificados e da chave deve-se executar o comando:

```
make install_server
```

que fará as seguintes ações:

- copiará o arquivo `pamrad` para o diretório `/usr/bin`;
- criará o diretório `/etc/pam_remote`, caso não exista. Este é o diretório de configuração do `PAMRAD`;

- copiará o certificado do diretório `PAMRAD/conf/server` para o diretório de configuração do `PAMRAD`;
- copiará a chave privada SSL do diretório `PAMRAD/conf/server` para o diretório de configuração do `PAMRAD`;
- copiará o arquivo de configuração do `PAMRAD` (`PAMRAD/conf/server/pamrad.conf`) para o diretório de configuração do `PAMRAD`;
- copiará o arquivo de configuração do serviço `PAMRAD` para o diretório `/etc/rc.d/init.d`
- copiará o arquivo de configuração do PAM (`PAMRAD/conf/server/pamrad`) para o diretório de configuração das aplicações PAM (`/etc/pam.d`) .

será preciso ter permissão nos diretórios para que tudo seja feito sem problemas, é aconselhado usar o usuário `root` para excutar estas tarefas.

O Arquivo de Configuração `pamrad.conf`

No servidor, onde é instalado a aplicação, é preciso editar o seguinte arquivo: `/etc/pam_remote/pamrad.conf`. Para informar a porta onde o servidor ficará "escutando" deve-se incluir a seguinte linha no arquivo:

```
PortNumber=<número da porta>
```

Para informar o arquivo com o certificado SSL deve-se incluir a seguinte linha no arquivo:

```
CertFile=<caminho do arquivo de certificado>
```

Para informar o arquivo com a chave privada SSL deve-se incluir a seguinte linha no arquivo:

```
keyFile=<caminho do arquivo de chave>
```

O padrão é que os arquivos do certificado e da chave privada fiquem no diretório criado durante a instalação `/etc/pam_remote`, mas podem ficar em qualquer lugar e com quaisquer nomes. Basta especificar no arquivo de configuração. Para colocar comentários nesse arquivo deve-se usar o símbolo # no início de cada linha.

5.2.4 Configuração e Instalação no Cliente

Para instalar o `pam_remote`, supondo que se esteja no `shell` da máquina cliente basta executar o comando

```
make install_client
```

que fará as seguintes ações:

- copiará o módulo `pam_remote.so` para o diretório `/lib/security`;
- copiará a biblioteca `libnss_remote.so.2.0` para o diretório de bibliotecas PAM `lib/` e fará um `link` simbólico desta biblioteca[3];
- criará o diretório `/etc/pam_remote`, caso não exista. Este é o diretório de configuração do PAMRAD na máquina cliente;
- copiará o arquivo de configuração do `pam_remote` (`PAMRAD/conf/client/pam_remote.conf`) para o diretório de configuração do PAMRAD no cliente;
- copiará o certificado do cliente do diretório `PAMRAD/conf/client` para o diretório de configuração do PAMRAD no cliente;
- cria um *backup* do arquivo de autenticação do Linux-PAM — `/etc/pam.d/system-auth`;
- copia um modelo de autenticação do Linux-PAM usando o PAMRAD, o arquivo `PAMRAD/conf/client/system-auth`;

será preciso ter permissão nos diretórios para que tudo seja feito sem problemas, é aconselhado usar o usuário `root` para excutar estas tarefas.

É importante ressaltar que a biblioteca NSS é usada pelo `ldconfig` através de um *link* simbólico, para mais informações consulte (WHEELER, 2003).

O Arquivo de Configuração `pam_remote.conf`

No cliente, onde é instalado o módulo, é preciso editar o seguinte arquivo: `/etc/pam_remote/pam_remote.conf`. Para informar o endereço do servidor de senhas é necessário incluir a seguinte linha no arquivo:

```
ServerAddr=<endereço do servidor:porta>
```

Para informar o arquivo com o certificado SSL é necessário incluir a seguinte linha no arquivo:

```
CertFile=<caminho do arquivo de certificado>
```

Para informar um servidor alternativo do PAMRAD é necessário incluir a seguinte linha:

```
ServerAddrOpt=<endereço do servidor:porta>
```

[3] `ln -sf /lib/libnss_remote.so.2.0 /lib/libnss_remote.so.2`

```
#
# /etc/nsswitch.conf
#
# An example Name Service Switch config file.
# This file should be sorted with the most-used
# services at the beginning.
#
passwd:     remote files
shadow:     remote files
group:      remote files

hosts:      files dns
```

Figura 5.5: Exemplo de Arquivo `/etc/nsswitch.conf`

este servidor alternativo vai ser usando quando o servidor especificado no parâmentro `ServerAddr` não estiver respondendo.

O padrão é que o arquivo do certificado fique no diretório criado durante a instalação `/etc/pam_remote`, mas pode ficar em qualquer lugar e com qualquer nome. Basta especificar no arquivo de configuração. Para colocar comentários nesse arquivo deve-se usar o símbolo # no início de cada linha.

5.2.5 Configuração do Arquivo `nsswitch.conf`

A configuração do arquivo `/etc/nsswitch.conf` é simples, basta colocar o nome `remote` após os serviços disponíveis — `groups`, `passwd` e `shadow` — que o *name service switch* passará a usá-los assim que a máquina reiniciar. A Figura 5.5 é um exemplo de configuração do NSS.

5.2.6 Configuração do Linux-PAM – Cliente

A configuração do Linux-PAM é feito nos arquivos do diretório `/etc/pam.d`, geralmente só há necessidade de modificar o arquivo `/etc/pam.d/system-auth` (pelo menos nas distribuições baseada em *Red Hat*), pois os programas de autenticação normalmente o utiliza para autenticar/autorizar um usuário. Um arquivo exemplo é copiado quando é executado o comando `make install_client`, descrito na Seção 5.2.4. A Figura 5.6 é um exemplo de configuração do arquivo `/etc/pam.d/system-auth` utilizando o `pam_remote.so`.

5.3 Comentários Finais

Esse capítulo apresentou o PAMRAD, que tem por objetivo autenticar/autorizar usuários remotamente em servidores Linux a partir de suas estações, sem que haja necessidade de uma cópia local dos arquivos `/etc/passwd`, `/etc/shadow` e `/etc/group`.

```
#%PAM-1.0
# local:  /etc/pam.d/
# User changes will be destroyed the next time authconfig is run.
auth required /lib/security/$ISA/pam_env.so
auth sufficient /lib/security/pam_remote.so likeauth
auth sufficient /lib/security/$ISA/pam_unix.so likeauth nullok
auth required /lib/security/$ISA/pam_deny.so

account sufficient /lib/security/pam_remote.so
account required /lib/security/$ISA/pam_unix.so

password required /lib/security/$ISA/pam_cracklib.so retry=3
password sufficient /lib/security/pam_remote.so use_authtok
password sufficient /lib/security/$ISA/pam_unix.so use_authtok md5 shadow
password required /lib/security/$ISA/pam_deny.so

session required /lib/security/$ISA/pam_limits.so
session sufficient /lib/security/pam_remote.so
session required /lib/security/$ISA/pam_unix.so
```

Figura 5.6: Ex. de Arquivo `/etc/pam.d/system-auth` para Autenticação.

Apesar do PAMRAD já possuir um mecanismo de autenticação/autorização implementado ele não acaba por aí. O projeto deve ser melhorado para sair da esfera acadêmica e vir a ser usado por ambientes reais, que são colocados à prova a toda hora, com o objetivo final de simplificar o trabalho dos administradores de sistemas Linux, sem que a confiabilidade e segurança seja sacrificada.

Capítulo 6

Conclusão

Este trabalho foi indicado como Trabalhos Futuros em (LIMA, 2003). Para desenvolvê-lo foi necessário compreender como o modelo de autenticação PAM funciona, como que estava implementado o modelo de autenticação e comunicação do PAMRAD, administração do PAM, desenvolvimento de aplicações e módulos PAM, criação de um *daemon*, funcionamento do *name service switch* e como implementá-lo.

Como resultado deste trabalho obteve-se uma aplicação e módulo PAM autenticando usuários remotos por meio de um canal criptografado, juntamente com um módulo NSS trabalhando com o mesmo modelo, ou seja, uma base de dados remota utilizando um canal criptografado.

6.1 Perspectivas

O PAMRAD encontra-se numa versão *alfa*, apresentando algumas instabilidades. Desta forma pretende-se dar continuidade ao trabalho para fazer melhorias e adicionar novas características. O primeiro passo foi disponilizar o projeto no *SourceForge* – `http://sourceforge.net/projects/pamrad`. Isso foi feito com o objetivo de divulgá-lo em listas de discussão e comunidades de *software* livre para que novos colaboradores possam participar com novas idéias, desenvolvimento, e especialmente como testadores do PAMRAD, de forma que possa colocá-lo a prova nos mais diferentes ambientes e distribuições do Linux.

Os passos mais importante agora são:

1. resolver os problemas críticos, como, compilação em novas versões da GLIBC e GCC, entre outros;

2. mudança de senha do usuário, a versão atual apresenta problemas quando o usuário tentar modificar sua senha e quando o usuário `root` precisa modificar a senha de um outro usuário é pedido a senha antiga, isso não deve ocorrer pois o usuário pode ter esquecido a senha;

3. melhorar o algortimo de concorrência fazendo com que mais requisições de um mesmo serviço possar ser executado, atualmente existe apenas um fluxo de execução para cada serviço;
4. adicionar novas características ao arquivo de configuração. Por exemplo, definir servidores PAMRAD com base nos grupos do usuário;
5. implementar um *pool*[1] de conexões e *threads* com o objetivo de melhorar o desempenho na criação destes objetos, pois instanciar novas conexões ou novas *threads* a toda hora é uma operação custosa.
6. terminar a implementação da classe `Thread` para ter um maior controle sobre as operações que podem ser realizadas com uma *thread*, como cancelar sua execução, suspender sua execução por um determinado período de tempo, etc.

Com a implementação desses itens, o PAMRAD elimirá problemas que hoje impedem sua adoção. Com isso, espera-se que ele se torne uma séria opção para os administradores de sistemas em autenticação distribuída segura.

[1] *Pool* é um mecanismo pelo qual objetos (conexões, *threads*, etc.) são mantidos em um *cache* para uso e reuso por partes diferentes de uma aplicação.

Referências Bibliográficas

BARRET, D. J.; SILVERMAN, R. *Linux Security Cookbook*. [S.l.]: O'Reilly, 2003.

CONECTIVA S.A. *Guia do Servidor Conectiva Linux*. julho 2004.

DIERKS, T.; ALLEN, C. *The TLS Protocol*. Internet Engineering Task Force (IETF), Janeiro 1999. (Request for Comments: 2222). URL: `http://www.ietf.org/`.

FREE SOFTWARE FOUNDATION. *The GNU C Library v.2.3.5*. 2005.

LIMA, W. V. F. *Desenvolvimento de um Módulo PAM para Autenticação Criptografada e Distribuída em Estações Linux*. Monografia (Monografia de Conclusãoo do Curso de Ciência da Computação) — Universidade Federal de Lavras, 2003.

MALèRE, L. E. P. *LDAP Linux HOWTO*. March 2004.

MATTHEW, N.; RICHARD. *Beginning Linux® Programming*. 3. ed. [S.l.]: Wiley Publishing, Inc., 2004.

MORGAN, A. G. *The Linux-PAM System Administrators' Guide*. julho 2002.

MYERS, J. *Simple Authentication and Security Layer (SASL)*. Internet Engineering Task Force (IETF), Outubro 1997. (Request for Comments: 2222). URL: `http://www.ietf.org/`.

NEMETH, G. S. S. S. E.; HEIN, T. R. *UNIX System Administration Handbook*. 3. ed. [S.l.]: Prentice Hall PTR, 2001.

OPENSSL PROJECT. *OpenSSL*. Disponível em: <http://www.openssl.org/>.

SAMAR, V.; LAI, C. Making login services independent of authentication technologies. *Third ACM Conference on Computer Communications and Security, March 1996*, March 1996.

SAMAR, V.; SCHEMERS, R. *UNIFIED LOGIN WITH PLUGGABLE AUTHENTICATION MODULES (PAM)*. Open Software Foundation, Outubro 1995. (Request for Comments: 86.0). URL: `http://www.opengroup.org/`.

SCHNEIER, B. *Applied Cryptography*. [S.l.]: John Wisley, 1996.

SUN MICROSYSTEMS. *System Administration Guide: Naming and Directory Services (DNS, NIS, and LDAP)*. 2003.

UCHôA, F. C. S. e J. Q. *ADMINISTRAÇÃO DE SISTEMAS LINUX*. [S.l.], 2003. v. 1. Capítulo 6.

WAHL, T. H. M.; KILLE, S. *Lightweight Directory Access Protocol (v3)*. Internet Engineering Task Force (IETF), Dezempbro 1997. (Request for Comments: 2251). URL: `http://www.ietf.org/`.

WATSON, D. *Linux Daemon Writing HOWTO*. Maio 2004.

WHEELER, D. A. *Program Library HOWTO*. The Linux Documentation Project, Abril 2003. URL: `http://www.tldp.org/HOWTO/Program-Library-HOWTO/shared-libraries.html`.

Apêndice A

Apêndice

A.1 Introdução

Definição de termos usados no texto.

A.1.1 Kerberos

De acordo (BARRET; SILVERMAN, 2003) Kerberos é um sofisticado sistema de autenticação, inicialmente desenvovido no *Massachusetts Institute of Technology* (MIT) como parte do projeto *Athena* na década de 80. Ele envolve uma autenticação centralizada Kerberos mantida numa base de dados em um ou mais *hosts*[1] altamente seguros atuando como *Kerberos Key Distribution Centers* (KDCs). A principal atuação no sistema Kerberos (usuários, *hosts*, etc) está em obter credenciais chamadas *"tickets"* de um KDC. Por exemplo, quando um usuário requisitar qualquer serviço de rede, a estação envia apenas o nome de usuário através da rede e o servidor gera uma *session key* (ou chave de sessão), baseada no nome do usuário e no serviço requisitado, que funciona ao mesmo tempo como uma chave de encriptação e autenticação, já que para ser decifrada e usada o cliente precisa ter a senha do usuário. É como se o servidor enviasse um enigma, cuja resposta só pudesse ser descoberta através da senha. Depois de decifrada a chave, o cliente envia de volta um pacote encriptado através dela e recebe de volta um *"ticket"*, que garante o acesso ao servidor. Todas as informações enviadas a partir daí são feitas de forma encriptada, o que garante a segurança dos dados. Mais informações sobre kerberos pode ser encontrada em `http://web.mit.edu/kerberos/`.

[1] Qualquer máquina ou computador conectado a uma rede.

A.2 Criptografia

A.2.1 Criptografia Simétrica

Ocorre quando duas partes trocam informações criptografadas e ambas utilizam a mesma chave criptográfica para decriptografar os dados transmitidos. Pode-se citar o `base64` como algoritmo de criptografia simétrica.

A.2.2 Criptografia Assimétrica

Acontece quando duas partes trocam informações criptografadas porém, a origem geralmente utiliza uma chave pública para criptografar os dados e o destino utiliza uma chave privada para fazer o caminho inverso (decriptografar). A origem da chave pública é a chave privada mas, é totalmente improvável (teoricamente) que através da chave pública reconstrua-se a chave privada.

A.2.3 RSA

RSA é um algoritmo de encriptação de dados, que deve o seu nome a três professores do MIT, Ron Rivest, Adi Shamir e Len Adleman, que inventaram este algoritmo, até então a mais bem sucedida implementação de sistemas de chaves assimétricas (SCHNEIER, 1996), e fundamenta-se em Teorias Clássicas dos Números. É considerado um dos mais seguros. Foi também o primeiro algoritmo a possibilitar encriptação e assinatura digital, e uma das grandes inovações em criptografia de chave pública.

RSA é combinado com a função *hashing SHA1* (*secure hash algorithm*) para cifrar a mensagem. A principal vantagem da criptografia baseada em chave pública é a sua maior segurança em relação a criptografia baseada em chave secreta. No sistema baseado em chave pública as chaves privadas nunca precisam ser transmitidas ou recebidas a ninguém. Num sistema de chave secreta, ao contrário, sempre existe uma chance de que um intruso possa descobrir a chave secreta enquanto esta está sendo transmitida. Maiores informações podem ser encontradas em (SCHNEIER, 1996).