Wanner Vinícius Fagundes Lima

Desenvolvimento de um Módulo PAM para Autenticação Criptografada e Distribuída em Estações Linux

Monografia apresentada ao Departamento de Ciência da Computação da Universidade Federal de Lavras, como parte das exigências do curso de Ceiência da Computação, para obtenção do título de Bacharel.

Orientador
Prof. Bruno de Oliveira Schneider

Lavras
Minas Gerais - Brasil
2003

Wanner Vinícius Fagundes Lima

Desenvolvimento de um Módulo PAM para Autenticação Criptografada e Distribuída em Estações Linux

Monografia apresentada ao Departamento de Ciência da Computação da Universidade Federal de Lavras, como parte das exigências do curso de Ceiência da Computação, para obtenção do título de Bacharel.

Aprovada em *27 de junho de 2003*

Prof. Joaquim Quinteiro Uchôa

Prof. Ricardo Martins de Abreu Silva

Prof. Bruno de Oliveira Schneider
(Orientador)

Lavras
Minas Gerais - Brasil

Dedico este trabalho
a meus amados pais Amado e Madalena
que sempre confiaram em mim
e são os responsáveis por
eu chegar até aqui.

Agradecimentos

Agradeço

a Deus;
aos meus pais, Amado e Madalena;
aos meus irmãos, Denílson, Leilla e Bruno;
aos meus amigos da Escolta Técnica Federal do Espírito Santo;
aos meus amigos da Universidade Vale do Rio Doce;
aos meus amigos da Universidade Federal de Lavras;
aos meus amigos do 201, Eduardo, Paulo Sérgio, Alexandre,
André, Júlio, Eric e Fernando(Calouro).

Desenvolvimento de um Módulo PAM para Autenticação Criptografada e Distribuída em Estações Linux

O objetivo deste trabalho é o desenvolvimento de um módulo PAM (*Plugable Authentication Modules*) para autenticação criptografada, distribuída e de fácil gerenciamento em estações Linux. Com esse módulo também será possível alterar as senhas dos usuários. Normalmente a autenticação de um usuário em Linux é feita comparando a senha fornecida pelo usuário com uma senha localizada em um arquivo na máquina local. Na maioria dos sistemas Linux atuais esse processo é feito pelo PAM. Em um ambiente de rede, principalmente rede local, onde se deseja que todos os usuários da rede possam se logar em qualquer máquina é interessante que o processo de autenticação seja centralizado, isto é, o arquivo de senhas fique localizado em apenas uma máquina da rede, um servidor por exemplo. Já existem meios para se fazer isso, mas ou não são seguros ou não são específicos para este fim. Para facilitar o processo de autenticação remota será desenvolvido um módulo PAM que possibilitará as várias aplicações existentes em Linux que precisam autenticar usuários (e utilizam o PAM para isso), poderem fazer essa autenticação remotamente e com segurança sem a necessidade de modificação dessas aplicações.

Development of a PAM Module for Cryptographic and Distributed Authentication on Linux Stations

The purpose of this work is the development of a PAM (Plugable Authentication Modules) module for cryptographic, distruibuted and easy to manage authentication on Linux stations. With this module it will also be possible to change users' passwords. Normally the authentication of a user is made by a comparison between a password provided by her and a password located in a file on the local machine. Nowadays, at most of Linux systems, PAM takes care of this process. In a network environment, especially a local network, where it is desired that all networks users be able to log in any machine, it is usefull that the authentication process be centralized, that is, the password file should stay in just one machine, a server, for example. Implementations that provide this already exist, but none is safe and specific for this purpose. To make the remote authentication process easy, a PAM module will be developed so that the many Linux applications that need to authenticate users (and use PAM to do it), will be able to authenticate remotely and safely without any modifications.

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

Em um sistema de computação distribuído, onde temos estações de trabalho e não terminais burros, quando um usuário faz o *login* numa dessas estações ele está se logando na estação e não, necessariamente, em um servidor único na rede. Com isso cada estação tem que ter seu controle sobre os usuários que terão acesso a ela. É mais interessante que cada estação faça esse controle, mas de maneira que os dados de usuários fiquem centralizados. Por exemplo, se temos vários usuários de um sistema de computação distribuído e desejamos que todos os usuários possam acessar qualquer estação da rede. Se cada estação fizer esse controle de acesso independente das outras estações todos os usuários teriam que ser cadastrados em todas as estações da rede. Vamos supor agora que um usuário mude sua senha em uma das estações da rede. Quando esse usuário tentar acessar outra estação ele não poderá utilizar sua nova senha, terá que usar a senha que estava cadastrada naquela estação. Uma desvantagem em se utilizar uma autenticação centralizada é que será necessário se preocupar com todos os problemas que são comuns em uma rede de computadores como tolerância a falhas, por exemplo.

Em Linux, fazer esse sistema de autenticação centralizado não é muito trivial. Normalmente o que é feito durante uma autenticação em Linux é obter o nome e a senha do usuário, comparar a senha fornecida pelo usuário com a senha desse usuário armazenada em um arquivo local e, se forem iguais, o acesso à máquina é liberado.

Antigamente, qualquer aplicação como *login*, *ssh*, *ftp* que precisasse autenticar usuários deveria implementar seu próprio esquema de autenticação. Com isso ficava difícil criar um novo esquema de autenticação para todas as aplicações que necessitassem de autenticação, pois teria que reescrever o código de cada aplicação. Imagine se a cada novidade ou mudança no sistema de autenticação do Linux tivéssemos que reescrever todas as aplicações que autenticam usuários.

Com o objetivo de eliminar esse problema a Sun® criou o PAM (*Plugable Authentication Modules*) em 1995 [Sme95] e divulgou as especificações em um RFC (*Request for Comment*) [Sun01]. A partir desse documento foi feita a implementação do PAM para Linux e ficou conhecido como Linux-PAM.

O PAM passa a ser o responsável para fazer a autenticação dos usuários, assim as aplicações que necessitam autenticar usuários devem ser reescritas apenas uma vez, para suportarem o PAM. Depois disso todo esquema de autenticação seria responsabilidade do PAM. Se desejássemos mudar ou criar um novo esquema de autenticação bastaria alterar ou desenvolver um novo módulo PAM que todas as aplicações poderiam usar esse novo esquema.

Já existem alguns métodos para utilizar um arquivo de senha centralizado em um servidor. O mais trivial é que qualquer alteração de senha seja feita no servidor e o arquivo de senhas desse servidor é copiado de tempos em tempos para cada estação. Esse método pode gerar muito tráfego na rede e as alterações nas senhas não têm efeito imediato nas outras estações da rede. Isso também pode ser feito utilizando o NIS (*Network Information Service*) [ENH01] [How98], que compartilha as informações dos usuários na rede. Esse método não é seguro, pois os dados trafegam sem criptografia alguma pela rede e, além disso, qualquer máquina na rede pode se passar como um servidor NIS e fornecer suas próprias senhas. Outro método para se fazer isso é o LDAP (*Lightweight Directory Access Protocol*). Esse método disponibiliza as informações dos usuários através de serviço de diretório [Con]. O LDAP foi desenvolvido para prover qualquer informação que se queira disponibilizar na rede. Embora se tenha percebido que ele pode ser usado para fornecer informações para autenticação ele não foi criado com esse propósito.

A facilidade oferecida pelo PAM para o desenvolvimento de um novo esquema de autenticação que possa ser usado por todas as aplicações já existentes com suporte a PAM sem a necessidade de reescrever essas aplicações, e a necessidade de um mecanismo simples e seguro para autenticação remota de usuários serviram de incentivo para o desenvolvimento deste trabalho.

Este trabalho é continuação do trabalho de conclusão de curso de Paulo Sérgio Rezende de Carvalho [dC02] e com isso o levantamento bibliográfico deste trabalho foi feito com base na monografia escrita por ele. Nesse trabalho Paulo Sérgio desenvolveu uma aplicação PAM para autenticação criptografada e distribuída de estações Linux que servirá como referência durante o desenvolvimento deste trabalho.

O objetivo deste trabalho é desenvolver um módulo PAM para controle de autenticação e de senhas que buscará os dados necessários em um servidor especificado em um arquivo de configuração. Esse módulo permitirá que aplicações autentiquem usuários e troquem as senhas dos usuário diretamente no servidor. Para garantir a segurança das informações, todos os dados envia-

dos para o servidor e recebidos dele serão criptografados. Todo esse processo de comunicação com o servidor será transparente para as aplicações que irão utilizar esse módulo. Para as aplicações será como um sistema local. Nos clientes ficará o módulo PAM e no servidor uma aplicação PAM. O sistema funciona do seguinte modo: a aplicação nas estações faz uma requisição ao módulo PAM, o módulo passa essa requisição para a aplicação PAM no servidor que, por sua vez passa essa requisição para os módulos PAM no servidor que realmente atenderão a requisição e envia a resposta obtida para o módulo na estação que, então, devolve essa resposta para a aplicação que o requisitou.

Utilizando esse módulo, o arquivo de senhas dos usuários poderá ficar somente em um servidor e será necessário somente modificar a configuração do PAM para as aplicações nas estações que desejarem utilizar a autenticação remota.

Capítulo 2

Autenticação em Linux

O Linux é um Sistema Operacional aonde um dos seus pontos fortes é a segurança. A segurança em Linux é feita normalmente com controle de usuários que podem ou não acessar o sistema e o que eles podem fazer no sistema. O processo utilizado para verificar se um determinado usuário é realmente quem ele diz ser é chamado de autenticação. O padrão do Linux, mesmo em uma máquina isolada, é pedir a autenticação do usuário para que ele possa usar a máquina.

A autenticação consiste basicamente em obter uma identificação do usuário e, quase sempre, uma chave de acesso, normalmente uma senha, e comparar esses dados com os dados de usuários armazenados no sistema. Se os dados forem iguais o sistema é liberado para o usuário com os privilégios que esse usuário possui.

No Linux o meio mais comum de autenticação é feito da seguinte forma: o usuário fornece ao sistema sua identificação e sua senha, então o sistema localiza esse usuário, normalmente no arquivo `/etc/passwd`, aplica na senha do usuário uma função *hash*[1]e compara com a senha do usuário armazenada, que também passou por essa função *hash* antes de ser gravada. Se as senhas conferem o sistema é liberado com os privilégios que esse usuário possui.

Um sistema de autenticação remota funciona da mesma forma, só que as informações dos usuários, ou pelo menos as senhas, estão em outra máquina. Geralmente esse processo segue o modelo cliente-servidor. A identificação do usuário e sua chave são enviadas para o servidor e esse faz a verificação e devolve a resposta seja ela positiva ou negativa.

[1]Função *hash* é uma função que recebe um conjunto de *bits* que pode ter tamanho variado e devolve um conjunto de *bits* de tamanho fixo. A função *hash* garante que dois conjuntos de *bits* distintos, que sejam recebidos, gerarão dois conjuntos de *bits* resultado também distintos. Também garante que um mesmo conjunto de *bits* passando duas ou mais vezes pela função sempre gera o mesmo conjunto resultado. De um conjunto de *bits* resultado é impossível se chegar ao conjunto de *bits* de entrada da função.

O Linux já possui sistemas de autenticação remota, sendo os mais conhecidos o LDAP, que utiliza serviço de diretório, o NIS que compartilha as informações dos usuários na rede, e a solução Radius e Portslave, um protocolo de autenticação e um emulador de hardware utilizando esse protocolo [dC02]. Apresenta-se a seguir uma breve descrição de dois desses métodos.

2.1 LDAP

É comum tratar LDAP como uma aplicação, mas na verdade LDAP é apenas um protocolo com as especificações de como prover e consultar informações em uma base de dados no modelo de diretórios (serviço de diretório) sobre uma rede rodando TCP/IP. Um exemplo de aplicação que implementa o LDAP é OpenLDAP, normalmente está presente nas distribuições Linux.

O LDAP foi criado para especificar uma interface para acessar, via TCP/IP, um servidor de diretórios chamado X.500 [YHK95]. Com a queda do X.500 o LDAP acabou tornando o padrão para serviços de diretório sobre redes TCP/IP [ENH01].

2.1.1 Entendendo um Serviço de Diretório

Um Serviço de Diretório é basicamente um banco de dados, mas um banco de dados com algumas considerações. Toda informação que atende essas considerações pode ser incluída num Diretório [ENH01]. As considerações básicas são:

- os dados são relativamente pequenos;
- o banco de dados pode ser replicado;
- as informações são baseadas em atributos;
- dados são lidos com freqüência e raramente escritos;
- busca é uma operação comum.

Em um Diretório os dados são armazenados em forma de árvore hierárquica, objetivando fornecer um serviço de busca muito eficiente e não se preocupando muito com o tempo das atualizações.

2.1.2 Autenticação com LDAP

O papel do LDAP em um esquema de autenticação remota é simplesmente incluir no servidor de diretórios as informações necessárias sobre os usuários. A patir daí o trabalho é com quem faz a autenticação. Por exemplo, existe

um módulo PAM, o `pam_ldap.so`, que autentica buscando os dados em um servidor LDAP. A autenticação com LDAP pode ser segura, pois o LDAP aceita trabalhar com serviços de criptografia como o TLS e o SSL que será explicado posteriormente.

Embora seja possível fazer autenticação via LDAP, esse protocolo não foi desenvolvido com essa finalidade. Ainda não se sabe se o LDAP vai se desenvolver na direção de ajudar os administradores de sistemas ou não. Na verdade o LDAP ainda não encontrou seu nicho [ENH01]. Então, se hoje é possível fazer autenticação remota via LDAP, não se sabe se continuará sendo nas próximas versões.

2.2 NIS

O NIS (*Network Information Service*), como o próprio nome diz, é um serviço de informação de rede [How98]. O NIS permite compartilhar informações administrativas do sistema pela rede. Essas informações podem ser dados dos usuários (arquivo `passwd`), dados dos grupos (arquivo `group`) etc.

As informações compartilhadas pelo NIS ficam armazenadas no servidor normalmente em seus respectivos arquivos como `passwd`, `group` e são editados normalmente. Esses arquivos são pré-processados, por uma biblioteca *hash*, tendo seus dados armazenados em arquivos de banco de dados para serem disponibilizados para consultas dos clientes. Esse processo melhora a eficiência das consultas aos dados. A cada alteração que se faça nos arquivos originais do servidor é necessário dizer ao NIS para refazer esse processo [ENH01].

Uma boa característica do NIS é que ele pode ser entendido por “meros mortais”. Ele é semelhante a fazer cópias dos arquivos pela rede. Em muitos casos o administrador não precisa saber como o NIS funciona internamente, pois a administração do servidor continua sendo feita da mesma forma que antes e são necessários aprender apenas um ou dois novos procedimentos [ENH01].

Se a segurança do sistema for importante não se deve usar o NIS [ENH01]. 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.

Capítulo 3

Linux-PAM

Atualmente quase todas (se não todas) as aplicações mais conhecidas que precisam autenticar usuários suportam PAM. Pode-se ver o papel do PAM através de um exemplo simples: o programa `login` faz duas coisas, ele primeiro checa se o usuário que está tentando se *logar* é relmente quem ele diz ser, depois fornece ao usuário o serviço requisitado, nesse caso um *shell* que será executado com a identificação do usuário. Quando o programa `login` passou a utilizar o PAM, é o PAM que passa a ser o responsável por verificar se o usuário é quem ele diz ser [Mor02b]. Esse é o papel fundamental do PAM.

Uma das vantagens do PAM é a flexibilidade, com ele o administrador do sistema tem liberdade para estipular qualquer esquema de autenticação que será usado. É possível definir um esquema para quaisquer ou todas aplicações que suportam PAM. O administrador pode definir desde um esquema de autenticação sem nenhuma segurança, liberando o acesso a qualquer pessoa, até um robusto esquema de segurança fazendo o usuário passar por vários níveis de identificação (exame de retina, impressão digital, senha etc). O exemplo a seguir mostra a flexibilidade que o PAM fornece: um administrador de sistema (pai) deseja melhorar a habilidade matemática de seus filhos. Ele configura o jogo preferido das crianças (o jogo tem suporte a PAM) para fazer a autenticação perguntando o valor de uma multiplicação de dois números randômicos menores que 12. Com certeza se o jogo é o preferido das crianças elas irão se esforçar para aprender a multiplicar. Com o passar do tempo o administrador do sistema ainda pode aumentar o grau de dificuldade das perguntas.

O Linux-PAM trabalha com quatro tipos de gerenciamento separadamente [Mor02b]: gerenciamento de autenticação, gerenciamento de conta, gerenciamento de sessão e gerenciamento de senhas. O esquema de autenticação utilizado por uma aplicação e o(s) módulo(s) PAM responsável(is) para executar esse esquema são definidos em um arquivo de configuração. A seção configuração do PAM trará mais detalhes sobre esse assunto.

3.1 Módulo PAM x Aplicação PAM

É importante destacar a diferença entre um módulo PAM e uma aplicação PAM. Uma aplicação PAM é uma aplicação específica que necessite autenticar usuário ou obter informações sobre a conta do usuário implementada para utilizar o PAM para realizar essas tarefas, como o `login, ssh` etc. Para isso a aplicação faz requisições aos módulos PAM. Um módulo PAM é o responsável por executar as tarefas requisitadas e devolver as respostas esperadas pela aplicação. Um módulo pode servir a várias aplicações diferentes.

3.2 Funcionamento do PAM

A figura 3.1 dá uma boa idéia de como o PAM funciona.

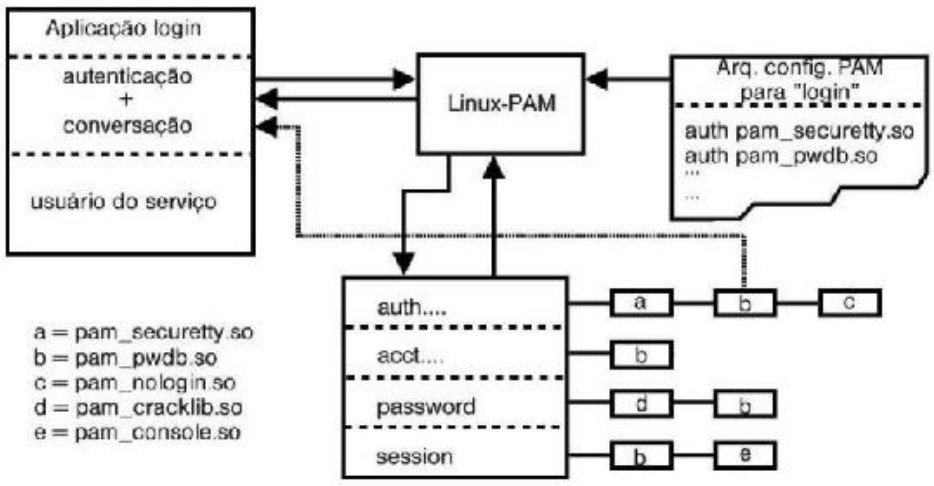

Figura 3.1: Funcionamento do PAM

O lado esquerdo da figura 3.1 representa a aplicação, no caso o `login` mas pode ser qualquer aplicação, seja em modo texto ou gráfico. A aplicação não sabe nada a respeito do esquema de autenticação que foi configurado pra ela. A biblioteca do Linux-PAM, no centro da figura 3.1, consulta o conteúdo do arquivo de configuração da aplicação e carrega os módulos necessários para o esquema de autenticação que será utilizado. Os módulos, quando chamados pelo Linux-PAM, realizam as várias rotinas de autenticação para a aplicação. Informações textuais fornecidas para o usuário ou requeridas dele podem ser

passadas pela função de conversação[1] fornecida pela aplicação.

3.3 Configuração do Linux-PAM

O Linux-PAM fornece duas alternativas para configurar as aplicações. Uma é descrever a configuração de todas as aplicações em um único arquivo chamado `pam.conf` localizado no diretório `/etc/` e a outra é colocar a configuração de cada aplicação em um arquivo separado que terá o nome do serviço a que ele se refere e ficará localizado no diretório `/etc/pam.d/`.

3.3.1 Arquivo Único

Uma linha normal de configuração do arquivo `/etc/pam.conf` tem cinco campos:

- `service-name`: nome do serviço associado com essa entrada. Normalmente o nome do serviço é o mesmo nome da aplicação.
- `module-type`: um dos quatro tipos de módulos:
  - `auth`: esse módulo é responsável por garantir que o usuário é quem ele diz ser e estabelecer suas credenciais.
  - `account`: esse módulo é responsável por verificar se a conta do usuário permite que ele tenha o acesso liberado nas circunstancias atuais como hora ou data, local de onde ele esta acessando etc.
  - `session`: esse módulo é responsável por executar tarefas que devem ser executadas antes e depois do usuário ter o serviço requisitado liberado para ele.
  - `password`: esse módulo é responsável por definir ou alterar o *token* de autenticação (normalmente é uma senha) do usuário.
- `control-flag`: usado para indicar como o PAM irá reagir no caso de sucesso ou falha do módulo a qual ele está associado. Como módulos podem ser empilhados esse campo pode indicar a importância de cada módulo na pilha. Existem quatro valores possíveis para esse campo:
  - `required`: indica que o sucesso do módulo associado é requerido para que a aplicação possa receber uma resposta de sucesso. No caso de falha a aplicação só tomará conhecimento após todos os módulos (do mesmo tipo) da pilha terem sido chamados.

[1] A função de conversação é explicada na seção 3.5.1

  - `requisite`: o mesmo que o item anterior, com a diferença de que no caso de falha do módulo a aplicação tomará conhecimento da falha imediatamente.
  - `sufficient`: indica que o sucesso do módulo atual é suficiente para retornar a aplicação com uma resposta de sucesso. Desde que nenhum módulo anterior com a opção de `required` tenha falhado e a resposta do módulo atual é de sucesso, a aplicação receberá a resposta de sucesso independentemente se tem outros módulos na pilha que não foram chamados.
  - `optional`: indica que tanto faz a resposta desse módulo. Um módulo com essa opção só fará efeito na ausência de resposta definitivas de falha ou sucesso dos módulos anteriores.

- `module-path`: caminho onde se encontra o arquivo do módulo, o módulo *plugável* em si. Se for passado só o nome do módulo ele será procurado no diretório padrão de módulos, normalmente `/lib/security/`.

- `args`: argumentos passados para o módulo quando ele é chamado. Normalmente os argumentos são específicos de cada módulo.

A tabela 3.1 mostra uma parte de um arquivo `pam.conf`. Além do `ftpd` poderiam estar configurados nesse mesmo arquivo outras aplicações como `sshd`, `login` etc.

`ftpd`	`auth`	`sufficient`	`pam_ftp.so`	
`ftpd`	`auth`	`required`	`pam_unix_auth.so`	`use_first_pass`
`ftpd`	`auth`	`required`	`pam_listfile.so`	`onerr = succeed`

Tabela 3.1: Exemplo de Configuração Usando Arquivo Único

3.3.2 Configuração Baseada em Diretório

Outra forma de configurar as aplicações PAM é colocar a configuração de cada aplicação em um arquivo separado, ou seja, existirá um arquivo de configuração para cada aplicação ou serviço PAM disponível. Esses arquivos de configuração deverão ficar no diretório `/etc/pam.d/`. O nome de cada arquivo é o mesmo nome da aplicação ou serviço a que ele se refere. As entradas nesses arquivos são semelhantes às entradas no arquivo único `pam.conf`, a única diferença é que nos arquivos separados o campo `service-name` não existe, pois o nome do arquivo já representa esse campo. A tabela 3.2 mostra um exemplo de configuração para a aplicação `ftpd`.

auth	sufficient	pam_ftp.so	
auth	required	pam_unix_auth.so	use_first_pass
auth	required	pam_listfile.so	onerr = succeed

Tabela 3.2: Exemplo de Configuração em Arquivos Separados: `ftpd`

Algumas vantagens podem ser listadas em se usar essa forma de configuração [Mor02b]. São elas:

- Menor chance de erros na configuração de uma aplicação.
- Menor chance de erro de digitação, pois tem um campo a menos para ser preenchido.
- Fácil de manter. Uma aplicação pode ser reconfigurada sem o risco de afetar outras aplicações.
- É possível fazer ligações simbólicas de diferentes serviços para um único arquivo de configuração.
- A leitura do arquivo de configuração pode ser mais rápida.
- Gerenciamento de pacotes fica mais simples. A cada nova aplicação instalada basta adicionar um arquivo no diretório `/etc/pam.d/`.

O Linux-PAM permite formar dois esquemas de configuração. Um é podendo utilizar as duas formas de configuração mas, não ao mesmo tempo. Se uma forma estiver sendo usada a outra não é checada. Outro é utilizando as duas formas de configuração ao mesmo tempo. Nesse caso se existir configuração para a aplicação nas duas formas de configuração prevalecerá a configuração baseada em diretório. Mais detalhes de configuração e de módulos PAM disponíveis podem ser encontrados em [Mor02b].

3.4 Desenvolvendo Módulos PAM

O módulo deve suprir um sub-conjunto das seis funções básicas para o desenvolvimento de um módulo. Juntas essas funções definem um Módulo Linux-PAM.

As seis funções são agrupadas em quatro grupos de gerenciamento independentes. São eles: *authentication*, *account*, *session* e *password*. Para se criar um módulo é preciso definir no mínimo um desses grupos por completo [Mor02c].

Para facilitar a vida dos administradores de sistemas diminuindo as possibilidades de erros nos arquivos de configurações, o desenvolvedor de módulos deve definir todas as seis funções básicas. Para aquelas funções que não serão abordadas pelo módulo, o módulo deve retornar um código de erro `PAM_SERVICE_ERR` e escrever uma mensagem apropriada no sistema de log.

3.4.1 Independência Entre os Grupos

A independência dos quatro grupos de gerenciamento faz com que o módulo permita que quaisquer um dos quatro possam ser chamados em qualquer ordem. Então, o desenvolvedor de um módulo não deve vincular a resposta de um grupo a outro, isto é, cada grupo deve ser totalmente independente dos demais. A utilidade dessa independência pode ser vista no seguinte exemplo: deseja-se trocar a senha de um usuário sem ter a necessidade de autenticá-lo primeiro. Em algumas situações isso é realmente válido, como quando o *root* deseja trocar a senha de um usuário qualquer. Mas também existem situações que isso não é uma boa política, como no caso em que um usuário mal intencionado deseja alterar a senha de um outro usuário.

3.4.2 Definição das Funções

Gerenciamento de Autenticação

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

Esta função realiza a tarefa de autenticar o usuário.

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

Esta função realiza a tarefa de alterar as credenciais do usuário de acordo com o esquema de autorização.

Gerenciamento de Conta

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

Esta função realiza a tarefa de verificar se o usuário tem permissão para obter acesso ao serviço requisitado na situação atual (data, hora, local de onde está acessando, etc).

Gerenciamento de Sessão

As duas funções seguintes controlam o início e o término de uma sessão. Por exemplo, no início de uma sessão o módulo pode desejar escrever uma mensagem no *log* do sistema avisando que uma nova sessão foi iniciada para um determinado usuário. Da mesma forma o módulo pode desejar escrever uma mensagem no *log* do sistema para informar que a sessão do usuário foi encerrada.

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

Esta função é chamada para começar um sessão.

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

Esta função é chamada para encerrar uma sessão.

Gerenciamento de Senhas

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

Esta função é usada para criar ou alterar a senha de um usuário.

Maiores detalhes sobre as funções, parâmetros, código de retorno, dicas sobre segurança na implementação de módulos e como compilar um módulo podem ser encontrados em [Mor02c].

3.5 Desenvolvendo Aplicações PAM

O Linux-PAM fornece uma série de funções para o desenvolvimento de aplicações baseadas em PAM. É importante perceber, quando escrevendo uma aplicação PAM, que essas funções são fornecidas de modo que elas ficam transparentes para a aplicação. A aplicação deve tratá-las como uma caixa preta, isto é, deve chamar as funções passando os parâmetros corretos e só aguardar a resposta, sem se preocupar com o que está sendo feito no interior das funções fornecidas pelo PAM [Mor02a].

O processo de autenticação é feito pelo Linux-PAM através de uma chamada a função `pam_authenticate()`. O valor retornado por essa função indicará se o usuário foi autenticado ou não. Se o PAM necessitar pedir alguma informação à aplicação, ele o fará por intermédio da função de conversação que deve ser fornecida pela aplicação. É importante ressaltar que a aplicação deve deixar a cargo do PAM as decisões de quando requisitar alguma informação do usuário.

O Linux-PAM pode ser usado da mesma forma por diferentes estilos de aplicações. Aplicações como o `login` e o `passwd` são aplicaçoes em modo texto, então trocas de informaçoes com o usuário são feitas com simples mensagens de texto. Entretanto aplicações em modo gráfico têm uma interface mais sofisticada. Elas geralmente interagem com o usuário através de caixas de diálogos. O PAM deixa a função de conversação a cargo da aplicação justamente para que cada aplicação possa estabelecer sua forma de apresentar ou requisitar informações ao usuário.

3.5.1 Função de Conversação

Uma aplicação PAM deve fornecer uma função de conversação. Ela é usada para comunicação direta entre o módulo PAM carregado e a aplicação, e o que ela fará normalmente é prover meios para que o módulo possa requisitar informações ao usuário como senha etc. O PAM define uma estrutura de conversação da seguinte forma:

```
struct pam_conv {
    int (*conv)(int num_msg,
        const struct pam_message **msg,
        struct pam_response **resp,
        void *appdata_ptr);
    void *appdata_ptr;
};
```

Note que o primeiro membro dessa estrutura é um ponteiro para uma função. A aplicação deve inicializar essa estrutura fazendo o primeiro membro apontar para uma função do mesmo tipo definida pela aplicação. Depois essa estrutura é passada para o PAM através de uma função de inicialização do PAM.

Quando um módulo faz uma chamada à função referenciada por `conv()`, o parâmetro `appdata_ptr` é apontado para o segundo membro da estrutura de conversação. Os outros argumentos da função dizem respeito a informação trocada pelo módulo e a aplicação. O `num_msg` guarda o número de mensagens que foram passadas para a função, `msg` guarda as mensagens passadas para a função, `resp` é o local onde as respostas da aplicação serão informadas.

O PAM ainda disponibiliza uma série de funções utilitárias para facilitar o desenvolvimento de aplicações PAM com segurança. A descrição completa dessas funções e das funções básicas, além de dicas de segurança e detalhes sobre como compilar uma aplicação PAM podem ser encontrados em [Mor02a].

Capítulo 4

Comunicação e Segurança

4.1 Sockets

Socket é o meio primário de comunicação com outras máquinas. Aplicações como `ftp`, `telnet`, `ssh` e outros programas conhecidos que trabalham em rede utilizam *sockets*. Em uma conexão o *socket* é identificado pelo endereço da máquina conectada e um número que identifica a porta que foi estabelecida a conexão [MCCI99]. *Sockets* também podem ser usados para comunicação entre processos distintos numa mesma máquina. Por exemplo, a função `printing` do UNIX utiliza *socket* [SM99].

Aplicações que utilizam *sockets* trabalham na política cliente/servidor. Uma conexão *socket* é estabelecida da seguinte forma:

1. O servidor fica constantemente "escutando" em uma porta pré-definida. O servidor não precisa saber onde o cliente está e nem em que momento acontecerá um pedido de conexão. Ele simplesmente fica esperando.

2. O cliente que deseja estabelecer uma conexão entra em contato com o servidor passando a identificação de seu *socket* para ele. O cliente já deve conhecer o endereço do servidor e qual a porta onde o servidor está "escutando" para iniciar uma conexão.

3. O servidor pode aceitar ou recusar a conexão. Uma vez aceita a conexão o servidor cria um novo *socket* e envia sua identificação para o cliente. Agora ambos os lados sabem o endereço (*socket*) um do outro e estão aptos a trocar mensagens até que um dos dois encerre a conexão.

4.1.1 Como Funciona uma Conexão

Existem dois grupos de rotinas (chamadas feitas ao *socket*) que são usadas em uma conexão [Rib00], rotinas primárias e rotinas de utilidades. Segue uma

breve descrição das rotinas primárias.

Chamada Socket

É utilizada para criar um novo *socket*. Essa rotina devolve um descritor de arquivos para o *socket* criado. Nessa chamada é passada a família de protocolos que poderão ser usados na conexão.

Chamada Connect

Essa chamada é feita pelo cliente quando este deseja estabelecer uma conexão com algum servidor. Com essa chamada é passado o *socket* (endereço e porta) onde o servidor está "escutando".

Chamada Bind

A chamada *bind* é usada para "amarrar" um *socket* recém criado a um endereço local, principalmente a porta. Normalmente essa chamada é utilizada pelo servidor para estabelecer em qual porta ele ficará "escutando". Geralmente nos clientes esse processo é feito automaticamente com a chamada *connect*.

Chamada Listen

Essa chamada é utilizada pelo servidor para colocar o *socket* para "escutar" na porta pré-estabelecida pela chamada *bind*. Com isso o servidor estará pronto para receber conexões dos clientes.

Normalmente um servidor utiliza um socket para ficar permanentemente "escutando" em uma porta, a cada conexão que chega um novo *socket* é criado para a conexão com uma chamada *accept* e o primeiro *socket* continua sua tarefa de ficar "escutando" na mesma porta de antes.

Chamada Accept

É utilizada pelo servidor para aceitar um pedido de conexão. Após o servidor ter sido posto para "escutar" pedidos de conexões a chamada *accept* é utilizada para verificar se tem algum pedido, caso exista algum pedido um novo *socket* é criado já com a conexão estabelecida e seu descritor é retornado para quem fez a chamada *accept*. Geralmente o servidor volta a "escutar" novos pedidos de conexão.

Chamada Write

A chamada *write* é usada sempre que se deseja enviar uma mensagem pela conexão estabelecida. Tanto o cliente quanto o servidor podem usá-la. Essa chamada requer o descritor do socket através do qual vai a mensagem, a mensagem e o tamanho da mensagem.

Chamada Read

É utilizada sempre que se deseja receber uma mensagem vinda pela conexão estabelecida. Como a chamada *write*, tanto o cliente quanto o servidor podem utilizar a chamada *Read*. Normalmente em uma conexão cliente/servidor, após estabelecida a conexão, o cliente usa a chamada *write* para fazer uma requisição ao servidor e em seguida usa a chamada *read* para receber a resposta. Por sua vez o servidor usa primeiro a chamada *read* para receber uma requisição do cliente e em seguida usa a chamada *write* para enviar sua resposta.

Essa chamada também recebe três parâmetros: um descritor do socket do qual será lido a mensagem, um *buffer* no qual será armazenada a mensagem e o tamanho do *buffer*. Se tiver mais dados do que o *buffer* suporta será necessário uma nova chamada *read* para receber o restante dos dados. No caso de o *buffer* ser maior do que a mensagem recebida ela é colocada normalmente no *buffer* e o tamanho da mensagem recebida é retornado.

Chamada Close

Também pode ser usada pelo cliente e pelo servidor. Quando um destes deseja encerrar a conexão deverá usar a chamada *close* para realizar essa tarefa. A chamada *close* encerra imediatamente a conexão e desaloca o socket.

A ordem dessas chamadas pode ser vista na figura 4.1.

Resumindo uma Conversação Cliente/Servidor

Normalmente em uma conversação cliente/servidor as chamadas ao *socket* são feitas da seguinte maneira:

1. O servidor faz uma chamada *socket* para criar um novo *socket*.
2. O servidor faz uma chamada *bind* para definir em qual porta ficará "escutando".
3. O servidor faz uma chamada *listem* para ficar "escutando" na porta predefinida.

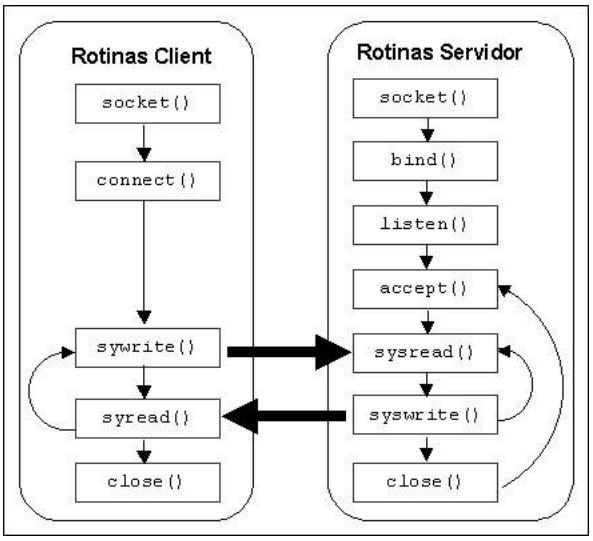

Figura 4.1: Fluxo da Transação de Sockets

4. O servidor faz uma chamada *accept* para verificar se tem algum pedido de conexão.

5. O cliente faz uma chamada *socket* para criar um novo *socket*.

6. O cliente faz uma chamada *connect* para fazer um pedido de conexão ao servidor.

7. Agora cliente e servidor usam as chamadas *read* e *write* para trocarem mensagens.

8. Cliente ou servidor usa a chamada *close* para encerrar a conexão.

4.2 Criptografia

Com o aumento do número de pessoas utilizando redes de computadores privadas ou públicas o aumento de negócios feitos através dessas redes traz junto o aumento de informações confidenciais trafegando nas redes. Com isso surge a necessidade de garantir a integridade e a segurança dessas informações. A criptografia surgiu como o principal meio para garantir que informações trafegando pela rede não possam ser entendidas por pessoas não autorizadas. A idéia básica da criptografia é garantir que só o destinatário legítimo da mensagem possa ler a mensagem.

Este trabalho necessita de um bom mecanismo de criptografia, pois estará trafegando entre o cliente e o servidor senhas de usuários da rede. Será apresentado nesta seção alguns conceitos, um breve histórico e alguns modelos de implementação.

4.2.1 Conceitos e Históricos

A palavra criptografia vem do grego cryptos que quer dizer secreto, oculto. Segundo o dicionário Aurélio criptografia significa "a arte de escrever em cifra ou em código". Um criptosistema é descrito da seguinte forma por [Fer95]:

> Dada uma mensagem e uma chave de codificação como entrada, o método de codificação produz como resultado uma mensagem codificada, a qual pode ser armazenada num meio qualquer ou enviada a um destinatário; para decodificar esta mensagem utiliza-se o método de decodificação passando como entradas a mensagem codificada e a chave de decodificação obtendo-se a mensagem original.

Esse sistema é ilustrado na figura 4.2.

Figura 4.2: Criptografar e Descriptografar

Segundo [Cou00] a finalidade da criptografia é estudar os métodos para codificar uma mensagem de modo que só seu destinatário legítimo consiga interpretá-la, ela é a arte dos códigos secretos.

Da mesma forma que tem pessoas procurando proteger as informações existem pessoas que procuram o inverso. Com isso à medida que a criptografia vai se desenvolvendo ela traz consigo a evolução da criptoanálise. O objetivo da criptoanálise é decifrar os códigos criptografados. É importante notar a diferença entre as palavras decodificar e decifrar. Segundo [Cou00], decodificar é a ação feita pelo destinatário legítimo de uma mensagem quando a recebe codificada e deseja ler a mensagem. Já decifrar é o que faz um usuário não autorizado quando pega uma mensagem codificada e deseja ler essa mensagem, ele terá que "quebrar" a criptografia utilizada na mensagem.

A criptografia não é uma coisa recente, ela já era utilizada a milhares de anos e por isso foi dividida da seguinte forma:

- Criptografia Tradicional: antes da computação a criptografia era orientada a caracter, pois ela se baseava na substituição de um caracter por outro ou na troca de posição dos caracteres no texto [Fer95]. Com o objetivo de aumentar a segurança, alguns métodos faziam tanto a troca quanto a substituição dos caracteres várias vezes. Esses métodos são chamados de simétricos.

- Criptografia Moderna ou Computacional: com o avanço da computação surgiram métodos criptográficos computacionais, que eram feitos por um computador ou por um circuito integrado específico, fazendo com que a codificação e decodificação das mensagens fossem feitos muito mais rápido [Oli02].

4.2.2 Métodos Criptográficos

A criptografia convencional ou criptografia de chave secreta ou simétrica utiliza uma chave única tanto para criptografar quanto para descriptografar. Um exemplo é o DES (*Data Encryption Standard*) [Sch96], ele é um método de criptografia convencional que foi utilizado pelo governo americano. O método de criptografia convencional é ilustrado pela figura 4.3.

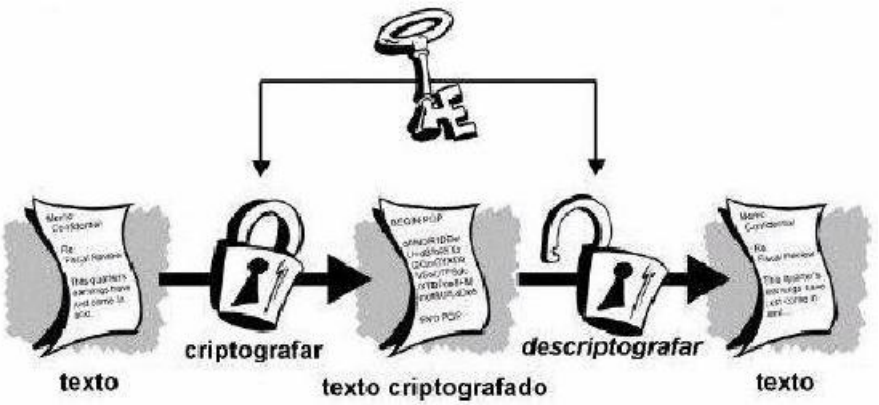

Figura 4.3: Criptografia Convencional ou Criptografia de Chave Simétrica

Os algoritmos criptográficos de chave simétrica tem uma grande vantagem, a velocidade de codificação e decodificação [Net98]. Neste método a chave deve ser enviada para o destinatário para que a mensagem possa ser decodi-

ficada, mas isto causa o problema de que se a mensagem for interceptada a chave estará junta.

Para resolver esse problema surgiu a criptografia de chave pública. Este conceito foi iniciado por Whitfild Diffie e Martin Hellman em 1975 [Sch96]. A criptografia de chave pública utiliza um par de chaves, uma chave pública que é utilizada para criptografar e uma chave privada ou secreta que é utilizada para descriptografar. A chave pública é divulgada para que todos possam criptografar as informações e a chave privada é mantida só com aqueles que poderão descriptografar as informações. Não é possível descriptografar utilizando a chave pública. Dependendo do tamanho das chaves, tentar descobrir a chave privada através da chave pública pode levar milhares de anos com os computadores mais potentes que existem hoje, ou seja, é impraticável tentar fazê-lo [Sch96]. A figura 4.4 ilustra o método de criptografia de chave pública.

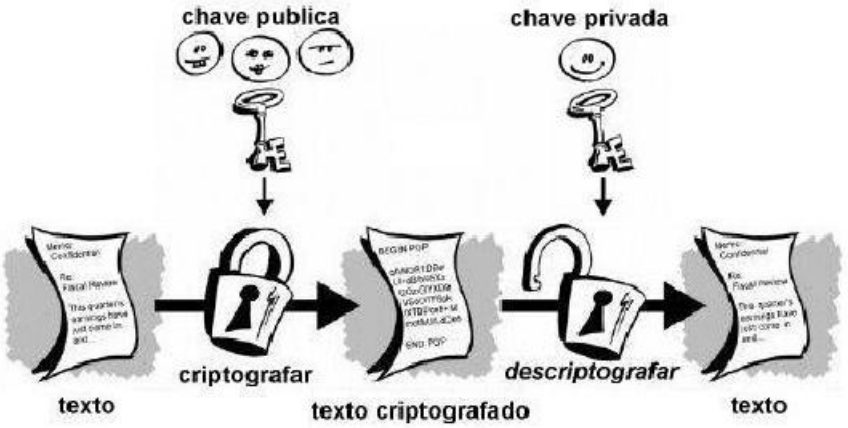

Figura 4.4: Criptografia de Chave Pública

Alguns Métodos Simétricos

- **Blowfish:** Este algoritmo de criptografia em bloco foi inventado por Bruce Schneier [Sch96] e permite a utilização de chaves de até 448 *bits*, sendo otimizado para ser executado em máquina de 32 ou 64 *bits*.

- **IDEA:** O algoritmo de criptografia de dados internacional - IDEA foi desenvolvido na Suíça por James Massey e Xuenjia Lai e publicado em 1990 [Sch96]. Ele é um algoritmo de blocos com tamanho de 64 *bits* e utiliza chaves de 128 *bits*. Acredita-se que ele seja um algoritmo bastante poderoso, visto que ainda não existe nenhum método de ataque

efetivo contra o mesmo e também por ele ter resistido bem contra métodos aplicados com êxito sobre outros algoritmos [Fer95].

- **DES:** O DES(*Data Encrytion Standard*) [Sch96] é o exemplo mais difundido de algoritmo criptográfico de chave única. Ele foi desenvolvido pela IBM e adotado pelos Estados Unidos em 1977. O DES é um algoritmo de bloco e trabalha dividindo o texto em blocos de 8 caracteres, cifrando cada bloco com uma chave de 56 *bits* (mais 8 *bits* de paridade, totalizando uma chave de 64 *bits*. Esse método é passível de ser quebrado usando o método da força bruta, bastando para tal testar as 2^{56} chaves possíveis [Sch96].

- **Triple-DES:** Este algoritmo [Sch96] é um método para tornar o DES mais seguro e para atingir tal objetivo aplica-se o algoritmo do DES três vezes com duas chaves diferentes: inicialmente cifra-se o texto com a chave *C1*, depois com a chave *C2* e finalmente com a chave *C1* novamente. Para quebrar tal método com o método da força bruta é necessário testar as 2^{112} chaves possíveis.

Alguns Métodos Assimétricos

- **RSA:** Este método de criptografia com chave pública ou assimétrico foi proposto em 1977 por Rivest, Shamir e Adleman [Sch96] e seu nome deriva das iniciais dos sobrenomes de seus inventores. Este método basea-se em uma chave pública composta por *n* e *e*, onde *n* é o produto entre dois números primos e *e* é um número inteiro positivo que seja inversível módulo $\phi(n)$. Já a chave privada é composta por *n* e *d*, onde *n* como já dito é o produto de dois números primos e *d* é o inverso de *e* em $\phi(n)$. Detalhes teóricos e de implementação podem ser encontrados em [Cou00].

- **Rabin:** Este método baseia-se na dificuldade de se extrair a raiz quadrada em aritmética modular de um número composto [Sch96]. Para aplicá-lo escolhe-se dois números primos, *p* e *q*, sendo que ambos devem ser congruentes a 3 módulo 4 e tais primos são a chave privada. Já a chave pública é calculada pelo produto de *p* e *q*. Pode-se encontrar mais detalhes em [Fer95] e [Sch96], sobre este algoritmo.

- **ElGamal:** Criado por Taher ElGamal [Sch96] é um sistema criptográfico de chave pública que pode ser usado tanto para cifrar mensagens quanto para assinatura digital. Tem sua segurança baseada na dificuldade de calcular logaritmos discretos e aritmética modular.

Estes são os principais métodos de criptografia, existem outros métodos, principalmente de chave pública, que podem ser encontrados em [Fer95] e [Sch96].

4.3 Comunicação Segura

Visto como é feita uma comunicação em rede e métodos de garantir segurança de informações que trafegam numa rede falta agora integrar essas duas coisas para que se tenha uma comunicação segura.

Esta seção descreve um método que integra *sockets* e criptografia. O método descrito aqui é o utilizado na implementação deste trabalho.

4.3.1 SSL - *Secure Sockets Layer*

O SSL foi criado pela *Netscape Corporation* e é um protocolo de segurança para fornecer autenticação e cifragem em redes TCP/IP, principalmente a internet. O SSL roda sobre protocolos de transporte confiável [Net96]. Com o protocolo SSL, clientes e servidores se autenticam e então podem trocar dados criptografados entre si.

O sigilo dos dados que trafegam entre o cliente o servidor é garantido pelo SSL através de uma criptografia simétrica. É utilizada a criptografia simétrica por ela ser mais rápida que a criptografia assimétrica, mas as chaves utilizadas são geradas de forma aleatória para cada conexão e trocadas entre o cliente e o servidor durante o processo de *handshake* onde é utilizada criptografia assimétrica. Com a finalidade de evitar que as informações, mesmo decifradas, sejam modificadas possibilitando um ataque de escuta ativa, o SSL incorpora à cada mensagem um MAC (*Message Authentication Code*). O MAC é calculado a partir de funções de *hash* seguras, garantindo a integridade das informações trocadas.

O SSL ainda faz a autenticação dos clientes e servidores com a finalidade de verificar a identidade uns dos outros. Para isso é utilizado criptografia assimétrica e certificados digitais.

Algumas características interessantes sobre o SSL podem ser destacadas:

- Apesar da maioria das implementações serem feitas para redes TCP/IP o SSL roda sobre qualquer protocolo de transporte confiável, garantindo uma independência de protocolos.

- Diferentes implementações do SSL são compatíveis devido a sua especificação bem detalhada e o uso de métodos de criptografia bastante conhecidos.

- Se for necessário incluir novos parâmetros ou métodos de criptografia no protocolo, isto pode ser feito sem a necessidade de criação de um novo protocolo ou a implementação de toda uma nova biblioteca.

- Para compensar a grande demanda por recursos que o SSL requer ele dispõe de uma opção de armazenamento em *cache* de informações referentes a sessão, fazendo com que diminua o esforço computacional em conexões sucessivas.

4.3.2 Estrutura do SSL

Segundo [Ope02] o protocolo SSL é dividido em duas camadas, uma de mais baixo nível chamada *Record* e outra que trabalha nos níveis superiores chamada de *Handshake*. É função da camada *record* encapsular os dados das camadas superiores em pacotes compactados e cifrados e repassá-los para a camada de transporte. Esta camada recebe dados não interpretados das camadas superiores em forma de bloco de dados de tamanhos variados. Os blocos são encapsulados em registros. Se um bloco não couber em um registro, este será fragmentado. Os registros ainda são compactados e criptografados pelos algoritmos e chaves definidos no processo de *handshake* [Ope02].

Nas camadas superiores está a outra camada do SSL, a camada *handshake*. É nesta camada que é feita a autenticação entre as partes envolvidas na comunicação. Também é nesta camada que é feita a escolha dos algoritmos criptográficos, troca das chaves para estes algoritmos e a definição da chave de sessão. Tudo isso é feito antes que a camada de aplicação receba ou envie algum *byte*. Nesta camada encontram-se os protocolos *Handshake*, *ChangeCipherSpec* (CCS) e *Alert*. Para entender melhor as funções desta camada é importante saber o conceito de sessão no SSL. Uma sessão no SSL é composta por um conjunto de dados que são gerados após um processo de *handshake* completo. Uma sessão é dependente do estado. É função do protocolo *Handshake* manter a consistência dos estados de uma sessão tanto no cliente quanto no servidor. Uma sessão suporta várias conexões. As sessões são compostas pelos seguintes dados:

- **session ID:** Um valor arbitrário escolhido pelo servidor para identificar esta sessão;

- **peer certificate:** Usado para certificar uma organização. Está no formato X.509 e dentre outras coisas encontra-se dentro dele a chave pública da entidade que está utilizando aquela aplicação;

- **compression method:** Algoritmo usado na compressão dos dados;

- **cipherspec:** Especifica que conjunto de algoritmos de cifragem e de *hash* serão utilizados;
- **Mastersecret:** Um segredo de 48 *bytes* compartilhado pelo servidor e pelo cliente;
- **IsResumable:** *Flag* utilizada para indicar se a sessão pode ou não ser retomada ao iniciar uma nova conexão.

Capítulo 5

Implementação

Neste trabalho foi implementado um sistema que permite que um usuário possa se autenticar e trocar sua senha em um servidor remoto, ou seja, quando o usuário requer uma autenticação os dados deste usuário são enviados para um servidor remoto e a autenticação é feita lá no servidor. E da mesma forma quando um usuário deseja mudar sua senha a mudança é realizada também no servidor.

5.0.3 Integrando o Módulo e a Aplicação

O módulo e a aplicação utilizam a filosofia cliente/servidor, sendo o módulo o cliente e a aplicação o servidor. A comunicação entre eles é feita utilizando *sockets* e o protocolo SSL através da biblioteca *OpenSSL* [Ope02].

A aplicação deve estar sempre rodando no servidor aguardando conexões dos clientes. É conveniente deixar a aplicação rodando em *background* e fazer com que ela seja chamada no momento da inicialização do Linux. No cliente, sempre que uma aplicação PAM configurada para usar o módulo `pam_remote` fizer uma requisição ao PAM, o módulo estabelece uma conexão segura com a aplicação no servidor e passa as requisições para ela. Sempre que o PAM no servidor precisar obter alguma informação para atender a requisição, a função de conversação da aplicação no servidor é ativada. Essa função então repassa o que foi pedido ao módulo no cliente. O módulo no cliente repassa o pedido para a aplicação que o chamou ativando a função de conversação desta aplicação, fazendo, assim, a conversação entre a aplicação no cliente e os módulos no servidor. Quando a aplicação no servidor recebe uma resposta definitiva esta resposta é enviada ao módulo no cliente e conexão é encerrada. A figura 5.1 mostra a comunicação que é feita entre o módulo `pam_remote` e a aplicação `pamrad` a cada requisição feita por uma aplicação no cliente.

A idéia é fazer com que as aplicações PAM nos clientes utilizem os serviços

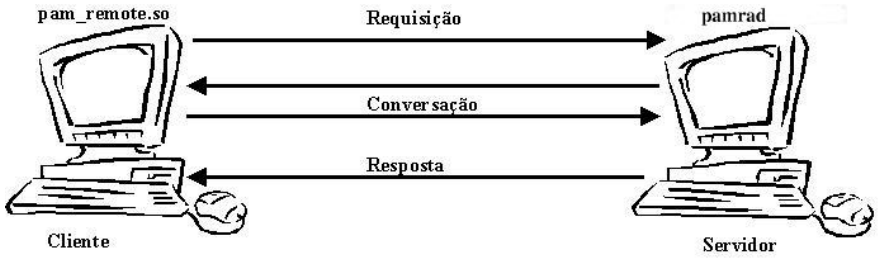

Figura 5.1: Comunicação Entre `pam_remote` e `pamrad`

do PAM no servidor como se estivessem utilizando localmente.

5.1 Funcionalidade do Sistema Desenvolvido

5.1.1 Cascata

Este módulo pode ser usado em cascata da seguinte forma: no cliente as aplicações utilizam o módulo `pam_remote` normalmente e no servidor a aplicação `pamrad` é configurada também para utilizar o módulo `pam_remote`. Desta forma um cliente busca as senhas em um servidor que por sua vez busca as senhas em outro servidor.

Este esquema pode ser estendido sucessivamente.

5.1.2 Restrições

O interessante é que todas as informações sobre os usuários fiquem somente no servidor, sem a necessidade de tê-los cadastrados também nos clientes. Um módulo PAM não é suficiente para isso. Algumas aplicações como o `login` não necessitam apenas autenticar usuários, elas precisam de mais informações sobre os usuários como o diretório *home* do usuário, a identificação do usuário, grupo etc. Geralmente estas informações ficam armazenadas no arquivo `passwd`. Estas informações não são buscadas nos módulos PAM e nem é função do PAM fornecer tais informações. Com isso o módulo desenvolvido neste trabalho não é suficiente para permitir que as informações a respeito dos usuários fiquem somente no servidor. A principal informação sobre os usuários que este módulo faz com que não seja necessário ter na máquina local é a senha.

5.1.3 O Que Pode Ser Feito

Cópia de Arquivos

Uma opção do uso deste módulo é fazendo com que as senhas no servidor fiquem armazenadas no arquivo `shadow` e não no `passwd`. Então faz a cópia do arquivo `passwd` de tempos em tempos para todos os clientes na rede. Como arquivo `passwd` não contém as senhas dos usuários não é necessário a preocupação de criptografar esses arquivos para trafegarem na rede.

O problema desta solução é que se alguma informação do usuário, exceto a senha, for alterada no servidor, esta alteração não terá efeito imediato nos clientes.

NIS Seguro

O problema no NIS, como já foi mencionado neste trabalho, é a falta de segurança, pois os dados trafegam em claro pela rede. Com o `pam_remote` é possível montar um esquema com NIS de forma que garanta a segurança no tráfego das senhas. O esquema pode ser da seguinte forma: faz com as senhas no servidor fiquem no arquivo `shadow`, disponibiliza o arquivo `passwd` para os clientes através do NIS e utiliza o `pam_remote` para fazer a autenticação no servidor. Como visto, o arquivo `passwd` não contém as senhas dos usuários, então não tem problema as informações contidas nele trafegarem em claro pela rede.

Implementação de Um Módulo NSS

Informações do tipo que aplicações como o `login` necessitam são obtidas, normalmente, com chamadas ao NSS (*Name Service Switch*). O NSS é parecido com o PAM, ele possui módulos que são responsáveis por fornecer informações do sistema a respeito de usuários, grupos, etc. Como o PAM, essas informações podem ser obtidas de maneiras diferentes de acordo com o módulo NSS chamado e para a aplicação que requisita a informação é indiferente. Então se pode implementar um módulo NSS que ao receber requisições a respeito dos usuários ele busque estas informações no servidor e repasse para a aplicação.

Desta forma, utilizando o `pam_remote` para fazer a autenticação remota e o módulo NSS buscando os dados no servidor tem-se um sistema completo de autenticação que permite a permanência dos dados dos usuários somente no servidor e sem a necessidade de utilizar um outro sistema qualquer como o NIS.

Estas são algumas idéias de utilização do `pam_remote`. Outras formas de

utilização podem ser encontradas, depende da criatividade dos administradores de sistemas.

5.2 Instalação e Configuração

Nesta seção será mostrado os passos necessários para instalar e configurar o módulo `pam_remote` e a aplicação `pamrad`.

5.2.1 Instalação

Os arquivos necessários para instalação do sistema estão listados na tabela 5.1.

pam_remote.cpp	fileconf.h
pam_remote_server.cpp	pam_remote.conf
pam_remote_server.h	pam_remote.crt
netssl.h	pam_remote.key
constants.h	pam_remote_server.conf
pam_remote_server.pam	Makefile

Tabela 5.1: Arquivos necessários para a instalação

Todos esses arquivos devem ser copiados para um diretório qualquer da máquina onde se deseja fazer a instalação.

Compilando o Módulo e a Aplicação

É necessário estar no diretório onde foram copiados os arquivos para executar os comandos descritos abaixo.
Para compilar o módulo utilize o seguinte comando:
`make pam_remote`
Para compilar a aplicação utilize o seguinte comando:
`make pamrad`
Para compilar o módulo e a aplicação ao mesmo tempo, o seguinte comando pode ser usado:
`make all`

Instalando o Módulo

Para fazer a instalação do módulo é necessário ter poderes de *root* no sistema, pois será feito escrita em diretórios onde só o *root* tem permissão para isso. Para instalar o módulo utilize o seguinte comando:

`make install_client`
Esse processo irá copiar o arquivo `pam_remote.so` para o diretório de módulos PAM `/lib/security` e também irá criar (caso não exista) o diretório `/etc/pam_remote` e copiar os arquivos `pam_remote.conf` e `pam_remote.crt` para esse diretório.

Se o diretório onde ficam os módulos PAM na máquina onde será instalado o módulo não for o `/lib/security` será necessário editar o `Makefile` para colocar o diretório correto.

Instalando a Aplicação

Para fazer a instalação da aplicação é necessário ter poderes de *root* no sistema, pois será feito escrita em diretórios onde só o *root* tem permissão para isso. Para instalar a aplicação utilize o seguinte comando:
`make install_server`
Esse processo irá copiar o arquivo `pamrad` para o diretório `/usr/bin`, criar (caso não exista) o diretório `/etc/pam_remote` e copiar os arquivos:
`pamrad.conf`;
`pam_remote.crt`;
`pam_remote.key`;
para esse diretório. Irá ainda copiar o arquivo `pamrad.pam` para o diretório `/etc/pam.d` com o nome de `pamrad`.

5.2.2 Configuração

Configurando o Cliente

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 inclua a seguinte linha no arquivo:
`ServerAddr=<porta:endereço do servidor>`
Para informar o arquivo com o certificado SSL inclua a seguinte linha no arquivo:
`CertFile=<caminho do arquivo de certificado>`
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 use o símbolo # no começo da linha que deseja colocar como comentário.

Configurando o Servidor

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" inclua a seguinte linha no arquivo:
`PortNumber=<número da porta>`
Para informar o arquivo com o certificado SSL inclua a seguinte linha no arquivo:
`CertFile=<caminho do arquivo de certificado>`
Para informar o arquivo com a chave privada SSL inclua 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 use o símbolo # no começo da linha que deseja colocar como comentário.

5.2.3 Certificado e Chave Privada SSL

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.

5.3 Dicas de Implementação

O desenvolvimento de módulos e aplicações PAM é razoavelmente simples. Não é uma boa estratégia tentar implementar um módulo PAM sem saber como funciona uma aplicação PAM e vice-versa. É importante dedicar um tempo para entender bem a função de conversação antes de começar a implementar.

Um módulo PAM é uma biblioteca que pode ser de ligação estática ou dinâmica e as funções que são acessadas de fora do módulo pelo PAM são declaradas nas bibliotecas que acompanham o linux como `extern` e isso é suficiente para funcionar com a linguagem C, mas caso se deseje implementar um módulo em C++ é necessário mudar a declaração dessas funções para `extern "C"` para que elas sejam interpretadas como funções no padrão C, pois o PAM só reconhece funções no padrão C.

5.4 Código Fonte

Como foi dito na seção anterior, para escrever módulo PAM em C++ é necessário fazer uma modificação na declaração das funções que são acessadas de fora do módulo. O arquivo `/usr/security/pam_modules.h` traz, entre outras coisas, as declarações dessas funções e é ele que deve ser modificado. A seguir apresenta-se um trecho desse arquivo.

```
1  .
2  .
3  .
4
5  #ifdef PAM_STATIC
6
7  #define PAM_EXTERN static
8
9  struct pam_module {
10     const char *name;      /* Name of the module */
11
12     /* These are function pointers to the module's key functions.  */
13
14     int (*pam_sm_authenticate)(pam_handle_t *pamh, int flags,
15                 int argc, const char **argv);
16     int (*pam_sm_setcred)(pam_handle_t *pamh, int flags,
17            int argc, const char **argv);
18     int (*pam_sm_acct_mgmt)(pam_handle_t *pamh, int flags,
19              int argc, const char **argv);
20     int (*pam_sm_open_session)(pam_handle_t *pamh, int flags,
21                 int argc, const char **argv);
22     int (*pam_sm_close_session)(pam_handle_t *pamh, int flags,
23             int argc, const char **argv);
24     int (*pam_sm_chauthtok)(pam_handle_t *pamh, int flags,
25              int argc, const char **argv);
26 };
27
28 #else /* !PAM_STATIC */
29
30 #define PAM_EXTERN extern
31
32 #endif /* PAM_STATIC */
33
34 /* Lots of files include pam_modules.h that don't need these
35  * declared.  However, when they are declared static, they
36  * need to be defined later.  So we have to protect C files
37  * that include these without wanting these functions defined.. */
38
```

```
39  #if (defined(PAM_STATIC) && defined(PAM_SM_AUTH)) || !defined(PAM_STATIC)
40
41  /* Authentication API's */
42  PAM_EXTERN int pam_sm_authenticate(pam_handle_t *pamh, int flags,
43                                     int argc, const char **argv);
44  PAM_EXTERN int pam_sm_setcred(pam_handle_t *pamh, int flags,
45                 int argc, const char **argv);
46  .
47  .
48  .
```

No caso do módulo ser uma biblioteca de ligação dinâmica o modo como as funções são declaradas é definido na linha 30 deste trecho de código. Para escrever um módulo PAM com C++ essa linha deve ser modificada, ficando da seguinte forma: `#define PAM_EXTERN extern "C"`. Desta forma, mesmo utilizando C++, as funções serão "vistas" de fora do módulo como funções no padrão C.

Todos os códigos fontes desenvolvidos neste trabalho são abertos segundo a licença GPL (*General Public License*) [Fou01].

Capítulo 6

Considerações Finais

Este trabalho foi indicado como Trabalhos Futuros na monografia de graduação de Paulo Sérgio [dC02] e para desenvolvê-lo foi necessário estudar métodos de autenticação em linux, administração do PAM, desenvolvimento de aplicações PAM, desenvolvimento de módulos PAM, comunicação em rede via *sockets*, criptografia e a biblioteca *OpenSSL*.

Com a conclusão deste trabalho tem-se como resultado um módulo PAM capaz de fazer o gerenciamento de autenticação, o gerenciamento de conta, o gerenciamento de sessão e o gerenciamento de senhas buscando os dados em um servidor remoto.

6.1 Trabalhos Futuros

A implementação do módulo NSS mencionado na seção anterior é sugerida como trabalho futuro para ser incorporado ao sistema desenvolvido por este trabalho tendo assim, um sistema de autenticação completo e independente.

6.1.1 Ideia Para Implementação do Módulo NSS

O que se deseja é fazer com que quando uma aplicação qualquer requerer uma informação do sistema como dados do usuário, por exemplo, o módulo NSS busque essas informações em um servidor remoto. Isso pode ser feito utilizando a mesma estratégia usada para a implementação do módulo PAM feita neste trabalho. O módulo NSS a ser implementado será somente uma interface entre a aplicação e os módulos NSS no servidor. Quando uma função do módulo NSS é chamada, este entra em contato com o servidor e passa a função que foi requisitada. Uma aplicação, que também deverá ser implementada, ficará no servidor esperando as requisições dos módulos no cliente. Quando a aplicação recebe uma requisição, ela faz a chamada referente a essa requisição

ao módulo NSS no servidor e ao receber a resposta, devolve esta para o módulo no cliente. Este então, passa a resposta, que provalmente é uma estrutura contendo os dados solicitados, para a aplicação que o requisitou.

Informações sobre implementação e configuração de módulos NSS podem ser obtidas na documentação das principais distribuições Linux. Utilize o comando `info libc` e escolha o item `Name Service Switch` no *Menu*.

Referências Bibliográficas

[Con] Conectiva S.A. *Guia do Servidor Conectiva Linux*.

[Cou00] Severino Collier Coutinho. *Números Inteiros e Criptografia RSA*. IMPA, 2000.

[dC02] Paulo Sérgio Rezende de Carvalho. Desenvolvimento de um cliente pam para autenticação criptografada e distribuída de estações linux. Monografia de graduação, Universidade Federal de Lavras, 2002.

[ENH01] Scott Seebass Evi Nemeth, Garth Snyder and Trent R. Hein. *UNIX System Administration Handbook*. Prentice Hall PTR, third edition, 2001.

[Fer95] Weber. Raul Fernando. Criptografia contemporânea. *http://www.módulo.com.br*, 1995.

[Fou01] Free Software Foundation. Gnu general public license, 2001. http://www.gnu.org/copyleft/gpl.html.

[How98] L. Howard. Network information service, 1998. http://www.cis.ohio-state.edu/cgi-bin/rfc/rfc2307.html.

[MCCI99] Steven W. Criffith Mark C. Chan and Anthony F. Iasi. *Java 1001 Dicas de Programação*, chapter Programação em Rede. Makron Books, 1999.

[Mor02a] Andrew G. Morgan. The application developers' manual, Junho 2002. Versão 0.76.

[Mor02b] Andrew G. Morgan. The linux-pam system administrators' guide, Junho 2002. Versão 0.76.

[Mor02c] Andrew G. Morgan. The module writers' manual, Junho 2002. Versão 0.76.

[Net96] Netscape Comunications, `http://wp.netscape.com/eng/ssl3/draft302.txt`. *Transport Layer Security Working Group*, November 1996. The SSL protocol Version 3.0.

[Net98] Network Associates, Inc. *An Introduction to Cryptography*, pgp*, version 6 edition, 1998.

[Oli02] Mário Luiz Rodrigues Oliveira. Uma análise da segurança e da eficiência do algorítmo de criptografia posicional. Monografia de graduação, Universidade Federal de Lavras, 2002.

[Ope02] OpenSSL, http://www.openssl.org/. *Welcome to the OpenSSL Project*, July 2002. The SSL protocol Version 3.0.

[Rib00] Nuno Valero Ribeiro. Computação em redes de computadores, 2000. Escola Superior de Tecnologia.

[Sch96] Bruce Schneier. *Applied Cryptography*. John Wisley, Inc., 1996.

[SM99] Richard Stones and Neil Mattew. *Beginning Linux Programming*, chapter Sockets. Wrox Press, 1999.

[Sme95] Jane Smeloff. Security technology unites disparate authentication mechanisms into common infrastructure. *http://wwws.sun.com/software/solaris/pam/osf-pr.html*, 1995.

[Sun01] SunSoft. *Linux-PAM*, May 2001. http://www.kernel.org/pub/linux/libs/pam/.

[YHK95] W. Yeong, T. Howes, and S. Kille. Lightweight directory access protocol, 1995. http://www.ietf.org/rfc/rfc1777.txt.