Serviços FTP, TELNET e SSH **fevereiro/2017**

Instalação e configuração dos serviços FTP, TELNET e SSH. Noções de rdesktop e VNC.

1 – Serviços em redes

Uma rede de computadores pode ser definida como um conjunto de módulos processadores[1] interligados por um meio físico[2], e que fazem uso de um protocolo comum de comunicação. Por protocolo de comunicação entendemos um conjunto de regras de comunicação comum às partes envolvidas nessa troca de dados.

Na nossa proposta de trabalhar em ambientes livres, usamos apenas protocolos de comunicação abertos. Daí a escolha pela suíte[3] Internet Protocol [TCP/IP].

Atualmente, na era da internet comercial, a suíte TCP/IP é o protocolo de comunicação *de fato*. De fato significa que não é o único protocolo de comunicação em rede existente, porém é o mais comumente praticado em qualquer realidade de rede de computadores.

O protocolo IP faz uso de endereços IPs, e ainda hoje a versão mais usada é o ipv4. O ipv6 já existe desde 1998, porém ainda é pouco usado.

Na versão 4, o endereço IP tem 4 bytes [32 bits], que possibilita apenas [teoricamente] 2^{32} endereços [4294967296]. Devido a isso, está previsto para os próximos anos a exaustão dos endereços na versão 4, pela expansão da base de usuários na internet. Consequentemente, a partir daí não será mais possível postergar a adoção em massa da versão 6, que permite teoricamente 2^{128} endereços !!

Host significa hospedador, ou máquina servidora que hospeda serviço em rede. Para acessar determinado serviço usa-se uma aplicação cliente, daí o conceito de cliente-servidor. Do ponto de vista do sistema operacional, tanto servidor quanto cliente, a aplicação servidora ou cliente não passa de uma aplicação comum, com a diferença de prover também acesso em rede. Isso implica em processos do lado cliente e processos do lado servidor, tendo a rede como veículo a transportar informações de um para o outro.

O endereçamento IP é hireárquico: para o cliente alcançar determinado serviço, precisa primeiro acessar a rede do host, chegando a essa rede alcança o host e, nesse host, busca pela porta do serviço.

Exemplo:

192.168.1.10:80

1 Por exemplo: computadores, roteadores, switches, hubs.
2 Cabo de rede, cabo coaxial, cabo ótico, wireless.
3 Suíte nos sentido de conjunto de protocolos divididos em camadas funcionais, num conceito de arquitetura de rede.

onde 192.168.1.0/24 é a rede [nesse caso, classe C] do host, 10 é o endereço desse host nessa rede e 80 é a porta do serviço hospedado por esse host [80 é a porta padrão (default) do serviço web, que usa protocolo de camada de aplicação HTTP].

Desse modo é possível, por exemplo, hospedar dois servidores web no mesmo host [mesmo endereço IP], desde que não usem a mesma porta TCP.

Pelo seu lado, para acessar o servidor, a aplicação cliente precisa apenas saber o endereço desse host, já que usa o mesmo protocolo de camada de aplicação do serviço e conhece a porta padrão [default] desse serviço. Se o serviço estiver em outra porta que não a default, essa informação precisa ser passada ao cliente ou ele não conseguirá acesso.

Um vez iniciado determinado serviço num host, ele abre a porta desse serviço e a mantém assim enquanto estiver rodando. Desse modo, aguarda pelo acesso dos clientes. Já o cliente, no acesso a esse serviço, abre uma porta de cliente para receber os dados enviados pelo servidor.

As portas de serviço normalmente estão entre 0 e 1023. Isso não implica em dizer que não possa haver serviço em portas fora dessa faixa. Por exemplo, o serviço proxy squid usa a porta padrão 3128.

Já as portas de cliente sempre estão acima de 1023.

Alguns exemplos de portas padrão associadas a protocolos de camada de aplicação são:

21:	FTP	[File Transfer Protocol]
22:	SSH	[Secure Shell]
23:	TELNET	[Telecommunications Network Protocol. Acesso remoto shell]
25 (587):	SMTP	[Simple Mail Transfer Protocol]
53:	DNS	[Domain Name System]
80:	HTTP	[Hyper Text Transfer Protocol]
110 (995):	POP	[Post Office Protocol]
143 (993):	IMAP	[Internet Message Access Protocol]
443:	HTTPS	[Hyper Text Transfer Protocol Secure]

O processo de serviço rodando num sistema operacional membro da família Unix é chamado de **daemon**, de **d**isk **a**nd **e**xecution **mon**itor. É devido a isso que, de um modo geral, os processos de serviço tem nomes que terminam com a letra **d**. Por exemplo, inetd, httpd, sshd, named, httpd.

Estes daemons podem ser de duas naturezas, **inetd** ou **standalone**. No Linux, é comum usar **xinetd** ao invés de inetd.

A diferença entre **inetd** e **xinetd** ocorre nos arquivos de conguração: enquanto inetd é configurado pelo arquivo **/etc/inetd.conf**, xinetd tem arquivos específicos de configuração no diretório **/etc/xinetd.d**.

Tipicamente, do ponto de vista do sistema operacional, um serviço em rede é constituído de pelo menos três partes:

- o executável [que ao rodar dá origem ao processo daemon];
- o arquivo de configuração;
- o script de inicialização do serviço.

Por exemplo, no servidor SSH de um CentOS [Red Hat] temos o executável "**/usr/sbin/sshd**", o arquivo de configuração "**/etc/ssh/sshd_config**" e o script de inicialização "**/etc/init.d/sshd**".

2 – Servidor inetd

O servidor **inetd** também é chamado de super servidor, pois ele gerencia conexões para diversos daemons.

Por exemplo, se configurarmos os serviços FTP e TELNET sobre o inetd, teremos apenas o daemon inetd rodando e escutando nas portas 21 e 23. Quando um cliente requisitar um destes serviços, o servidor inetd dispara o daemon específico [neste caso, ftpd ou telnetd] que gera um processo de serviço e atende ao cliente. Com isso, existe uma economia de processos daemons rodando no host.

Por outro lado, no caso de um serviço muito requisitado, não parece boa idéia disponibilizá-lo como inetd, pois isto acabaria produzindo muita carga num único daemon, que resultaria em maior tempo de espera para o cliente.

Além disso, existe o aspecto disponibilidade do serviço. Por exemplo, se determinados serviços forem configurados como inetd e este daemon for comprometido [digamos, por ter sofrido algum ataque], todos estes serviços se tornarão indisponíveis pois o daemon inetd não estará mais respondendo corretamente às requisições dos clientes.

O principal argumento em favor do serviço rodar sobre o inetd é a redução da carga no sistema de forma geral, quando comparado a se executar cada daemon individualmente. Executar o daemon individualmente é o caso de servidor standalone.

3 – Servidor standalone

O servidor é standalone quando disponibiliza um serviço inependente do inetd. Em outras palavras, o serviço é standalone quando mantém seu próprio daemon rodando.

Por exemplo, o serviço SSH standalone mantém rodando o tempo todo o daemon sshd.

Em comparação com inetd, disponibilizar alguns serviços standalone no mesmo host representa maior carga no sistema, pois os processos daemons estarão lá consumindo recursos independente de haver cliente acessando ou não. Por outro lado, se existe uma carga média grande de acessos ao serviço, justifica usar daemon standalone em lugar de inetd. Aliás, essa é a tendência

atual: cada vez mais serviços standalone e menos inetd.

Em princípio, qualquer serviço pode ser disponibilizado standalone ou inetd, basta fazer a escolha e configurar.

O conceito de daemon standalone em contraposição a inetd não tem a ver com servidor standalone do ponto de vista do domínio Windows.

4 – Serviço FTP: File Transfer Protocol

FTP é um protocolo padrão para transferência de arquivos, desenvolvido numa arquitetura cliente-servidor.

É um dos serviços mais antigos da internet, porém continua popular ainda hoje. Inicialmente, haviam apenas as interfaces clientes de linha de comando, mas atualmente existe também uma diversidade de interfaces clientes gráficas, que justifica a popularidade desse serviço.

5 – Serviço TELNET: acesso shell remoto

O serviço telnet oferece acesso em linha de comando [shell] a um cliente remoto. O protocolo TELNET foi desenvolvido em 1969.

NOTA:

À época de criação do TELNET e FTP, a segurança no acesso em rede era um aspecto secundário, e por isso tanto TELNET quanto FTP não implementam criptografia na transmissão de dados. Por isso, atualmente esses serviços deveriam ser substituídos por SSH.

6 – Instalação do servidor xinetd com serviço TELNET

Para descobrir se o servidor xinetd está instalado, uma dica simples é procurar pelo seu script de inicialização em **/etc/init.d**:

```
shell# ls  /etc/init.d  |  grep xinetd
/etc/init.d/xinetd
```

Se não houver saída no comando acima é indicativo de que o servidor x**inetd** não está

instalado. Nesse caso, instalar com o comando **yum**:

```
shell# yum  install  xinetd
```

Num Ubuntu, o comando seria "**apt-get install xinetd**".

Após instalado, verificar se existem os seguintes arquivos:

```
shell# file  /etc/init.d/xinetd
/etc/init.d/xinetd: Bourne shell script text executable
shell# file  /usr/sbin/xinetd
/usr/sbin/xinetd: ELF 32-bit LSB shared object, Intel 80386, version 1 (SYSV), dynamically linked (uses
shared libs), for GNU/Linux 2.6.18, stripped
shell# file  /etc/xinetd.d
/etc/xinetd.d: directory
```

onde:

- **/etc/init.d/xinetd** é o script de inicialização do servidor xinetd;
- **/usr/sbin/xinetd** é o executável que ao rodar dá origem ao processo daemon;
- **/etc/xinetd.d** é o diretório onde estão os arquivos de configuração dos serviços xinetd.

Verificar se existe o processo daemon rodando:

```
shell# ps  -ef  |  grep  xinetd
root     3234     1  0 15:05 ?        00:00:00 /usr/sbin/xinetd
```

portanto, se houver saída no comando indica que o processo está rodando e deve ser parado esse serviço:

```
shell# /etc/init.d/xinetd  stop
Parando o xinetd:                                  [  OK  ]
```

É importante lembrar que o servidor xinetd apenas disponibiliza serviços a partir do daemon xinetd, então é necessário também verificar se o serviço telnet está instalado para rodar sobre o xinetd.

Para essa verificação, comandar

```
shell# ls  /etc/xinetd.d/telnet
/etc/xinetd.d/telnet
```

Se não houvesse saída no comando acima, seria necessário instalar o serviço telnet para rodar sobre o daemon xinetd:

```
shell# yum  install  telnet-server
```

Num Ubuntu, o comando seria "**apt-get install telnetd**".

NOTA:
O fato de instalar o serviço TELNET sobre o xinetd não significa que esse serviço não possa ser disponibilizado de outro modo, por exemplo inetd ou standalone. Porém, na maior parte dos casos de instalação em Linux, TELNET será um serviço disponibilizado sobre o daemon xinetd.

Agora, devem estar instalados os executáveis **/usr/sbin/in.telnetd** [daemon do serviço TELNET] e **/usr/sbin/tcpd** [TCP-Wrapper]. Isso pode ser verificado com o comando:

```
shell# ls  /usr/sbin/in.telnetd   /usr/sbin/tcpd
/usr/sbin/in.telnetd   /usr/sbin/tcpd
```

Porém, no CentOS a instalação do serviço telnet não instala o arquivo de configuração desse serviço em **/etc/xinetd.d/telnet**. Desse modo, precisa ser criado ou copiado de algum lugar. O jeito mais fácil é baixar de www.jairo.pro.br com o comando **wget**:

```
shell# cd  /etc/xinetd.d
shell# wget  www.jairo.pro.br/telnet
```

Caso haja um proxy no meio do caminho, que é o caso de acesso à internet a partir dos laboratórios acadêmicos da Uninove, antes de comandar **wget** precisa passar a instrução de usuário e senha para a variável **http_proxy**, e isso é feito com o comando **export**:

```
shell# export  http_proxy=http://RA:SENHA@186.251.39.92:3128
```

Onde:

RA:	é o RA do aluno;
SENHA:	é a senha de acesso do aluno;
186.251.39.92:	é o IP do serviço proxy, que atende na porta **3128** (é um Squid).

No arquivo de configuração **/etc/xinetd.d/telnet** baixado de www.jairo.pro.br deverá ter o seguinte conteúdo:

========== arquivo **/etc/xinetd.d/telnet** ==========

```
# default: on
# description: The telnet server serves telnet sessions; it uses \
# unencrypted username/password pairs for authentication.
service telnet
{
        disable         = yes
        flags           = REUSE
        socket_type     = stream
        wait            = no
        nice            = 10
        user            = root
        server          = /usr/sbin/in.telnetd
        log_on_failure  += USERID
}
```

==========

Nessa configuração, precisa alterar "*disable = yes* " para "***disable = no***".

Outra maneira de obter o arquivo de configuração **/etc/xinetd.d/telnet** é através do comando **man xinetd.conf**:

```
shell# man  xinetd.conf
```

Neste manual, existe a descrição da configuração do serviço **xinetd**, que inclui exemplos. Basta então procurar pelo serviço telnet e recortar 9 linhas abaixo do padrão encontrado:

```
shell# man  xinetd.conf  |  grep  -A  9   "service telnet"
```

Feita esta verificação, basta redirecionar a saída para o arquivo **telnet**:

```
shell# man  xinetd.conf  |  grep  -A  9   "service telnet"  >  /etc/xinetd.d/telnet
```

E com isso está instalado e configurado o serviço TELNET para rodar sobre o servidor xinetd. Mas antes de iniciar o servidor xinetd, verificar quais portas TCP estão abertas. Para isso, é necessário a aplicação **nmap** para fazer um scan de portas. Se esta aplicação não estiver instalada, é necessário instalar:

```
shell# yum  install  nmap
```

Num Ubuntu, o comando acima seria "**apt-get install nmap**".

Agora, é só fazer o scan de portas:

```
shell# nmap  localhost

Starting Nmap 4.76 ( http://nmap.org ) at 2009-09-07 15:31 BRT
Interesting ports on localhost (127.0.0.1):
Not shown: 999 closed ports
PORT    STATE SERVICE
631/tcp open  ipp

Nmap done: 1 IP address (1 host up) scanned in 0.11 seconds
```

que mostra apenas a porta 631 [servidor de impressão] aberta.

E também, antes de iniciar o servidor xinetd, verificar se existe o processo daemon xinetd rodando:

```
shell# ps  -ef  |  grep  xinetd
```

que não deve ter saída, indicando que esse daemon não está rodando.

Agora, então, iniciar o servidor xinetd:

```
shell# /etc/init.d/xinetd  start
Iniciando o xinetd:                        [  OK  ]
```

Porém, o comando acima só mostra que o serviço **xinetd** iniciou, mas como saber se foi habilitado telnet? Para saber o número de serviços habilitados pelo xinetd, consultar os logs do sistema:

```
shell# tail  /var/log/messages
...
xinetd: 1 available service
...
```

que indica 1 serviço disponível debaixo do xinetd.

Num Ubuntu, usar o comando "**tail /var/log/syslog**".

Depois disso, o scan de portas vai mostrar que a porta 23 também está aberta:

```
shell# nmap  localhost

Starting Nmap 4.76 ( http://nmap.org ) at 2009-09-07 15:38 BRT
Interesting ports on localhost (127.0.0.1):
Not shown: 998 closed ports
PORT    STATE SERVICE
23/tcp  open  telnet
631/tcp open  ipp

Nmap done: 1 IP address (1 host up) scanned in 0.08 seconds
```

E o comando **ps** vai mostrar que o daemon **xinetd** está rodando:

```
shell# ps  -ef  |  grep  xinetd
root     3314     1  0 15:41 ?        00:00:00 /usr/sbin/xinetd
```

7 – Testar o acesso com a aplicação cliente telnet

Sabendo que foi disponibilizado serviço TELNET na máquina remota com endereço IP, por exemplo, 192.168.1.10, usar a aplicação cliente telnet para fazer o acesso a esse serviço:

```
shell$ telnet  192.168.1.10
Trying 192.168.1.10...
Connected to 192.168.1.10.
Escape character is '^]'.
Ubuntu 9.04
Lab10 login: aluno
Password:
Last login: Sun Sep 13 11:36:57 from 192.168.1.13
[aluno@192.168.1.10]$ hostname
```

Uma vez conectado no acesso telnet, pode disparar comandos no servidor remoto.

O acesso telnet é usado para administrar hosts remotos e equipamentos de rede como roteadores, switches e firewalls.

8 - Instalação do serviço FTP standalone

Será usado o serviço FTP **v**ery **s**ecure **f**ile **t**ransfer **p**rotocol **d**aemon, **vsftpd**.

Verificar se o serviço **vsftpd** está instalado:

```
shell# ls /etc/init.d | grep vsftpd
/etc/init.d/vsftpd
```

Se não houver saída no comando acima é indicativo de que o servidor **vsftpd** não está instalado. Nesse caso, instalar com o comando **yum**:

```
shell# yum install vsftpd
```

Num Ubuntu, o comando seria "**apt-get install vsftpd**".

Depois de instalado o serviço, deverá haver o arquivo **/etc/init.d/vsftpd**. Verificar as configurações desse arquivo com o comando **more**:

```
shell# more /etc/init.d/vsftpd
```

Também deverá estar instalado o executável **/usr/sbin/vsftpd** [daemon do serviço FTP]. Isso pode ser verificado com o comando **ls**:

```
shell# ls /usr/sbin/vsftpd
/usr/sbin/vsftpd
```

E no arquivo de configuração **/etc/vsftpd/vsftpd.conf** deverá haver o seguinte conteúdo:

========== arquivo **/etc/vsftpd/vsftpd.conf** =========
local_enable=YES
write_enable=YES
chroot_local_user=NO
tcp_wrappers=YES
==========

NOTA:
Num Ubuntu, esse arquivo está em **/etc/vsftpd.conf**. Além disso, precisa incluir na última linha a configuração "**tcp_wrappers=YES**".

Para visualizar esse conteúdo no arquivo de configuração, basta comandar:

```
shell# grep -v "^#" /etc/vsftpd/vsftpd.conf
```

9 – Testar o acesso com as aplicações cliente ftp

Sabendo que foi disponibilizado serviço FTP na máquina com endereço IP, por exemplo, 192.168.1.10, usar a aplicação cliente ftp para fazer o acesso a esse serviço:

```
shell$ ftp  192.168.1.10
Connected to 192.168.1.10.
220 Lab10 FTP server (Version 6.4/OpenBSD/Linux-ftpd-0.17) ready.
Name (Lab10:aluno): aluno
331 Password required for aluno
Password:
230 User aluno logged in.
Remote system type is UNIX.
Using binary mode to transfer files.
ftp>
```

obtendo o acesso, ganha o prompt do shell cliente ftp.

Nesse prompt, comandar "**?**" para visualizar a lista de comandos possíveis no acesso ftp:

```
ftp> ?
Commands may be abbreviated.  Commands are:

!               debug           mdir            sendport        site
$               dir             mget            put             size
account         disconnect      mkdir           pwd             status
append          exit            mls             quit            struct
ascii           form            mode            quote           system
bell            get             modtime         recv            sunique
binary          glob            mput            reget           tenex
bye             hash            newer           rstatus         tick
case            help            nmap            rhelp           trace
cd              idle            nlist           rename          type
cdup            image           ntrans          reset           user
chmod           lcd             open            restart         umask
close           ls              prompt          rmdir           verbose
cr              macdef          passive         runique         ?
delete          mdelete         proxy           send
```

Para enviar um arquivo, usar "**put**", para enviar vários arquivos de uma única vez, usar "**mput**". Para baixar um arquivo, usar "**get**", para baixar vários arquivos de uma única vez usar "**mget**". O comando "**ascii**" configura a transferência para arquivo de texto [e não binário].

Por exemplo, o usuário aluno logado no servidor FTP em 192.168.1.10 irá baixar o arquivo

/etc/passwd para o diretório **/tmp** da sua estação de trabalho Linux:

```
ftp> pwd
257 "/home/aluno" is the current directory
ftp> cd /etc
250 CWD command successful
ftp> lcd /tmp
Local directory now /tmp
ftp > ascii
200 Type set to A
ftp> get passwd
local: passwd remote: passwd
ftp> bye
221 Goodbye.
shell$
```

Se fosse o caso de enviar um arquivo do cliente para o servidor, o comando seria "**put**" e não "**get**". Com "**mget**" ou "**mput**" podem ser usados curingas "*****", e assim transferir vários arquivos de uma única vez. Mas antes de usar "**mget**" ou "**mput**", é usual comandar "**prompt off**" para não precisar confirmar o download ou upload de arquivo por arquivo.

10 - TCP-Wrappers

TCP-Wrappers é um controle no acesso pela rede, usado como filtro para impedir clientes de determinados endereços ou redes de acessarem serviços disponibilizados por algum host.

Ter controle no acesso a determinado serviço é muito importante para a segurança, especialmente quando estamos disponibilizando serviços típicos da internet, onde os acessos podem ocorrer de qualquer lugar do mundo.

As configurações do TCP-Wrapper são locais ao host, nos arquivos "**/etc/hosts.allow**" e "**/etc/hosts.deny**", mas dependem da biblioteca ***libwrap***. Ou seja, apenas os serviços compilados contra essa biblioteca é que podem ser configurados para ter esse controle no acesso. Por questões históricas, o servidor **xinetd** suporta TCP-Wrappers, bem como uma parte das implementações de serviços SSH. O serviço FTP vsftpd também suporta TCP-Wrappers.

Tipicamente, as configurações TCP-Wrapper envolvem bloquear o acesso para todos no arquivo **/etc/hosts.deny**, e posteriormente liberar o acesso no arquivo **/etc/hosts.allow** para alguns endereços IPs ou redes.

Desse modo, as configurações abaixo:

===== arquivo **/etc/hosts.deny** =====
ALL: ALL
==========

===== arquivo **/etc/hosts.allow** =====
in.telnetd: 10.1.2.3, 192.168.1.0/255.255.255.0
vsftpd: ALL
==========

liberam o acesso aos serviços TELNET e FTP. No caso do TELNET, apenas para clientes com IPs da rede 192.168.1.0/24 e também para o IP 10.1.2.3. No caso do serviço FTP, essas configurações liberam o acesso para qualquer IP.

Convém notar que a configuração em **/etc/hosts.deny** está impedindo o acesso a todo serviço [ALL] para todo IP ou rede [ALL], daí a notação **ALL:ALL**. Desse modo, no **/etc/hosts.allow** precisa declarar alguma liberação de acesso para o serviço que use TCP-Wrapper.

11 – Instalação e configuração do serviço SSH standalone

SSH vem de **S**ecure **SH**ell, ou shell seguro, e foi criado para resolver os problemas de segurança dos protocolos antigos TELNET e FTP que, embora funcionais, não implementam criptografia de dados.

Desse modo, quando se faz um acesso a um serviço FTP ou TELNET, tanto usuário e senha quanto dados enviados ou recebidos são transportados pela rede expostos como texto. E qualquer um que intercepte essa informação obtém diretamente a senha de acesso ao serviço, além dos dados que estão trafegando pela rede.

Existe pelo menos quatro vantagens em usar SSH ao invés de TELNET e FTP:

- 1) o SSH usa uma única porta, que é a 22, tanto para aceso shell remoto quanto transferir arquivos;
- 2) o SSH implementa criptografia de dados;
- 3) o SSH tem pelo menos três aplicações clientes: *ssh*, *sftp* e *scp*;
- 4) com SSH é possível criar relação de confiança com par de chave pública e privada.

O serviço SSH usa criptografia assimétrica, com chaves pública e privada. Nesse acesso [por exemplo], os dados são criptografados no servidor usando a chave pública do cliente [previamente recebida pelo servidor], e então decriptografados no cliente com a chave privada do cliente. Portanto, isso não impede que os dados sejam interceptados, porém só quem conseguirá extrair a informação é o detentor da chave privada.

A implementação mais comum de serviço SSH é o OpenSSH.

Para descobrir se o servidor **sshd** está instalado, uma dica simples é procurar pelo seu script de inicialização em **/etc/init.d**:

```
shell# ls  /etc/init.d  |  grep sshd
/etc/init.d/sshd
```

NOTA:
Num Ubuntu, o arquivo é "**/etc/init.d/ssh**".

Se não houvesse saída no comando acima, seria indicativo de que o servidor **sshd** não está instalado. Nesse caso, poderia ser instalado com o comando yum:

```
shell# yum  install  openssh-server
```

Num Ubuntu, o comando seria "**apt-get install openssh-server**".

O serviço SSH foi instalado como standalone [que é o padrão] e não inetd. Após instalado, verificar se existem os seguintes arquivos:

```
shell# file  /etc/init.d/sshd
/etc/init.d/sshd: Bourne shell script text executable
shell# file  /usr/sbin/sshd
/usr/sbin/sshd: ELF 32-bit LSB shared object, Intel 80386, version 1 (SYSV), dynamically linked (uses
shared libs), for GNU/Linux 2.6.18, stripped
shell# file  /etc/ssh/sshd_config
/etc/ssh/sshd_config: ASCII English text
```

onde:

- **/etc/init.d/sshd** é o script de inicialização do servidor sshd (Ubuntu: **/etc/init.d/ssh**);
- **/usr/sbin/sshd** é o executável que ao rodar dá origem ao processo daemon;
- **/etc/ssh/sshd_config** é o arquivo de configuração do servidor sshd.

Verificar se existe o processo daemon rodando:

```
shell# ps  -ef  |  grep  sshd
root     3234     1  0 15:05 ?        00:00:00 /usr/sbin/sshd
```

portanto, se houver saída no comando indica que o processo está rodando e deve ser parado:

```
shell# /etc/init.d/sshd  stop
Parando o sshd:                            [  OK  ]
```

Nessa situação, o scan de portas **nmap** não deve mostrar a porta 22 aberta.

Agora, então, iniciar o servidor sshd:

```
shell# /etc/init.d/sshd  start
Iniciando o sshd:                          [  OK  ]
```

Depois disso, o scan de portas vai mostrar que a porta 22 está aberta:

```
shell# nmap  localhost

Starting Nmap 4.76 ( http://nmap.org ) at 2009-09-07 17:38 BRT
Interesting ports on localhost (127.0.0.1):
Not shown: 998 closed ports
PORT    STATE SERVICE
22/tcp  open  ssh
631/tcp open  ipp

Nmap done: 1 IP address (1 host up) scanned in 0.08 seconds
```

E o comando **ps** vai mostrar que o daemon **sshd** está rodando:

```
shell# ps  -ef  |  grep  sshd
root      1327     1  0 19:22 ?        00:00:00 /usr/sbin/sshd
```

12 – Testar o acesso com as aplicações clientes ssh, scp e sftp. Criar relação de confiança ssh.

IMPORTANTE:
Se estiver usando TCP-WRAPPERS, para acessar via ssh precisa antes incluir a seguinte linha no /etc/hosts.allow:

=== /etc/hosts.allow ==========
sshd: 192.168.1.0/255.255.255.0
==========
para liberar o acesso a todos os clientes na rede 192.168.1.0/24.

A aplicação cliente **ssh** é semelhante a **telnet**, com a segurança da criptografia.

Sabendo que foi disponibilizado serviço SSH na máquina remota com endereço IP 192.168.1.10, usar a aplicação cliente **ssh** para fazer o acesso a este serviço:

```
shell$ ssh  aluno@192.168.1.10
The authenticity of host 'localhost (192.168.1.10)' can't be established.
RSA key fingerprint is 66:39:9d:7a:8f:4c:af:1b:40:c4:a7:35:42:15:3b:d6.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added 'localhost' (RSA) to the list of known hosts.
aluno@192.168.1.10's password:
Last login: Sun Sep 13 11:37:30 2009 from 192.168.1.13
[aluno@192.168.1.10]$
```

Uma vez conectado no acesso ssh, pode disparar comandos no servidor remoto.

O acesso ssh também é usado para administrar hosts remotos e equipamentos de rede como roteadores, switches e firewalls.

Para desconectar do serviço ssh, comandar **exit**:

```
shell$ ssh  aluno@192.168.1.10
[aluno@192.168.1.10]$ exit
shell$
```

Uma alternativa à notação "**ssh aluno@192.168.1.10**" é usar:

```
shell$ ssh  -l  aluno  192.168.1.10
```

onde a letra *l* vem de login_name.

Uma outra maneira de usar a aplicação cliente **ssh** é disparar diretamente comandos remotos, sem necessidade de fazer o acesso remoto shell. Por exemplo:

```
shell$ ssh  aluno@192.168.1.10  'hostname'
aluno@192.168.0.10's password:
lab-10
shell$ hostname
lab-11
```

onde a máquina local com IP 192.168.1.11 tem hostname lab-11, e o servidor remoto é lab-10.

A aplicação cliente **scp** [**s**ecure **co**p**y**] serve para transferir arquivos usando o protocolo SSH.

Por exemplo, para transferir o arquivo **/etc/passwd** do host remoto 192.168.1.10 para o diretório **/tmp** local [download], comandar:

```
shell$ scp  aluno@192.168.1.10:/etc/passwd  /tmp
aluno@192.168.1.10's password:
passwd                                        100% 2023     2.0KB/s   00:00
shell$
```

Para enviar o arquivo local **/etc/group** para o diretório **/var/tmp** do servidor remoto [upload], comandar:

```
shell$ scp  /etc/group  aluno@192.168.1.10:/var/tmp
aluno@192.168.1.10's password:
group                                         100%  875     0.9KB/s   00:00
shell$
```

Outra vantagem do cliente scp é possibilitar transferir diretórios com todo o conteúdo, usando a opção "**-r**" (recursivo):

```
shell$ scp  -r  /bin  aluno@192.168.1.10:/tmp
```

neste exemplo, todo o diretório local /bin (e seu conteúdo) foi transferido para o /tmp da máquina em 192.168.1.10.

NOTA:
O **ftp** não transfere diretórios.

A aplicação cliente **sftp** tem esse nome de **s**ecure **ftp**, pois transfere arquivos usando criptografia de dados.

A funcionalidade do **sftp** é similar a do cliente **ftp**, incluindo a interatividade, o que significa dizer que todos os comandos ftp funcionam do mesmo jeito no sftp.

Por exemplo, para acessar o host no IP 192.168.1.10, comandar:

```
shell$ sftp  aluno@192.168.1.10
aluno@192.168.1.10's password:
sftp> ?
Available commands:
cd path                      Change remote directory to 'path'
lcd path                     Change local directory to 'path'
chgrp grp path               Change group of file 'path' to 'grp'
chmod mode path              Change permissions of file 'path' to 'mode'
chown own path               Change owner of file 'path' to 'own'
df [path]                    Display statistics for current directory or filesystem containing 'path'
help                         Display this help text
get remote-path [local-path] Download file
lls [ls-options [path]]      Display local directory listing
ln oldpath newpath           Symlink remote file
lmkdir path                  Create local directory
lpwd                         Print local working directory
ls [path]                    Display remote directory listing
lumask umask                 Set local umask to 'umask'
mkdir path                   Create remote directory
progress                     Toggle display of progress meter
put local-path [remote-path] Upload file
pwd                          Display remote working directory
exit                         Quit sftp
quit                         Quit sftp
rename oldpath newpath       Rename remote file
rmdir path                   Remove remote directory
rm path                      Delete remote file
symlink oldpath newpath      Symlink remote file
version                      Show SFTP version
!command                     Execute 'command' in local shell
!                            Escape to local shell
?                            Synonym for help
sftp>
```

Mas o **sftp** não fica só na imitação do **ftp**, ele também tem a sua própria sintaxe, que é bastante eficiente para transferir arquivos.

No exemplo abaixo, temos alguns destes comandos próprios do cliente **sftp**:

```
shell$ sftp  aluno@192.168.1.10
aluno@192.168.1.10's password:
sftp> get  /bin/c*  /tmp
sftp> put  /tmp/c*  /var/tmp
sftp> get  -r  /bin  /tmp
sftp> exit
```

onde o comando get é para baixar arquivos e put para enviar. Não é necessário usar mget ou mput.

NOTA:

Algumas versões do cliente **sftp** suportam o modo recursivo "**-r**" para transferir diretórios, enquanto outras não.

Com ssh, é possível (e recomendado) criar relação de confiança entre o cliente e o serviço, desse modo dispensando o uso de senhas. Isso é seguro, recomendado e muito importante no caso de transferência agendada de arquivos, pelo uso de scripts. Neste caso, não existe a necessidade de escrever a senha em algum lugar, como ocorre com o cliente ftp.

A relação de confiança é obtida a partir da criação de um par de chaves, com o comando **ssh-keygen**:

```
shell$ ssh-keygen -b 1024 -t rsa
Generating public/private rsa key pair.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/aluno/.ssh/id_rsa.
Your public key has been saved in /home/aluno/.ssh/id_rsa.pub.
The key fingerprint is:
d8:b8:de:5b:2c:2d:1d:ed:c8:ea:a1:10:a0:63:c4:63 aluno@lab-11
The key's randomart image is:
+--[RSA 1024]---+
|                 |
|.                |
| E.              |
|o...   +   .     |
|o.  . o S . .    |
|..   . . = +     |
|    . . + B .    |
|     o o *       |
|      o.=.       |
+-----------------+
```

onde a opção "**-b**" especifica o número de bits na chave e "**-t**" o tipo da chave, no caso algoritmo **rsa**. Não foi usado passphrase para esta chave, pois a ideia é usar num script, portanto não interativo.

NOTA:

Como não foi especificado o local onde seria criado o par de chaves, estas foram criadas no subdiretório **.ssh** na home do usuário, na máquina lab-11 (192.168.1.11).

Para ver as chaves, comandar:

```
shell$ ls -l /home/aluno/.ssh/id_rsa*
-rw------- 1 aluno aluno 891 Fev 11 17:20 /home/aluno/.ssh/id_rsa
-rw-r--r-- 1 aluno aluno 227 Fev 11 17:20 /home/aluno/.ssh/id_rsa.pub
```

O passo seguinte é enviar a chave pública **id_rsa.pub** para o servidor ssh remoto, no caso 192.168.1.10. Lá, esta chave deverá ser renomeada para **authorized_keys**, no caminho absoluto

/home/aluno/.ssh/authorized_keys.

Para este envio ser seguro, deve ser usado o acesso ssh:

```
shell$ cd  /home/aluno/.ssh
shell$ scp  id_rsa.pub  aluno@192.168.1.10:/home/aluno/.ssh/authorized_keys
```

NOTA:
Caso já existisse um arquivo **/home/aluno/.ssh/authorized_keys** em 192.168.1.10, para não perder a(s) chave(s) que houvesse(m) lá, o conteúdo do arquivo id_rsa.pub deveria ser acrescentado no final de authorized_keys.

No servidor ssh, as permissões de authorized_keys devem ser, no mínimo, 600. Para garantir isso, comandar:

```
shell$ ssh  aluno@192.168.1.10  'chmod  600  /home/aluno/.ssh/authorized_keys'
aluno@192.168.0.10's password:
```

Depois disso, já deverá funcionar o acesso sem necessidade de senha:

```
shell$ ssh  aluno@192.168.1.10
[aluno@192.168.1.10]$
[aluno@192.168.1.10]$ exit
shell$
```

Um exemplo bem simples de script para agendar copia de arquivos de 192.168.1.11 (o cliente) para 192.168.1.10 (o servidor SSH remoto), é:

```
=== copy.sh ==================
#!/bin/bash
file=$1
scp  $file  aluno@192.168.1.10:/tmp
==========================
```

Dar permissão de execução para o script:

```
shell$ chmod  755  /home/aluno/copy.sh
```

Depois, disparar este script para copiar para o /tmp do servidor o arquivo passado como primeiro argumento:

```
shell$ /home/aluno/copy.sh  /etc/passwd
passwd                                        100% 2746     2.7KB/s   00:00
shell$
```

Uma configuração importante no servidor SSH é impedir o acesso do usuário root diretamente ao host, pois isso caracteriza um acesso genérico com privilégios. O modo correto [e recomendado] é o administrador logar com o seu usuário e depois comandar **su** ou **sudo** para ganhar privilégios de root, e com isso registrar em logs do sistema quem é a pessoa que está executando estes comandos.

Para impedir o acesso do usuário root no acesso SSH basta que haja a seguinte linha de configuração no arquivo **/etc/ssh/sshd_config**:

=== arquivo **/etc/ssh/sshd_config** ===
PermitRootLogin no
==========

13 – Noçoes de rdesktop e vnc

A aplicação cliente **rdesktop** é usada para acessar sistemas Windows com Terminal Services. Essa aplicação roda praticamente em todos os sistemas da família Unix, e serve para acessar ou administrar sistemas Windows a partir de uma estação de trabalho Linux [por exemplo].

O cliente **rdesktop** exige ambiente gráfico na estação onde for rodar.

O nome **vnc** vem de **v**irtual **n**etwork **c**omputing, que é um sistema de compartilhamento do desktop para controle a partir de outro computador. O **vnc** transmite os eventos de teclado e mouse de um computador para o outro.

Uma grande vantagem do **vnc** é a independência de plataforma: um **vnc** viewer [visualizador] num determinado sistema operacional pode conectar a um **vnc** server em outro sistema operacional.

Com isso, uma máquina com **vnc** server pode ser administrada remotamente por outra, mesmo de sistema operacional diferente, desde que tenha um vnc viewer.

Porém, na prática, a administração de sistemas operacionais da família Unix e equipamentos de rede é feita via linha de comando, de modo que tanto **rdesktop** quanto **vnc** estão mais voltados para administração e acesso a sistemas Windows.