

# **CURSO DE SEGURANÇA EM REDES LINUX**

Autor: Renato Martini  
rmartini@cipsga.org.br

**Março de 2000**

Este documento esta sob a egide da GPL (General Public License). Cópias, mesmo que parciais do texto, estão previamente autorizadas, desde que se mantenha integralmente junto a reprodução ou citação do texto, todas as informações desta página.

## **Curso de Segurança em Redes**

Comite de Incentivo a Produção  
do Software Gratuito e Alternativo  
CIPSGA

Autor:

**Renato Martini**  
rmartini@cipsga.org.br

Março de 2000

## Índice

1	SEGURANÇA: FUNDAMENTOS.....	4
1.1	Introdução .....	4
1.2	Segurança: o conceito .....	4
1.3	Segurança e as ferramentas de rede.....	6
1.4	Nosso objetivo .....	7
2	0 FIREWALL: DUAS SOLUÇÕES EM AMBIENTE LINUX.....	9
2.1	Uma palavra inicial sobre firewalls .....	9
2.2	Firewalls e acesso remoto: o Secure Shell .....	10
2.3	Firewalls: solução Linux.....	15
2.4	A filtragem de pacotes .....	16
2.5	IPCHAINS (The Enhanced IP Firewalling Chains Software for Linux) .....	17
2.6	The SINUS Firewall - a TCP/IP packet filter for Linux .....	32
3	MONITORAÇÃO DA REDE.....	56
3.1	Os scanners de rede.....	56
3.2	Saint: Security Administrator's Integrated Network Tool.....	72
	<b>ANEXOS .....</b>	<b>80</b>
	<u>ANEXO A:</u> .....	81
	<u>ANEXO B:</u> .....	82
	<u>ANEXO C:</u> .....	90
	<u>ANEXO D:</u> .....	96
	<u>ANEXO E:</u> .....	99
	<b><u>BIBLIOGRAFIA GERAL</u>.....</b>	<b>100</b>
	<b><u>SOBRE O AUTOR DA APOSTIL</u> .....</b>	<b>102</b>

# 1 Segurança: Fundamentos

## 1.1 Introdução

---

O objetivo deste documento é mostrar como o Linux oferece uma solução *completa* para o problema da segurança. Observe-se, aqui, que se falamos “solução” não significa a existência de um sistema *absolutamente* seguro. O Linux oferece, isto sim, as ferramentas necessárias para a gestão da segurança num micro isoladamente ou numa intranet, e, então, tal intranet conectada à Grande Rede. Este documento descreve introdutoriamente tais ferramentas e problemas de segurança.

## 1.2 Segurança: o conceito

---

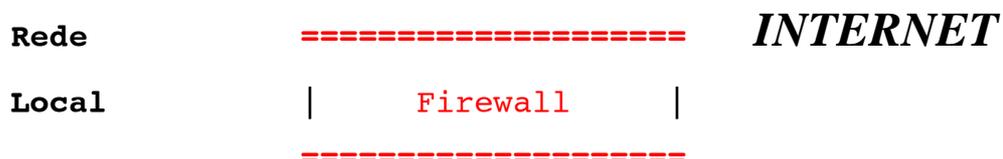
Trataremos segurança como sendo a restrição dos recursos de *um* micro-computador, ou de uma rede, ou de porções desta rede para outros usuários ou computadores. Segurança *nada mais é do que a gestão de tal restrição*, - o que constitui portanto uma *política de segurança*, ou como se diz em inglês: *security policy*. O que significa tão-somente que numa rede há determinados recursos (arquivos, dispositivos de hardware, etc.) que estão disponíveis para este computador ou *tipo* de usuário, mas que por outro lado ficam restritos a tal ou qual computador, esteja ele fora ou dentro da rede.

Mesmo um sistema como o antigo DOS implementava um *tipo* de segurança, claro que singela frente aos SO multi-usuários, como o Linux ou qualquer tipo de sistema UNIX. No DOS há determinados arquivos *ocultos*, há arquivos com atributos para que se permita que este somente seja lido, etc. Por conseguinte, há restrição de

“áreas” do SO. Claro, que tudo isso num sistema DOS, ou mesmo na sua sequência histórica, o Windows (9x), possuem restrições que facilmente burlamos. Mas o DOS-Windows não foram feitos para rede. O recurso do *networking* não é embutido (*built-in*), intrínseco ao Windows, por isso mesmo todos os embaraços que seus usuários possuem ao operarem na Internet, como por exemplo, no famoso *Back Orifice*.

O Linux é um sistema multi-usuário, essencialmente um SO capacitado para as Redes, não é um recurso que lhe foi acrescentado *externamente*. E todo SO de rede estabelece privilégios como forma de segurança: há usuários especiais, na nomenclatura correta Super-Usuários ou usuário *Root*, que podem alterar arquivos especiais do sistema, montar partições, sejam locais ou remotas, desligar a rede, etc. O usuário *comum*, sem privilégios, não pode fazê-lo. Porém, quando nos deslocamos da ótica de um computador pensado isoladamente, isto é, uma única estação de trabalho, usado por umas 6 pessoas, digamos, vemos que se coloca sem dúvida o tema da segurança. Mas quando observamos uma rede local, a segurança é um tema ainda mais urgente. Numa intranet, às vezes diversos recursos de vários micros estão interditados de um departamento, por exemplo. E, sobretudo, quando esta intranet conecta-se a Internet, o que é quase inevitável, a segurança torna-se ainda mais urgente e fundamental.

Poderíamos esquematizar assim:



### 1.3 Segurança e as ferramentas de rede

---

Num único micro ou numa rede local (de pequeno, médio ou grande porte) e, assim, quando esta rede está conectada na Grande Rede, a segurança tem que ser gerida, e, logo veremos, o Linux tem todas as ferramentas e a documentação necessárias para isso.

Deve-se lembrar que tais ferramentas são inúteis se o administrador não tiver uma política de segurança. Não constitui objetivo deste documento tratar explicitamente do tema. Leituras iniciais obrigatórias para tanto são: o *RFC*<sup>1</sup> 2196 intitulado de **Site Security Handbook** (<http://www.rfc-editor.org>) escrito por B. Fraser (setembro de 1997); e o *Linux Security HOWTO* (<ftp://sunsite.unc.edu/pub/Linux/docs/HOWTO>) de K. Fenzi e D. Wreski (maio de 1998); e o livro de Paul Sery *Ferramentas Poderosas para Redes em Linux* (ed. Ciência Moderna). Ali o leitor encontrará por certo os ensinamentos iniciais para a construção da *política de segurança de sua rede*, - e que o *RFC* 2196 propriamente define como uma declaração formal de *regras* que concedem acesso a recursos de informação e tecnologia, e que logo devem ser cumpridas. Devemos lembrar ainda que o aluno poderá encontrar uma reunião bastante abrangente de ferramentas, para todos os "sabores" de sistemas UNIX, no site <http://www.securityfocus.com>, visita obrigatória para todo aquele que se ocupa com segurança.

Gostaríamos de resumir aqui alguns pontos do *Linux Security HOWTO* (2.3 "*What are you trying to protect?*" E o 2.4 "*Developing a security policy*"), e eles devem ser levados em consideração, antes mesmo de se criar uma política de segurança para a sua rede. Antes de proteger seu sistema você deve saber de que tipo de ameaça se

está tentando proteger, se atacado o que está em jogo... Vejamos então:

*\*O risco é a possibilidade que um intruso possa ter sucesso ao tentar invadir seus computadores. Um intruso pode, ao acessar seus arquivos, danificar dados críticos? Não se esqueça, também, que ao possuir uma conta de sua rede, o intruso pode se passar por você.*

*\*As ameaças serão sempre no sentido de se obter acesso não-autorizado em sua rede ou computador. Há portanto vários tipos de intrusos e, então, diferentes tipos de ameaça a sua rede.*

*\*Há o curioso: esse tipo de intruso se interessa pelo tipo de dado e sistema você possui.*

*\*Há o malicioso: esse quer em síntese derrubar o seu sistema, destruir dados, destruir os documento publicados no seu Web server, etc. É o chamado cracker.*

*\*Há o intruso de “alto-nível” (High-Profile): ele quer obter popularidade mostrando suas habilidades ao invadir seu sistema.*

*\*Há o competidor: esse quer conhecer seus dados para obter algum ganho com isso.*

*\*Por fim, “vulnerabilidade” descreve o quão bem protegido é seu computador, e o que se perderá se alguém obter acesso não-autorizado a algum(ns) computador(res).*

Portanto, crie uma *política de segurança* para sua rede que seja simples e genérica e que todos os usuários possam prontamente compreender e seguir. Você pode proteger dados tanto quanto respeitar a privacidade dos usuários

---

## 1.4 Nosso objetivo

---

Em suma, nossa intento é mostrar - como já dissemos - as ferramentas necessárias para a construção e gerência de uma política de segurança em ambiente Linux. Todos são softwares abertos e cobertos pela GPL (*General Public License* da GNU), assim como a documentação disponível. Para tanto nosso caminho não começará da segurança local

---

<sup>1</sup>RFC é o acrônimo de *Request for Comments*, um enorme conjunto de documentos organizados pela INTERNIC reunindo

(ou seja, uma rede interna, segundo a nomenclatura do *Site Security Handbook*) em direção à segurança extranet (rede externa). Contrariamente, iremos da segurança externa - passando obviamente pelas questão de segurança na rede intranet, pois o *firewall* também ocupa-se com este ponto -, para a segurança interna.

## 2 0 *Firewall*: duas soluções em ambiente Linux

### 2.1 Uma palavra inicial sobre *firewalls*

---

*Firewall* é uma “parede” que separa sua intranet do *mundo exterior*, isto é, a internet. Mas, também, um *firewall* pode ser usado eficientemente numa intranet que não chega a ter acesso à internet. Pois, pode filtrar *pacotes* que passam de uma máquina a outra numa intranet.

Definindo:

Um *firewall* é todo sistema conjunto de *hardware* e *software* que é elaborado para proteger uma intranet de usuários potencialmente perigosos, visto que não autorizados.

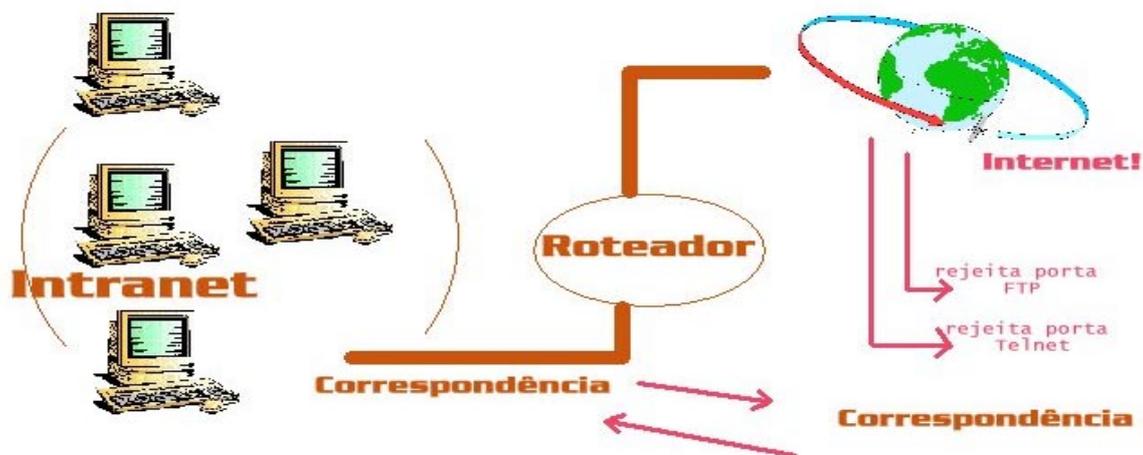
Entretanto, a capacidade efetiva de controle de um *firewall* é implementada ao pôr-se entre a rede local e a internet, de forma a evitar que o mundo tenha acesso a dados particulares. Mas não estamos face a face com algo assim como um “programa” fácil e simples de executar. Pois, como já vimos, um *firewall* tem inquestionavelmente uma natureza abrangente.

Não haveria problemas de segurança, se você como que isolasse seus computadores ou sua rede do mundo. Mas, uma intranet é fundamentalmente “uma rede interna que usa tecnologias World Wide Web para partilhar tarefas e informações entre departamentos e/ou locais remotos”.<sup>2</sup> Não posso instalar um servidor Apache numa máquina e fechar o acesso a seus dados em meu *firewall*, tão pouco posso fechar o acesso a todos os computadores que formam a rede mundial.

---

<sup>2</sup>Definição retirada de *SCO OpenServer*© *Internet Services* (v.5.0.4 may 1997), p. 11.

Este esquema mostra a implementação simples de um *firewall*:



Todavia, deveríamos notar em nosso esquema que nossa rede tem que possuir um *host* (um *bastion host*), que ligado ao roteador nos põe em contato com o mundo. É bastante consensual que tal máquina deve ter seus serviços desligados, o mínimo de portas devem estar ativas. Para tanto o aluno deve seguir as orientações básicas para desligar serviços no seu *inetd*. O Linux da Red Hat possui uma ferramenta de *setup* em modo texto. Pode-se usá-la como uma interface valiosa para desligar serviços como *ftp*, *tftp*, *telnet*, etc. Você pode chamá-la no console a partir do comando **setup**, ou chamar diretamente pelo comando **netsysv**, pois a ferramenta *setup* é apenas uma interface para diversos outros programas de configuração. Igualmente, não se faz necessário a instalação do sistema gráfico Xfree. O *host* responsável pelo *firewall* de nossa rede deve ser uma máquina "enxuta", com a prioridade óbvia de cumprir as funções de *firewall*.

## 2.2 Firewalls e acesso remoto: o *Secure Shell*

Para o acesso remoto ao nosso *firewall* devemos usar o *software SSH Secure Shell* (*ssh*: <http://www.ssh.org>), e não *telnet* ou *rsh*, ou *rlogin*, mesmo porque já tratamos de desligá-los. O *ssh* é um pacote completo para *login* remoto que tem a capacidade de criptografar, utilizando-se da autenticação forte RSA, a comunicação entre redes e *hosts não-confiáveis*. Assim sendo todas as comunicações são

criptografadas. A RSA é usada para um intercâmbio de “chaves” (num esquema de chaves públicas e privadas<sup>3</sup>), e um método de criptografia (IDEA, Blowfish, etc.) é usado também para criptografar a sessão remota a ser aberta. O que é importante no *ssh* é que o software dispara a criptografia antes do processo de autenticação, ou seja, antes da checagem das senhas. Portanto, nenhuma senha é enviada pela rede em aberto, sem criptografia.

A maior parte das distribuições atuais do Linux já vem com a versão 1.2.27 deste programa (a última versão é a 2.0.13, chamada de *ssh2*). Se conseguir o *ssh* através de um pacote RPM basta instalá-lo com o comando “**rpm -ivh**”. Ele instalará os seguintes arquivos:

<b>Arquivos</b>	<b>Descrição</b>
sshd	<b>Daemon que roda na máquina-servidor, espera o pedido do cliente ssh, autentica a conexão e inicia a sessão.</b>
ssh ou slogin	Cliente: programa usado para login e execução de outros comandos,
scp	Usado para copiar arquivos de um computador para outro em segurança
ssh-keygen	Usado para criar chaves RSA
ssh-agent	Agente para a autenticação das chaves
ssh-add	Usado para registrar novas chaves
make-ssh-known-hosts	Script perl usado para criar o arquivo /etc/ssh_known_hosts a ser usado pelo DNS

Realizada a instalação com êxito, vamos geramos então uma chave RSA para o usuário root num host chamado *alpha* usando o comando descrito acima **ssh-keygen**. Encontraríamos, por exemplo, a seguinte saída:

```
[root@alpha /]# ssh-keygen
Initializing random number generator...
Generating p: .....++ (distance 250)
Generating q: .....++ (distance 314)
Computing the keys...
Testing the keys...
Key generation complete.
Enter file in which to save the key (/root/.ssh/identity):
Enter passphrase: (entre com a senha - nada será ecoado)
```

<sup>3</sup>A autenticação RSA é baseada numa *chave pública de criptografia*. Uma chave para criptografar e a outra para descriptografar. A chave pública é usada para criptografar, e a chave para descriptografar por sua vez é privada, mas *jamais poderemos derivar esta chave de descriptografar da outra*.

```
Enter the same passphrase again: (idem)
Your identification has been saved in /root/.ssh/identity.
Your public key is:
1024 27 7979797979793729732739277556658683028389748648364634683648
979479274937493749797397492794729749348793475154551542455251454514
686486348638463826427864863861525415199494879757808883282083082038
12112288201820820181280281080374683658597938408 root@cipsga.org.br
Your public key has been saved in /root/.ssh/identity.pub
```

Depois de gerarmos nossas chaves (a pública e a privada) e estas serem gravadas no diretório de nossa escolha (aqui aceitamos a melhor opção, a default do ssh-keygen) vamos tentar abrir uma sessão segura. Mas antes temos de carregar o programa servidor do ssh, o *sshd*. Executaremos *sshd* manualmente, para tanto o aluno deve digitar: **/etc/rc.d/init.d/sshd**, é com esse script que o Red Hat Linux sabidamente carrega em toda inicialização o *daemon*, sem, portanto, a necessidade de uma execução manual. Depois aluno deve verificar com a ferramenta **netstat** do Red Hat se o *sshd* está marcado para execução automática. Deve se ter atenção para tal detalhe porque se o ssh, o cliente, não encontrar o *sshd* rodando, ele oferece a possibilidade de uma sessão não-segura via *rsh*. Como no exemplo abaixo:

```
[root@beta /root]# ssh alpha
Secure connection to alpha refused; reverting to insecure method.
Using rsh. WARNING: Connection will not be encrypted.
Password: (...)
```

Assim, certos que o *sshd* está ativo no *host* alpha, numa outra máquina que chamamos *beta* carregamos o cliente *ssh* para conectar com o *host* alpha, vejamos:

```
[root@beta /]# ssh alpha.cipsga.org.br
Host key not found from the list of known hosts.
Are you sure you want to continue connecting (yes/no)? Yes
Host 'beta.cipsga.org.br' added to the list of known hosts.
Creating random seed file ~/.ssh/random_seed. This may take a while.
root@alpha.cipsga.org.br's password: [entra com senha nada é ecoado]
Last login from: Sat Jan 29 23:12:00 2000 from alpha
[root@alpha /root]#
```

Vamos entender as mensagens da tela. Quando o servidor *sshd* no *host* alpha recebeu a solicitação ele nos advertiu não ter encontrado a *host key* da lista de *hosts* conhecidos, - ainda assim aceitamos

continuar a conexão, ele põe a máquina beta na lista e continua com a autenticação, depois que ela é executada com êxito recebemos o terminal virtual. Se realizarmos o *logout* e novamente entrar com um pedido de sessão ssh, o daemon lê o arquivo `~/.ssh/randon_seed` e nos permite a sessão segura e a autenticação padrão do sistema. Porém o aluno deve escolher entre este método, o importar sua chave pública (armazenada no arquivo *identity.pub*) para a máquina remota. Isso possibilitará ao programa usar a autenticação baseada em RSA. Como nossa máquina habilitada para firewall não nos possibilita transferência de arquivo (ftp, NFS, etc.), usaremos o velho disquete usando um programa das ferramentas **mttools**, o **mcopy**. Coloque um disquete na unidade na e digite: `"mcopy ~/.ssh/*.pub a:"`. Com isso copiaremos para o diretório *remoto* `~/.ssh` nossa chave pública (atenção: só levamos nossa chave pública! A chave privada **não** deve ser divulgada), mas agora com um novo nome a ser utilizado nas conexões, a saber, **authorized\_keys**. Tal arquivo corresponde ao convencional arquivo **.rhosts**; ele possui uma *chave por linha*, suportando portanto diversas chaves públicas de diferentes usuários e *hosts*. Atenção para não truncar o arquivo, pois as chaves são longas, e devem entrar numa *única* linha. Logo depois disso, o usuário pode executar o login sem a autenticação padrão, mas apenas dando o *passphrase* da chave RSA, que é sobremaneira mais seguro.

Com a criação do **authorized\_keys** quando abrirmos a nova sessão com o cliente ssh, veremos primeiramente então:

```
[root@alpha/]#ssh beta
Host key not found from the list of known hosts.
Are you sure you want to continue connecting (yes/no)? Yes
Host 'beta' added to the list of known hosts.
Enter the passphrase for RSA key 'root@alpha.cipsga.org.br': (nada ecoa)
Last login: Sun Jan 30 21:00:12 on tty2
You have mail
[root@beta /root]#
```

Numa posterior conexão encontraremos o seguinte cenário:

```
[root@alpha /]#ssh beta
Enter the passphrase for RSA key 'root@alpha.cipsga.org.br': (nada ecoa)
Last login: Sun Jan 30 21:05:10 from alpha
You have mail
[root@beta /root]#
```

Assim o aluno entra com a sua *passphrase*, as chaves são checadas e a sessão segura criptografada é posta em ação.

```
SCO OpenServer(TM) Release 5 (sprache.iis.com.br) (tty0)
login: root
Password:
Last successful login for root: Sun Jan 30 17:12:57 2000 on tty0
Last unsuccessful login for root: Thu Jan 27 00:27:02 2000 on tty02

          SCO OpenServer(TM) Release 5
          (C) 1976-1997 The Santa Cruz Operation, Inc.
          (C) 1980-1994 Microsoft Corporation
          All rights reserved.

          For complete copyright credits,
          enter "copyrights" at the command prompt.

you have mail
# ssh 192.168.0.2
Enter passphrase for RSA key 'root@sprache.iis.com.br':
Last login: Sun Jan 30 17:27:56 2000 from leben
You have mail.
[root@machen /root]# _
```

Outro mérito indiscutível do *Secure Shell* é o fato do cliente para acesso remoto já estar portado para diversas plataformas (UNIX, OS/2, Win32, etc.) - o que é bastante importante quando temos redes "híbridas". Visite o site <ftp://ftp.cs.hut.fi/pub/ssh> para uma visão de versões disponíveis. Podemos sugerir ao aluno pesquisar as seguintes versões não-comerciais para Win32 e OS/2 respectivamente:

☛<http://www.chiark.greenend.org.uk/~sgtatham/putty/> (PuTTY é uma versão livre para telnet e SSH).

<http://dome.weeg.uiowa.edu/pub/domestic/sos/ports/ssh-1.2.27-cygwinb20.tar.bz2>  
(Versão cliente e servidor SSH dentro do projeto Cygnus)

☛<ftp://ftp.cs.hut.fi/pub/ssh/os2/sshos203.zip> (Versão SSH para OS/2 3.x)

Nesta tela veremos um cliente compilado e rodando numa máquina UNIX, SCO OpenServer v.5.0.4 (<http://www.sco.com>), e abrindo uma conexão ssh com checagem da chave RSA num *host* Linux:

## 2.3 Firewalls: solução Linux

---

Não discutiremos aqui a *implementação física* de um *firewall*. Nosso objetivo é mostrar as ferramentas disponíveis no Linux. O aluno tem que ter em mente, também, que não existe uma solução única em termos de firewall. Você tem que conhecer as ferramentas, e então adaptá-las às necessidades específicas de sua rede, montar sua política de segurança, e adaptá-la a um firewall. Você terá uma introdução, e depois terá que pesquisar suas demandas específicas. São tantos os cenários<sup>4</sup> possíveis que o tema torna-se inesgotável: uma pequena rede doméstico ou um pequeno escritório que se conecta a Internet via ppp; uma rede com um roteador conectado a Internet; uma rede média com dois *firewalls*, etc., etc. Daremos aqui os conceitos principais das ferramentas, seus comandos, opções e utilização.

Um *firewall* é implementado no Linux em nível de kernel, por isso todo o texto sobre o tema começa com as opções para *recompilar* o kernel. Sem dúvida para termos um *firewall* no Linux necessitamos de sua habilitação no kernel. Mas isso não se faz necessário pois o Red Hat 6.x já vem com o kernel para *firewalling* tanto quanto o SuSE 6.x (por exemplo, o OpenLinux da Caldera v.2.2, por ser mais voltado para um sistema *desktop*, não possui um kernel compilado para *firewall*). Ainda assim, se desejar checar se suas máquinas estão habilitadas para *firewall*, digite: **"ls /proc/net/ip\_fwchains"** - se tal arquivo estiver aí, ok temos *firewall*. Apenas como informação listamos abaixo as opções a serem habilitadas e desabilitadas num kernel recompilado para *firewall*:

**Em 'general setup':**

1. turn networking support => ATIVO

**Em 'networking options':**

1. turn network firewalls => ATIVO

---

<sup>4</sup>O *Linux IPCHAINS-HOWTO* por exemplo apresenta 4 diferentes cenários possíveis.

2. `turn TCP/IP networking => ATIVO`
3. `turn IP forwarding/gatewaying => ATIVO`
4. `turn IP firewalling => ATIVO`
5. `turn IP firewall packet logging => ATIVO`
6. `turn IP masquerading => ATIVO`
7. `turn IP accounting => ATIVO`
8. `turn IP tunneling => DESATIVO`
9. `turn IP aliasing => ATIVO (pode-se optar por seu uso modular...)`
10. `turn IP (PC/TCP mode) => DESATIVO`
11. `turn IP (reverse ARP) => DESATIVO`
12. `turn drop source routed frames => ATIVO`

Em 'network device support':

1. `turn network device support => ATIVO`
2. `turn net driver support => ATIVO`
3. `turn ethernet (10/100 mbit.) => ATIVO`

## 2.4 A filtragem de pacotes

---

As informações que atravessam uma rede são desmembradas e enviadas na forma de *pacotes* ou *datagramas*. Tais pacotes têm um formato que inclui um *cabeçalho* e um *corpo de dados*. No cabeçalho do pacote IP encontramos algumas informações, tais como:

- a) **endereço de origem**
- b) **endereço de destino**

Por conseguinte, um “filtro de pacotes” possui a habilidade de *olhá-los* quando eles passam e assim decidir o seu destino. Podendo escolher por **negar** (*deny*) um pacote, descartando-o totalmente, **aceitar** (*accept*) o pacote, deixando-o passar, ou, por fim, **rejeitar** (*reject*) o pacote, porém retornando uma mensagem ao *endereço de origem*. Eis aí três conceitos fundamentais em qualquer software de *firewall*.

É por essas características que um *firewall* pode nos garantir segurança e controle numa rede - o administrador por evitar o envio de pacotes para fora da rede, ou então que pacote de fora entre em certas partes da intranet.

## **2.5 IPCHAINS (The Enhanced IP Firewalling Chains Software for Linux)**

O Ipchains<sup>5</sup>, escrito por Rusty Russel (*ipchains@rustcorp.com*), tem a habilidade de filtrar os pacotes que passam pelo kernel. Não é um software simples, mas conhecendo o seu mecanismo e seus conceitos o administrador de rede pode escolher a política de segurança de sua rede. A versão trabalhada aqui será a 1.3.8 (ipchains-1.3.8-3.i386.rpm).

Para lidar com um pacote o kernel do Linux possui três *regras* principais, que o Ipchains chama de *firewall chains* ou *chains* - não usaremos tradução para a palavra chains/chain, iremos portanto conservá-la em inglês. São elas:

**IP INPUT CHAIN: quando um pacote entra ...**  
**IP OUTPUT CHAIN: quando um pacote sai ...**  
**IP FORWARD CHAIN: quando um pacote é roteado para  
outra máquina ...**

É neste tripé que se fundamenta a construção de nosso *firewall*. Monta-se, portanto, as regras para gerir esses chains principais: um pacote entra, posso recusá-lo de tal endereço de origem, aceitá-lo para tal destino, mas se atravessa meu *firewall* em direção a tal ou qual destino recuso, etc.

As quatro operações mais básicas e comuns no Ipchains são: *acrescentar* (*append*) uma nova regra com a flag **-A**, *excluir* (*delete*) uma regra com a flag **-D**; e *listar* as regras existentes com a flag **-L**, e *limpar* (*flush*) indiscriminadamente todas as regras com a flag **-F**.

---

<sup>5</sup>Uma questão terminológica: o aluno não deve confundir **ipchains** com o antigo **ipfwadm**: o primeiro está e distribuições com o kernel 2.2.x e o último no kernel 2.0.x; os parâmetros são *essencialmente* diferentes, portanto não servem os scripts feitos para o **ipfwadm**.

Vejamos alguns caminhos com o `Ipchains` em ação. É curioso notar que toda publicação ensinando uma linguagem de programação em geral comece mostrando como gerar, nesta linguagem, uma frase do tipo 'Hello World!'... No caso das regras do *firewall*, começamos mostrando como impedir o **ping** para a interface de "loopback" (lo), teríamos o seguinte:

```
ipchains -A input -s 127.0.0.1 -p icmp -j DENY
```

*Elucidando os parâmetro usados:*

- A input ➔ acrescentamos uma regra para a entrada
- s ➔ eis o endereço de origens dos pacotes
- p ➔ o protocolo usado (ICMP)
- j DENY ➔ o que faremos com o pacote (*jump-to*): aqui negamos

Agora, se executamos um **ping** para o endereço de *loopback*, iremos ver a tradicional mensagem de '**100% packet loss**'. A regra que estabelecemos diz que deve-se *negar* todos pacotes ICMP que chegarem do endereço **127.0.0.1**, - o parâmetro **DENY** é chamado tecnicamente de *alvo* (*target*).

Vejamos outro exemplo, agora numa interface de rede. Em nosso cenário desejamos fechar as portas **22** e **23** do nosso *host* alpha (192.168.0.1) para beta (192.168.0.2); em suma: beta não poderá usar tais serviços da máquina alpha. Estas são as portas padrão para Telnet e FTP no protocolo TCP. Segundo a lógica da suite de protocolos TCP/IP, necessitamos além do endereço IP, de um outro nível de atribuição de endereços que é chamado *porta* (de 0 a 65535) - é ela que direciona os dados que chegam pelo endereço IP para um determinado serviço. De forma que pode-se manter um número de conexões simultâneas e, ao mesmo tempo, separadas. As portas são

separadas em dois grupos distintos: de 0 a 1023 são as “portas privilegiadas” são usadas por daemons confiáveis e com privilégios de root. As restantes, de 1024 até 65535, são chamadas de “não-privilegiadas” e são usadas livremente. Por isso, ao montar um *firewall* toda a atenção com os movimentos de portas não privilegiadas. Abaixo está uma lista resumida com os principais serviços, seus portas *padrão* e o protocolo correspondente:

SERVIÇO	PORTA	PROTOCOLO
netstat	15	tcp
ftp	21	tcp
ssh	22	tcp/udp
telnet	23	tcp
smtp	25	tcp
whois	43	tcp
finger	79	tcp
www	80	tcp
pop-3	110	tcp/udp
https	443	tcp/udp

Se estiver confuso sobre qual o número que corresponde a tal ou qual serviço, e também desejar uma listagem mais completa, consulte o arquivo `'/etc/services'`, é ele que armazena a correspondência entre o nome do serviço (Telnet) e o número da porta (23) e, também, o protocolo que é usado (TCP)<sup>6</sup>.

Observe, então, nosso exemplo:

```
[root@alpha /]# ipchains -A input -s 192.168.0.2 -d 192.168.0.1 :23 -p tcp -J DENY
[root@alpha /]# ipchains -A input -s 192.168.0.2 -d 192.168.0.1 :21 -p tcp -J DENY
```

Com essas regras negamos (**DENY**) a entrada de pacotes do endereço de origem (*source address*, `'-s'`) 192.168.0.2 para o endereço de destino (`'-d'`) 192.168.0.1, nas portas **23** e **21**, no protocolo TCP. Ok, sem mensagens de erro, nossas regras foram aceitas. Tente agora

<sup>6</sup>Pode-se consultar também o **RFC 177** para uma lista completa das portas.

abrir uma sessão Telnet ou FTP da máquina beta, a tentativa obviamente será sem sucesso, - a porta está fechada para esta máquina. A flag **-J** especifica um **alvo**, aqui usamos DENY; os outros alvos mais importantes são **ACCEPT**, **REJECT** e **MASQ**. Os dois primeiros já mencionamos anteriormente, um aceita a entrada e o rejeita, porém respondendo ao endereço de origem, já o alvo MASQ só é usado quando usamos o chain *forward* ('**ipchains -A forward**'), deixa o pacote passar, porém usando a técnica do mascaramento.

Feito isso, vamos entrar outra opção do Ipchains: vamos pedir uma lista (flag **-L**) de nossas regras:

```
[root@alpha /]# ipchains -L
Chain input (policy ACCEPT):
target      prot opt      source                destination            ports
DENY        tcp  -----  beta.cipsga.org.br   alpha.cipsga.org.br   any ->  0:telnet
DENY        tcp  -----  beta.cipsga.org.br   alpha.cipsga.org.br   any ->  0:ftp
Chain forward (policy ACCEPT):
Chain output (policy ACCEPT):
```

A saída que obtemos da tela é muito elucidativa e também muito clara. De início devemos observar que ele separa os três chains principais (**input**, **forward** e **output**), e nos mostra quais regras implementamos para eles; observe que 'input' e 'forward' estão vazios, pois nada implementamos ainda. Outro detalhe a ser observado: a presença da expressão entre parênteses '**policy ACCEPT**'. Isto quer dizer que a política *default* de nosso *firewall* é **aceitar** pacotes; o que é feito pela flag **-P**, ao digitarmos por exemplo: '**ipchains -P input DENY**', ou ainda: '**ipchains -P output REJECT**'. No segundo caso, se nada for especificado, o kernel *por padrão* rejeitará a saída de qualquer pacote.

Seguem em colunas as características que escolhemos. Vejamos:

O alvo	Prot (o protocolo)	Opt (as opções)	Source	Destination	As portas	O serviço
Qual alvo			O endereço	O endereço	As portas	Os

usamos...	O protocolo que escolhemos	Nenhuma opção específica foi usada...	de origem, que foi traduzido para um FQDN*	de destino, que foi traduzido para um FQDN*	em jogo... No sentido de ⇨ para	serviços, traduzidos de 'portas' para seu nome...
-----------	----------------------------	---------------------------------------	--	---	---------------------------------	---

\*Fully Qualified Domain Name

Vamos observar agora como podemos *excluir* as regras que criamos. Há, primeiramente, uma solução digamos radical. Podemos simplesmente aplicar o parâmetro '**flush**', isto é, limpamos *totalmente* as duas regras estabelecidas:

```
[root@alpha /]# ipchains -F
[root@alpha /]# ipchains -L
Chain input (policy ACCEPT):
Chain forward (policy ACCEPT):
Chain output (policy ACCEPT):
```

Depois do **flush**, vimos a nossa lista e ela está vazia...

Entretanto, isso não seria uma solução se desejarmos aproveitar outras regras estabelecidas. No nosso exemplo, temos duas regras. A primeira fecha a porta 23 (Telnet), e já a segunda fecha a porta 21 (FTP). Podemos derrubar apenas a primeira regra:

```
[root@alpha /]# ipchains -D input 1
```

E depois para nos certificarmos digitamos:

```
[root@alpha /]# ipchains -L
Chain input (policy ACCEPT):
target    prot opt      source                destination            ports
DENY      tcp  ----|-  beta.cipsga.org.br   alpha.cipsga.org.br   any ->  0:ftp
Chain forward (policy ACCEPT):
Chain output (policy ACCEPT):
```

Só nos resta então uma regra, a da porta de FTP. É importante aqui o aluno perceber a flag **input**, pois é ela que define que queremos eliminar a regra 1 dos *firewall chains* de entrada, e não de saída (**output**), etc.

Um outro exemplo. Estabelecemos fechar a conexão via TCP *sem especificar porta, nem endereço de origem* (atenção para esses detalhes). E igualmente negamos qualquer conexão FTP da máquina alpha seja para qual destino for. Veja (lembrando: alpha=192.168.0.1 e beta=192.168.0.2):

```
[root@alpha /]# ipchains -A input -d 192.168.0.1 -p tcp -J DENY
[root@alpha /]# ipchains -A output -s 192.168.0.1 :21 -p tcp -J DENY
[root@alpha /]# ipchains -L
Chain input (policy ACCEPT):
target    prot opt      source                destination           ports
DENY      tcp  ----- anywhere            alpha.cipsga.org.br  any ->  any
Chain forward (policy ACCEPT):
Chain output (policy ACCEPT):
target    prot opt      source                destination           ports
DENY      tcp  ----- alpha.cipsga.org.br  anywhere              0:ftp ->  any
```

Queremos então liberar as conexões de FTP da máquina 192.168.0.1? Simples: **'ipchains -D output 1'**.

Com esse cenário acima, se usarmos uma ferramenta de varredura de portas, teremos um quadro interessante. Podemos usar o **nmap** (*Network Exploration Tool and Security Scanner*) escrito por Fyodor, versão **2.3 beta10** ([www.insecure.org/nmap](http://www.insecure.org/nmap)), ele nos mostra as portas abertas e/ou filtradas, e mesmo se nossa conexão foi totalmente derrubada. Nesta tela ele nos mostra os *sockets* que o Ipchains filtrou e os que deixamos abertos:

```
[root@alpha /]#/usr/local/bin/nmap -v alpha.cipsga.org.br
Starting nmap V. 2.3BETA10 by Fyodor (fyodor@dhp.com, www.insecure.org/nmap/)
Host alpha.cipsga.org.br (192.168.0.1) appears to be up ... good.
Initiating TCP connect() scan against alpha.cipsga.org.br (192.168.0.1)
Adding TCP port 139 (state Open).
Adding TCP port 514 (state Open).
(ecoam todas as portas investigadas...)
The TCP connect scan took 4 seconds to scan 1510 ports.
Interesting ports on alpha.cipsga.org.br (192.168.0.1):
Port      State      Protocol  Service
1         filtered  tcp       tcpmux
2         filtered  tcp       compressnet
3         filtered  tcp       compressnet
4         filtered  tcp       unknown
5         filtered  tcp       rje
6         filtered  tcp       unknown
7         filtered  tcp       echo
8         filtered  tcp       unknown
9         filtered  tcp       discard
10        filtered  tcp       unknown
11        filtered  tcp       systat
12        filtered  tcp       unknown
13        filtered  tcp       daytime
```

```
15    filtered    tcp    netstat
16    filtered    tcp    unknown
17    filtered    tcp    qotd
18    filtered    tcp    msp
19    filtered    tcp    chargen
20    filtered    tcp    ftp-data
21    filtered    tcp    ftp
22    open        tcp    ssh
23    open        tcp    telnet
25    open        tcp    smtp
79    open        tcp    finger
80    open        tcp    http
111   open        tcp    sunrpc
113   open        tcp    auth
139   open        tcp    netbios-ssn
443   open        tcp    https
513   open        tcp    login
514   open        tcp    shell
515   open        tcp    printer
3128  open        tcp    squid-http
```

Nmap run completed -- 1 IP address (1 host up) scanned in 5 seconds

Agora que entramos com a regra para derrubar as conexões TCP, o software nos relata:

```
[root@alpha /]#/usr/local/bin/nmap -v alpha.cipsga.org.br
Starting nmap V. 2.3BETA10 by Fyodor (fyodor@dhp.com, www.insecure.org/nmap/)
Host alpha.cipsga.org.br (192.168.0.1) appears to be up ... good.
Initiating TCP connect() scan against alpha.cipsga.org.br (192.168.0.1)
The TCP connect scan took 151 seconds to scan 1510 ports.
Interesting ports on alpha.cipsga.org.br (192.168.0.1):
(Ports scanned but not shown below are in state: filtered) ↵
Port      State      Protocol  Service
```

Nmap run completed -- 1 IP address (1 host up) scanned in 151 seconds

A linha de flags do Ipchains é bastante maleável, e apenas com o uso e o *debug* constante das regras que o administrador implementa é que se pode dominá-las. Pelo menos até o kernel 2.5...

Relembrando algumas flags vimos que podemos usar no parâmetro **-s** ou **-d** tanto o FQDN ou o endereço IP expresso numericamente. Na especificação das portas podemos usar o número ou o nome do serviço: telnet ou 23, ssh ou 22, etc. Podemos igualmente especificar faixas (*rangs*) de portas: **'-d 192.168.0.2 21:23'**, ou seja, manejamos aqui todas as portas de 21 até 22 (inclusive). Podemos escrever **'-s 192.168.0.3 :22'**, ou **'-s gama.cipsga.org.br ssh'**. O mesmo se passa com os endereços IP que podemos manejar, podemos usar faixas

inclusivas, por exemplo: `'-p tcp 192.168.0.1/10 :25'`, - incluimos com isso os IPs de x.x.x.1 até 192.168.0.10.

Outro recurso fundamental é o da *inversão*. Quando usamos a flag '!'. Ela tem o sentido *negativo*; quando o usamos devemos ler mentalmente: 'todo x menos y'. Assim: `'-p tcp -s 192.168.0.5 ! www'` significa: todas as portas TCP menos a 80 (WWW)...

Exemplo. Digitamos no console da máquina alpha:

```
ipchains -A input -s 192.168.0.2 !ssh -d 192.168.0.1 !ssh -p tcp -j REJECT
```

Com isso fechamos todas as portas TCP, mas deixamos a **22** para o Secure Shell operar. Se tentamos um Telnet, nossa conexão obviamente falhará. Mas o ssh pode estabelecer sua sessão remota sem problemas. Uma listagem na máquina nos mostra:

```
[root@alpha /]# ipchains -L
Chain input (policy ACCEPT):
target      prot  opt      source                destination            ports
DENY        tcp  -----  beta.cipsga.org.br    alpha.cipsga.org.br    !ssh -> !ssh
Chain forward (policy ACCEPT):
Chain output (policy DENY):
```

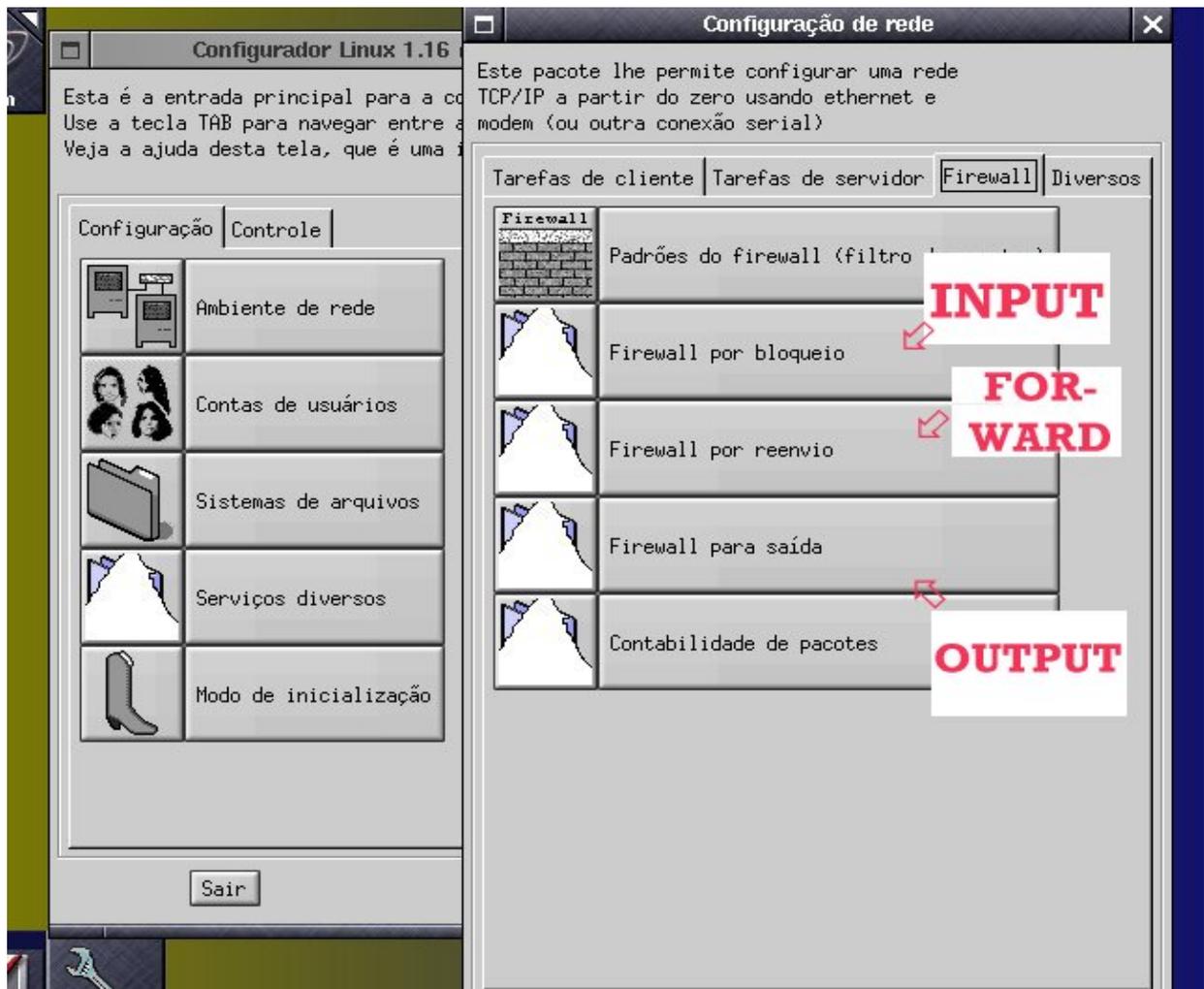
A conhecida flag `'-L'` esclarece que o protocolo TCP tem estar **DENY**, mas *não as portas do ssh* (observe as setas...). Podemos usar o '!' na especificação de endereços IP, nos protocolos (exceto com o ICMP) e nas interfaces de rede. Poderíamos configurar a rejeição de pacotes de qualquer protocolo *menos o UDP*: `'-p ! udp'`. Ou ainda: `'-A input -p tcp -d ! 192.168.0.10 www -j ACCEPT'`; especificamos com isso que aceitamos a entrada de qualquer pacote TCP para a porta 80 menos da máquina 192.168.0.10, que fica excluída. Contudo: `'-A input -p tcp -d 192.168.0.10 ! www -j ACCEPT'` é essencialmente diferente... Agora aceitamos qualquer conexão TCP da máquina 192.168.0.10, menos a da porta Web.

Acima, fizemos menção as interfaces de rede (*network interface*)<sup>7</sup>. O `Ipchains` pode gerir todas as interfaces do sistema, para isso usamos a flag `-i`. Podemos usar `-i eth0` (a sua primeira placa ethernet) ou `-i ppp0` (a conexão de modem via `pppd`), para nos atermos em exemplos mais usuais. Quando uma máquina de nossa rede obtém acesso a internet via `dialup`, o parâmetro é fundamental. Sendo que podemos mesmo especificar uma interface que não está ativa, o que quase sempre é o caso da interface `ppp`. Quando conectamos nossa provedora a interface `ppp0` está presente, o `ifconfig` lista sua presença. Quando desfazemos a conexão a interface desaparece. Pode-se também usar um curinga na opção `-i`, lançando mão do símbolo `+`: `eth+` designa qualquer interface possível (`eth0`, `eth1`, etc.). A regra `'ipchains -A output -d 192.168.0.9 -p udp -i eth+ -j DENY'` aplica-se a qualquer interface ethernet de nossa máquina.

Existe uma ferramenta de configuração chamada **Linuxconf**. Ela está presente em quase nas distribuições da Red Hat, e em língua portuguesa no Conectiva Linux. É uma interface gráfica que funciona dentro do X-Window para editar e gerir as funções do sistemas Linux. Como todo recurso GUI pode às vezes confundir facilidade com desconhecimento. Só devemos nos habilitar a usar tais ferramentas, portanto, quando temos um conhecimento conceitual daquilo que vamos nos ocupar. Vamos dar uma rápida olhada na interface, dentro das funções do `Linuxconf`, para alterar as regras do *firewall*. Numa janela do `xterm` chama o programa: `'linuxconf &'`. Na tela que surge escolha **'Ambiente de rede'**, e depois a guia **'firewall'**. Temos seguinte tela, já com o as equivalências apontadas:

---

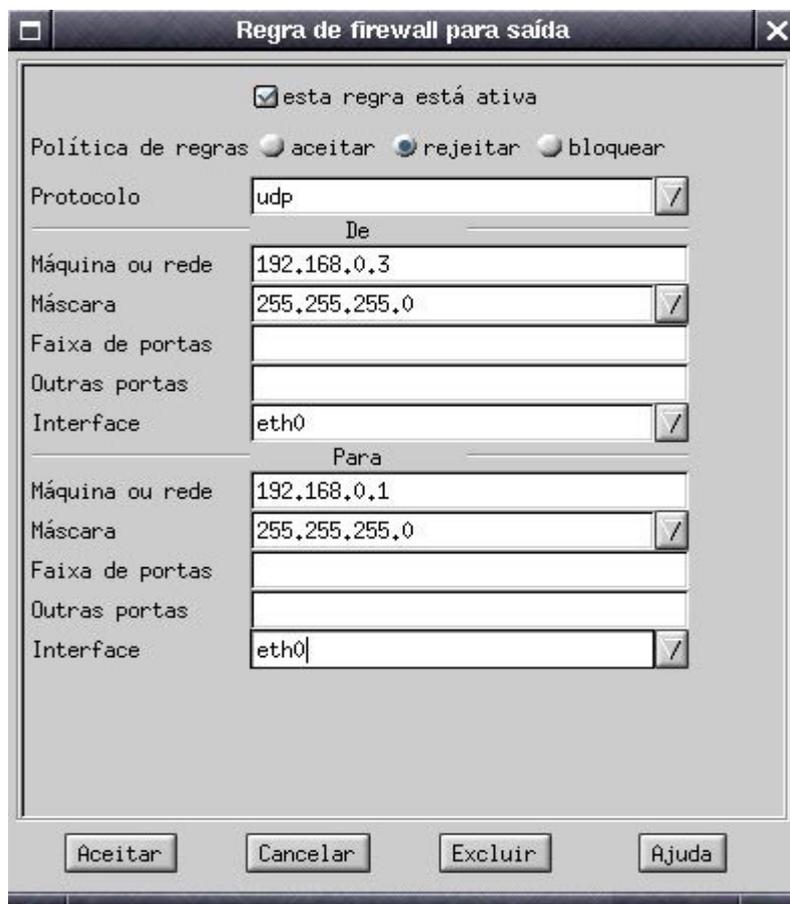
<sup>7</sup>Consulte a documentação sobre o tema, principalmente sobre o comando `ifconfig`, que configura e lista as interfaces existentes nu *host*.



Destacamos os três botões que representam o tripé 'input-forward-output' de todas as regras de *firewalling*. É aqui que você entrará com as regras do *firewall*. Vamos editar os ip chains output (note-se que curiosamente **input** foi traduzido por *bloqueio...*), clique no botão '**saída**' e aparecerá este quadro de diálogo:



Clique, então, no botão '**adicionar**'. Teremos o quadro seguinte que pode nos proporcionar a possibilidade de múltiplas edições do ipchains:



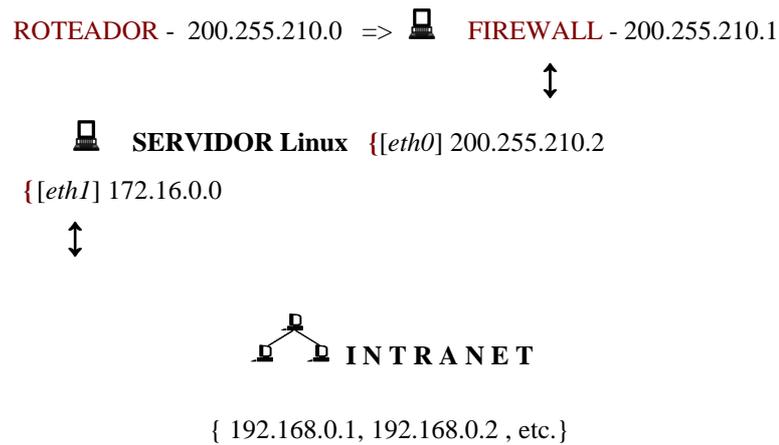
Não há dados novos nessa tela do Linuxconf. A única coisa que não tocamos ainda foi a entrada para a **netmask** (máscara de rede) que é especificada ao lado do endereço IP. Atenta para os alvos aceitar, rejeitar e bloquear (o **DENY**). Traduzindo os campos da janela teríamos: `'ipchains -A output -s 192.168.0.3/255.255.255.0 -s 192.168.0.1/255.255.255.0 -p udp -i eth0 -j reject'`. Então, marcamos o quadradinho '**ativo**' e aceitamos a regra. Para desativá-la, usamos a mesma interface.

Outra opção para gravar as regras que criamos, são os scripts de inicialização tradicionalmente guardados no diretório **rc.d**. Testa-se exaustivamente as regras, executa-se o *debug*, e então criamos um script que carrega as regras de *firewall*, na partida do Linux (consulte-se a documentação existente para operar tais alterações em seu sistema). Há dois scripts que ajudam muito o Sysadmin a montar seu *firewall*: o **ipchains-save** e o **ipchains-restore**. O primeiro grava num arquivo via redirecionamento as regras que você criou: por exemplo, '**ipchains-save > ~/firewall-testado**' - evidentemente você pode usar o nome que desejar. Já o **ipchains-restore** restaura as regras gravadas pelo **ipchains-save**: '**ipchains-restore < ~/firewall-testado**'. Para executá-los é necessário os privilégios de usuário root. Outro recurso muito útil é a capacidade do Ipchains de agrupar várias regras num chain. Como vimos, o programa tem três chains básicos, chamados *built-in chains*, estão embutidos no programa, se assim podemos traduzir. A flag usada é '**-N**'. Digitamos:

```
[root@alpha /]# ipchains -N nao-udp
[root@alpha /]# ipchains -A input -i eth1 -j nao-udp
[root@alpha /]# ipchains -A nao-udp -p udp -s 192.168.0.10 -j DENY
[root@alpha /]# ipchains -A nao-udp -p udp -s 192.168.0.9 55:65 -j REJECT (etc.)
```

Assim agrupamos sob o nome '**nao-udp**' um conjunto de regras. Podemos com o **ipchains-save**, como vimos, gravá-las e, enfim, restaurá-las.

Encerrando... Vamos analisar um cenário clássico em redes médias, muito presente nos dias atuais. Tenha em mente uma intranet com um servidor oferecendo vários serviços típicos (Web, FTP, etc.), conectados a uma máquina *firewall* Linux, e este, por sua vez, a um roteador com uma linha dedicada. Observe os IPs na nossa representação:



Vejamos as configurações:

No Firewall...

**Desligamos os serviços desnecessários, deixando o SSH para conexão remota com o servidor somente. E vamos proteger nossa intranet...**

```
ipchains -A input -p tcp -s 172.16.0.0 -j DENY
```

```
ipchains -A input -p tcp -s 192.168.0.1/10 -j DENY
```

```
ipchains -A input -p tcp -s ! 200.255.210.2 ssh -j DENY
```

No Servidor...

**Como nosso servidor irá ter serviços da Internet públicos precisamos liberá-los...**

```
ipchains -A input -p tcp -s 0.0.0.0/0 -d 255.210.02 www -j ACCEPT
```

```
ipchains -A input -p tcp -s 0.0.0.0/0 -d 255.210.02 ftp -j ACCEPT
```

*Inclua aqui todos  
os serviços que serão oferecidos...*

```
ipchains -A input -p tcp -s 192.168.0.1/10 -j DENY
```

*(Fechando nossa intranet...)*

```
ipchains -A input -p tcp -i lo -j ACCEPT
```

*(Abrindo interface 'lo'...)*

```
ipchains -P forward DENY
```

```
ipchains -A forward -s 192.168.0.1/10 -d 200.255.210.2 -j MASQ
```

*(Usando a técnica do mascaramento de IPs...)*

⊕ Ou seja: qualquer faixa de IPs...

Uma palavra sobre o tópico de mascaramento de IPs (*IP masquerading*). Ao configurarmos o Ipchains para usar o alvo MASQ, com a flag '**-j MASQ**', possibilitamos que um ou mais endereços IPs "falsos" ou privativos possam ser usados na extranet. No nosso exemplo, todos os endereços de nossa intranet aparecem usando o IP do servidor (**200.255.210.2**). Para tanto não se esqueça de configurar o arquivo **/etc/sysconfig** adicionando '**forward\_ipv4 = yes**'. Se você não editar o arquivo, terá que acionar o IP *forwarding* manualmente. Exemplo, se tentar efetivar o IP chains *forward* sem a linha acrescida, terei o seguinte em minha tela:

```
[root@alpha /]# ipchains -A forward -p tcp -s 200.255.210.2 -d 192.168.0.10 -j ACCEPT
Warning: you must enable IP forwarding for packets to be forwarded at all:
Use 'echo 1 > /proc/sys/net/ipv4/ip_forward' ↵
```

[root@alpha /]#

Basta seguir a orientação (observe a seta). Este comando irá ativar o IP forwarding. E, então, os pacotes poderão passar *através* do *host* alpha para o destino desejado. Estas técnicas tornam-se poderosas quando estabelecidas em conjunto com as *rotas de rede, gateways, etc.* Porém, são tópicos que fogem do escopo deste documento.

Abaixo, uma lista das opções do software Ipchains. Para uma listagem completa, use o '*IP Chains Quick Reference*' de S. Bronson, feito em PostScript, e que pode ser trazido da página Web oficial em <http://www.adelaide.net.au/~rustcorp/linux/ipchains>.

```
Uso:  ipchains -[ADC] chain regra-especificada [opções]
      ipchains -[RI] chain nº-da-regra regra-especificada [opções]
      ipchains -D chain nº-da-regra [opções]
      ipchains -[LFZNX] [chain] [opções]
      ipchains -P chain alvo [opções]
      ipchains -M [ -L | -S ] [opções]
      ipchains -h [icmp] (mostra informação de ajuda, ou lista ICMP)
```

#### Comandos:

Permite-se a forma longa ou curta.

```
--add      -A chain          Acrescenta chain
--delete   -D chain          Exclui regra
--delete   -D chain nº-da-regra
                                     exclui regra nº-da-regra (1 = primeira) do chain
--insert   -I chain [nº-da-regra]
                                     Insere no chain como nº-da-regra (padrão 1=primeiro)
--replace  -R chain nº-da-regra
                                     Substitui regra nº-da-regra (1 = primeira) no chain
--list     -L [chain]        Lista regras num chain ou todos os chains
--flush    -F [chain]        Exclui todas regras no chain or todos os chains
--zero     -Z [chain]        Zera os chains
--check    -C chain          Testa um pacote num chain
--new      -N chain          Cria um novo 'user-defined chain'
--delete-chain
                                     -X chain          Exclui um 'user-defined chain'
--policy   -P chain alvo
                                     Muda a política num chain para tal alvo
--masquerade -M -L          Lista as conexões mascaradas atuais
--set      -M -S tcp tcpfin udp
                                     Põe valores de 'timeout' para mascaramento
```

#### Opções:

```
--bidirectional -b          insere duas regras: uma com -s & -d invertido
--proto        -p [!] proto  protocolo: pelo nº or nome, ex. `tcp'
--source       -s [!] endereço[/masc.] [!] [porta[:porta]]
                                     origem especificação
--source-port  [!] [porta[:porta]]
                                     origem porta especificação
--destination -d [!] endereço[/masc.] [!] [porta[:porta]]
                                     destino especificação
--destination-port [!] [porta[:porta]]
                                     destino porta especificação
--icmp-type   [!] tipo       especifica tipo de ICMP
```

```
--interface  -i [!] nome[+]
               nome da interface de rede ([+] para curinga)
--jump      -j alvo [porta]
               alvo para a regra ([porta] para REDIRECT)
--mark      -m [+-]mark n° para 'mark' no pacote correspondente
--numeric   -n                saída numerica de endereços e portas
--log       -l                habilita registro (log) no kernel
--output    -o [tamanhomax.] saída de pacote para dispositivo 'netlink'
--TOS       -t e xor e/xor máscaras para TOS
--verbose   -v                modo 'verbose'
--exact     -x                expande números (mostra valores exatos)
[!] --fragment -f            combina somente o segundo ou mais fragmentos
[!] --syn     -y            combina pacotes TCP somente quando configura 'SYN'
[!] --version -v            mostra versão.
```

---

## 2.6 The SINUS Firewall - a TCP/IP packet filter for Linux

---

Veremos agora uma outra solução de gerenciamento de *firewall* em ambiente Linux. Não faremos comparações entre o *Ipchains* e o *SINUS*. O primeiro já um produto pronto para funcionar, o segundo um produto em *amplo* desenvolvimento. Pelo menos, a versão 0.1.5, que foi desenvolvida para o kernel 2.2.x do Linux<sup>8</sup>. O *SINUS Firewall* (*sf Firewall*) é um projeto da Universidade de Zurique, com o concurso da SWITCH, da Telekurs Payserv AG e da ETH Zürich, desenvolvido por Robert Muchsel & Roland Schmid. Informações sobre o projeto e *download* do programa estão em:

✓ <http://www.ifi.unizh.ch/ikm/SINUS/firewall.html>

✓ <ftp://ftp.ifi.unizh.ch/pub/security/firewall>

Os desenvolvedores advertem do estado de desenvolvimento do software. Portanto, ele não ser usado *ainda* para a segurança de redes com dados críticos. Advertem ainda: use-o para aprendizado e se você não tem nenhum *firewall*, ou não confia no que tem.

Os aspectos positivos do programa estão no seu modo de atuação. Ele trabalha com um arquivo de configuração `'/etc/firewall.d/firewall.conf'`; ao carregar ele lê as configurações aí estocadas. Há ainda com o pacote (**sifi-0.1.5.tar.gz**) um servidor (o software para *firewall* propriamente dito) e um cliente, escrito

---

<sup>8</sup>Há uma versão *estável* para versões de kernel mais antigos.

em **Java**, uma interface para facilitar a edição do **firewall.conf**, chamado *Firewall Control Panel*. Há, outrossim, a contribuição de Benedict Trefzer para a leituras dos *logs* gerados pelo SINUS. Uma ferramenta baseada em HTML e mSQL.

Como não há pacotes RPM, nem binários TAR.GZIP, com SINUS, teremos que passar os passos de compilação do software. Mesmo porque há detalhes que não podem ser negligenciados pelo aluno.

Já vimos que um *firewall* no Linux é implementado em nível de kernel. O arquivo '**README**' do SINUS adverte o usuário da necessidade de recompilação do kernel. Também já mencionamos que várias distribuições Linux já vem com o kernel prontos para *firewalling*. Por conseguinte, certos disso, não se faz necessário recompilar todo o kernel. No entanto, para a compilação do SINUS precisamos de somente *um* arquivo das fontes do kernel. Um *object file* usado no '*link*' para gerar o executável do programa, - sem ele a compilação falhará. Mas temos uma *dica* para resolver o problema... ;-)

Certifique-se que você tem em sua máquina um ambiente de desenvolvimento instalado (*gcc*, *headers files*, etc.). Depois, instale o arquivo que vem com as fontes do kernel: **kernel-source-2.2.13.i386.rpm**. Digite o seguinte:

```
[root@alpha /]# cd /usr/lib/linux/arch/i386/lib
[root@alpha lib]# gcc -D__ASSEMBLY__ -traditional -c checksum.S    ⇨ atenção
[root@alpha lib]# ls c*
[root@alpha lib]# checksum.S    checksum.o
                             ↑
```

Observe a existência do arquivo que desejamos: o '**checksum.o**'<sup>9</sup>. Agora podemos descompactar o arquivo com as fontes do SINUS supracitado. Escolha um diretório a seu gosto. Se você possui o JDK 1.1.6x e o pacote SWING (<http://www.java.sun.com>), pode compilar o

---

<sup>9</sup>Se você optar por recompilar o kernel, **não** execute o '**make clean**' no fim do processo! Pois sabidamente ele apagará todos os arquivos **\*.o** gerados...

sf Control Panel. Se desejar edite os *paths* no diretório '**client/Makefile.in**', para ajustar o caminho para as *classes* Java. Há uma versão não-oficial, já compilada, na página <http://www.sinusfirewall.org> do cliente do Control Panel, pode ser uma boa opção, se você tiver problemas com o JDK, com as classes SWING, ou mesmo se não desejar compilar em ambiente Java... Execute então no diretório aonde você instalou as fontes:

**./configure**

```
make { 'make client' somente gerará o sf Control Panel...  
      { 'make server' somente gerará o servidor...
```

Antes do **make install** você deve criar um novo 'usuário' chamado **firewall** adicionando a seguinte linha em seu arquivo '**/etc/passwd**' - use um **user id** e **group id** desocupados:

```
firewall:x:888:888::/dev/null:/bin/false
```

Após, crie um novo grupo também chamado **firewall** adicionando estas linhas no seu arquivo '**/etc/group**' (não se esqueça de efetuar um *backup* nos arquivos alterados):

```
firewall:x:888:
```

Feito isso a instalação pode seguir...

```
make install { pode ser 'make server_install'  
              OU 'make client_install'...
```

A instalação por padrão, - e também se a compilação transcorreu normalmente -, põe os seguintes arquivos do servidor (no momento nos ocuparemos com eles apenas...) nestes diretórios:

```

/usr/local/sbin/sfcaccount

/usr/local/sbin/sfc  ⇒      o daemon...

/var/log/firewall    { 'LOGS': arquivos de registros...
/var/log/firewall.spy
/var/log/firewall.report

/etc/firewall.d/ { reservado para os arquivos firewall.conf
                  e firewall.passwd

/lib/modules/misc/sf.o  ⇒      um modulo instalável...

```

Vamos iniciar o SINUS Firewall. Para isso precisamos de uma configuração, pois o diretório `/etc/firewall.d` está vazio após o `'make install'`, falta o arquivo `firewall.conf`. Depois que editar uma configuração ao nosso gosto e necessidades, instalamos o módulo<sup>10</sup> com `'insmod /lib/modules/misc/sf.o'`. Depois damos partida no daemon:

```

[root@alpha /]# insmod /lib/modules/misc/sf.o
[root@alpha /]# /usr/local/sbin/sfc start
Initializing daemon...
No socket security: firewall-to-firewall communication disable

[root@alpha /]#

```

Quando o daemon `'sfc'` dá partida, ele lê a sua configuração e executa as regras desejadas de `firewall`. Um aspecto interessante do programa daemon é que ele checa as inconsistências possíveis contidas no `firewall.conf`: tanto de regras quanto erros de digitação. Se isso acontecer ele pára, e devolve um `"parse error"` ao usuário. Cheque tais incongruências e repita o processo. Quanto a

<sup>10</sup>Consulte a documentação disponível a respeito dos módulos instaláveis no Linux, ou mesmo as páginas manuais (`'man lsmod'`, `'man insmod'`, `'man rmmod'`).

mensagem ecoada "**no socket security...**" não se preocupe, ela aparece quando o SINUS é compilado sem o programa ENskip. Ele possibilita a conexão criptografada entre *firewalls*. Mas como temos o SSH para esse intento, podemos abrir de tal suporte. Há também sérios problemas de compilação do Enskip (<ftp://ftp.tik.ee.ethz.ch/pub/packages/skip/>) no Linux.

Para desligar o SINUS, remova o modulo da memória ('**rmmod sf.o**'), e pare o daemon com o comando '**/usr/local/sbin/sfc stop**'.

Quando do *debug* de nossa configuração estiver pronto, e estivermos satisfeitos com a montagem das regras do *firewall*, podemos usar um script para iniciar o nosso daemon '**sfc**'. Abaixo está um exemplo de um script que usamos em nossa rede, pode-se adaptá-lo às mais diversas situações:

```
# -----
#      Script para iniciar daemon do SINUS Firewall@
#      e instalar modulo 'sf.o' *** Janeiro de 2000
#      Usado num Sistema Linux Red Hat 6.0 kernel 2.22
# -----

. /etc/rc.d/init.d/functions #Inicia aqui o script 'functions'...

case "$1" in
  start)
    gprintf "Starting %s: " "sf Firewall"
    if test -r /var/run/sfc.pid && kill -0 `cat /var/run/sfc.pid`
    then
      gprintf "%s exists, %s already running.\n" \
        "/var/run/sfc.pid" "sfc"
    else /sbin/insmod /lib/modules/misc/sf.o ; /usr/local/sbin/sfc start

      echo sfc
    fi
    touch /var/lock/subsys/sfc
    ;;
  stop)
    gprintf "Stopping %s: " "sf Firewall"
    /usr/local/sbin/sfc stop
    gprintf "Removing SINUS module..."
    /sbin/rmmod sf
    rm -f /var/run/sfc.pid
    rm -f /var/lock/subsys/sfc
    echo "sfc"
    ;;
  restart)
    $0 stop
    $0 start
    ;;
  status)
    status sfc

```

```
    /usr/local/sbin/sfc show
    ;;
*)
    gprintf "Usage: %s {start|stop|status|restart}\n" "sfc"
    exit 1
esac
exit 0
```

Vamos agora analisar a construção de nossas regras de *firewall*, que ficaram contidas no **firewall.conf**. Os autores colocam no diretório '**samples/**' quatro arquivos de configuração à título de exemplo, para que o administrador estude-os:

↗ [CheswickBellovin.conf](#)

↗ [MediumEnterprise.conf](#)

↗ [SimpleSample.conf](#)

↗ [SmallBusiness.conf](#)

Consulte-os depois de dominar as técnicas de *firewalling*. Os nomes dos arquivos já demonstram o seu intento. Somente o primeiro talvez não o seja. Quem não está familiarizado com a bibliografia sobre segurança, pode não conhecê-los: W. R. Cheswick e S. M. Bellovin são autores de um clássico sobre o tema chamado *Firewalls and Internet Security: Repelling the Wily Hacker*. O primeiro arquivos tenta aplicar os preceitos descritos neste livro.

Vamos usar aqui um **firewall.conf** alternativo, bem simples para estudar a dinâmica do SINUS Firewall. Ele está no manual do programa, o aluno o encontrará em formato *html* no diretório '**doc/html/index.html**' da distribuição. Modificamos algumas partes, mas em síntese ele está aí. Vejamos:

```
# .....
# | Sample configuration for the sf Firewall|
# .....

setup

internalnets 192.168.0.1,192.168.0.2,192.168.0.3,192.168.0.4;
mail_default "root@localhost";
```

```
rules

block tcp to port 87, /* link */
      port 95 /* supdup */
      notification_level 99;

block options loose_source_route, strict_source_route all
      notification_level 13;

# detecta rotas secundárias para a Internet

block rip from inside notification_level 12;

# Permite a entrada de requisições FTP para nosso servidor FTP

accept tcp to 192.168.0.1 port 22 notification_level 4;
accept tcp to 192.168.0.1 port 21 notification_level 1;

# Permite a entrada de requisições FTP para o servidor Telnet/Login

accept tcp to 192.168.0.1 port 23 notification_level 2;
accept tcp to 192.168.0.1 port 80 notification_level 3;
accept all from 127.0.0.1 to 127.0.0.1 notification_level 0;

# bloqueia qualquer outra requisição TCP e soa o alarme...

block tcp notification_level 99;

notification

level 1: /* registra todas as chegadas de conexão FTP permitidas */
  message "Requisicao de conexao FTP.";

level 2: /* registra todas as chegadas de conexão Telnet permitidas */
  message "Requisicao de conexao Telnet.";

level 3: /* registra todas as chegadas de conexão www permitidas*/
  message "Requisicao de conexao WWW.";

level 4: /* registra todas as chegadas de conexão SSH permitidas*/
  message "Requisicao de Secure shell.";

level 11:
  message "Redirecionamento ICMP recebido.";

level 12:
  message "Ha uma rota secundária para a Internet.";

level 13:
  message "Pacote IP com opcao _source route_ detectada...";
  let sr:sourcehost := sr:sourcehost + 1 timeout 300;
  if sr:sourcehost = 1 then
    message "Pacote IP com opcao _source route_ detectada...";
    spy;
  endif;

level 99:
  message "Conexao TCP ilegal!";
  let illtcp:sourcehost := illtcp:sourcehost + 1 timeout 600;
  if illtcp:sourcehost = 1 then
    message "Conexao TCP ilegal!";
    mail;

    spy;
  endif;

level 999: /* Desliga INTERFACE ETH0!! */
  exec "ifconfig eth0 down";
```

end.

O **firewall.conf** possui três blocos de configuração, são eles:

➤ **SETUP**

➤ **RULES** (pode-se usar também **PRIMARY\_RULES**)

➤ **NOTIFICATION**

Entre os blocos de configuração podemos ter comentários aceitando o símbolo # ou as barras tradicionais do C /\* (...) \*/.

O bloco SETUP é bem curto, aí colocamos a informação sobre nossa topologia de rede (**internalnets**) para trabalhar contra o *ip spoofing*, listando assim nossa intranet. Os IPs devem ser separados por vírgulas e terminados com ';', Pode-se aqui também especificar o **netmask**: '192.168.0.1 mask 255.255.255.0, (...)'. Segue-se depois a informação do(s) endereço(s) usado(s) (**mail\_default**) pelo SINUS para o eventual envio de e-mails, coloque quantos desejar, separando-os por vírgulas.

O bloco RULES é o principal do arquivo. Nesta seção as linhas começam obrigatoriamente com as palavras: **ACCEPT** (aceita os pacotes enviados), **BLOCK** (O correspondente ao **DENY** do Ipchains, abandona os pacotes sem maiores considerações), **REJECT** (rejeita gerando justificativa) e **OBSERVE** (construída para gerir os *logs* do SINUS. A rigor, não aceita nem recusa, mas deixa a decisão para a próxima regra). Por fim, a regras são direcionadas para um "nível de notificação" que se encontra no último bloco, e terminadas com um ';'. Os conceitos são basicamente os mesmos do Ipchains, e você deve trazer os seus conhecimentos. O que muda é sintaxe nas linhas de *configuração*. Poder-se-ia resumi-la assim:

[regra] proto. IP (de) porta XX IP (para) porta XX -> nível de notificação

Retirando um exemplo de nosso arquivo encontramos:

```
accept tcp to 192.168.0.1 port 22 notification_level 4;
```

Qual regra estabelece-se aqui? Aceitamos pacotes TCP do IP 192.168.0.1 provenientes da porta 22, e se entram os pacotes executa-se o **notification\_level** de número 4. Que estabelece:

```
level 4:  
message "Requisicao de Secure shell.";
```

Envia para o arquivo de registro do SINUS (em **'/var/log/firewall'**) a informação que houve uma requisição na porta 22.

Algumas observações ainda sobre os parâmetros do arquivo: podemos estabelecer uma regra para todos os protocolos aceitos com a flag **'all'**, também estabelecer faixas de portas (xx..xx), e configurar as faixas de endereços usando **'to ... from'** (**'do IP tal para o IP tal...'**). Exemplo:

```
block all to 192.168.0.10 port 1024..65535 from 192.168.0.7 1024..65535  
notification_level 3;
```

Outro parâmetro importantíssimo é **'inside'** que não aparece em nosso arquivo-exemplo. Inside é nossa *intranet*, definida como vimos, na flag **internalnets**. Ele é combinado com o parâmetro **'to (...) from'**. E então se estabeleço, por exemplo:

```
block tcp from inside to 200.255.216.1 port 21 notification_level 9;
```

Bloqueio qualquer pacote TCP de minha intranet para o IP 200.255.216.1. Posso inverter **'inside'** usando **'outside'** - isto é, qualquer IP fora de minha intranet.

```
block tcp from outside to 200.255.216.1 port 21 notification_level 9;
```

Por fim, o bloco NOTIFICATION. Nesta seção cada nível de notificação será estabelecido num parágrafo. Iniciando-se com a palavra '**level**', seguido de um número identificador e ':'. Pode-se especificar as ações necessárias a serem implantadas. O nível de notificação funciona em nível de script. Em pseudo-código ficaria:

**SE**  
determinadas circunstâncias ocorrem...  
Mande uma MENSAGEM...  
E CHAMA (**call**) o Nível 900



NÍVEL 900  
Se as circunstâncias ocorrem 10 vezes  
**DESLIGUE A INTERFACE ETH0!**

É claro que demos um exemplo radical. Vejamos outro caso. Todo acesso ao meu servidor Telnet em uma ou mais *hosts*, redireciono para um nível de notificação:

```
level 10:  
  message "Solicitacao de sessao TELNET na porta 23...";  
  call 8000;
```

Com a linha '**call 8000**' reenvio para uma **sub-rotina**, o '**level 8000**' em meu **firewall.conf**. Onde:

```
level 8000:  
  let connections:sourcehost := connections:sourcehost + 1 timeout 15;  
  if connections:sourcehost > 50 then  
    report;  
    message "Alguem esta tentando ativar mais de 50 conexoes Telnet!"  
    "Bloqueando acesso por 30 segundos...";  
    block all from sourcehost notification_level 0 timeout 30;  
    call 20000;  
  endif;
```

Esclarecendo: a palavra '**let**' que inicia a frase, pode estabelecer ou mudar o valor de uma variável. Na sentença '**connections:sourcehost**', o primeiro termo é a variável e **sourcehost** (tanto quanto **desthost**) funciona como um qualificador. Configuramos que se nosso servidor receber mais do (> 50) cinquenta conexões de um endereço IP (o **sourcehost**): envio mensagem ao *log*, bloqueio de

acesso ao endereço IP de origem por 30 segundos, e vamos para a sub-rotina 20000. O aluno pode consultar o manual HTML que nos referimos para checar as interessantes opções de script que o SINUS oferece. São estas as opções de *script* que podem ser usadas no bloco **notification\_level**:

**MESSAGE:**

*Define a mensagem a ser enviada para o arquivo de registro, e também é usado no e-mail especificado no bloco 'setup'. As mensagens devem ser escritas entre aspas (" ").*

**SYSLOG:**

*Coloca o registro corrente também no 'log' do sistema.*

**REPORT:**

*Copia o registro corrente para o arquivo 'firewall.report'.*

**MAIL:**

*Envia e-mail para os endereços especificados na configuração.*

**SPY:**

*Item muito interessante. Na definição de Muchsel & Schmid a declaração 'spy' inicia um "counter intelligence". Seu objetivo é para ser claro espionar o host e o usuário que está executando uma ação da qual se desconfia. No nosso exemplo, um usuário em um host tentou fazer 50 conexões num certo prazo de tempo em nosso telnet server. Poderíamos ter configurado com 'spy', vejamos:*

```
if connections:sourcehost > 50 then
    spy;                               ⇐
    report;
```

*Isto é, se houver mais de 50 conexões então comece o 'spy'... O 'spy' é um processo de identificação de um host/usuário usando o DNS para obter o nome do host, o protocolo ident (válido somente em*

conexões TCP), *finger* e *rusers*. A coleta dessas informações, tenha ela êxito ou não, vão para o log, como foi configurado com o 'report' (poder-se-ia utilizar também mail, e enviar por aí os resultados do spy).

**RELEVEL:**

Muda o nível de notificação para a regra que causou a ação.

**EXEC:**

Declaração bastante poderosa. Pois pode abrir um shell local e executar um comando. Assim, na certeza de um ataque efetivo podemos desligar a nossa interface de rede (*eth0*, *eth1*, etc.) com a seguinte declaração 'exec':

```
exec "ifconfig eth0 down";
```

**CALL:**

Chama outro nível de execução e o executa.

**LET:**

Esta declaração pode mudar o valor de uma variável. O valor de timeout (em segundos) pode ser especificado no fim da declaração 'let'. Observe o exemplo abaixo:

```
accept icmp icmp_echo to inside notification_level 10;
```



```
notification
```

```
level 10:
```

```
let pingcount:sourcehost := pingcount:sourcehost + 1 timeout 2;  
if pingcount:sourcehost > 100 then  
  block all from sourcehost notification_level 0 timeout 600  
endif;
```

Um máquina tentou um 'ping' para alguma máquina de nossa intranet, o protocolo é ICMP. Ele é usado para diagnósticos e condições de erro numa rede. Mas esta máquina executou o 'ping' 100 vezes (> 100) dentro com o tempo máximo de 2 segundos (timeout 2) a cada 2 'pings'. Portanto, bloqueia-se a conexão por 600 segundos...

**IF (if ... then ... endif):**

*Use como num shell script: para executar um fluxo de ações, se tal ou qual condição acontece.*

O programa **sf Firewall Control Panel** (versão 0.4.0) pode ser usado para a edição do **firewall.conf**. As mesmas advertências feitas nas lições sobre o **Ipchains** e sua edição via **Linuxconf**, valem aqui. Um programa GUI como **sf Control Panel** pode ajudar, mas é preciso saber o que está fazendo, conhecer os conceitos e os fundamentos de um **firewall**, em geral, e da proposta do **SINUS**, em particular. O **sf Control Panel** é um programa feito em **Java**, não o utilizamos no **Windows**, ou em qualquer outra plataforma. De maneira que não podemos fazer uma avaliação de seus resultados.

O arquivo ('**run**') que chama o programa é um script que ajusta o **classpath** **Java** para possibilitar a execução do programa. Faça os ajustes se forem necessários, abre uma sessão **Xterm** e o execute do diretório onde ele foi instalado: '**./run**'. As funções do **sf Control** são basicamente auto-explicativas. O interessante do software é sua capacidade de "auto-configuração", ou seja, você montar uma topologia de rede, editar *host*, o *host* **firewall**, a rede em si, e o acesso a **Internet**. E depois salvar num arquivo e analisá-lo com os posteriores ajustes. Você pode abrir a sua máquina *firewall* pelo **sf Control**, isso se tiver compilado o servidor **SINUS** com o pacote da **ENskip**, podendo estão usar o protocolo **Firewall-to-Firewall**.

As operações com o **Sf Control** começam com a edição da topologia de rede. Para isso se escolhe no menu a opção "**Edit topology**":



Depois é necessário *montar* a topologia propriamente dita, escolhendo as unidades a serem configuradas. Na tela acima está um exemplo da configuração de uma máquina *firewall*. Podemos com um *host* servidor configurá-lo com vários serviços possíveis (FTP, Secure WWW server, SSH server, etc.), liberando ou não para o acesso público. Encerrada a identificação de nossa rede, saímos do 'modo topologia', nesse instante o sf Control avalia se há inconsistências em nossa topologia. Se tudo está ok, podemos escolher o '**autoconfig**'. Depois podemos rever conforme nosso gosto as configurações. Na tela seguinte estamos editando o bloco de *setup* '**rules**':



Rule	Action	Protocol	From	To	Notification	Valid for
Rule 1 autoconf priority	accept	TCP no FTP data conn.	CONFCLIENTS	FW-CONFPORT Port 7227	None	All
Rule 2 autoconf	accept	all protocols	LOCALHOST 127.0.0.1/255.255.25	LOCALHOST 127.0.0.1/255.255.25	None	All
Rule 3 autoconf	accept	all protocols	OWNADDR Own addresses	OWNADDR Own addresses	None	All
Rule 4 autoconf	accept	TCP no FTP data conn.	FW-UNPRIV Ports 1024..65535	SMTP-PORT Port 25	None	All
Rule 5 autoconf	accept	TCP no FTP data conn.	FW-UNPRIV Ports 1024..65535	FINGER-PORT Port 79	None	All
Rule 6 autoconf	reject with tcp reset	TCP		FW-IDENTPORT Port 113	None	All
Rule 7 autoconf	accept	TCP	FW-UNPRIV Ports 1024..65535	IDENT-PORT Port 113	None	All

Agora vamos ver o mecanismo de registro do SINUS Firewall. Sem dúvida um dos pontos fortes do software. Já que a montagem de um *firewall* significa em grande parte *vigilância*. Assim, registrar em arquivos seguros o movimento de entrada e saída de pacotes, e além do mais, graças à capacidade de script do programa, executar ações é fundamental para o administrador de uma rede.

Já vimos que o arquivo de *log* fundamental do SINUS chama-se `'/var/log/firewall'`. Você deve investigá-lo sempre, pois todo o movimento de pacotes é nele gravado, assim como as operações configuradas no `notification_level`. Se a sua rede é de grande porte, e portanto recebe um tráfego muito intenso, é aconselhável usar uma partição separada para o diretório `'/var/log'`. Com efeito, podemos usar o comando `'tail'` para inspecionar o *log* gravado no arquivo (com o acréscimo da flag `-f`, para forçar a leitura do fim do arquivo). Por exemplo:

```
[root@alpha local]# tail -f /var/log/firewall
Feb 02 21:03:02 (s 33) accept TCP 192.168.0.2:1042->192.168.1.1:telnet
Feb 02 21:03:02 Telnet connection request.
Feb 02 21:09:04 (s 38) block TCP 192.168.0.3:1125->192.168.0.1:www
Feb 02 21:09:04 Illegal TCP connection.
Feb 02 21:09:04 sending mail to root
Feb 02 21:09:04 started counter intelligence against host 192.168.0.3
Feb 02 21:09:04 (s 38) block TCP 192.168.0.5:1065->192.168.0.3:auth
Feb 02 21:09:04 Illegal TCP connection.
Feb 02 21:09:04 sending mail to root
Feb 02 21:14:04 (s 38) block TCP 192.168.0.5:1065->192.168.0.1:auth
Feb 02 21:14:04 Illegal TCP connection.
Feb 02 21:14:04 (s 38) block TCP 192.168.0.10:1066->192.168.0.1:finger
Feb 02 21:14:04 Illegal TCP connection.
Feb 02 21:14:04 (s 38) block TCP 192.168.0.12:1066->192.168.0.1:finger
Feb 02 21:14:04 Illegal TCP connection.
Feb 02 21:14:13 (s 38) block TCP 200.255.214.1:1065->192.168.0.1:auth
Feb 02 21:14:13 Illegal TCP connection.
Feb 02 21:14:13 (s 38) block TCP 192.168.0.9:1066->192.168.0.1:finger
Feb 02 21:14:13 Illegal TCP connection.
```

(Segue...)

Temos aqui uma mostra da sequência de entradas registradas, com o mês, o dia, a hora, a advertência ocorrida, endereços IP, protocolos

em questão, etc. Há duas entradas de envio de *mail* para o usuário *root* (linhas 5 e 9). O conhecido leitor de *mails* PINE acusa o recebimento da mensagem:

```
PINE 4.10 MESSAGE TEXT Folder: INBOX Message 2 of 9 ALL
Date: Wed, 2 Feb 2000 21:09:04 -0400
From: firewall@192.168.0.1
Reply-To: nobody@192.168.0.1
Subject: Notification, Feb 02 21:09:04

Illegal TCP connection.

Triggered by: (s 38) block TCP 192.168.0.3:1125->192.168.0.1:www
```

O comando **tail** usado assim serve como uma “visão” instantânea da tráfego de rede, ele ecoará sem parar os logs contidos em **firewall**, até que execute-se um '**Control+C**'. A forma mais produtiva é usar os terminais virtuais, por exemplo **7** e **8**, que sabidamente são alcançados com '**ALT+F7**' (ou use o terminal que desejar...). Para tanto edite o arquivo **inittab**, edite as seguintes linhas:

```
➤ 7:2345:respawn:/usr/bin/tail -f /var/log/firewall > /dev/tty7
```

Você pode também querer “ler” o seu '**/var/log/messages**' num terminal virtual, use o **8** (portanto **ALT+F8** vê-lo):

```
➤ 8:2345:respawn:/usr/bin/tail -f /var/log/messages > /dev/tty8
```

Aconselhamos também o uso de um ótimo script em *Perl*<sup>11</sup> escrito por Michael Babcock (<http://www.linuxsupportline.com/~pgp/linux>) chamado **Logcolorise** (**logcolorise-1.0.7.tar.gz** é a versão que usamos). Ele tem a capacidade de pôr cores nos arquivos de *log*, de maneira que pode acentuar determinadas palavras específicas.

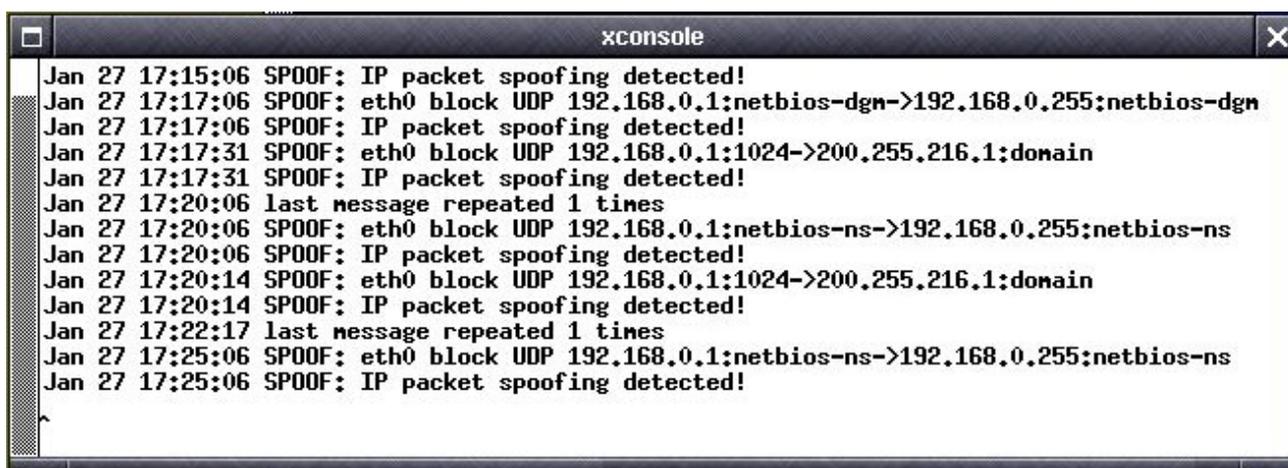
Se você desejar “ver” os *logs* gerados, mas iniciar o seu sistema em '*init 5*', ou mesmo, estiver sempre em ambiente gráfico, e não desejar sair dele, pode usar a o programa **xconsole**, presente na quase totalidade das distribuições Linux. O **xconsole** monitora as mensagens do console no X-Window. Você pode usar no seu '**.xinitrc**' o comando:

```
tail -f /var/log/firewall > /dev/console &
```

E então:

```
xconsole -exitOnFail &
```

Nesta tela vemos um exemplo de execução no **xconsole**:



```
xconsole
Jan 27 17:15:06 SPOOF: IP packet spoofing detected!
Jan 27 17:17:06 SPOOF: eth0 block UDP 192.168.0.1;netbios-dgm->192.168.0.255;netbios-dgm
Jan 27 17:17:06 SPOOF: IP packet spoofing detected!
Jan 27 17:17:31 SPOOF: eth0 block UDP 192.168.0.1;1024->200.255.216.1;domain
Jan 27 17:17:31 SPOOF: IP packet spoofing detected!
Jan 27 17:20:06 last message repeated 1 times
Jan 27 17:20:06 SPOOF: eth0 block UDP 192.168.0.1;netbios-ns->192.168.0.255;netbios-ns
Jan 27 17:20:06 SPOOF: IP packet spoofing detected!
Jan 27 17:20:14 SPOOF: eth0 block UDP 192.168.0.1;1024->200.255.216.1;domain
Jan 27 17:20:14 SPOOF: IP packet spoofing detected!
Jan 27 17:22:17 last message repeated 1 times
Jan 27 17:25:06 SPOOF: eth0 block UDP 192.168.0.1;netbios-ns->192.168.0.255;netbios-ns
Jan 27 17:25:06 SPOOF: IP packet spoofing detected!
```

O daemon do SINUS ('**sfc**') aceita algumas importantes opções na sua linha de comando. São elas:

<sup>11</sup>Portanto, para usá-lo você precisa de Perl em seu sistema (5.x)...

<b>start</b> [config]	Inicia o daemon usando por padrão o arquivo '/etc/firewall.d/firewall', entre com outro arquivo se quiser (exemplo: 'sfc start /usr/local/tmp/firewall.teste')
<b>stop</b>	Desliga o daemon, mas não descarrega o modulo.
<b>reconfig</b> [config] [flush ou flush_all]	Desliga o daemon, e reinicia-o com um novo arquivo de configuração. `reconfig` não paralisa as conexões TCP, ao menos que se use 'flush' ou 'flush_all'. `flush` destrói todas as conexões que não são permitidas na nova configuração, já `flush_all` destrói todas as conexões TCP existentes.
<b>checkconfig</b> [config]	Checa o arquivo de configuração por erros de sintaxe. (Padrão: /etc/firewall.d/firewall.conf).
<b>show</b>	Mostra a configuração usada e as conexões TCP.
<b>showtcp</b>	Mostra apenas as conexões TCP.
<b>kill</b> id+key+criado	Mata conexão TCP especificada.
<b>killall</b>	Mata todas as conexões TCP.
<b>killidle</b> minutos	Mata todas as conexões TCP ociosas por X minutos.
<b>resetusage</b>	'Reset usage counters'.

Estas opções são muito importantes. Sobretudo, na construção do **notification\_level**. Podemos construir interessantes estratégias usando a declaração '**exec**' em conjunto com estas flags. Uma opção deveras importante é a saída possibilitada por '**sfc show**', além de listar nossa configuração no essencial, lista as conexões ativas, mostras suas 'id/keys', que poderemos usar para derrubar conexões que se mostrem perigosas. Eis uma saída do daemon, observe com as informações são facilmente identificadas:

```
[root@alpha /adm]# /usr/local/sfc show
Active rules -- usage counters last reset at Sat Feb  5 20:53:00 2000
-----
```

```

1 (line 13) static -- usage: 0 times
  block, notification level 99
  protocol 6 (TCP)
  to      port 87 (link)
         port 95 (supdup)

2 (line 17) static -- usage: 0 times
  block, notification level 13
  all protocols
  options: `loose source route' `strict source route'

3 (line 23) static -- usage: 0 times
  block, notification level 12
  protocol RIP
  from    192.168.0.11

4 (line 28) static -- usage: 1 time, 3202 bytes
  accept, notification level 4
  protocol 6 (TCP) `ftp_all'
  to      192.168.0.10 port 22 (ssh)

5 (line 29) static -- usage: 0 times
  accept, notification level 1
  protocol 6 (TCP) `ftp_all'
  to      192.168.0.10 port 21 (ftp)

6 (line 33) static -- usage: 1 time, 3279 bytes
  accept, notification level 2
  protocol 6 (TCP) `ftp_all'
  to      192.168.0.2 port 23 (telnet)

7 (line 34) static -- usage: 0 times
  accept, notification level 3
  protocol 6 (TCP) `ftp_all'
  to      192.168.0.3 port 80 (www)

8 (line 35) static -- usage: 4 times, 288 bytes
  accept, notification level 0
  all protocols `ftp_all'
  from    127.0.0.1
  to      127.0.0.1

9 (line 39) static -- usage: 46 times, 2696 bytes
  block, notification level 99
  protocol 6 (TCP)

```

TCP connections at 21:08:21:

```

-----
id/key   created   last use  timeout  state/hosts client          server
-----
00000001 0017d987 0017dc34 00000000 ESTABLISHED 00000000 000000 00000000 000000
          0090 20:54:18 20:54:25 -none-    192.168.0.10:1276->192.168.0.1:telnet
00000002 00180fb4 001811e7 00000000 ESTABLISHED 00000000 000000 00000000 000000
          008f 20:56:36 20:56:42 -none-    192.168.0.10:1023->192.168.0.1:ssh

```

variables:

-----

sr = 0

```

illtcp = 40, timeout Feb 05 21:17:27
  host 192.168.0.15 = 40, timeout Feb 05 21:17:27

```

Poder-se-ia dividir este saída de status do daemon em **dois** grandes blocos. O primeiro é o essencial de nossa configuração. Isto é, as

regras ativas separadas em 9 divisões. Identificando inclusive a linha do **firewall.conf** onde se encontram as regras. Também lista-se o nível de notificação que é chamado, e o número de vezes de acesso e os bytes usados, por exemplo na linha “.4” temos: **'usage: 1 time, 3202 bytes'**.

No segundo bloco encontramos as **'conexões TCP'**. Listam-se todas as conexões ativas (chamadas **'ESTABLISHED'**), o servidor e o cliente, a porta utilizada (usando-se o nome do serviço), e ao final, todas conexões consideradas “ilegais” (**'illtcp'**). A linha de conexões ativas guarda três dados importantes para o uso da flag **'kill'** (deve-se lembrar que esta é uma habilidade que o Ipchains não possui, salvo informação contrária). Com o uso de **'sfc killall'** matamos toda e qualquer conexão TCP ativa em nossa rede, sem discriminação alguma. Com **'sfc killidle XXX\_minutos'** podemos matar qualquer conexão que esteja ocioso por digamos 25 minutos. Mas **'sfc kill id+key+criado'** mas *uma* entre tantas conexões em nossa rede, se uma conexão se mostra perigosa por alguma razão. Mas o **kill** somente funciona com as três informações, como uma medida de preocupação, segundo relato dos próprios desenvolvedores. Como identificá-las? Vejamos:

```
id/key   created  last use  timeout  state/hosts  client          server
-----  -
00000001 0017d987 0017dc34 00000000 ESTABLISHED 00000000 000000 00000000 000000
      0090 20:54:18 20:54:25 -none-    192.168.0.10:1276->192.168.0.1:telnet
```

↑

**00000001 é o ID + 0090 é o KEY + 0017d987 é o CREATED**

Para matarmos tal conexão estabelecida em nossa rede usaríamos então:

✓ ✓ ✓

```
[root@alpha ~]# /usr/local/sbin/sfc kill 00000001 0090 0017d987
Searching for TCP connection 00000001 0090 0017d987...
Connection terminated.
[root@alpha ~]#
```

A opção de *timeout* em conjunto com a possibilidade de derrubar as conexões TCP ativas é uma importante ferramenta do SINUS. Podemos, por exemplo, monitorar as conexões em nosso servidor Web evitando qualquer tentativa de DoS ('*Denial of Service*') através de um SYN-Flooding<sup>12</sup>. Vejamos. Quando uma máquina executa uma conexão ela envia um pacote chamado SYN (de *synchronise*), quando um Servidor Web, - como em nosso exemplo -, recebe nosso pacote SYN, ele responde com um SYN-ACK, um pacote que possui uma sequência numérica de valor randômico. Quando nosso *host* recebe o SYN-ACK, acreditamos no reconhecimento (*acknowledge*) e, então, respondemos com um ACK. Porém, um servidor após enviar um SYN-ACK espera cerca de 40 segundos pelo ACK de resposta, se ele não chega, cancela-se a conexão. E, também, somente oito máquinas podem tentar a conexão com o servidor ao mesmo tempo. O SYN-Flooding consiste em enviar vários pacotes com 'endereços de origem' imaginários sucessivamente. Quando o servidor recebe o SYN, responde, mas como o endereço não existe, não recebe resposta. Espera 40 segundos e cancela a conexão. Se a cada 5 segundos chegam pacotes SYN inexistentes de oito conexões, um servidor praticamente pára de funcionar... Poderíamos esquematizar em três momentos:

**SYN**

①      CLIENTE → SERVIDOR

**SYN-ACK ... 40 segs.**

②      SERVIDOR → CLIENTE

**ACK**

③      CLIENTE → SERVIDOR

---

<sup>12</sup>Flooding em inglês quer dizer inundação...

Um outro típico ataque baseado em SYN-ACK foi descrito no livro Tsutomu Shimomura<sup>13</sup>. No entanto, o objetivo do invasor não era operar simplesmente um **DoS**. Seu intento era penetrar numa rede privada, *fingindo-se* ser um *trusted host*, - como veremos -, usando uma outra técnica: o *IP Spoofing* ("to spoof", ou seja, ludibriar, enganar). Que é, segundo descrição do especialista S. Bellovin<sup>14</sup>:

*“Um dos mais fascinantes buracos na segurança, e que foi primeiramente descrito por Morris. Em suma: usa-se uma predição dos números de sequência TCP para construir uma sequência de pacote TCP sem em qualquer momento ter recebido qualquer resposta do servidor. Isto lhe permite enganar (spoof) um host confiável numa rede local.”*

Para tanto o invasor anulou uma das máquinas (1) com vários SYNs, de maneira que ela não pode responder a um outro *host* (2) confiável. A máquina 2 então recebeu uma série de 20 SYNs, o hacker estudou a sequência numérica usada e respondeu corretamente no lugar da máquina 1, ainda parada. Fazendo-se passar pela máquina 1 da intranet enviou o simples comando '**echo ++ > /.rhosts**', destruindo qualquer controle de acesso ao *host*. A história é assim contada, e de forma ainda mais detalhada pelo autor em seu livro.

Deve-se dizer, por fim, que o aluno não deve confundir a troca de SYN-ACK que ocorre entre um servidor e um cliente, com um *processo de autenticação*. É, isto, sim, uma *sincronização*. O protocolo TCP, diferentemente do UDP que prima pela *velocidade*, é caracterizado pela confiabilidade na entrega dos pacotes, o que, portanto, tem um custo. Dentro de um pacote TCP, além das informações sobre a porta de origem e da destino, de um *checksum*, etc., há algumas flags que garantem a confiabilidade da conexão, SYN e ACK são duas dessas flags ('RST' por exemplo reinicia a conexão, ou FIN que encerra a conexão). Em suma, para que a conexão seja confiável, é necessário que haja critérios para a troca de dados entre dois *host*.

---

<sup>13</sup>*Contra-Ataque. A História da captura do Pirata Cibernético mais procurado dos Estados Unidos*. Ed. Companhia da Letras, 1996, p. 104 ss.

<sup>14</sup>“Security Problems in the TCP/IP Protocol Suite”. *Computer Communication Review*, Vol. 19, nº2, p. 32.



## 3 Monitoração da rede

### 3.1 Os *scanners* de rede

---

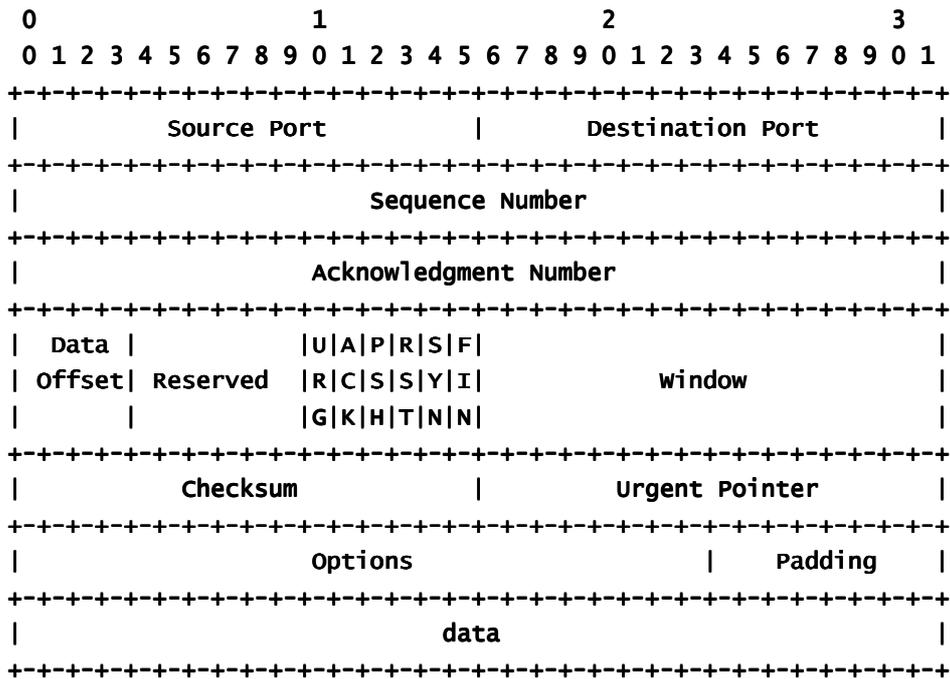
Depois de montado o nosso *firewall* necessitamos de ferramentas, não somente para validá-lo como para a depuração necessária. Assim como para num eventual falha de nossa política detectarmos os intrusos em nossa rede.

O ambiente Linux dispõe de um grande número de ferramentas, veremos assim algumas delas, que às vezes cumprem a mesma função, porém com suas particularidades.

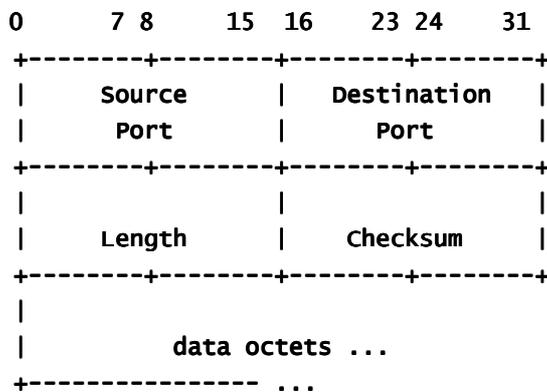
Gostaríamos de voltar ao fim do capítulo anterior. Para tomarmos o exemplo citado no livro de Tsutomu Shimomura. Vimos lá que o invasor pôde através de um *IP-flooding* enganar a rede, e entrar como um *host* confiável. Ele estudou a sequência emitida por um dos *hosts*, e então fez a resposta exata. Como isso foi possível? *A priori*, advertimos que o processo de reconhecimento do SYN-ACK não é uma autenticação, mas sim uma negociação *end-to-end*, entre *hosts*, - que é corretamente chamada de "*tcp handshaking*", o que significa literalmente uma espécie de "aperto de mãos". Muito pelo contrário é baseado numa sequência lógica que iremos entender com facilidade. Os *números sequenciais* não são algo assim como "senhas", não recebem nenhuma espécie de criptografia. Um programa do tipo *sniffer* (de *to sniff*: farejar), pode investigar os dados de um cabeçalho TCP, e executar as manobras necessárias visando a ruptura da segurança.

Precisamos, agora, visualizar primeiramente um cabeçalho TCP, e para uma contraposição um cabeçalho UDP.

Formato de um cabeçalho TCP<sup>15</sup>



Note agora, a diferença para um cabeçalho UDP, bem mais simples...



O motivo principal está na natureza mesma dos protocolos. A complexidade do cabeçalho TCP está em sua necessidade estabelecer a confiabilidade da conexão de dados.

Nos campos iniciais podemos observar a indicação para as portas de origem e de destino (tanto no TCP quanto no UDP, diferentemente de

<sup>15</sup>Fontes: RFC 793 para o cabeçalho TCP e RFC 768 para o UDP.

um cabeçalho IP que traz os *endereços IP*). Três campos no cabeçalho TCP nos importam de imediato: **sequence number** (número sequencial), **acknowledgment number** (número de reconhecimento) e as seis **flags** num total de 6 bits: **URG**, **ACK**, **PSH**, **RST**, **SYN** e **FIN**.. São tais flags que controlam a transmissão de dados de forma confiável. E são aí, igualmente, que se escondem buracos na segurança.

Relembrando: vimos anteriormente que um cliente quando se conecta com um servidor envia pacote, onde no seu cabeçalho, a flag SYN é marcada. Ele solicita uma conexão... Seguem o SYN/ACK do servidor, e o ACK obtendo a conexão, a flag FIN marca o encerramento de envio de dados. Mas quando estes cabeçalhos são trocados, a “negociação” se dá também nos campos que assinalam os 'números sequenciais' e o 'número de reconhecimento'. Já observamos que é uma sequência *lógica*. Ele acontece em incrementos de uma unidade. Observe o seguinte esquema:

```

                SYN
Cliente =====>> Servidor
  Seq. number: (RANDOM) 7
  Ack. number: 0

                SYN/ACK
Cliente <<===== Servidor
  Seq. number: (RANDOM) 9
  Ack. Number: 7 + 1 = 8

                ACK
Client =====>> Servidor
  Seq. number: 8
  Ack. Number: 9 + 1 = 10
```

Assim, o *host* A pede uma sessão Telnet com o *host* B, o servidor. No campo **sequence number** do *host* A que envia o SYN e é escolhido aleatoriamente um número (em nosso exemplo 7...). O campo **acknowledgment number** está obviamente zerado. O *host* B responde com um SYN/ACK, e escolhe aleatoriamente no **sequence number** um número

(9), mas já no campo do número de reconhecimento incrementa 1: soma o **sequence number** do cliente mais 1, resultando 8. No ACK enviado pelo *host* A vemos no **sequence number** 8, isto é, a soma do último **acknowledgment number**, e no *seu* número de reconhecimento está a soma do **sequence number** mais 1. Observe o esquema anterior que esta sequência ficará mais clara. Foi assim que o invasor pode entrar na rede privada, fazendo-se passar por uma máquina confiável.

Vejamos um exemplo real. Três ferramentas são úteis nesse processo de monitoração. Cada um tem suas qualidades, e fica quase no estilo do administrador optar por este ou aquele software. São eles: **Tcpdump** da *Lawrence Berkeley National Laboratory* (<ftp://ftp.ee.lbl.gov>), o **IPTraf** escrito por Gerald Java que estão presentes em quase todas as distribuições Linux, e o **APS**, *Advanced Packet Sniffer* versão 0.13, escrito por Christian Schulte (<http://www.swrtec.de>).

O APS nos possibilita uma radiografia completa do cabeçalho de um pacote TCP ou UDP recebido. Podemos com ele vislumbrar o arranjo de números sequenciais que explicamos. Suas opções são:

```
aps [opções] [-p <da_porta> <para_porta>][-o <tipo>]
```

Onde '**tipo**' evidencia o tipo *frame*, se *smb*, *arp*, *tcp*, *udp*., etc. Nas '**opções**' podemos restringir decisivamente os pacotes a serem mostrados: não mostrar os endereços IP, não mostrar os endereços ethernet, não mostrar tal ou qual faixa de portas, etc. O comando '**aps --help**' lista os parâmetros desejados. Se chamamos o programa sem opções ou restrições ele põe-se simplesmente a monitorar a rede. Vamos realizar como um teste uma simples conexão Telnet, protocolo TCP, porta 23. Aqui o *host* 192.168.0.3 é cliente, e 192.168.0.1 é o servidor. Não há sentido em se mostrar toda a saída

que o *sniffer* nos proporciona, pois é demasiado longa. Congelamos os três momentos significativos. Vejamos:

❶

```
HW-ADDR: 00:10:4b:0b:a9:dc -----> 00:c0:df:e1:15:c4
IP-ADDR: 192.168.0.3 -----> 192.168.0.1
IP-Ver4 || Head:0x0a (bytes) || Service(TOS):0 || Length over all:0060
Fragmentation: ID:0x413b - Flags: 0 1 0 - Offset:00000
TTL:064 || Protokoll:006 (TCP) || HeaderCRC:0x782c
TCP-HEADER:
Ports: 1033-->0023 (telnet) Seq./Ack. Nr.:0x9ab41f84 / 0x00000000 ↵
Data-Offset:0x0a Reserved-6Bit:00 Flags:-urg-ack-psh-rst-SYN-fin- ↵
Window:0x7d78 CRC:0x1729 Urgent-Pointer:0x0000
```

❷

```
HW-ADDR: 00:c0:df:e1:15:c4 -----> 00:10:4b:0b:a9:dc
IP-ADDR: 192.168.0.1 -----> 192.168.0.3
IP-Ver4 || Head:0x0a (bytes) || Service(TOS):0 || Length over all:0060
Fragmentation: ID:0x1c51 - Flags: 0 1 0 - Offset:00000
TTL:064 || Protokoll:006 (TCP) || HeaderCRC:0x9d16
TCP-HEADER:
Ports: 0023-->1033 (telnet) Seq./Ack. Nr.:0xd9408a18 / 0x9ab41f85 ↵
Data-Offset:0x0a Reserved-6Bit:00 Flags:-urg-ACK-psh-rst-SYN-fin- ↵
Window:0x7d78 CRC:0x5017 Urgent-Pointer:0x0000
```

❸

```
HW-ADDR: 00:10:4b:0b:a9:dc -----> 00:c0:df:e1:15:c4
IP-ADDR: 192.168.0.3 -----> 192.168.0.1
IP-Ver4 || Head:0x0a (bytes) || Service(TOS):0 || Length over all:0052
Fragmentation: ID:0x413c - Flags: 0 1 0 - Offset:00000
TTL:064 || Protokoll:006 (TCP) || HeaderCRC:0x7833
TCP-HEADER:
Ports: 1033-->0023 (telnet) Seq./Ack. Nr.:0x9ab41f85 / 0xd9408a19 ↵
Data-Offset:0x08 Reserved-6Bit:00 Flags:-urg-ACK-psh-rst-syn-fin- ↵
Window:0x7d78 CRC:0x7edc Urgent-Pointer:0x0000
```

Assinalamos com a seta as linhas que nos interessam aqui. Note como as flags sucedem-se como foi explicado. Elas são grifadas pelo APS em letras maiúsculas:

No quadro 1 vemos:

Flags:-urg-ack-psh-rst-SYN-fin → Um SYN enviado

No quadro 2 vemos:

**urg-ACK-psh-rst-SYN-fin** → Um SYN-ACK respondido

No quadro 3 vemos:

**urg-ACK-psh-rst-syn-fin** → E finalmente um ACK ...

O aluno deve também notar a sucessão dos *sequence* e *acknowledgments numbers*. Observe como ocorre o incremento que foi explicado anteriormente:

No SYN:

**Seq./Ack. Nr.:0x9ab41f84 / 0x00000000**  
↳ Número de reconhecimento zerado ...

No SYN-ACK:

**Seq./Ack. Nr.:0xd9408a18 / 0x9ab41f85**  
↳ Random ↳ Seq. Nr. Anterior + 1 ...

No Ack:

**Seq./Ack. Nr.:0x9ab41f85 / 0xd9408a19**  
↳ Ack. Nr anterior ↳ Seq. Nr Anterior + 1

O programa APS é sem dúvida um dos mais úteis *sniffers* para o acompanhamento de rede. É claro que programas deste tipo (incluo aqui o **tcpdump**) executam um *dumping* completo e sempre exaustivo do tráfego de rede. Um redirecionamento para um arquivo-*log* numa pequena intranet, durante poucos minutos, gerará arquivos enormes. E não há outra solução para o estudo de monitoramento, e para a montagem de um IDS efetivo, senão o estudo cuidadoso dos registros em texto. Que nos ajudará a encontrar as pegadas deixadas por uma invasão...

O **tcpdump** trabalha como o **APS**, porém seus *layouts* são diferentes. Vejamos então. O programa aceita diversas opções restritivas e inclusivas na linha de comando (consulte a página manual). Sua saída possui o seguinte formato em pacotes TCP:

**End. origem > End. destino: flags seq.number ack.number tamanho\_de\_janela urgent opções**

Já nos pacotes UDP em geral:

End. origem > End. destino: protocolo tamanho

Aqui temos uma saída típica do **tcpdump**. Observamos uma conexão TCP, abrindo uma sessão FTP:

```
18:57:07.436267 192.168.0.3.1026 > 192.168.0.1.ftp: S 1169389928:1169389928(0)
win 32120 <mss 1460,sackOK,timestamp 527641 0,nop,wscale 0> (DF)
18:57:07.437344 arp who-has 192.168.0.3 tell 192.168.0.1
18:57:07.437400 arp reply 192.168.0.3 (0:10:4b:b:a9:dc) is-at 0:10:4b:b:a9:dc
(0:c0:df:e1:15:c4)
18:57:07.437712 192.168.0.1.ftp > 192.168.0.3.1026: S 2223359636:2223359636(0)
ack 1169389929 win 32120 <mss 1460,sackOK,timestamp 71124 527641,nop,wscale 0>
(DF)
18:57:07.437836 192.168.0.3.1026 > 192.168.0.1.ftp: . 1169389929:1169389929(0)
ack 2223359637 win 32120 <nop,nop,timestamp 527641 71124> (DF)
```

Note que a saída do **tcpdump** não assinala explicitamente aos SYN-ACKs. E igualmente no **sequence number** inicial (**ISN**), ele não zera, mas repete os valores. Nesta conexão, assinalada em negrito, temos, é óbvio, a repetição da lógica de sincronização dos pacotes TCP.

O software **IPtraf** traz a comodidade de uma interface feita em *ncurses*. Podemos rolar a tela, por exemplo, para uma visão instantânea do tráfego de rede. Além disso, o **Iptraf** separa o tráfego UDP do TCP. Há diversas outras opções de monitoramento. No menu de abertura temos listadas tais opções:

#### IPtraf 1.2.0

```
*-----*
| IP traffic monitor
| General interface statistics
| Detailed interface statistics
| TCP/UDP service monitor
| Ethernet station monitor
| TCP display filters
| Other protocol filters
| Ethernet host descriptions
| Options
| Exit
*-----*
```

Displays current IP traffic information  
Up/Down-Move selector Enter-execute



## TELA 02

```

IPtraf 1.2.0
* Statistics for eth0 -----*
| Packets      Bytes
| Total: 806   146692
| IP: 806     133434
| TCP: 677    105882
| UDP: 115    26376
| ICMP: 14    1176
| Other IP:   0 0
| Non-IP:    0 0
|
| Activity:    30.60 kbits/sec
|             13.60 packets/sec
|
| IP Checksum Errors: 0
|
| Pkt Size (bytes)      Count Pkt Size (bytes)      Count
| 1 to 100:             343  801 to 900:           18
| 101 to 200:           126  901 to 1000:          1
| 201 to 300:           296 1001 to 1100:          0
| 301 to 400:             5 1101 to 1200:          0
| 401 to 500:            10 1201 to 1300:          1
| 501 to 600:             1 1301 to 1400:          0
| 601 to 700:             0 1401 to 1500:          4
| 701 to 800:             1 1500+:                 0
|
* Elapsed time: 0:01 -----*
X/Ctrl+X-Exit

```

Agora podemos, à título de exemplo, monitorar uma simulação de SYN-flooding numa intranet. E observá-lo em nossos programas, APS e IPtraf. Usaremos um software chamado **Synk4**, o código é escrito por um hacker chamado Zakath. Seus parâmetros são simples:

```
synk4 (end._de_origem) (end._de_destino) porta0 portal
```

Pode-se optar na flag 'end.\_de\_origem' por **0**, o que fará que os SYNs sejam de diferentes e aleatórios endereços IP. '**Porta0**' e '**Portal**', como chamamos, referem-se às portas a serem conseqüentemente geradas; por exemplo: se escolhermos as portas **20** a **500**, mandaremos pacotes da porta **20** até a porta **500**. Os pacotes são assim enviados de forma *intermitente*, até a interrupção pelo usuário com um **Control+C** ou **Control+break**.

Mostramos aqui o IPtraf detectando um SYN-flooding, note como os endereços IP são gerados aleatoriamente, e como as portas são

varridas em sequência (atenção: há também uma conexão Telnet regular sendo mostrada na linha 9...), note também a flag SYN marcada com um **S**:

```

IPTraff
+ Source ----- Destination ----- Packets --- Bytes Flags Iface +
+246.97.200.146:1561 192.168.0.1:101 1 40 S--- eth0 +
+192.168.0.1:101 246.97.200.146:1561 0 0 ---- eth0 +
+6.227.7.183:1561 192.168.0.1:102 1 40 S--- eth0 +
+192.168.0.1:102 6.227.7.183:1561 0 0 ---- eth0 +
+127.12.176.249:1561 192.168.0.1:103 1 40 S--- eth0 +
+192.168.0.1:103 127.12.176.249:1561 0 0 ---- eth0 +
+229.76.20.102:1561 192.168.0.1:104 1 40 S--- eth0 +
+192.168.0.1:104 229.76.20.102:1561 0 0 ---- eth0 +
+192.168.0.1:23 192.168.0.5:1039 > 333 89557 -PA- eth0 +
+192.168.0.5:1039 192.168.0.1:23 > 331 13240 --A- eth0 +
+224.62.23.178:1561 192.168.0.1:105 1 40 S--- eth0 +
+192.168.0.1:105 224.62.23.178:1561 0 0 ---- eth0 +
+ TCP: 3329 entries ----- Active -----
+
+ UDP (78 bytes) from 192.168.0.1:137 to 192.168.0.255:137 on eth0
+ UDP (90 bytes) from 192.168.0.5:137 to 192.168.0.1:137 on eth0
+ ARP (60 bytes) from 00104b0ba9dc to 00c0dfe115c4 on eth0
+ ARP (42 bytes) from 00c0dfe115c4 to 00104b0ba9dc on eth0
+
+ Top ----- Elapsed time: 0:01 -----
IP: 236085 TCP: 235917 UDP: 168 ICMP: 0 Non-IP: 102
Up/Dn/PgUp/PgDn-scr1 actv win W-chg actv win M-more TCP info X/Ctrl+X-Exit

```

Agora, um ataque do mesmo tipo ecoando nesta captura do APS:

(...)

```

HW-ADDR: 00:10:4b:0b:a9:dc -----> 00:c0:df:e1:15:c4
IP-ADDR: 45.240.4.72 -----> 192.168.0.1
IP-Ver4 || Head:0x0a (bytes) || Service(TOS):0 || Length over all:0040
Fragmentation: ID:0x96da - Flags: 0 0 0 - Offset:00000
TTL:030 || Protocol:006 (TCP) || HeaderCRC:0x1315
TCP-HEADER:
Ports: 1307-->0023 (telnet) Seq./Ack. Nr.:0x28376839 / 0x00000000
Data-Offset:0x05 Reserved-6Bit:00 Flags:-urg-ack-psh-rst-SYN-fin-
Window:0xffff CRC:0x275f Urgent-Pointer:0x0000

```

00 00 00 00 00 00

```

HW-ADDR: 00:10:4b:0b:a9:dc -----> 00:c0:df:e1:15:c4
IP-ADDR: 199.27.151.167 -----> 192.168.0.1
IP-Ver4 || Head:0x0a (bytes) || Service(TOS):0 || Length over all:0040
Fragmentation: ID:0x4b0d - Flags: 0 0 0 - Offset:00000
TTL:030 || Protocol:006 (TCP) || HeaderCRC:0x3257
TCP-HEADER:
Ports: 1307-->0024 (unknown) Seq./Ack. Nr.:0x28376839 / 0x00000000
Data-Offset:0x05 Reserved-6Bit:00 Flags:-urg-ack-psh-rst-SYN-fin-
Window:0xffff CRC:0xfad2 Urgent-Pointer:0x0000

```

00 00 00 00 00 00

```

HW-ADDR: 00:10:4b:0b:a9:dc -----> 00:c0:df:e1:15:c4
IP-ADDR: 106.43.186.138 -----> 192.168.0.1
IP-Ver4 || Head:0x0a (bytes) || Service(TOS):0 || Length over all:0040
Fragmentation: ID:0x731e - Flags: 0 0 0 - Offset:00000

```

```

TTL:030 || Protocol:006 (TCP) || HeaderCRC:0x4453
TCP-HEADER:
Ports: 1307-->0025 (smtp) Seq./Ack. Nr.:0x28376839 / 0x00000000
Data-Offset:0x05 Reserved-6Bit:00 Flags:-urg-ack-psh-rst-SYN-fin-
window:0xffff CRC:0x34df Urgent-Pointer:0x0000

00 00 00 00 00 00

HW-ADDR: 00:10:4b:0b:a9:dc -----> 00:c0:df:e1:15:c4
IP-ADDR: 61.54.16.99 -----> 192.168.0.1
IP-Ver4 || Head:0x0a (bytes) || Service(TOS):0 || Length over all:0040
Fragmentation: ID:0xb4b2 - Flags: 0 0 0 - Offset:00000
TTL:030 || Protocol:006 (TCP) || HeaderCRC:0xd9db
TCP-HEADER:
Ports: 1307-->0026 (unknown) Seq./Ack. Nr.:0x28376839 / 0x00000000
Data-Offset:0x05 Reserved-6Bit:00 Flags:-urg-ack-psh-rst-SYN-fin-
window:0xffff CRC:0x0bfb Urgent-Pointer:0x0000

00 00 00 00 00 00

HW-ADDR: 00:10:4b:0b:a9:dc -----> 00:c0:df:e1:15:c4
IP-ADDR: 26.79.212.92 -----> 192.168.0.1
IP-Ver4 || Head:0x0a (bytes) || Service(TOS):0 || Length over all:0040
Fragmentation: ID:0x2b54 - Flags: 0 0 0 - Offset:00000
TTL:030 || Protocol:006 (TCP) || HeaderCRC:0xc227
TCP-HEADER:
Ports: 1307-->0027 (nsw-fe) Seq./Ack. Nr.:0x28376839 / 0x00000000
Data-Offset:0x05 Reserved-6Bit:00 Flags:-urg-ack-psh-rst-SYN-fin-
window:0xffff CRC:0x6ae7 Urgent-Pointer:0x0000

(...)

```

Não poderíamos deixar de mencionar uma ferramenta presente em quase todos os sistemas UNIX, chamada '**netstat**'. Como já demosntra o nome, trata-se um programa para mostrar as estatísticas em nossa rede. Porém, diferentemente dos outros softwares, o **netstat** é de certa forma "estático", pois nos mostra os *estado atual de um interface de rede*, segundo o *protocolo* de nossa escolha. Por exemplo, temos aqui um *SYN-flooding* captado pelo **netstat**:

```

[root@alpha /bin]# netstat -n -t
Active Internet connections (w/o servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp      0      0 127.0.0.1:1032          127.0.0.1:1033         ESTABLISHED
tcp      0      0 127.0.0.1:1033          127.0.0.1:1032         ESTABLISHED
tcp      0      0 127.0.0.1:1030          127.0.0.1:1031         ESTABLISHED
tcp      0      0 127.0.0.1:1031          127.0.0.1:1030         ESTABLISHED
tcp      0      0 127.0.0.1:1028          127.0.0.1:1029         ESTABLISHED
tcp      0      0 127.0.0.1:1029          127.0.0.1:1028         ESTABLISHED
tcp      0      0 127.0.0.1:1026          127.0.0.1:1027         ESTABLISHED
tcp      0      0 127.0.0.1:1027          127.0.0.1:1026         ESTABLISHED
tcp      0      0 127.0.0.1:1024          127.0.0.1:1025         ESTABLISHED
tcp      0      0 127.0.0.1:1025          127.0.0.1:1024         ESTABLISHED
tcp      0      0 192.168.0.1:80          177.166.149.230:1004   SYN_RECV
tcp      0      0 192.168.0.1:80          62.230.207.11:2457    SYN_RECV
tcp      0      0 192.168.0.1:80          45.188.9.136:2314     SYN_RECV
tcp      0      0 192.168.0.1:80          60.232.0.196:1342     SYN_RECV

```

```

tcp      0      0 192.168.0.1:80      81.133.200.206:1676  SYN_RECV
tcp      0      0 192.168.0.1:25      33.92.2.15:2457     SYN_RECV
tcp      0      0 192.168.0.1:25      51.211.45.194:1796  SYN_RECV
tcp      0      0 192.168.0.1:25      121.152.205.144:2314 SYN_RECV
tcp      0      0 192.168.0.1:25      164.10.34.192:1676  SYN_RECV
tcp      0      0 192.168.0.1:25      111.143.44.218:1759 SYN_RECV
tcp      0      0 192.168.0.1:22      30.174.175.2:2457   SYN_RECV
tcp      0      0 192.168.0.1:22      137.239.112.222:1796 SYN_RECV
tcp      0      0 192.168.0.1:22      140.3.60.30:2314    SYN_RECV
tcp      0      0 192.168.0.1:22      71.206.136.178:1342 SYN_RECV
tcp      0      0 192.168.0.1:22      17.43.150.45:1676   SYN_RECV
tcp      0      0 192.168.0.1:22      107.180.175.84:1759 SYN_RECV
tcp      0      0 192.168.0.1:22      173.99.112.246:1004 SYN_RECV
tcp      0      0 192.168.0.1:98      171.171.225.56:1004 SYN_RECV
tcp      0      0 192.168.0.1:98      77.96.75.102:2457   SYN_RECV
tcp      0      0 192.168.0.1:98      91.249.132.72:1676  SYN_RECV
tcp      0      0 192.168.0.1:98      183.203.56.221:1796 SYN_RECV
tcp      0      0 192.168.0.1:98      196.46.169.155:2314 SYN_RECV
tcp      0      0 192.168.0.1:98      203.75.7.61:1342    SYN_RECV
tcp      0      0 192.168.0.1:79      79.182.34.165:1004  SYN_RECV
tcp      0      0 192.168.0.1:79      157.119.64.69:2457  SYN_RECV
tcp      0      0 192.168.0.1:79      64.95.212.209:1796  SYN_RECV
tcp      0      0 192.168.0.1:79      60.199.58.237:1342  SYN_RECV
tcp      0      0 192.168.0.1:79      158.101.240.106:1676 SYN_RECV
tcp      0      0 192.168.0.1:23      215.159.71.120:2457 SYN_RECV
tcp      0      0 192.168.0.1:23      39.125.118.27:2314  SYN_RECV
tcp      0      0 192.168.0.1:23      1.222.77.0:1342     SYN_RECV
tcp      0      0 192.168.0.1:23      50.203.24.189:1676  SYN_RECV
tcp      0      0 192.168.0.1:23      202.224.254.255:1759 SYN_RECV
tcp      0      0 192.168.0.1:23      112.62.198.114:1004 SYN_RECV

```

✓IPs 'falsos'...      ✓SYNS...

Não se deve estranhar a expressão usada “*SYN\_RECV*”. O **netstat**, como falamos, é uma ferramenta tradicional dos Sistemas operacionais UNIX, por isso usa a expressão canônica presente no RFC 793. Ali uma conexão TCP foi interpretada como uma progressão de **estados** com diferentes significados, onde SYN-RECEIVED é um deles, tais como LISTEN (espera por uma conexão), ESTABLISHED (estabelecimento da conexão TCP), FIN-WAIT (espera pelo término da conexão), etc.

Entretanto, a análise de tais ferramentas de monitoração, não nos respondem seguramente *uma* questão de um ataque IP *spoofing*, um ponto permanece aberto. Qual seja, como o invasor pode enviar um pacote, com um endereço de um *host* confiável, e além do mais, respondendo a um SYN-ACK? Como *montar* um *sequence number* ou um *ack number*?

Um software aberto chamado **SendIP** (versão 0.0.1) de Mike Richards (mike@earth.li), - criado dentro do chamado *Project Purple* (<http://www.earth.li>) -, por exemplo, pode fazer tal operação, sendo

uma importantíssima ferramenta de *debug* e administração de rede e firewall.

O **SendIP** aceita inúmeras opções na linha de comando. Cobrindo quase todas as opções existentes nos protocolos UDP, TCP e ICMP. Vejamos:

**sendip <hostname> -p <tipo> -d <dados> <opções>**

**<hostname>** is the name of the host to send the packet to, defaults to localhost

**<type>** is the packet type: currently UDP, TCP or ICMP, defaults to IP

**<data>** is the data payload of the packet as a string of hexadecimal numbers, defaults to 10 bytes of 0s

OPÇÃO	DESCRIÇÃO	PADRÃO
<b>Opções de soquete:</b>		
-sd	Debug	0
-sr	Não rotear	0 (ie do route)
-sb	Permite <i>broadcast</i>	0
<b>Opções IP:</b>		
-is	IP de origem	127.0.0.1
-id	IP de destino	Correto
-ih	Tamanho de cabeçalho	Correto
-iy	'Type of service'	0 (ver RFC791)
-il	Tamanho	Correto
-ii	Identificação	Aleatório
-ifr	Flag reservada	0
-ifd	Não fragmenta	0
-ifm	Mais fragmentos	0
-if	'Frag offset'	0
-it	TTL	255
-ip	Protocolo	Correto para o tipo
-ic	'Checksum'	Correto
<b>Opções ICMP:</b>		
-ct	Tipo	8 (Ver RFC792)
-cd	Código	0 (Ver RFC792)
-cc	'Checksum'	Correto
<b>Opções UDP:</b>		
-us	Porta de Origem	0
-ud	Porta de Destino	0
-ul	Tamanho	Correto
-uc	'Checksum'	Correto
<b>Opções TCP:</b>		
-ts	Porta de Origem	0
-td	Porta de Destino	0
-tn	'Sequence number'	Aleatório

```

-ta      'Acknowledgement'      0
-tt      'Data offset'          Correto
-tr      Reservado              0
-tfu     Conf. 'urg bit'        0 a menos que -tu seja especificado
-tfa     Conf. 'ack bit'        0 a menos que -ta seja especificado
-tfp     Conf. 'psh bit'        0
-tfr     Conf. 'rst bit'        0
-tfs     Conf. 'syn bit'        1
-tff     Conf. 'fin bit'        0
-tw      Janela                 65535
-tc      'Checksum'            Correto
-tu      Indicador/Urgência     0
    
```

Podemos agora enviar um pacote TCP, a nossa escolha, para um *host* qualquer de nossa rede, e então analisar com o APS e o IPtraf, o conteúdo do pacote. Faremos com as seguintes opções:

```

sendip 192.168.0.1 -p tcp -is 200.123.120.0 -ts 100 -td 22
↳ (cont.) -ta 2345 -tu 1
    
```

O primeiro endereço IP é o *host* que desejamos enviar o pacote, o protocolo TCP, o endereço de um servidor que usaremos para *spoofing* (a máquina que esta enviando é originalmente 192.168.0.3). Seguem-se as opções típicas do protocolo: '-ts' porta de origem valor 100, '-td' porta destino 22 (ssh), '-ta' o número de reconhecimento forjado: '2345' em decimais, e, por fim, marcamos e, então ativamos, com '1' a flag URG do cabeçalho.

Com o APS monitorando nosso *host*, ele nos reporta:

```

AdvancedPacketSniffer V0.13      (CONFIG-DATE:16:58:49-02/13/00)
    
```

```

Copyright (C) 1999 Christian Schulte (dglns@saturn2.franken.de)
    
```

```

This program is free software; you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation.
    
```

```

HW-ADDR: 00:10:4b:0b:a9:dc -----> 00:c0:df:e1:15:c4
IP-ADDR: 200.123.120.0 -----> 192.168.0.1
IP-Ver4  || Head:0x0a (bytes)  || Service(TOS):0  || Length over all:0050
Fragmentation:  ID:0x6745 - Flags: 0 0 0 - Offset:00000
TTL:255  || Protokoll:006 (TCP)  || HeaderCRC:0x505b
    
```

**TCP-HEADER:**

Ports: 0100-->0022 (newacct) Seq./Ack. Nr.:0xc6237b32 / 0x00000929  
 Data-Offset:0x05 Reserved-6Bit:00 Flags:-URG-ACK-psh-rst-SYN-fin-  
 Window:0xffff CRC:0x2133 Urgent-Pointer:0x0001

00 00 00 00 00 00 00 00 00 00

HW-ADDR: 00:10:4b:0b:a9:dc -----> 00:c0:df:e1:15:c4  
 ARP: IP->Ethernet  
 Request from 192.168.0 .3 (HW:00:10:4b:0b:a9:dc)  
 Goes to 192.168.0 .1 (HW:00:00:00:00:00:00)

01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01

HW-ADDR: 00:c0:df:e1:15:c4 -----> 00:10:4b:0b:a9:dc  
 ARP: IP->Ethernet  
 Answer from 192.168.0 .1 (HW:00:c0:df:e1:15:c4)  
 Goes to 192.168.0 .3 (HW:00:10:4b:0b:a9:dc)

(...)

Agora, no IPtraf podemos igualmente observar:



```

IPtraf 1.2.0
* Source ----- Destination ----- Packets --- Bytes Flags Iface *
|-200.123.120.0:100 192.168.0.1:22 > 1 50 S-AU eth0
|-192.168.0.1:22 200.123.120.0:100 > 0 0 ---- eth0
* TCP: 1 entries ----- Active ---*
*Bottom-----Elapsed time: 51:03-----*
IP: 645231100 TCP: 3909080 UDP: 234992 ICMP: 12 Non-IP: 0
Up/Dn/PgUp/PgDn-scr| actv win w-chg actv win M-more TCP info X/Ctrl+X-Exit
    
```

Aqui observamos, em ambas as telas, um único pacote, e exatamente com os valores que criamos. Os campos em destaque nos mostram isso. A leitura do cabeçalho nos mostra facilmente os bits de urgência marcados, e o SYN-ACK do pacote, também (o APS) pode nos mostrar o número de reconhecimento que estabelecemos ('2345'), expresso em hexadecimais ('0x00000929').

Outro detalhe a ser notado é a requisição ARP (*Address Resolution Protocol*). Os pacotes ARPs não são extremamente importantes em termos de segurança. Pois estamos num nível mais baixo da pilha TCP/IP. Mas, por outro lado, pode revelar alguns traços importantes para o monitoramento da rede. Vejamos. ARP é um protocolo que liga um endereço ethernet, chamado comumente de *Media Access Control* (MAC) ou endereço de hardware (ethernet), a um endereço IP atribuído. Segundo a definição precisa de Douglas Comber:

*“ARP é um protocolo de baixo nível que abstrai o endereçamento físico, permitindo que seja atribuído um endereço IP para cada máquina. Idealizamos ARP como parte física do sistema de redes, e não como parte do protocolo de interligação em redes.”<sup>16</sup>*

Quando um *host* da rede deseja saber o MAC de um outro *host*, ele envia uma requisição ou um pacote ARP por *broadcast*, perguntando “qual o endereço de hardware da máquina com IP, por exemplo, '192.168.0.1'?”. Todas as máquinas aceitam o pacote ARP, mas como só há uma máquina com tal endereçamento, somente ela responde, enviado seu MAC. Depois é formado um *cache* com uma listagem com os endereços Ips e os endereços de hardware. O comando “**arp -nv**” pode mostrar o *cache* criado:

```
[root@beta /bin]# arp -nv
Address          Hwtype  Hwaddress          Flags  Mask              Iface
192.168.0.1      ether   00:FF:46:13:00:F7  C      C                  eth0
192.168.0.11     ether   00:FF:46:56:01:FF  C      C                  eth0
(...)
```

O MAC é um endereço único. Ou seja, cada adaptador de rede possui um endereço singular no formato de 48 bits (6 bytes), escrito em hexadecimais e separados por “:” ou “-”. Os três primeiros bytes

---

<sup>16</sup>COMBER, D. *Interligação em Rede com TCP/IP*. Rio de Janeiro, ed. Campus, 1998, p. 84.

designam o código do fabricante da placa, determinado pela IEEE (*Institute of Electrical and Electronics Engineers*). O MAC de sua placa de rede pode ser mostrado com o comando `'ifconfig'`; na primeira da linha da interface ethernet, pode-se observar a entrada `"HWaddr X:X:X:X:X:"`. Em nosso exemplo encontramos uma placa da empresa 3COM, seu MAC é:

**00:10:4B:0B:A9:DC**

↳ *cód. da empresa 3COM*

Ainda em nosso exemplo, vimos que a máquina que enviou pacotes *spoofing* deixará com certeza rastros. A requisição ARP está expressa no APS nas seguintes linhas:

```
HW-ADDR: 00:10:4b:0b:a9:dc -----> 00:c0:df:e1:15:c4
ARP: IP->Ethernet
Request from 192.168.0 .3 (HW:00:10:4b:0b:a9:dc)
Goes to 192.168.0 .1 (HW:00:00:00:00:00:00)
↳ endereço ainda desconhecido...
```

```
01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01
```

```
HW-ADDR: 00:c0:df:e1:15:c4 -----> 00:10:4b:0b:a9:dc
ARP: IP->Ethernet
Answer from 192.168.0 .1 (HW:00:c0:df:e1:15:c4)
Goes to 192.168.0 .3 (HW:00:10:4b:0b:a9:dc)
↳ requisição completada!
```

O pacote *spoofing* foi enviado pelo host 192.168.0.3, mas mesma a máquina que enviou o pacote "artificial", imediatamente envia um pacote ARP... E a outra máquina responde casando o endereço IP com o MAC.

---

### ***3.2 Saint: Security Administrator's Integrated Network Tool***

---

Vamos agora analisar a ferramenta **Saint (Security Administrator's Integrated Network Tool)** versão 1.4 release 17, um software aberto e

muito poderoso, desenvolvido pela *World Wide Digital Security* (<http://www.wwdsi.com/saint>). Ele funciona em quase todos os sistemas UNIX, - para sua utilização necessita da linguagem Perl versão 5.005 ou superior; e por rodar dentro de um *browser*, necessita evidentemente de qualquer tipo de *browser* para Web (até mesmo os em *modo texto*, tal como o **Lynx**, podem ser usados). O Saint é uma evolução da conhecida ferramenta **Satan** (escrita por D. Farmer & W. Venema), e possui mesmo grandes porções de *copyright* deste software. Mas, sem dúvida, o Saint está bem melhor adaptado às distribuições Linux. Quem tentou compilar o Satan no Linux 2.0 ou 2.2x, estará de acordo conosco.

Nem todas as distribuições Linux, incluem uma versão do Saint. A exceção talvez seja a da SuSE (v.6.3), que já distribui o pacote compilado. Para fazê-lo execute os seguintes passos:

1. Digite '**make linux**' para a compilação dos binários.
2. Depois ajuste os *paths* para os scripts em Perl com '**perl reconfig**', com isso também o Saint encontrará a ajustará o *path* para o browser disponível no sistema se desejar alterá-lo edite o arquivo '**config/paths.pl**').
3. Você necessita desligar o servidor proxy quando for usar o Saint, pois ele precisa de acesso *direto* aos *hosts* da rede. Você pode fazer editando a sua variável de ambiente (algo do tipo **\$http\_proxy**), e também desfazendo o uso do proxy no netscape. Sem esse ajuste o Saint não funcionará!

Agora já podemos lançar o Saint. Se usarmos um browser gráfico, como o netscape ou o NSCA mosaic, precisamos estar no X-Windows. De uma janela do terminal gráfico (o Xterm, por exemplo), carregamos o programa a partir do seu diretório:

**cd /usr/local/saint**

e então:

**./saint**

A tela inicial mostra um menu de funções do programa. O Saint não é apenas um software de gerenciamento de rede. Mas sim, um programa de análise e relatório de diagnósticos de nossa rede e seus *hosts*. O Saint procura vulnerabilidades e, então, as reporta, formatando-as numa página Html.

Vejamos o seu menu de abertura:

DATA MANAGEMENT ⇒ configura o arquivo (o padrão é: **/var/saint/saint-data**), onde o programa irá registrar suas análises e resultados; o outro item *"Merge with existing SAINT data base"* é usado para fundir dois arquivos de dados produzidos pelo Saint.

TARGET SELECTION ⇒ configura o *host* ou os *hosts* de uma sub-rede primária. Aqui serão escolhidas as funções básicas do programa que iremos detalhar à frente.

DATA ANALYSIS ⇒ mostra os relatórios e análises depois da investigação de um *host*.

CONFIGURATION MGT (management) ⇒ É uma página que gerencia as configurações gerais do Saint, editando o arquivo '**config/saint.cf**'. Tais como o arquivos de dados, o 'nível de pesquisa' (*level probe*), o valor de *timeout*, quais sufixos iremos excluir da pesquisa (*mil, gov*, por exemplo), se usamos um DNS, etc.

DOCUMENTATION e TROUBLESHOOTING ⇒ são *links* para a documentação armazenada localmente.

A interface construída como uma página Web é bastante auto-explicativa. Começaremos com a escolha de um *host* de nossa rede, e seguiremos os passos do programa. Escolhemos no menu: “**target selection**”, e preenchemos com o endereço do *host*. Podemos “varrer” toda a nossa rede também, - se estivermos numa máquina com bastante recurso de memória. Ou podemos escolher a execução de um **Denial Of Service**. Abaixo encontramos a seleção do “**scanning level**”: leve, normal, pesado (evitando *portas* problemáticas do Windows NT) e o último nível o ‘**heavy+**’ (não evitando as *portas* problemáticas do Windows NT). Na escolha final, habilitamos ou não o *firewall* na máquina a ser investigada.

Feitos os ajuste, iniciamos a investigação: “**Start the scan**”.

Assim o Saint começa sua varredura no *host*, e a subsequente coleta de dados, dependendo do tráfego de rede, e mesmo do hardware do equipamento, pode-se levar um certo tempo até finalizar a varredura. A página de varredura irá mostrar as *portas* e os serviços encontrados, até anunciar o fim da tarefa: “...**Saint run completed**”. Vemos aqui o processo de varredura do Saint num *host*:

.....  
**Data collection in progress...**

```
10:32:45 bin/timeout 30 bin/ping 192.168.0.5
10:32:45 bin/timeout 20 bin/finger.saint 192.168.0.5
10:32:46 bin/timeout 20 bin/ostype.saint 192.168.0.5
10:33:06 bin/timeout 20 bin/dns.saint 192.168.0.5
10:33:07 bin/timeout 20 bin/rpc.saint 192.168.0.5
10:33:07 bin/timeout 45 bin/udpscan.saint 1-1760,1763-2050,31337,32767-33500 192.168.0.5
10:33:42 bin/timeout 45 bin/tcpscan.saint 1-1525,1527-9999,12345 192.168.0.5
10:34:04 bin/timeout 20 bin/smb.saint 192.168.0.5
10:34:24 bin/timeout 20 bin/showmount.saint 192.168.0.5
10:34:24 bin/timeout 60 bin/nfs-chk.saint -t 10 192.168.0.5
10:34:25 SAINT run completed
```

**Data collection completed (1 host(s) visited).**

.....

[Back to the SAINT start page](#) | [Continue with report and analysis](#)

Ele nos mostra todos os serviços e suas respectivas portas... É um mapa completo das portas que estão *abertas* num *host* de nossa rede, resta-nos checar se este estado de coisas é correto ou não.

Escolhemos agora a opção **“Continue with report and analysis”**. Com essa opção o Saint irá analisar o quadro encontrado na máquina 192.168.0.5.

A tela baixo mostra as opções oferecidas pelo software. São três blocos de opções: 'Vulnerabilidades', 'Informações sobre o *host*' e, por fim, uma listagem de '*Trusted hosts*'. Sem dúvida, o primeiro bloco é o mais importante do ponto de vista da segurança. O segundo e o terceiro são mais *informativos*.

## SAINT Reporting and Analysis

---

### Table of contents

#### Vulnerabilities

- [By Approximate Danger Level](#)
- [By Type of Vulnerability](#)
- [By Vulnerability Count](#)

#### Host Information

- [By Class of Service](#)
- [By System Type](#)
- [By Internet Domain](#)
- [By Subnet](#)
- [By Host Name](#)

#### Trust

- [Trusted Hosts](#)
- [Trusting Hosts](#)

É, por conseguinte, no nível **I** ('**By Approximate Danger Level**') das "Vulnerabilidades", que o Saint irá fazer primeiramente uma análise muita rica dos problemas potenciais encontrados, - se ele porventura os encontrar. Aqui, como já anuncia o *menu*, ele lista os problemas segundo o nível de *perigo aproximado*. Os problemas podem ser os mais diversos possíveis. Ele fará uma procura exaustiva, e colocará uma pequena *bola vermelha* para assinalar a existência de problemas críticos, que possam ser encontrados. Ao mesmo tempo, são montados interessantes *links* para explicações bastante proveitosas, - que estão em arquivos em diretórios locais.

No item abaixo, nível **II**, ("**By Type of Vulnerability**"), encontramos uma lista dos problemas encontrados segundo o "tipo de *vulnerabilidade*". Como mostra a tela abaixo:

## Vulnerabilities – By Type

---

### Number of hosts per vulnerability type.

- [Possible DoS \(fraggle\) problem - 1 host\(s\)](#)
- [unrestricted NFS export - 1 host\(s\)](#)
- [Windows detected - 1 host\(s\)](#)

**Note: hosts may appear in multiple categories.**

---

[Back to the SAINT start page](#) | [Back to SAINT Reporting and Analysis](#)

O *host* em questão é uma máquina rodando o Windows NT 4.0 (workstation). O Saint aponta três *links*:

- 1. Adverte para problemas de DoS característicos de sistemas Windows.**
- 2. O arquivo export do NFS rodando no Windows NT não impõe restrições para diretórios e arquivos.**
- 3. Ele, imaginem, detecta e lista o próprio Windows como um problema...**

O *link* 1 e 2 basicamente advertem para o uso de *patches* para corrigir problemas de DoS no ambiente Windows.

Assim se dermos um clique em '**Possible DoS problem**' ou '**Windows detected**', encontraremos uma tela com o(s) endereço(s) do(s) *host(s)*, e se seguimos então o *link* encontraremos o seguinte:

## Results – 192.168.0.5

---

### General host information:

- Host type: [Windows](#)
- NB Name:
- Subnet [192.168.0](#)
- Scanning level: heavy
- Last scan: Tue Feb 29 10:42:24 2000

### Network Services:

- [NFS server](#)

### Vulnerable Services:

- [Is your host a DoS threat?](#)
- [Is your Windows patched for DoS?](#)
- [Exports /drivec to everyone](#)

A bola vermelha marca um problema crítico, a saber, o diretório `'/drivec'` da máquina NT pode ser montado por qualquer usuário indiscriminadamente! Feita a advertência resta ao administrador da rede tomar as medidas necessárias (uma dica útil aqui é conferir o arquivo `'rules/drop'`, ali podemos configurar “fatos” a serem *negligenciados* pelo Saint. Exemplo: é bastante comum nas redes que usam NFS liberarem a montagem de unidades de CD-ROM nos seus arquivos `/etc/exports`, podemos configurar então para que o Saint não reporte tal fato como um *problema*).

As *bolas marrons* (`'Is your host a DoS threat?'` e o seguinte...), como dissemos anteriormente, esclarecem-nos da necessidade de correções contra as ações DoS.

---

---

# ANEXOS

---

**Anexo A:** O arquivo '/etc/protocols'

**Anexo B:** O arquivo '/etc/services'

**Anexo C:** Um 'script' para IPCHAINS

**Anexo D:** Os cabeçalhos de pacotes IP, ICMP, TCP, UDP e ARP

**Anexo E:** Como um pacote viaja através da pilha TCP/IP?

## ANEXO A:

```
#
# protocols This file describes the various protocols that are
#           available from the TCP/IP subsystem. It should be
#           consulted instead of using the numbers in the ARPA
#           include files, or, worse, just guessing them.
#

ip      0      IP      # internet protocol, pseudo protocol number
icmp    1      ICMP    # internet control message protocol
igmp    2      IGMP    # internet group multicast protocol
ggp     3      GGP     # gateway-gateway protocol
tcp     6      TCP     # transmission control protocol
pup     12     PUP     # PARC universal packet protocol
udp     17     UDP     # user datagram protocol
idp     22     IDP     # WhatsThis?
raw     255    RAW     # RAW IP interface

# End.
```

**ANEXO B:**

```

#
# Network services, Internet style
#
# Note that it is presently the policy of IANA to assign a single well-known
# port number for both TCP and UDP; hence, most entries here have two entries
# even if the protocol doesn't support UDP operations.
#
# This list could be found on:
#     http://www.isi.edu/in-notes/iana/assignments/port-numbers
#
# Not all ports are included, only the more common ones.
#
#           0/tcp      Reserved
#           0/udp      Reserved
tcpmux     1/tcp          # TCP Port Service Multiplexer
tcpmux     1/udp          # TCP Port Service Multiplexer
compressnet 2/tcp          # Management Utility
compressnet 2/udp          # Management Utility
compressnet 3/tcp          # Compression Process
compressnet 3/udp          # Compression Process
rje        5/tcp          # Remote Job Entry
rje        5/udp          # Remote Job Entry
echo       7/tcp          Echo
echo       7/udp          Echo
discard    9/tcp          Discard sink null
discard    9/udp          Discard sink null
systat     11/tcp         users          # Active Users
systat     11/udp         users          # Active Users
daytime    13/tcp         Daytime       # (RFC 867)
daytime    13/udp         Daytime       # (RFC 867)
netstat    15/tcp          # Unassigned [was netstat]
qotd       17/tcp         quote         # Quote of the Day
qotd       17/udp         quote         # Quote of the Day
msp        18/tcp          # Message Send Protocol
msp        18/udp          # Message Send Protocol
chargen    19/tcp         ttytst source # Character Generator
chargen    19/udp         ttytst source # Character Generator
ftp-data   20/tcp          # File Transfer [Default Data]
ftp-data   20/udp          # File Transfer [Default Data]
ftp        21/tcp          # File Transfer [Control]
fsp        21/udp
ssh        22/tcp          # SSH Remote Login Protocol
ssh        22/udp          # SSH Remote Login Protocol
telnet     23/tcp         Telnet
telnet     23/udp         Telnet
#          24/tcp         any private mail system
#          24/udp         any private mail system
smtp       25/tcp         mail          # Simple Mail Transfer
smtp       25/udp         mail          # Simple Mail Transfer
nsw-fe     27/tcp          # NSW User System FE
nsw-fe     27/udp          # NSW User System FE
msg-icp    29/tcp          # MSG ICP
msg-icp    29/udp          # MSG ICP

```

```

msg-auth      31/tcp      # MSG Authentication
msg-auth      31/udp      # MSG Authentication
dsp           33/tcp      # Display Support Protocol
dsp           33/udp      # Display Support Protocol
#            35/tcp      any private printer server
#            35/udp      any private printer server
time         37/tcp      # Time timeserver
time         37/udp      # Time timeserver
rap          38/tcp      # Route Access Protocol
rap          38/udp      # Route Access Protocol
rlp          39/tcp      resource      # Resource Location Protocol
rlp          39/udp      resource      # Resource Location Protocol
graphics     41/tcp      Graphics
graphics     41/udp      Graphics
name         42/tcp      # Host Name Server
name         42/udp      # Host Name Server
nameserver   42/tcp      # Host Name Server
nameserver   42/udp      # Host Name Server
nicname      43/tcp      # Who Is
nicname      43/udp      # Who Is
whois        43/tcp      # Who Is
whois        43/udp      # Who Is
mpm-flags    44/tcp      # MPM FLAGS Protocol
mpm-flags    44/udp      # MPM FLAGS Protocol
mpm          45/tcp      # Message Processing Module [recv]
mpm          45/udp      # Message Processing Module [recv]
mpm-snd      46/tcp      # MPM [default send]
mpm-snd      46/udp      # MPM [default send]
ni-ftp       47/tcp      # NI FTP
ni-ftp       47/udp      # NI FTP
auditd       48/tcp      # Digital Audit Daemon
auditd       48/udp      # Digital Audit Daemon
tacacs       49/tcp      # Login Host Protocol (TACACS)
tacacs       49/udp      # Login Host Protocol (TACACS)
re-mail-ck   50/tcp      # Remote Mail Checking Protocol
re-mail-ck   50/udp      # Remote Mail Checking Protocol
la-maint     51/tcp      # IMP Logical Address Maintenance
la-maint     51/udp      # IMP Logical Address Maintenance
xns-time     52/tcp      # XNS Time Protocol
xns-time     52/udp      # XNS Time Protocol
domain       53/tcp      # Domain Name Server
domain       53/udp      # Domain Name Server
xns-ch       54/tcp      # XNS Clearinghouse
xns-ch       54/udp      # XNS Clearinghouse
isi-gl       55/tcp      # ISI Graphics Language
isi-gl       55/udp      # ISI Graphics Language
xns-auth     56/tcp      # XNS Authentication
xns-auth     56/udp      # XNS Authentication
#           57/tcp      any private terminal access
#           57/udp      any private terminal access
xns-mail     58/tcp      # XNS Mail
xns-mail     58/udp      # XNS Mail
#           59/tcp      any private file service
#           59/udp      any private file service
ni-mail      61/tcp      # NI MAIL
ni-mail      61/udp      # NI MAIL
acas         62/tcp      # ACA Services
acas         62/udp      # ACA Services

```

```

whois++      63/tcp      whois++
whois++      63/udp      whois++
covia        64/tcp      # Communications Integrator (CI)
covia        64/udp      # Communications Integrator (CI)
tacacs-ds    65/tcp      # TACACS-Database Service
tacacs-ds    65/udp      # TACACS-Database Service
sql*net      66/tcp      # Oracle SQL*NET
sql*net      66/udp      # Oracle SQL*NET
bootps       67/tcp      # Bootstrap Protocol Server
bootps       67/udp      # Bootstrap Protocol Server
bootpc       68/tcp      # Bootstrap Protocol Client
bootpc       68/udp      # Bootstrap Protocol Client
tftp         69/tcp      # Trivial File Transfer
tftp         69/udp      # Trivial File Transfer
gopher       70/tcp      # Gopher
gopher       70/udp      # Gopher
netrjs-1     71/tcp      # Remote Job Service
netrjs-1     71/udp      # Remote Job Service
netrjs-2     72/tcp      # Remote Job Service
netrjs-2     72/udp      # Remote Job Service
netrjs-3     73/tcp      # Remote Job Service
netrjs-3     73/udp      # Remote Job Service
netrjs-4     74/tcp      # Remote Job Service
netrjs-4     74/udp      # Remote Job Service
#            75/tcp      any private dial out service
#            75/udp      any private dial out service
deos         76/tcp      # Distributed External Object Store
deos         76/udp      # Distributed External Object Store
#            77/tcp      any private RJE service
#            77/udp      any private RJE service
vettcp       78/tcp      vettcp
vettcp       78/udp      vettcp
finger       79/tcp      Finger
finger       79/udp      Finger
http         80/tcp      # World Wide Web HTTP
http         80/udp      # World Wide Web HTTP
www          80/tcp      # World Wide Web HTTP
www          80/udp      # World Wide Web HTTP
www-http     80/tcp      # World Wide Web HTTP
www-http     80/udp      # World Wide Web HTTP
hosts2-ns    81/tcp      # HOSTS2 Name Server
hosts2-ns    81/udp      # HOSTS2 Name Server
xfer         82/tcp      # XFER Utility
xfer         82/udp      # XFER Utility
mit-ml-dev   83/tcp      # MIT ML Device
mit-ml-dev   83/udp      # MIT ML Device
ctf          84/tcp      # Common Trace Facility
ctf          84/udp      # Common Trace Facility
mit-ml-dev   85/tcp      # MIT ML Device
mit-ml-dev   85/udp      # MIT ML Device
mfcobol      86/tcp      # Micro Focus Cobol
mfcobol      86/udp      # Micro Focus Cobol
#            87/tcp      any private terminal link
#            87/udp      any private terminal link
kerberos     88/tcp      Kerberos
kerberos     88/udp      Kerberos
su-mit-tg    89/tcp      # SU/MIT Telnet Gateway
su-mit-tg    89/udp      # SU/MIT Telnet Gateway

```

```

dnsix          90/tcp          # DNSIX Securit Attribute Token Map
dnsix          90/udp          # DNSIX Securit Attribute Token Map
mit-dov       91/tcp          # MIT Dover Spooler
mit-dov       91/udp          # MIT Dover Spooler
npp           92/tcp          # Network Printing Protocol
npp           92/udp          # Network Printing Protocol
dcp           93/tcp          # Device Control Protocol
dcp           93/udp          # Device Control Protocol
objcall       94/tcp          # Tivoli Object Dispatcher
objcall       94/udp          # Tivoli Object Dispatcher
supdup        95/tcp          # SUPDUP
supdup        95/udp          # SUPDUP
dixie         96/tcp          # DIXIE Protocol Specification
dixie         96/udp          # DIXIE Protocol Specification
swift-rvf     97/tcp          # Swift Remote Virtual File Protocol
swift-rvf     97/udp          # Swift Remote Virtual File Protocol
tacnews       98/tcp          # TAC News
tacnews       98/udp          # TAC News
metagram      99/tcp          # Metagram Relay
metagram      99/udp          # Metagram Relay
newacct       100/tcp         # [unauthorized use]

hostnames     101/tcp
iso-tsap      102/tcp
csnet-ns      105/tcp
csnet-ns      105/udp
rtelnet       107/tcp
rtelnet       107/udp
pop2          109/tcp
pop2          109/udp
pop3          110/tcp
pop3          110/udp
sunrpc        111/tcp
sunrpc        111/udp
auth          113/tcp
sftp          115/tcp
uucp-path     117/tcp
nntp          119/tcp
ntp           123/tcp
ntp           123/udp
netbios-ns    137/tcp
netbios-ns    137/udp
netbios-dgm   138/tcp
netbios-dgm   138/udp
netbios-ssn   139/tcp
netbios-ssn   139/udp
imap2         143/tcp
imap2         143/udp
snmp          161/udp
snmp-trap     162/udp
cmip-man      163/tcp
cmip-man      163/udp
cmip-agent    164/tcp
cmip-agent    164/udp
xdmcp         177/tcp
xdmcp         177/udp
nextstep      178/tcp
nextstep      178/udp

hostname      # usually from sri-nic
tsap          # part of ISODE.
cso-ns        # also used by CSO name server
cso-ns        # Remote Telnet

postoffice    # POP version 2
# POP version 3

tap ident authentication

readnews untp # USENET News Transfer Protocol

# Network Time Protocol
# NETBIOS Name Service
# NETBIOS Datagram Service
# NETBIOS session service

imap imap4    # Interim Mail Access Proto v2
imap imap4

# Simple Net Mgmt Proto
snmptrap      # Traps for SNMP
# ISO mgmt over IP (CMOT)

# X Display Mgr. Control Proto

NeXTStep NextStep # NeXTStep window
NeXTStep NextStep # server

```

```

bgp          179/tcp          # Border Gateway Proto.
bgp          179/udp
prospero    191/tcp          # Cliff Neuman's Prospero
prospero    191/udp
irc         194/tcp          # Internet Relay Chat
irc         194/udp
smux        199/tcp          # SNMP Unix Multiplexer
smux        199/udp
at-rtmp     201/tcp          # AppleTalk routing
at-rtmp     201/udp
at-nbp      202/tcp          # AppleTalk name binding
at-nbp      202/udp
at-echo     204/tcp          # AppleTalk echo
at-echo     204/udp
at-zis      206/tcp          # AppleTalk zone information
at-zis      206/udp
z3950       210/tcp          wais          # NISO Z39.50 database
z3950       210/udp          wais
ipx         213/tcp          # IPX
ipx         213/udp
imap3       220/tcp          # Interactive Mail Access
imap3       220/udp          # Protocol v3
ulistserv   372/tcp          # UNIX Listserv
ulistserv   372/udp
https       443/tcp          # http protocol over TLS/SSL
https       443/udp          # http protocol over TLS/SSL
#
# UNIX specific services
#
exec         512/tcp
biff        512/udp          comsat
login       513/tcp
who         513/udp          whod
shell       514/tcp          cmd          # no passwords used
syslog      514/udp
printer     515/tcp          spooler      # line printer spooler
talk        517/udp
ntalk       518/udp
route       520/udp          router routed # RIP
timed       525/udp          timeserver
tempo       526/tcp          newdate
courier     530/tcp          rpc
conference  531/tcp          chat
netnews     532/tcp          readnews
netwall     533/udp          # -for emergency broadcasts
uucp        540/tcp          uucpd        # uucp daemon
remotefs    556/tcp          rfs_server rfs # Brunhoff remote filesystem
klogin      543/tcp          # Kerberized `rlogin' (v5)
kshell      544/tcp          # Kerberized `rsh' (v5)
kerberos-adm 749/tcp          # Kerberos `kadmin' (v5)
#
webster     765/tcp          # Network dictionary
webster     765/udp
#
# From ``Assigned Numbers'':
#
#> The Registered Ports are not controlled by the IANA and on most systems
#> can be used by ordinary user processes or programs executed by ordinary

```

```
#> users.
#
#> Ports are used in the TCP [45,106] to name the ends of logical
#> connections which carry long term conversations. For the purpose of
#> providing services to unknown callers, a service contact port is
#> defined. This list specifies the port used by the server process as its
#> contact port. While the IANA can not control uses of these ports it
#> does register or list uses of these ports as a convenience to the
#> community.
#
ingreslock 1524/tcp
ingreslock 1524/udp
prospero-np 1525/tcp      # Prospero non-privileged
prospero-np 1525/udp
rfe        5002/tcp      # Radio Free Ethernet
rfe        5002/udp      # Actually uses UDP only
#
#
# Kerberos (Project Athena/MIT) services
# Note that these are for Kerberos v4, and are unofficial. Sites running
# v4 should uncomment these and comment out the v5 entries above.
#
#kerberos  750/udp      kdc  # Kerberos (server) udp
#kerberos  750/tcp      kdc  # Kerberos (server) tcp
krbupdate  760/tcp      kreg # Kerberos registration
kpasswd    761/tcp      kpwd # Kerberos "passwd"
#klogin    543/tcp      # Kerberos rlogin
eklogin    2105/tcp     # Kerberos encrypted rlogin
#kshell    544/tcp      krcmd # Kerberos remote shell
#
# Unofficial but necessary (for NetBSD) services
#
snpp       444/tcp      # Simple Network Paging Protocol
supfilesrv 871/tcp      # SUP server
supfiledbg 1127/tcp     # SUP debugging
hylafax    4559/tcp     # HylaFAX client-server protocol
axnet      5492/tcp     # Applix Server
aagtwy     5493/tcp     # Applix Anyware
rplay      5555/udp
http-rman  6711/tcp     #man to html on the fly
sql30      7200/tcp     #adabasd remote sql
btx        20005/tcp    ceptd
vboxd      20012/tcp    # for vbox daemon
vboxd      20012/udp
isdnlog    20011/tcp    isdnlog
LNXDBMS    26150/tcp
midinet    40001/tcp    midinetd

#
# services for hp eloquence software
#
runsrv     8010/tcp     # HP Eloquence RUNSRV (Windows integration)
eloqsd     8100/tcp     # HP Eloquence A.06.00 eloqsd server
eloqdb     8102/tcp     # HP Eloquence A.06.00 data base server
eloqdb5    8104/tcp     # HP Eloquence A.06.00 ELOQDB5 server

#
# services for amanda backup software
```

```
#
amanda          10080/udp
amandaidx       10082/tcp
amidxtape       10083/tcp

#
# services for radius software
#
radius          1645/udp          radiusd
radacct         1646/udp

#
# make apple talk more friendly
#
rtmp            1/ddp          # Routing Table Maintenance Protocol
nbp             2/ddp          # Name Binding Protocol
echo           4/ddp          # AppleTalk Echo Protocol
zip            6/ddp          # Zone Information Protocol

#
# Appletalk
#
afpovertcp     548/tcp          # AFP over TCP
afpovertcp     548/udp          # AFP over UCP

#
# MySQL Daemon
#
mysql          3306/tcp
mysql          3306/udp

#
# the rsync daemon
#
rsync          873/tcp

#
# netplan
#
netplan        12000/tcp

#
# swat is the Samba Web Administration Tool
#
swat           901/tcp

#
# Entries for IBM DB2 Database
#
db2cdb2inst1   50000/tcp        # Connection port for DB2 instance db2inst1
db2idb2inst1   50001/tcp        # Interrupt port for DB2 instance db2inst1

#
# Entry for Mimer database
#
mimer          1360/tcp        # Mimer Database Server
```



**ANEXO C:**

[o símbolo ↵ marca a continuação da linha; os acentos foram suprimidos do script]

```
#!/bin/sh

#Controla cores na tela ...
bold=`tput smso`
offbold=`tput rmso`
BOLDON="setterm -bold on"
BOLDOFF="setterm -bold off"
GREEN="setterm -foreground green"
DEF="setterm -default"
clear
# fim ...

# -----
# Interface
# Coloque aqui a interface desejada...
# eth0, eth1, ppp0, etc.
LOCALIF="eth0"

# -----
#
# Localizacao do ipchains...

IPCHAINS="/sbin/ipchains"

# Retira enderecos a partir do ifconfig... Nao altere!

LOCALIP=`ifconfig $LOCALIF | grep inet | cut -d : -f 2 | cut -d \ -f 1`
LOCALMASK=`ifconfig $LOCALIF | grep Mask | cut -d : -f 4`
LOCALNET="$LOCALIP/$LOCALMASK"

echo "x.....x"

echo "${bold}[Firewall!}${offbold} IP: $LOCALNET"

REMOTENET="0/0"

# -----
# Limpa todas as regras de firewall, flag '-F'...

echo -n "${bold}[Firewall!}${offbold} Limpando regras.."

#Pacotes que chegam _de fora_ da rede...

$IPCHAINS -F input
echo -n "."
```

```
# Pacotes que saem da intranet

$IPCHAINS -F output
echo -n "."

$GREEN
echo "Terminado!"
$DEF

# -----

echo -n "${bold}[Firewall!}${offbold} Interface 'lo'.."

$IPCHAINS -A input -i lo -s 0/0 -d 0/0 -j ACCEPT
$IPCHAINS -A output -i lo -s 0/0 -d 0/0 -j ACCEPT
echo -n ".."

$GREEN
echo "Terminado!"
$DEF

# -----
#Configura telnet, WWW e FTP com um prazo minimo, manipulando os bits de #TOS do
pacote...
#ATENCAO! Deve-se ter habilitado no kernel a opcao CONFIG_IP_ROUTE_TOS!
# Ver "Ipchains How-To", p. 18.
# Os chamados TOS bits afetam a forma como os pacotes sao tratados. As
# configuracoes possiveis sao:
# 'maximum delay'
# 'maximum throughput'
# 'maximum reliability'
# 'minimum cost'

echo -n "${bold}[Firewall!}${offbold} Type Of Service flags.."

$IPCHAINS -A output -p tcp -d 0/0 www -t 0x01 0x10
$IPCHAINS -A output -p tcp -d 0/0 telnet -t 0x01 0x10
$IPCHAINS -A output -p tcp -d 0/0 ftp -t 0x01 0x10
echo -n "...

# Configura ftp-data para o maximo de 'saida'...
$IPCHAINS -A output -p tcp -d 0/0 ftp-data -t 0x01 0x08
echo -n "."

$GREEN
echo "Terminado!"
$DEF

# -----
# Poe regras para permitir conexoes de redes e hosts confiaveis...

# echo -n "${bold}[Firewall!}${offbold} Redes confiaveis.."

# $IPCHAINS -A input -s [IPs host/net] -d $LOCALNET <ports> -j ACCEPT
# echo -n "."
```

```

# $GREEN
# echo "Terminado!"
# $DEF

# -----
# Poe regras para impedir conexoes com redes e hosts nao-confiaveis. Os
# pacotes sao registrados...

# echo -n "${bold}[Firewall!}${offbold} Redes nao-confiaveis.."

# $IPCHAINS -A input -l -s [IPs host/net] -d $LOCALNET <ports> -j DENY
# echo -n "."

# Regra para impedir ataques ICMP...

# $IPCHAINS -A input -l -b -i $LOCALIF -p icmp -s [IPs] -d $LOCALNET -j
↳DENY
# echo -n "."

# $GREEN
# echo "Terminado!"
# $DEF

# -----

echo -n "${bold}[Firewall!}${offbold} Servicos diversos.."

# $IPCHAINS -A input -p tcp -s alpha.cipsga.org.br -d $LOCALNET ↳1023:65535 -j
ACCEPT
# $IPCHAINS -A input -p tcp -s mail.cipsga.org.br -d $LOCALNET ↳1023:65535 -j
ACCEPT
# $IPCHAINS -A input -p tcp -s www.cipsga.org.br -d $LOCALNET 1023:65355 ↳-j
ACCEPT
# $IPCHAINS -A input -p tcp -s proxy.cipsga.org.br -d $LOCALNET ↳1023:65535 -j
ACCEPT
# echo -n "...."

# Posso usar os enderecos abaixo para manejar meu DNS ...
DNSHOST=200.215.0.1 #por exemplo...
# $IPCHAINS -A input -p tcp -s $DNSHOST -d $LOCALNET 1023:65535 -j ↳ACCEPT
# $IPCHAINS -A input -p tcp -s $DNSHOST -d $LOCALNET 1023:65535 -j ↳ACCEPT
# echo -n ".."

# Use esse espaco para negar acesso a rede a maquina ou grupo de #maquinas...
$IPCHAINS -A input -l -s [IPs] -d $LOCALNET -j DENY
echo -n "."

$GREEN
echo "Terminado!"
$DEF

#-----
# Esta parte nega acesso a servicos e portas sabidamente problematicas
# para fora da rede local ...

```

```
echo -n "${bold}[Firewall!]}${offbold} Portas negadas.."

# NetBEUI/Samba
$IPOCHAINS -A input -p tcp -s $REMOTENET -d $LOCALNET 139 -j DENY
$IPOCHAINS -A input -p udp -s $REMOTENET -d $LOCALNET 139 -j DENY
echo -n "."

# Microsoft SQL
$IPOCHAINS -A input -p tcp -s $REMOTENET -d $LOCALNET 1433 -j DENY
$IPOCHAINS -A input -p udp -s $REMOTENET -d $LOCALNET 1433 -j DENY
echo -n "."

# Postgres SQL

$IPOCHAINS -A input -p tcp -s $REMOTENET -d $LOCALNET 5432 -j DENY
$IPOCHAINS -A input -p udp -s $REMOTENET -d $LOCALNET 5432 -j DENY
echo -n "."

# Network File System (NFS)
$IPOCHAINS -A input -p tcp -s $REMOTENET -d $LOCALNET 2049 -j DENY
$IPOCHAINS -A input -p udp -s $REMOTENET -d $LOCALNET 2049 -j DENY
echo -n "."

# X-Windows Displays :0-:2-
$IPOCHAINS -A input -p tcp -s $REMOTENET -d $LOCALNET 5999:6003 -j DENY
$IPOCHAINS -A input -p udp -s $REMOTENET -d $LOCALNET 5999:6003 -j DENY
echo -n "."

# X Font Server :0-:2-
$IPOCHAINS -A input -p tcp -s $REMOTENET -d $LOCALNET 7100 -j DENY
$IPOCHAINS -A input -p udp -s $REMOTENET -d $LOCALNET 7100 -j DENY
echo -n "."

# Back Orifice (Registrado! Flag -l)
$IPOCHAINS -A input -l -p tcp -s $REMOTENET -d $LOCALNET 31337 -j DENY
$IPOCHAINS -A input -l -p udp -s $REMOTENET -d $LOCALNET 31337 -j DENY
echo -n "."

# NetBus (Registrado! Flag -l)
$IPOCHAINS -A input -l -p tcp -s $REMOTENET -d $LOCALNET 12345:12346 -j DENY
$IPOCHAINS -A input -l -p udp -s $REMOTENET -d $LOCALNET 12345:12346 -j DENY
echo -n "."

$GREEN
echo "Terminado!"
$DEF

# -----
# Tais portas sao abertas para permitir soquetes criado por conexoes
# permitidas pelo ipchains...

echo -n "${bold}[Firewall!]}${offbold} Portas 'altas' e nao-privilagiadas.."

$IPOCHAINS -A input -p tcp -s $REMOTENET -d $LOCALNET 1023:65535 -j ACCEPT
$IPOCHAINS -A input -p udp -s $REMOTENET -d $LOCALNET 1023:65535 -j ACCEPT
echo -n "."
```

```
$GREEN
echo "Terminado!"
$DEF

# -----
# Servicos de carater basico...

echo -n "${bold}[Firewall!]{offbold} Servicos.."

# ftp-data (20) and ftp (21)
# $IPCHAINS -A input -p tcp -s $REMOTENET -d $LOCALNET 20 -j ACCEPT
# $IPCHAINS -A input -p tcp -s $REMOTENET -d $LOCALNET 21 -j ACCEPT
# echo -n ".."

# ssh (22)
# $IPCHAINS -A input -p tcp -s $REMOTENET -d $LOCALNET 22 -j ACCEPT
# echo -n "."

# telnet (23)
# $IPCHAINS -A input -p tcp -s $REMOTENET -d $LOCALNET 23 -j ACCEPT
# echo -n "."

# smtp (25)
# $IPCHAINS -A input -p tcp -s $REMOTENET -d $LOCALNET 25 -j ACCEPT
# echo -n "."

# DNS (53)
$IPCHAINS -A input -p tcp -s $REMOTENET -d $LOCALNET 53 -j ACCEPT
$IPCHAINS -A input -p udp -s $REMOTENET -d $LOCALNET 53 -j ACCEPT
echo -n ".."

# DHCP do lado da Rede Local (to make @Home DHCP work) (67/68)
# $IPCHAINS -A input -i $LOCALIF -p udp -s $REMOTENET -d 255.255.255.255/24 67 -
j ACCEPT
# $IPCHAINS -A output -i $LOCALIF -p udp -s $REMOTENET -d 255.255.255.255/24 68
-j ACCEPT
# echo -n ".."

# http (80)
# $IPCHAINS -A input -p tcp -s $REMOTENET -d $LOCALNET 80 -j ACCEPT
# echo -n "."

# POP-3 (110)
# $IPCHAINS -A input -p tcp -s $REMOTENET -d $LOCALNET 110 -j ACCEPT
# echo -n "."

# identd (113)
$IPCHAINS -A input -p tcp -s $REMOTENET -d $LOCALNET 113 -j REJECT
echo -n "."

# nntp (119)
# $IPCHAINS -A input -p tcp -s $REMOTENET -d $LOCALNET 119 -j ACCEPT
# echo -n "."

# https (443)
# $IPCHAINS -A input -p tcp -s $REMOTENET -d $LOCALNET 443 -j ACCEPT
```

```
# echo -n "."

# ICQ Services (4000)
# $IPCHAINS -A input -p tcp -s $REMOTENET -d $LOCALNET 4000 -j ACCEPT
# echo -n "."

$GREEN
echo "Terminado!"
$DEF

# -----

echo -n "${bold}[Firewall!]{offbold} Regras ICMP.."

# Use isso para negar ataques ICMP de enderecos especificos...
$IIPCHAINS -A input -b -i $EXTERNALIF -p icmp -s <address> -d 0/0 -j DENY
# echo -n "."

# Permite a chegada de pacotes ICMP
$IIPCHAINS -A input -p icmp -s $REMOTENET -d $LOCALNET -j ACCEPT
$IIPCHAINS -A input -p icmp -s $REMOTENET -d $LOCALNET -j ACCEPT
echo -n ".."

# Permite a saida de pacotes ICMP
$IIPCHAINS -A output -p icmp -s $LOCALNET -d $REMOTENET -j ACCEPT
$IIPCHAINS -A output -p icmp -s $LOCALNET -d $REMOTENET -j ACCEPT
echo -n "...."

$GREEN
echo "Terminado!"
$DEF

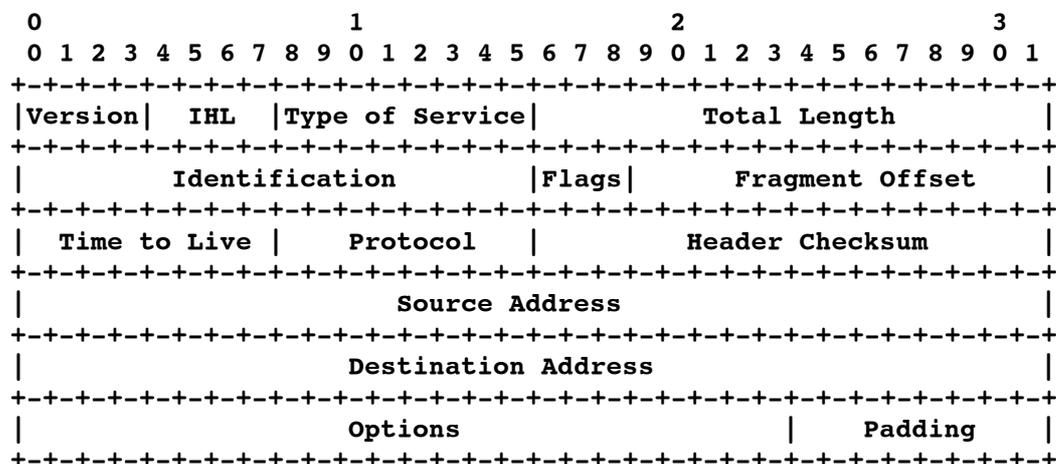
# -----
# Configura politica padrao!

$IIPCHAINS -A input -j DENY
$IIPCHAINS -A output -j ACCEPT

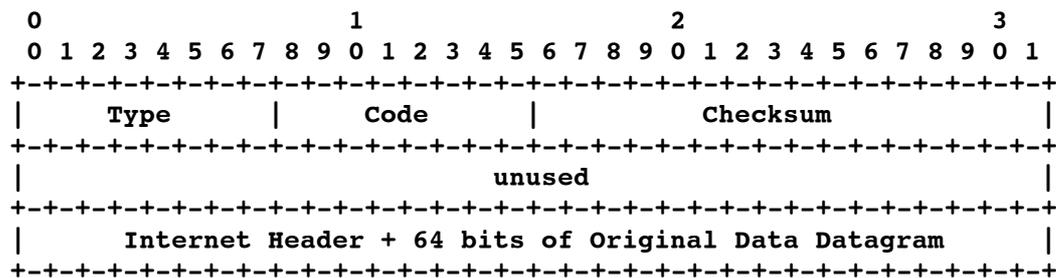
echo ""
$BOLDON
echo "${bold}[Firewall!]{offbold} Estabelecimento de firewall completo."
$BOLDOFF
echo "x.....x"
$DEF
```

## Anexo D:

### a.1 O cabeçalho IP:



### a.2 O cabeçalho ICMP



#### Campos ICMP:

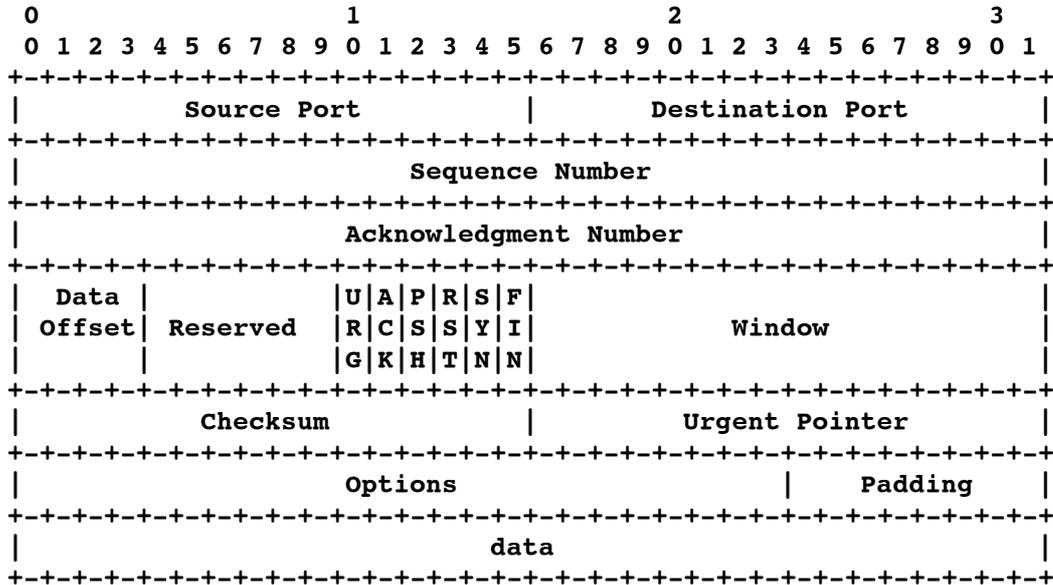
##### Type

3

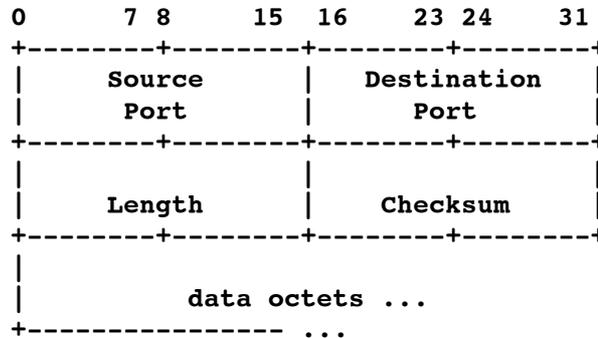
##### Code

- 0 = net unreachable;
- 1 = host unreachable;
- 2 = protocol unreachable;
- 3 = port unreachable;
- 4 = fragmentation needed and DF set;
- 5 = source route failed.

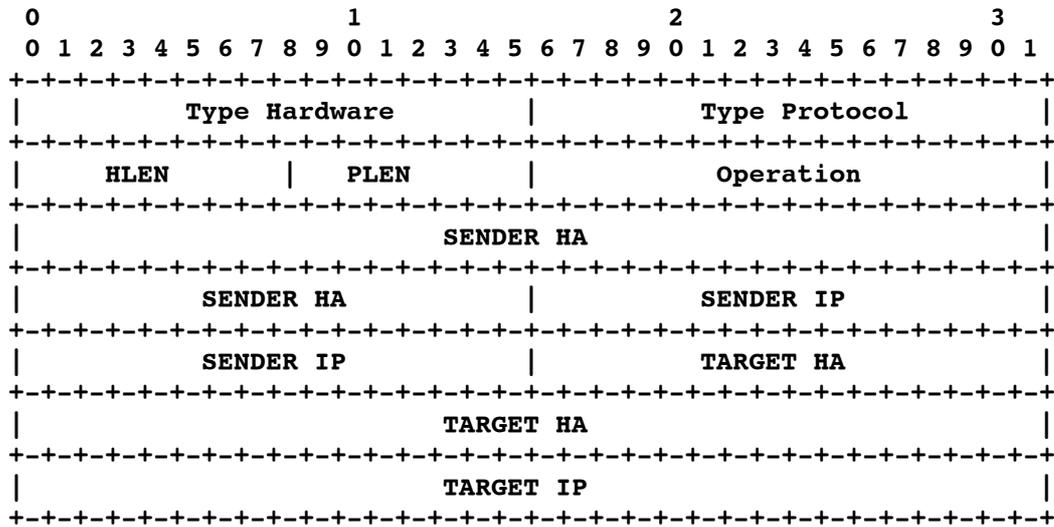
a.3 O cabeçalho TCP:



a.4 O cabeçalho UDP:



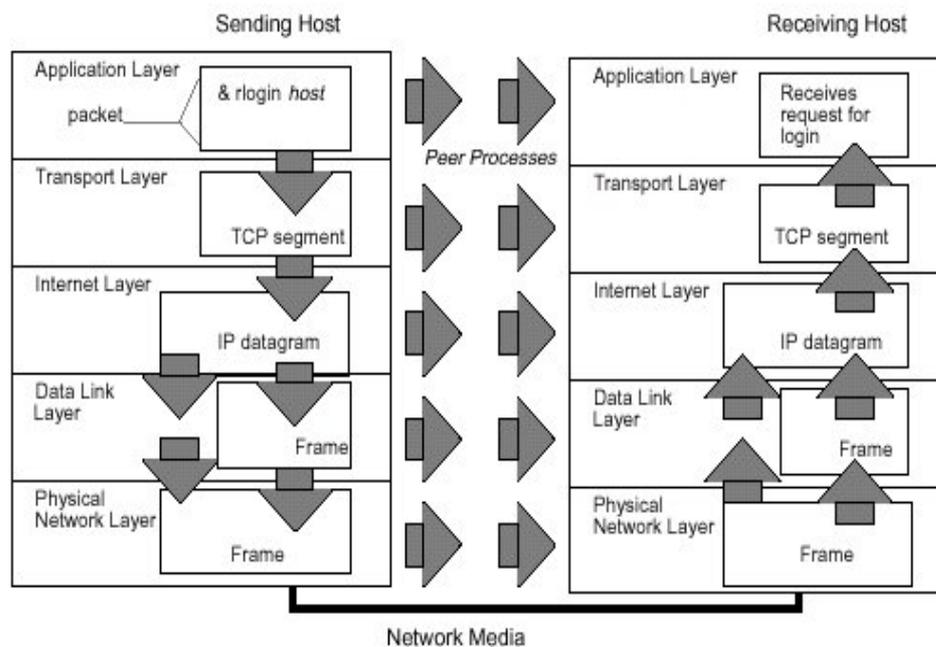
a5. O cabeçalho ARP



## Anexo E:\*

*Como um pacote viaja através de um aplicativo (cliente) até o host servidor?*

No diagrama abaixo nós observamos um *host* (cliente) abrindo uma conexão via **rlogin** com outro *host* (servidor):



\***Fonte:** *TCP/IP and Data Communications Administration Guide*, Sun Microsystems, Inc, october 1998, p. 23.

## **Bibliografia geral**

---

- ARNETT, M. et alii. *Desvendando o TCP/IP*. Rio de Janeiro, ed. Campus, 1997.
- BELLOVIN, S. *Defending Against Sequence Number Attacks*, RFC 1948.
- \_\_\_\_\_. "Security Problems in the TCP/IP Protocol Suite". *Computer Communication Review*, vol 19, nº2, 1989.
- COMER, D. E. *Interligação em Rede com TCP/IP. Volume I: Princípios, Protocolos e Arquitetura*. Rio de Janeiro, ed. Campus, 2ª edição, 1998.
- CRISCUOLO, P. *Distributed Denial of Service. Trin00, Tribe Flood Network, Tribe Flood Network 2000. and Stacheldraht*. CIAC-2319, Department of Energy, LLNL, february 14, 2000.
- DAWSON, T. *NET-3 HOWTO*, v.1.1, march, 1997.
- DYSON, P. *Dicionário de Redes*. Rio de Janeiro, ed. Campus - Novell Press, 1995
- FARMER, D. & Wietse V. *Improving the Security of your Site by Breaking into it*, 1993, fonte: 'ftp://ftp.win.tue.nl'.
- FENZI, K. & WRESKI D. *Linux Security-HOWTO*, v.0.9.11, may 1998.
- FRASER, B. *Site Security Handbook*, RFC 1244.
- HUNT, R. "Internet/Intranet Firewall Security - Policy, Architecture and Transaction Services", *Computer Communications*, 23 (1998).
- MARCELO, A. *Intranet em Ambiente Linux*. Rio de Janeiro, BRASPORT, 1999.
- MUCHSEL, R. *sf Firewall Software Version*, feb. 1997.
- NORTHCUTT, S. *Como Detectar Invasão em Rede. Um Guia para Analistas*. Rio de Janeiro, ed. Ciência Moderna, 2000.
- OLIVEIRA, W. J. *Hacker: Invasão e Proteção*. Florianópolis, Visual Books, 1999.
- POSTEL, J. (ed.). *Transmission Control Protocol. DARPA Internet Program Protocol Specification*, RFC 793.
- POSTEL, J. *User Datagram Protocol*, RFC 768.
- RUSSEL, P. *Linux IPCHAINS-HOWTO*, v.1.0.6, jan. 1999.
- SCO OpenServer™ Internet Services, v.5.0.4, may 1997.
- SERY, P. *Ferramentas Poderosas para Redes em Linux*. Rio de Janeiro, Ciência Moderna, 1998.
- SHIMOMURA T. & MARKOFF, J. *Contra-Ataque: a História da Captura do Pirata Cibernético mais Procurado dos EUA*. São Paulo, Companhia das Letras, 1996.
- SUN Microsystems™. *TCP/IP and Data Communications Administration Guide*. Palo Alto, oct. 1998.

*SuSE Linux™ 6.3. Installation, Configuration and First Steps.* Nürnberg, 16ª edição

## Sobre o autor da Apostila

---

Esta apostila é de autoria de **Renato Martini** <[rmartini@cipsga.org.br](mailto:rmartini@cipsga.org.br)>, doutor em Filosofia pela Pontifícia Universidade Católica do Rio de Janeiro, onde lecionou as disciplinas de Filosofia da Ciência e Lógica para o CTC. Foi bolsista do CNPq ao longo dos anos pós-graduação. Usuário e entusiasta há mais de 5 anos do Linux, - mas essencialmente como um Sistema Operacional voltado para Rede, e igualmente apto ao conhecimento e aos desafios acadêmicos e pedagógicos de nosso país.

Desenvolve no momento um site sobre Segurança em RedesLinux, que será hospedado pelo CIPSGA. Também participa do desenvolvimento de um 'sniffer', o APS, sob os termos da GNU GPL, com Christian Schulte.

Numa escola da Secretaria de Educação do Rio de Janeiro, em Niterói, desenvolve um Laboratório com uma Rede Linux, para o ensino introdutório de Linux e Redes para jovens do ensino médio.

Seu projeto futuro de ensino também é um Curso que proponha a integração do Linux com os outros Sistemas Operacionais UNIX tradicionais, tais como SCO OpenServer, o Solaris, e o FreeBSD.

# Comitê de Incentivo a Produção do Software Gratuito e Alternativo



Fundado em 29 de janeiro de 1999.

1ª Diretoria

Djalma Valois Filho

**Diretor Executivo**

dvalois@cxpostal.com

José Luiz Nunes Poyares  
Diretor Administrativo

Paulo Roberto Ribeiro Guimarães  
Diretor Institucional

CIPSGA  
Rua Professora Ester de Melo, numero 202,  
Parte, Benfica, Rio de Janeiro, RJ, CEP. 20930-010;  
Telefone (Fax/Dados): 021-5564201;  
e-mail: administracao@cipsga.org.br  
CNPJ: 03179614-0001/70