A 1ª LINHA DE DEFESA

por **ANDRÉ ONOFRE LIMA**

HTTP://PWP.NET.IPL.PT/ALUNOS.ISEL/28838/

Nº 28838 IRS 09/10 ISEL

Índice

Introdução

Este trabalho tem como base comum a todos os grupos a instalação e configuração de um sistema Gentoo; a implementação de um servidor DNS utilizando o Bind9; e a implementação de um router que separe uma rede interna (10.62.74.176/28) da rede externa (10.62.75.0/24), comunicando a existência da rede interna através de OSPF, utilizando o Quagga. O desenvolvimento destes sistemas desta forma, tem a vantagem de serem bastante eficientes devido à reduzida quantidade de aplicações, estritamente necessárias, para o cumprimento das funções do servidor/router, facto esse que traz também outras vantagens como maior segurança e menor complexidade.

Em seguida, é implementada a parte do trabalho específica a cada grupo, neste caso: implementação de uma firewall utilizando o Iptables. O objectivo é o de realçar algumas regras essenciais à prevenção de ataques muito conhecidos, e também o de demonstrar aplicações práticas de vários módulos do próprio Iptables, em forma de cenários comuns no dia a dia de administradores de redes e sistemas.

Instalação do Gentoo

A instalação do sistema foi feita na máquina virtual, arrancando a máquina a partir da imagem do ficheiro "install-x86-minimal-20091020.iso". Este CD detecta o *hardware* contido na máquina, carrega os *drivers* necessários, e fornece ao utilizador um ambiente Gentoo para que se proceda com a configuração da placa de rede com uma ligação à internet, necessária para fazer *download* do *kernel*, e para que se proceda com o particionamento do disco e outras configurações necessárias ao alojamento do sistema operativo.

A instalação do Gentoo divide-se em três fases, conhecidas por *stages*. Na *stage1* tem-se te fazer *bootstrap* ao sistema, ou seja, compilar o compilador gcc, a biblioteca de C e mais alguns programas essenciais. Na *stage2* o que se faz basicamente é um *emerge system* para instalar os programas necessários para se ter um sistema base funcional. No caso deste projecto, foi escolhido a instalação *stage3*, ou seja, os *stages* 1 e 2 já implementados (no CD install-x86-minimal-20091020.iso) e o que se vai explicar em seguida consiste na *stage3*.

Início da instalação

Após constatar que a rede já se encontrava configurada (#/sbin/ifconfig eth0), procedeu-se à partição do disco /dev/sda que alojará o sistema. Este *block device* constitui uma i*nterface* que abstrai o administrador e as aplicações das especificidades do disco, nomeadamente se são SCSI, IDE, ou outro tipo. Partições são divisões lógicas do disco e existem três tipos: primário, estendido e lógico. As partições primárias têm a sua informação armazenada no *Master Boot Record* (MBR), e como este tem tamanho reduzido (512 *bytes*), só podem ser definidos quatro partições primárias. As partições estendidas são partições primárias especiais, na medida em que, dada a necessidade de se terem mais do que quatro partições, estas permitem a criação de mais partições dentro delas, sendo essas últimas conhecidas por partições lógicas, cujos dados não são mantidos no MBR.

Com base no que até então foi explicado, criaram-se as seguintes partições (/sbin/fdisk /dev/sda):

```
   Device Boot    Start       End    Blocks   Id  System
/dev/sda1   *         1        14    105808+  83  Linux
/dev/sda2            15        81    506520   82  Linux swap
/dev/sda3            82      3876  28690200   83  Linux
```

A partição sda1 tem 32MB e é a partição de *boot*, ou seja, é onde se armazena o *kernel* utilizado pelo sistema operativo e o *boot loader*, que é das primeiras aplicações a serem executadas no arranque do computador. Neste caso, mais especificamente, o BIOS tenta encontrar o *boot sector* do primeiro dispositivo de arranque, ou seja, o disco utilizado. Isto significa que o BIOS executará o *boot loader* que se situa no MBR, *boot loader* esse que dá ao utilizador a opção do sistema operativo a arrancar. Para esta instalação optou-se, de entre os *boot loaders* LILO e GRUB, pelo GRUB. A razão para essa escolha remete-nos às várias vantagens que o GRUB tem sobre o LILO. A primeira, e provavelmente a de maior importância, consiste no facto do GRUB permitir que se aceda a uma linha de comandos e se carregue outro *kernel* que não apenas os especificados no seu ficheiro de configuração. Isto é particularmente útil quando se altera a configuração do LILO para carregar um *kernel* novo e esquece-se de manter a velha configuração. Se o novo *kernel* rebentar, é-se obrigado a reiniciar a partir do CD/DVD/disquete. Outra vantagem do GRUB é a de esta apenas ter de ser instalada uma única vez, tendo-se que alterar em caso de necessidade apenas o respectivo ficheiro de configuração. O LILO obriga a que a cada alteração na configuração, se tenha de voltar a instalar o *boot loader* no MBR visto ser lá que se armazena a informação relativa às suas configurações.

A partição sda2 tem 512MB e é a partição de *swap* que serve como memória virtual quando a RAM se esgota. Finalmente, a partição sda3, cujo tamanho ocupou o resto do disco de 8GB atribuído à máquina virtual, é onde será montado (*mount*) a raiz do sistema de ficheiros utilizado (*root*).

Em seguida, foram formatadas as partições sda1 (mke2fs /dev/sda2) e sda3 (mke2fs -j /dev/sda3) com os sistemas de ficheiros ext2 e ext3 respectivamente. Apesar do ext3 já ser um sistema de ficheiros bastante testado e consequentemente confiável, visto que o sda1 tem apenas 32MB, optou-se por utilizar o ext2 para não desperdiçar o espaço necessário ao armazenamento da *metadata* referente ao *journaling*.

A seguir, foram montadas as partições sda3 e sda1 para posteriormente se criar toda a estrutura do sistema de ficheiros. Essa estrutura será criada com a ajuda dos dois seguintes ficheiros: "stage3-i686-<release>.tar.bz2" e "portage-latest.tar.bz2" obtidos no site http://www.gentoo.org/main/en/mirrors.xml com o comando *links*. O *stage3 tarball* contém a estrutura básica do sistema de ficheiros usado no Gentoo Linux. Essa estrutura é a necessária para prosseguir com a instalação do sistema. Quanto ao *portage*, este consiste numa aplicação de gestão de software, tida para muitos como o melhor gestor de software para ambientes Linux, e é também considerada uma das grandes vantagens da distribuição Gentoo. O *portage* permitirá que sejam feitas actualizações ao sistema, e que sejam adicionadas e removidas aplicações de forma bastante simples.

Configurando opções de compilação (kernel e outras aplicações)

Apesar de se poderem alterar essas opções utilizando as variáveis de ambiente, o que tornaria essas alterações temporárias mediante o reinicio do sistema, o mais adequado aqui é torná-las permanentes no ficheiro /etc/make.conf que é um ficheiro de configuração do *portage*. Dentro deste ficheiro encontram-se as variáveis CFLAGS e CXXFLAGS que definem as opções de optimização do compilador gcc para C e C++ respectivamente. Outra variável que se deve adicionar a este ficheiro, com a ajuda do comando *mirrorselect* é o GENTOO_MIRRORS que define de onde o *portage* deverá efectuar os *downloads* necessários. Finalmente, com a ajuda do mesmo comando mas com parâmetros diferentes, dever-se-á adicionar a variável SYNC que indica qual o servidor *rsync* a utilizar quanto se estiver a actualizar a chamada *portage tree*. Esta consiste na colecção de executáveis e *scripts* necessários ao *download* e instalação de *software*.

Agora que já se configuraram as opções de compilação, o próximo passo consiste em mudar do ambiente de execução do LiveCD para o do sistema instalado com o comando *chroot*. Este comando altera a raíz do sistema / para a directoria que se lhe indicar. Contudo, antes de se fazer isso, há que garantir que a resolução de nomes, no que diz respeito à rede, continua a funcionar no novo ambiente. Para isso é copiado o ficheiro /etc/resolv.conf para /mnt/gentoo/etc/, sendo /mnt/gentoo a directoria que servirá de raiz ao novo ambiente de execução. Outro facto importante a garantir é o de, no novo ambiente, se poder obter informação referente ao *hardware* em utilização e aos dispositivos (*devices*) a que se tem acesso. Para isso são montados (*mount*) o /proc (mount -t proc none /mnt/gentoo/proc) e o /dev (mount -o bind /dev /mnt/gentoo/dev) respectivamente. O primeiro *mount* (relativo aos proc) faz com que seja montado um sistema de ficheiros do tipo proc na directoria indicada. Esta pasta contém informação sobre que processos encontram-se em execução e pormenores relativos a essas execuções, e também contém informação sobre recursos que se encontram alocados naquele instante pelo *kernel*. Quanto ao segundo *mount* (relativo ao dev), este faz com que o mesmo conteúdo /dev seja acessível em dois sítios simultaneamente. Isso é feito para que os dispositivos detectados pelo actual sistema (Live CD) possam ser acedidos pelo novo ambiente.

O novo ambiente de execução

Um ambiente de execução é basicamente definido pela raiz do sistema de ficheiros desse ambiente (neste caso pretende-se que seja o /mnt/gentoo) e pelas variáveis de ambiente. A alteração da raiz do sistema é feito com o comando *chroot*, enquanto que a alteração das variáveis de ambiente é feita pelo comando *env-update* e pelo *source /etc/profile*. O ficheiro /etc/profile é um ficheiro que contém variáveis de ambiente que abrangem todo o sistema.

Para que melhor se perceba o *env-update*, há que se conhecer a directoria /etc/env.d. Nesta pasta encontram-se ficheiros específicos a cada aplicação que utilize variáveis de ambiente na sua execução, ou que necessite delas para ser encontrada. Um exemplo disto é o gcc:

```
PATH="/usr/i686-pc-linux-gnu/gcc-bin/3.2"
ROOTPATH="/usr/i686-pc-linux-gnu/gcc-bin/3.2"
MANPATH="/usr/share/gcc-data/i686-pc-linux-gnu/3.2/man"
INFOPATH="/usr/share/gcc-data/i686-pc-linux-gnu/3.2/info"
CC="gcc"
CXX="g++"
LDPATH="/usr/lib/gcc-lib/i686-pc-linux-gnu/3.2.3"
```

Com esta arquitectura, única do Gentoo, consegue-se organizar melhor as variáveis de ambiente do sistema. Por exemplo se se quisessem acrescentar variáveis, tudo o que se teria de fazer seria criar um ficheiro /etc/env.d/99local com, por exemplo, a informação de *proxy* HTTP.

Percebendo-se este esquema, o que o *env-update* faz é concatenar as várias variáveis idênticas dos vários ficheiros. Esta concatenação na verdade apenas acontece para algumas variáveis como o PATH, importante na localização de ficheiros nomeadamente executáveis, e como MANPATH, importante para o comando *man* localizar os *manpages* específicos ao programa que se pretenda. O resultado dessas concatenações, ou sobreposições, será colocado então no ficheiro /etc/profile.env que é, por sua vez utilizado por /etc/profile. O *env-update* também extrairá a informação de LDPATH (contém lista de directorias onde se situam as bibliotecas utilizadas pelo *dynamic linker*) para criar /etc/ld.so.conf. Após fazer isso, o *env-update* invocará internamente o comando *ldconfig* para recriar /etc/ld.so.cache que é utilizado pelo *dynamic linker* - parte do sistema operativo que carrega e liga as bibliotecas utilizadas pelas aplicações em execução.

Para que se notem as diferenças provocadas pelo comando anterior imediatamente, é necessário a execução do comando *source /etc/profile*. que basicamente faz um *refresh* às variáveis de ambiente do sistema.

Instalando o Kernel

O primeiro passo foi o de actualizar o *portage tree* para a versão mais recente (*emerge --sync --quiet*). A seguir foi definido o perfil do sistema (eselect profile list). Este perfil é muito importante na definição de valores por omissão nas principais variáveis de ambiente do sistema, neste caso o USE, e é também importante no bloqueio do sistema de forma a que este aceite apenas um determinado conjunto de aplicações. Por exemplo, se o *profile* utilizado for o *desktop*, no USE constarão opções que pressupõem que a máquina deseja instalar aplicações como o kde ou o gnome, e mesmo aplicações de suporte a multimédia (áudio/video).

Procedeu-se então com o *download* do *kernel* (*emerge gentoo-sources*) e posteriormente foram feitas as seguintes configurações no *kernel* tendo em atenção o facto de que todos os *drivers,* vitais ao arranque do *kernel,* fossem compilados dentro do *kernel* e não como módulos, de forma a que o sistema arranque sem problemas, visto que os módulos só são carregados no *kernel* após este arrancar.

Configurações: escolha da família correcta de processadores; suporte aos sistemas de ficheiros utilizados (ext2 e ext3, proc e virtual filesystem); suporte ao multi-processamento; e suporte a dispositivos USB.

Em seguida foram compilados o *kernel* e os respectivos módulos e, finalmente, colocado a imagem gerada na partição de *boot*.

Após a instalação do *kernel* foi configurado o ficheiro /etc/fstab (contém informação relativa a que dispositivos de bloco - /dev - devem ser montados, onde e como), foi configurada a rede (/etc/conf.d/net) de forma a pedir endereço IP usando DHCP por omissão; foi configurada a placa de rede eth0 para se iniciar com o arranque do *runlevel* 3 (*default*); foi configurado o teclado para a língua portuguesa (/etc/conf.d/keymaps); e foram instalados o syslog-ng (logs do sistema), vixie-cron (possibilidade de agendar tarefas únicas ou periódicas no sistema), slocate (indexação dos ficheiros do sistema para rápida localização dos mesmos), dhcpcd (pedidos dhcp como cliente), sudo (delegação de autorização de execução de comandos), e vim (editor de texto preferido do autor deste relatório).

Antes de terminar a instalação do Gentoo com o LiveCD, teve-se de instalar o GRUB, cuja escolha já se explicou na secção "Início da instalação". Finalmente fez-se *reboot* ao sistema, mas sem antes desmontar (*umount*) todos os *devices* montados: /mnt/gentoo/boot , /mnt/gentoo/dev , /mnt/gentoo/proc e /mnt/gentoo/.

Com o reinicio do sistema, já a partir do disco, removeram-se os ficheiros correspondentes ao *stage3* e ao *portage*. Em seguida foi acrescentada uma *password* à conta *root*, e foi criada uma conta ao utilizador *alima*, com *password* e respectiva pasta em /home/alima, sendo este utilizador pertencente aos grupos *users*, *wheel*. Este último grupo é particularmente útil para a configuração do /etc/sudoers onde se especifica que todos os utilizadores que pertencem ao grupo *wheel* tem permissão de acesso a pastas/ficheiros/programas cujos acessos são normalmente vedados ao *root*.

DNS (bind)

Existem três tipos de servidores DNS: primários, secundários e de *cache*. Os primários são considerados servidores DNS com autoridade sobre uma determinada zona (sub-domínio). É sobre eles que são feitas as alterações à configuração das diferentes zonas contidas no domínio em causa. Os secundários são servidores de *backup* ou de distribuição de carga relativamente aos primários. Quando um cliente faz uma interrogação a um servidor secundário, este responde com a mesma autoridade que o primário, contudo correndo-se o risco da resposta não conter uma possível actualização recentemente feita sobre o primário. Em relação ao sincronismo entre os servidores primários e secundários, o primário pode notificar os servidores secundários que estejam indicados no seu ficheiro de configuração, ou os secundários podem periodicamente requisitar actualizações ao primário (*updates*). Quanto aos servidores *cache*, é possível distinguir as suas respostas das dos primários e secundários, pois contêm nessa resposta a indicação de que essa resposta não tem autoridade. O que estes servidores fazem é, caso não encontrem a *Fully Qualified Domain Name* (FQDN) na *cache*, fazer o pedido recursivamente a um servidor primário/secundário. Caso encontre o FQDN pedido na interrogação, é então dada a resposta ao cliente.

Ficheiros de Configuração

Eis o ficheiro de configuração principal /etc/bind/named.conf :

- Anexo A tabela A.1 -

No /etc/bind/named.conf configura-se o bind para ser tanto um servidor de *caching*, para os domínios que este desconheça, reencaminhando-os para um dos *root servers* contidos no ficheiro named.ca, e como servidor de primário para o sub-domínio atribuído ao grupo 2: "g2.lrcd.local". Outro ficheiro importante na configuração, é o indicado na configuração da zona responsável pelo sub-domínio do grupo, o "pri/g2.lrcd.local.zone". A configuração contida nesse ficheiro é mostrado na seguinte tabela:

- Anexo A tabela A.2 -

Neste ficheiro dá-se um TTL aos registos de uma semana, ou seja, este é o tempo máximo que um servidor cache poderá manter os registos retornados deste ficheiro, a não ser que no próprio *record* seja indicado outro valor que se sobreporá ao TTL global. O *serial number* é um número na forma AAAAMMDDxx onde AAAA corresponde ao ano, MM aos dois dígitos do mês, DD aos dois dígitos do dia, e xx ao número da actualização deste ficheiro. Este número é importante porque, quando um servidor secundário, que tenha este servidor como primário, quiser actualizar a sua base de dados, é com base neste número que o secundário saberá se é necessária ou não uma actualização. Mais especificamente, se o número de série do servidor primário for maior (indicando que houve uma nova actualização) que o número de série da última actualização retirada desse mesmo primário por parte do secundário, então é necessário actualizar-se. Quanto ao valor de *Refresh*, este indica com que periodicidade deve ser verificado o número de série, para ver se é necessária uma actualização. O *Retry* indica, num cenário onde o primário não responde ao secundário que pretende ver se há ou não a necessidade de se actualizar, com que periodicidade deve novamente tentar comunicar. O *Expire* indica aos servidores secundários que façam *cache* após quanto tempo, sem conseguir contactar com um servidor primário, devem descartar o registo. O *Minimum* indica aos servidores de *cache* após quanto tempo, caso não consigam contactar com um servidor primário, devem invalidar os registos.

Quanto ao resto da configuração, indica-se o ip do *host* "server", e dá-se o nome "andre" também a este *host*.

No ficheiro pri/g2.lrcd.local.rev é indicado como proceder com os *reverse lookups*. Neste caso o único registo é o do *host* "server.g2.lrcd.local" que tem o ip 10.62.75.132.

- Anexo A tabela A.3 -

Para pôr em prática estas configurações, colocou-se o *daemon* named em execução (/etc/init.d/named start). De salientar que que existe um ficheiro de configuração chamado /etc/conf.d/named que contém uma variável OPTIONS que indica os parâmetros a serem passados na execução do *daemon* named. Por omissão, esta

variável encontra-se vazia. Isto significa que o *daemon* procurará o fichiro de configuração principal named.conf na directoria /etc/bind e que, também por omissão, será executado com as permissões do utilizador named, especialmente criado para este efeito. Caso, por exemplo, se quisesse executar o *daemon* com as permissões de root, e com o ficheiro named.conf definido noutra directoria, apenas ter-se-ia de colocar na variável OPTIONS o valor "-u root -c /outra_dir".

Testes efectuados

Para testar toda esta implementação executaram-se, com a ajuda do comando *dig*, as seguintes instruções:

- Anexo A tabela A.4 -

Quagga

O Quagga consiste num pacote de software que disponibiliza serviços de *routing* como RIP (RIPv1, RIPv2 e RIPng), OSPF (OSPFv2 e OSPFv3) e BGP (BGP-4 e BGP-4+). Este software suporta tanto IPv4 como IPv6, e também tem suporte para SNMP.

Arquitectura do Sistema

Com o Quagga instalado, o sistema comporta-se como um *router* dedicado apossando-se das interfaces do próprio sistema. As actualizações recebidas através dos protocolos de *routing* servem posteriormente para actualizar a tabela de *routing* do *kernel*. Essa actualização é feita, directamente à tabela de *routing* do *kernel*, por um *daemon* chamado zebra. Indirectamente, a informação obtida pelo zebra é-lhe fornecida por *daemons* específicos à implementação de cada protocolo de *routing*, por exemplo, o OSPFv2 tem o *daemon* ospfd. Ou seja, o quagga utiliza a arquitectura demonstrada na figura seguinte, com a flexibilidade digna dum software *open-source*:

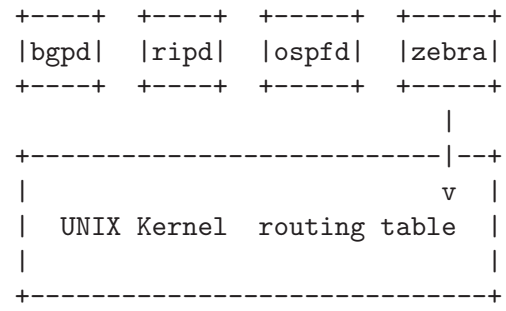


Figura1 - Arquitectura de um Sistema Quagga

(retirada do manual de referência Quagga 0.99.4, 7/2006 - Kunihiro Ishiguro)

Com esta arquitectura tem-se: extensibilidade, dada à facilidade de adicionar suporte a novos protocolos com a simples construção do *daemon* próprio; e modularidade, dado que o sistema encontra-se dividido em módulos a facilidade de manutenção aumenta visto que podem-se confinar os problemas respectivos a módulo, apenas a esse módulo.

A comunicação entre os vários *daemons* com o zebra, dá-se através de um protocolo denominado por ZEBRA PROTOCOL. Esta comunicação é tipicamente feita utilizando sockets TCP sobre o *localhost* (127.0.0.1), sendo que o zebra escuta no porto 2601 e, por exemplo, o ospfd escuta sobre o 2604. Isto consiste numa das grandes vantagens da utilização deste sistema, que é a de se poderem executar *daemons* em máquinas remotas, estando o zebra centralizado noutra.

Cenário de Testes

A máquina virtual que se tem vindo a desenvolver (a que se vai referir neste capítulo como QUAGGA) necessitará de duas interfaces. Contudo até agora só existe a interface eth0 no QUAGGA, devidamente configurada em /etc/conf.d/net como "*config_eth0 = ("dhcp")* ", sendo o ip atribuido ao grupo 2 o 10.62.75.132. A segunda interface eth1 foi então criada (cd /etc/init.d; ln -s net.lo net.eth1; rc-update add net.eth1 default) e configurada, em /etc/conf.d/net com o último endereço da rede que me foi atribuída 10.62.74.176/28 "*config_eth1 = ("10.62.74.190 netmask 255.255.255.240")* ". Outra configuração necessária é a activação da máquina para efectuar o re-encaminhamento de pacotes de uma interface para a outra. Isto é normalmente feito da seguinte forma: #echo "1" > /proc/sys/net/ipv4/ip_forward ou #sysctl -w net.ipv4.ip_forward=1. Contudo, apesar disto funcionar, só o faz até que a máquina se reinicie. Caso se pretenda que a alteração seja permanente há que inserir manualmente no ficheiro /etc/sysctl.conf a entrada "net.ipv4.ip_forward=1", entrada essa que já deverá constar no ficheiro, mas com o valor por omissão zero.

Em relação às redes interna (10.62.74.176/28) e externa (10.62.75.132/24), colocou-se o eth0 ligado à rede externa numa configuração *bridge* na porta ethernet do pc, e colocou-se o eth1 ligado à rede interna numa configuração também *bridge* na placa de rede wireless do pc, onde se criou uma rede ad-hoc com WEP128 (apenas para impedir que qualquer um acedesse à rede). Isto é melhor explicado na seguinte figura:

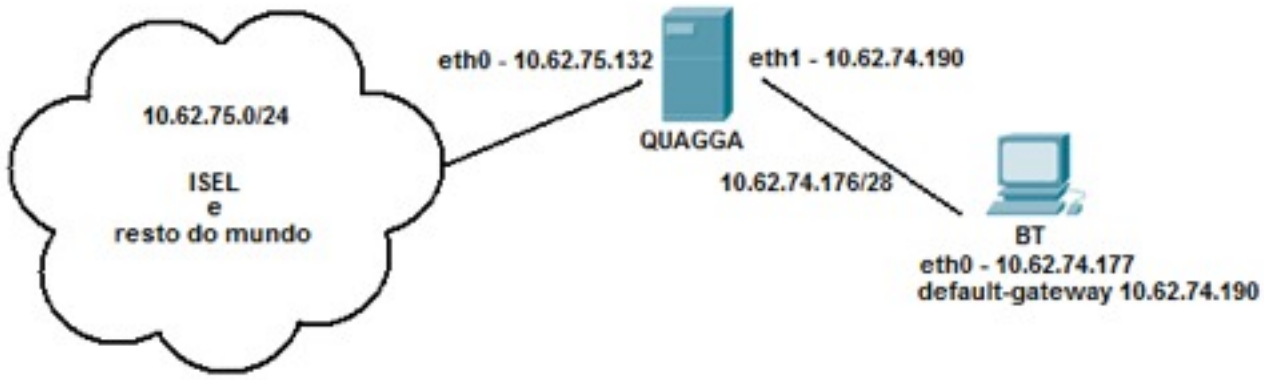


Figura2 - Estrutura lógica do cenário montado

Como se pode constatar pela figura anterior, é ainda necessária uma estação interna (a que chamaremos de BT - backtrack) à rede privada que me foi atribuída (10.62.74.176/28). Para esse efeito, foi utilizada uma imagem de um backtrack, apenas porque era uma imagem que já tinha no pc, e porque o objectivo é o de testar a ligação da rede interna com o exterior, o que se faz com um simples ping. No BT foi necessário configurar a sua única interface eth0, ligada fisicamente através da máquina virtual à placa de rede wireless, com o ip estático 10.62.74.177 (primeiro disponível) visto que o QUAGGA não tem um servidor DHCP que forneça essa informação. Outra informação importante dada por servidores DHCP é o *default gateway*, que neste caso terá também de ser configurada manualmente (route add default gateway 10.62.74.190). Para verificar se tudo ficou bem configurado, para além do comando #ifconfig eth0, foi executado o comando #netstat -rn (imprime a tabela de routing do kernel sem tentar resolver os ips recorrendo a DNS) tendo sido verificado o seguinte output:

Kernel IP routing table

Destination	Gateway	Genmask	Flags	MSS Window	irtt	Iface
10.62.74.176	0.0.0.0	255.255.255.240	U	0 0	0	eth0
0.0.0.0	10.62.74.190	0.0.0.0	UG	0 0	0	eth0

Configuração do sistema

Para o cenário que se está a desenvolver, o protocolo de *routing* utilizado será o OSPFv2. Isto significa que ter-se-á de configurar tanto o zebra (obrigatório) como o ospfd.

Em relação às configurações relativas ao *routing* o zebra e os outros *daemons* (neste caso ospfd) permitem o acesso via telnet, onde será apresentado um terminal que busca a total semelhança com o Cisco IOS. Esta é outra grande vantagem do Quagga, que faz com que a transição de administradores habituados ao Cisco IOS, que é o sistema mais utilizado no mercado, seja feita de forma transparente. Outra vantagem do Quagga é o de fornecer um terminal especial, denominado por vtysh, que abstrai o administrador das inúmeras transições de um terminal de um *daemon* para outro. Este vtysh recebe os comandos e conforme o contexto destes, assim os reencaminha para um *daemon* ou para outro. A gravação das configurações feitas num terminal é feita com o comando *write*, que guarda-as nos ficheiros de configuração respectivos a cada *daemon* na directoria /etc/quagga/.

Quanto à configuração do *daemon* propriamente dito, tem-se o ficheiro /etc/conf.d/zebra onde temos a opção ZEBRA_OPTIONS onde se indicam os parâmetros a passar ao *daemon* quando se lhe colocar em execução. É preciso indicar que a aplicação (zebra) se executará em modo *daemon* (-d) podendo-se assim executá-lo em *background*; é preciso indicar qual o ficheiro de configuração, com as configurações iniciais, e onde deverá armazenar as configurações alteradas no acesso ao seu terminal (-f); é preciso indicar exactamente onde ficará o ficheiro com o seu *process id* (-i); é também preciso indicar o endereço (-A), o porto (-P) onde deverá esperar que venham acessos ao terminal, o utilizador e o grupo cujas permissões serão atribuídas ao *daemon* zebra em execução.

A ordem de tarefas: 1º acede-se a todos os terminais (ou só a um - vtysh) e acrescentam-se as configurações todas que se pretendem; 2º armazenam-se as configurações feitas com o comando *write*.

Eis o ficheiro de configuração do ospfd (/etc/quagga/ospfd.conf):

```
! OSPFd sample configuration file
!
!
hostname ospfd
password zebra
enable password xpto_123
!
interface eth0
  ip ospf authentication message-digest
  ip ospf message-digest-key 20 md5 mesmosimples
!
router ospf
  network 10.62.74.176/28 area 75
  network 10.62.75.0/24 area 75
  area 75 authentication message-digest
  area 75 default-cost 1000
  passive-interface eth1
!
```

Teste final

O teste final basicamente consiste em testar a ligação da rede interna à externa, o que se vai fazer com um ping do BT a uma estação externa à rede.

Após todas as configurações previamente descritas no QUAGGA, foram colocados em execução tanto o zebra (#/etc/init.d/zebra start #rc-update add zebra default) como o ospfd (#/usr/sbin/ospfd -d -u daemon -g daemon). Para se constatar que o ospf funciona bem entrou-se no terminal do ospf via telnet (normalmente #telnet 127.0.0.1 ospfd mas neste caso #vtysh) onde, após se fazer #enable, executou-se o comando #show ip route pelo que se obteve o seguinte output:

```
server# sh ip route
Codes: K - kernel route, C - connected, S - static, R - RIP, O - OSPF,
       I - ISIS, B - BGP, > - selected route, * - FIB route

O   0.0.0.0/0 [110/1010] via 10.62.75.253, eth0, 00:00:16
K>* 0.0.0.0/0 via 10.62.75.254, eth0
O>* 10.62.74.0/28 [110/20] via 10.62.75.57, eth0, 00:00:17
O>* 10.62.74.16/28 [110/20] via 10.62.75.57, eth0, 00:00:17
O>* 10.62.74.160/28 [110/110] via 10.62.75.131, eth0, 00:00:11
O   10.62.74.176/28 [110/10] is directly connected, eth1, 00:00:17
C>* 10.62.74.176/28 is directly connected, eth1
O   10.62.75.0/24 [110/10] is directly connected, eth0, 00:00:17
C>* 10.62.75.0/24 is directly connected, eth0
K * 127.0.0.0/8 is directly connected, lo
C>* 127.0.0.0/8 is directly connected, lo
```

Para verificar quais os *neighbors* executou-se o comando #show ip ospf neighbor:

```
server# sh ip ospf  neighbor

Neighbor ID     Pri   State           Dead Time   Address         Interface            RXmtL RqstL DBsmL
10.62.75.57       1   Full/Backup     00:00:32    10.62.75.57     eth0:10.62.75.132    1     0     0
10.62.74.174      1   Init/DROther    00:00:39    10.62.75.131    eth0:10.62.75.132    0     0     0
10.62.75.254      1   Full/DR         00:00:38    10.62.75.253    eth0:10.62.75.132    0     0     0
```

Finalmente, para verificar a ligação do BT com redes externas, foi executado no próprio BT o comando #ping -c 1 -R 193.137.220.1:

```
PING 193.137.220.1 (193.137.220.1) 56(124) bytes of data.
64 bytes from 193.137.220.1: icmp_seq=1 ttl=61 time=4.86 ms
RR:     10.62.74.177
        10.62.75.132
        193.137.221.3
        193.137.220.125
        193.137.220.1
        193.137.220.1
        193.137.221.10
        10.62.75.253
        10.62.74.190


--- 193.137.220.1 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 4.869/4.869/4.869/0.000 ms
```

Avaliação Final

O Quagga tem várias vantagens trazidas pela sua arquitectura, que a tornam muito flexível - dado por exemplo se poderem executar *daemons* em máquinas diferentes - e também que a tornam modular e consequentemente menos exposta a erros globais que contaminem todo o sistema. O suporte ao IPv6 é outra funcionalidade que demonstra a intenção deste sistema em ser uma opção não só agora como no futuro. Uma funcionalidade que desperta interesse é o suporte a MPLS, o que demonstra empenho por parte da equipa que desenvolve o Quagga em trazê-lo para o mesmo patamar de soluções mais caras e há mais tempo no mercado.

Contudo, o Quagga também tem, pelo menos ainda, algumas desvantagens. Um exemplo é o facto do sistema não suportar alguns comandos básicos do Cisco IOS como é o caso do #enable secret *xpto*. Outro exemplo é o facto do Quagga não suportar múltiplos processos, no caso verificado, de OSPF. Estas desvantagens obrigam a que se conclua que, apesar deste sistema ser uma muito boa solução para ambientes menores, quando a complexidade aumenta, tanto a nível de rede, como a nível de segurança, torna-se necessária a passagem a soluções dedicadas e proprietárias com maiores funcionalidades.

Firewall Iptables

Um mínimo de segurança é praticamente obrigatório nos dias que correm. E, numa boa e sólida implementação da chamada segurança em camadas, o *firewall* é a primeira linha de defesa. É deste facto que advém toda a importância desta ferramenta.

A implementação iptables da Netfilter é uma *stateful packet inspection* (SPI) *packet filtering firewall*, ou seja, é capaz de filtrar pacotes da rede enquanto mantém estado sobre o contexto destes mesmos pacotes. Isto é importantíssimo para a detecção de ataques típicos a *stateless packet filters* (ataques explicados mais à frente) que não mantêm estado. Não obstante o facto do iptables ser um SPI, este contém extensões que lhe permite restringir e/ou aceitar acessos com base em *strings* contidas no interior do datagrama (camada 7), e também com base nos endereços MAC (camada 2). No entanto, a análise, nomeadamente aos dados da camada de aplicação degradam consideravelmente o desempenho da firewall, e nunca substituirá um proxy que perceba de raiz o protocolo aplicacional que se pretenda analisar.

Um dos ataques que se fazem aos *stateless packet filters*, aproveitando o facto destes não manterem estado das ligações, é explorar as regras que se inserem na firewall para permitir que as respostas a pedidos internos passem pela firewall. Isto pode ser feito de duas formas. A primeira é permitindo que passem, no sentido de fora para dentro, todos os pacotes com origem em portos inferiores a 1024 e com destino a portos superiores a esse mesmo valor. A segunda consiste em permitir passagem a todos os pacotes que tenham os bits ACK ou RST (cabeçalho TCP) activos. Qualquer um dos casos é facilmente explorado. O primeiro exemplo não consegue negar acesso a atacantes que tenham conseguido instalar *backdoors* em computadores dentro do perímetro de segurança que fiquem à escuta de ligações em portos acima do 1024. O segundo cenário é batido pela capacidade de se manipularem os cabeçalhos dos protocolos, nomeadamente activar o ACK do TCP. Ambos estes cenários são fraquezas que já não se põem no caso de SPIs, dado que se, por exemplo, se receber um pacote com o ACK activo, mas não se tenha em memória a informação de que houve um pacote de dentro para fora que justificasse esse ACK, é negado o acesso ao pacote.

Outro ataque específico aos *stateless packet filters* é a exploração da divisão de pacotes IP, que excedam o MTU de um canal, em fragmentos. O problema da fragmentação advém do facto da informação relativa ao cabeçalho TCP/UDP (ou outro protocolo sobre IP), importante para a filtragem, só aparecer no primeiro fragmento - o fragmento 0. Consequentemente, as primeiras implementações de firewalls (*stateless packet filters*) deixavam passar todos os fragmentos iguais ou superiores a 1, baseando-se no facto de que caso não tenha sido permitido o acesso ao fragmento 0, os seguintes não fariam sentido e seriam descartados pela máquina destino. Contudo esse não foi o caso, dado que várias implementações erróneas do TCP/IP, em vários sistemas operativos, ordenavam e reconstruíam o pacote mesmo que faltasse o fragmento 0. Isto significa que um atacante apenas tem de enviar pacotes fragmentados com o cabeçalho TCP/UDP/ICMP no fragmento n, sendo n>0. Outras implementações, ao não receber o fragmento 0, resultavam num *crash* do sistema. Estes problemas não se põem com SPIs, dado que aos pacotes fragmentados com o mesmo identificador de fragmentação é-lhes aplicada a mesma decisão tomada para o fragmento 0 correspondente. No caso específico ao iptables, se se utilizar o *connection tracking* (módulo que efectivamente o torna no SPI) ou NAT, tem-se a garantia de que o pacote é juntado de novo antes de se atingir o código de filtragem.

O percurso de um pacote através do Iptables

Explicar-se-ão três casos na figura 3: recepção, emissão e re-encaminhamento de um pacote.

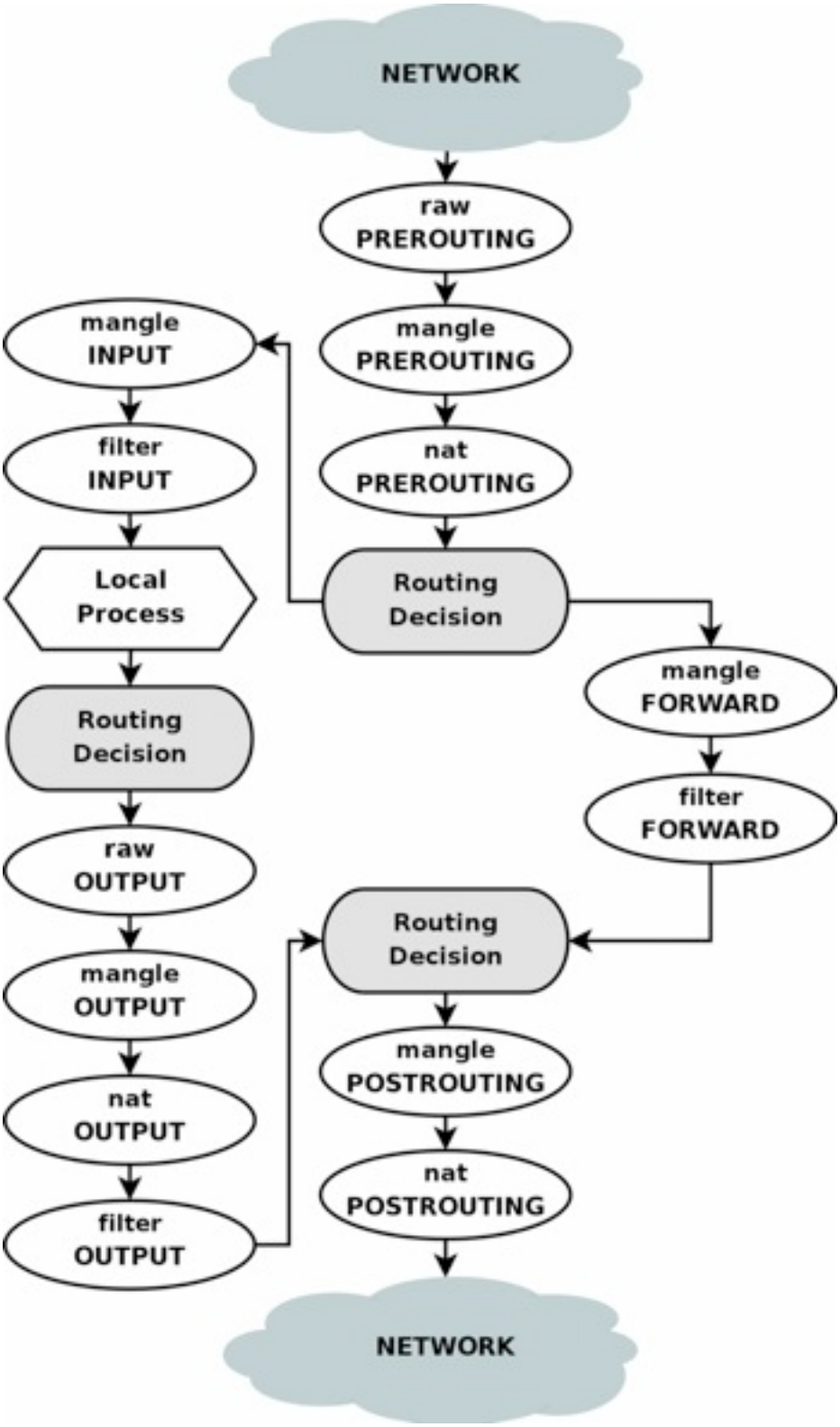


Figura 3 - Caminho efectuado por um pacote atravessando o Iptables
http://www.frozentux.net/iptables-tutorial/iptables-tutorial.html

Chegando um pacote da rede, este pode ser acedido e/ou manipulado em diversos pontos (retratados na figura 3 pelas formas ovais). De salientar que não é obrigatória nenhuma acção em qualquer um desses pontos. O primeiro ponto é a tabela *raw* da cadeia PREROUTING. Este ponto é utilizado para aceder a pacotes antes de se aplicar o *connection tracking* (ver secção "Máquina de Estados" a seguir), podendo-se inclusivamente indicar que não se quer que o pacote em causa passe pelo *connection tracking*. A seguir, entra em acção o *connection tracking* e só depois se chega ao segundo ponto, a tabela *mangle* também da cadeia PREROUTING. Nos pontos de *mangling* é dada a opção de se alterar o pacote, neste caso por exemplo, alterar o TOS do cabeçalho IPv4. O terceiro ponto é a tabela *nat* ainda da cadeia PREROUTING. Neste colocam-se, caso necessário as regras de DNAT. Isto tem de ser feito antes do re-encaminhamento para que o pacote seja re-encaminhado de acordo com o novo IP destino. E o passo seguinte, é exactamente a decisão de re-encaminhamento, onde apenas se faz a distinção entre se o pacote tem destino local ou remoto.

Caso o pacote tenha destino local, o próximo ponto é o da tabela *mangle* da cadeia INPUT. Neste ponto são efectuadas alterações a cabeçalhos, depois do pacote ter sido encaminhado e antes de ser recebido pela aplicação. O ponto seguinte é o da tabela *filter* e cadeia INPUT. Neste são aplicados os filtros necessários. De notar que independentemente de qual a interface receptora, se o pacote tem destino local, passa obrigatoriamente por este ponto. Finalmente, caso o pacote seja aceite pelas regras de filtragem, chega então à aplicação.

Caso o pacote tenha destino remoto, o próximo ponto é o da tabela *mangle* da cadeia FORWARD. Neste pode ser alterado o pacote, de acordo com necessidades muito específicas, depois de ter passado pelo encaminhamento inicial mas antes da última decisão de encaminhamento (dado que neste ponto podem ser feitas alterações ao pacote - ex TOS - que alterem a interface de saída). Em seguida tem-se o ponto da tabela *filter* ainda do cadeia FORWARD, por onde passam todos os pacotes que sejam re-encaminhados caso sejam aceites pelas regras aqui colocadas. A seguir é tomada nova decisão de encaminhamento, devido a possíveis alterações inseridas pelo ponto da tabela *mangle* da cadeia FORWARD já referida. Finalmente, o pacote passa por dois pontos, o da tabela *mangle* da cadeia POSTROUTING, e o da tabela *nat* da mesma cadeia. Neste último, é onde normalmente se faz o SNAT, tipicamente usado para permitir acesso, aos servidores internos mapeados por DNAT, dos computadores também internos que tentem aceder aos IPs públicos desses servidores (IP da interface pública do router/firewall externo), ou para esconder IPs internos do exterior.

O último cenário a ser considerado é do envio de um pacote. Nestes casos, o pacote é submetido a uma decisão de encaminhamento e, a seguir, passa pelas tabelas *raw*, *mangle*, *nat*, e *filter* da cadeia OUTPUT. Na tabela *raw* é onde se pode aceder ao pacote antes deste passar pelo *connection tracking* que tem lugar logo em seguida, para atribuir estado ao pacote antes deste sair da máquina. A seguir o pacote passa pela tabela *mangle*, e *nat* que têm funções idênticas às já explicadas nas cadeias anteriores. Após o *nat*, dado que o pacote pode ter sido alterado tanto pelo *mangle* como pelo próprio *nat*, há que passar novamente por outra decisão de re-encaminhamento. Finalmente o pacote passa pela filtragem e pela cadeia POSTROUTING explicada já no parágrafo anterior.

Apenas de salientar que, na inserção de regras utilizando Iptables, caso não se indique explicitamente que tabela se pretende utilizar, assume-se por omissão a tabela *filter*.

Máquina de Estados - Connection Tracking

O conntrack é um módulo do kernel (apesar de poder também poder ser compilado como parte interna deste) que permite ao kernel atribuir um estado a um pacote recebido, por enviar, ou em trânsito. Isto permite que o Iptables configure regras que filtrem pacotes com base nesses estado. Os estados acessíveis no modo *user space* são: NEW, ESTABLISHED, RELATED, INVALID e UNTRACKED. Este módulo, apesar de poder ser desactivado, hoje em dia já não se lhe desactiva devido às falhas inerentes aos *stateless packet filters*. Quando activo, o conntrack actua sempre nas cadeias PREROUTING (para pacotes recebidos ou em trânsito) e OUTPUT (para pacotes a enviar), e garante que haja sempre desfragmentação de fragmentos recebidos antes de proceder com a filtragem.

Em seguida, para melhor se perceber todo o poder da flexibilidade do Iptables, vai-se tirar partido de algumas das suas extensões, desenvolvendo-se cenários que utilizem alguns dos vários módulos desta implementação.

Cenário 1 - Firewall base

Este cenário tem como objectivo demonstrar e justificar a implementação de uma firewall com as regras mínimas para proteger a rede interna contra os ataques mais conhecidos. A listagem da firewall pode ser vista a seguir:

- Anexo B Output 1 (e código B.1 - script que originou a firewall) -

Em primeira análise, a cadeia INPUT, ou seja, todos os pacotes que tenham como destino a própria firewall. As regras 1 e 2 registam e negam respectivamente o acesso a pacotes no estado INVALID. Destes podem constar mensagens de erro ICMP que não estejam relacionadas com quaisquer ligações, ou pacotes que não tenham sido identificados devido à falta de recursos como memória na máquina onde se encontra o firewall.

As regras 3 e 4 registam e negam acesso a pacotes IP que contenham opções no cabeçalho. Isto resulta de um tipo de ataque muito popular em casos onde a autenticação é feita com base em endereços IP de origem (ex: TCP *wrappers* em sistemas UNIX; e ACLs da IIS em sistemas NT). Falsificar o seu IP (IP *spoofing*) por si só não é suficiente na maior parte das vezes onde o atacante estará fora da rede do IP falsificado, ou seja, no retorno da comunicação os pacotes não passarão por ele. Contudo, se se utilizar o *source routing*, pode-se utilizar o *loose routing* para fazer com que os pacotes de resposta passem pelo próprio atacante. Daí a importância de negar estes pacotes em firewalls de fronteira, dado que normalmente o *debug* (para o qual estas opções foram concebidas) é feito na rede interna, e não entre a rede interna e externa (cenário típico de um ataque).

A regra 5 aceita pacotes cujas ligações já tenham sido estabelecidas (ESTABLISHED), ou seja, comunicações que tenham sido originadas pela própria firewall, e também aceita pacotes que estejam relacionados (RELATED) com outras comunicações já estabelecidas, como são os casos de FTP em modo activo, e mensagens de erro ICMP como *host unreachable*. A regra 6 aceita pacotes cuja origem seja a inteface de *loopback*.

As regras 7 e 8 aceitam *queries* DNS (udp) ao porto 53. De salientar que isto apenas foi feito para facilitar a implementação do cenário de testes com máquinas virtuais, pois a filtragem de pacotes deve ser o único objectivo da máquina onde se implementa uma firewall. Isto porque não se quer deixar toda a rede interna vulnerável caso um atacante consiga penetrar dentro da firewall devido a uma vulnerabilidade de outra aplicação, neste caso o Bind9. Outro pormenor importante é o de proteger a resolução de nomes DNS de ataques de Negação de Serviço (DoS). Isto não foi aqui feito porque o objectivo deste cenário base é o de demonstrar as várias funcionalidade desta firewall, sendo que esta funcionalidade de limitação de tráfego é demonstrada na cadeia FORWARD, onde são protegidos os utilizadores da rede interna de ataques *Syn Flood* e de *portscans*.

A regra 9 aceita tráfego OSPF (*protocol type* 89) para que a implementação do Quagga se comunique com os vizinhos. Neste caso, não se aplica a preocupação de uma aplicação extra na mesma máquina que a firewall, pois esta normalmente se executa na mesma que o router, função esta desempenhada com a ajuda do Quagga. A única forma de se explorar neste caso o Quagga é encontrando uma vulnerabilidade no protocolo OSPF o que, a ser possível, ter-se-ão problemas muito maiores do que somente a segurança da rede interna. Nesta regra 9 poder-se-ia utilizar o módulo ttl para filtrar pacotes que, sendo vizinhos, têm de certeza TTL igual a 1 (... -m ttl --ttl-eq 1) e também poderia, apesar de perder alguma flexibilidade, filtrar os IPs origem dos OSPF NEIGHBOURS. Porém, tanto o TTL como o IP origem podem ser facilmente falsificáveis, e como o protocolo OSPF trata da sua própria segurança (autenticidade e integridade com MD5) achou-se não ser necessário um controlo muito apertado do tráfego OSPF.

A cadeia INPUT termina com a regra 10 que regista todos os pacotes que cheguem a este ponto, ou seja, aqueles cujo acesso será negado, dado que a política de todas as cadeias (INPUT/OUTPUT/FORWARD) são de negação.

Na cadeia OUTPUT por onde passam todos os pacotes que tenham sido gerados pela própria máquina, as regras 1 e 2 têm as mesmas funções que as respectivas à cadeia INPUT. As duas regras seguintes (3 e 4) permitem restringir tráfego de saída a apenas aquele que tenha os IPs origem respectivos à máquina, ou seja, de uma das interfaces. Isto é apenas um pormenor (irrelevante se se assumir que a máquina é gerida apenas por

administradores de confiança) que impossibilita o *spoofing* a partir da máquina firewall. A última regra 5 aceita o envio de tráfego à interface *loopback*. De notar que aqui também pode ser inserida uma 6ª regra para registar os pacotes que atinjam o fim da lista e sejam negados.

Um ponto importante de salientar é o do registo dos pacotes - *logging*. É importante que o administrador perceba que podem ser feitos ataques DoS, neste caso ao disco rígido, inundando os ficheiros de *log* com pacotes até que não mais haja espaço em disco para registar mais nenhum pacote, podendo um atacante efectuar uma enorme quantidade de ataques sem sequer se preocupar com o IP *spoofing*, o que também quer dizer que não se tem de preocupar com regras que neguem acesso a pacotes que utilizem *loose routing*. Existem módulos, como o nth, que dão a opção à firewall de não registar todos os pacotes, mas apenas 1 em cada 10 por exemplo. Outra forma de combater este tipo de ataques é o de automatizar, recorrendo a aplicações de monitorização (ex: Nagios), a compactação de ficheiros que atinjam um determinado tamanho, ou mesmo movê-los para um dispositivo de armazenamento central (ex: NAS).

A última cadeia é a de FORWARD, por onde passam todos os pacotes que nem têm origem na própria máquina, nem têm-na como destino. As regras 1 e 2 registam e negam acesso a pacotes no estado INVALID, a regra 3 impede a passagem de *broadcasts*, desta forma evitando ataques *Smurf*. Nestes ataques é enviado por um atacante um ICMP *echo request* com destino broadcast e origem falsificada com o IP da vítima. Quando os computadores todos da rede interna responderem com o ICMP *reply*, a vítima será inundada de pacotes (DoS). O impedimento do encaminhamento de pacotes com destino *broadcast* é suportado a nível do próprio *kernel*, contudo em versões mais antigas isso poderá não ser verdade.

As regras 4 e 5 impedem a passagem de pacotes com opções no cabeçalho IP, pelas mesmas razões anteriormente apresentadas, desta feita protegendo a rede interna.

A regra 6 foi inicialmente concebida para proteger a própria firewall, mais especificamente no porto 22 (administração remota via SSH), de ataques *Syn Flood*. Neste o atacante explora uma fraqueza do protocolo TCP, no *three-way-handshake*, onde envia um syn, fazendo com que o servidor armazene estado (consumindo memória) com os dados da ligação. Contudo, após o servidor responder com o syn+ack, o atacante já não mais responde. Isto é tipicamente feito falsificando o IP origem para um IP que não esteja atribuído. Este era o único serviço a correr sobre TCP e consequentemente vulnerável a este tipo de ataques. De realçar o facto do próprio sistema Linux ter formas de evitar este tipo de ataques (/etc/sysctl.config - syn cookies, redução de timeouts, etc) porém, sendo o objectivo do trabalho a demonstração de como fazê-lo com o Iptables, optou-se por fazer isso mesmo com a regra 6. Nesta, são encaminhados os pacotes TCP que tenham estado NEW, tipicamente as características de um pacote que pretenda iniciar uma sessão SSH. Esses pacotes são então encaminhados para uma nova cadeia de regras chamada syn_flood_protect. Nesta cadeia utiliza-se o módulo *limit* para limitar o fluxo desses pacotes a 1 por segundo. Dado eu ser o único administrador da máquina pareceu-me ser perfeitamente razoável esse valor.

Apesar de tudo o que foi dito no parágrafo anterior, esta regra encontra-se não na cadeia INPUT mas sim na FORWARD. Isto para proteger os servidores internos com serviços sobre TCP. Esta regra só não se encontra na cadeia INPUT porque não é lá necessária visto que não se permite acesso ao porto 22 antes de se autenticar o cliente (método SPA explicado mais à frente).

A regra seguinte é a 7 que permite a passagem de qualquer tráfego interno para a rede externa. A regra 8 permite a passagem de pacotes relacionados com ligações criadas a partir de dentro, e a regra 9 regista os pacotes que chegaram ao fim da lista e serão descartados dada a política de negação comum a todas as cadeias principais.

De salientar três ataques aos quais nem a firewall nem a rede interna estão vulneráveis: Ping of Death - onde se envia um ICMP echo request com mais de 65536 bytes obrigando implementações antigas de SOs a terem comportamentos estranhos como o bloqueio completo; Teardrop - onde se adulteravam os *fragment offsets* de forma a também explorar falhas em implementações antigas do stack TCP/IP em SOs; e Land - onde se envia um pacote syn com o endereço e porto origens iguais aos do destino (isto provocava que o Win95 se bloqueasse). Estes ataques não têm efeito sobre o firewall, nem sobre a rede interna, porque o *kernel* utilizado é considerado "moderno" no que diz respeito às falhas que estes ataques exploravam, e também porque, no caso do Ping of Death e Teardrop, utiliza-se o *conntrack* que garante a desfragmentação dos fragmentos antes

destes serem re-encaminhados, local ou remotamente, e consequentemente a má formação do pacote será logo detectada (sendo então descartada). Caso o *conntrack* não fornecesse protecção, um simples "iptables -A INPUT -p icmp --icmp-type echo-request -m length --length 86:0xffff -j DROP", não permitindo ICMP echo request maiores que 85 bytes seria mais do que suficiente.

Single Packet Authorization - SPA

Esta técnica consiste em deixar um *daemon* à escuta de um pacote com dados que servem para autenticar um cliente. Quando esse pacote chega, o *daemon* insere uma regra na firewall permitindo acesso ao porto 22, por parte de um IP específico e por um determinado espaço de tempo. A implementação utilizada foi a *Forward Knock Operator* (fwknop) por ser uma das melhores a nível de implementação de autenticação considerada segura. Nomeadamente, protege de ataques de repetição, onde um atacante que obtenha uma cópia de um pacote já enviado, ao re-enviá-lo o servidor aceita-o como válido. O Fwknop protege-se deste ataque com a inserção de 16 bits de dados aleatórios no pacote. Juntamente com mais alguns outros dados, é calculado um *hash* desses dados com uma palavra-chave simétrica. Esse *hash* é anexado ao pacote que é então enviado ao servidor.

Com esta técnica não mais foi necessário proteger o porto 22 de *syn floods*. Uma das vantagens do SPA é também tornar o sistema operativo, da máquina que contém a firewall, irreconhecível, dado que apenas o porto 53 (que como já referido sequer lá deveria estar) que apenas responde a *queries* DNS (udp). Isto foi testado com o NMap (http://nmap.org/ -> nmap -O -p 1-1024 10.62.75.132) que respondeu: "*Too many fingerprints match this host to give specific OS details*". Outra vantagem é o de não mais ser necessário proteger o porto 22 (Secure Shell - SSH) de falhas/vulnerabilidades da implementação utilizada: o OpenSSH (www.google.com/search?q=OpenSSH+Vulnerability+site%3Acert.org) como é o caso do SSL *request floods* que se resume no mesmo que *syn floods* mas a nível de estabelecimento de ligação SSL (sobre o qual passa o SSH).

O cliente executado sobre um MacBook Pro deverá invocar:

/Users/andreflima/bin/fwknop -A tcp/22 -a 10.62.75.4 -D 10.62.75.132

Enquanto que no servidor já deverá estar a se executar:

/usr/bin/perl -w /usr/sbin/fwknopd -d -v # para poder ver a informação de debug

O servidor foi configurado através dos ficheiros /etc/fwknop/access.conf (informação relativa ao acesso do cliente como a palavra passe) e na mesma pasta fwknop.conf (com informação como o modo de escuta - PCAP, ULOG, syslog; que comandos executar na recepção de um pacote válido e após o *timeout*; e também o *timeout* que por omissão é de 30 segundos).

Protecção contra Port Scans

A preocupação contra *portscans* não se coloca no caso da cadeia INPUT pois os pacotes enviados pelo atacante serão todos descartados, com a excepção do porto 53 UDP. Após um teste com:

nmap -O -p 1-1024 10.62.75.132

e colocando a protecção contra *syn floods* na cadeia INPUT para efeitos de teste, notou-se que os pacotes , dada a sua rapidez de envio, foram maioritariamente (cerca de 2000/20XX) descartados no final da cadeia syn_flood_protect. Consequentemente considerou-se que esta cadeia inicialmente criada para a protecção contra *syn floods* teve êxito na intercepção de *portscans*, dada a sua característica alta velocidade de envio facilmente confundível com ataques DoS.

A utilização do módulo *hashlimit* do Iptables foi considerada, contudo esse módulo pode ser enganado com IP *spoofing* onde o atacante utiliza todos os IPs da gama atribuída à sua rede e utilizando um *sniffer* esperar pelas respostas, daí que se tenha optado pela manutenção no uso do módulo *limit*.

Cenário 2 - Controlo Parental

Neste cenário pretende-se demonstrar alguns módulos que poderão ser utilizados para restringir acessos à rede externa (Internet) aos mais pequenos. Isto será feito limitando as horas de navegação, estabelecendo quantidades limites de *downloads*, e restringindo acesso a conteúdos com base em strings.

A limitação do tempo de navegação é feita utilizando o módulo *time*:

```
# iptables -A FORWARD -i eth1 -o eth0 -m time --timestart 12:00 --timestop 21:00 --days Mon,Tue,Wed,Thu,Fri -j ACCEPT
# iptables -A FORWARD -i eth1 -o eth0 -m time --timestart 9:00 --timestop 23:00 --days Sat,Sun -j ACCEPT
```

Permite-se assim que se aceda à rede exterior nos dias úteis do meio-dia às 21h e nos fins de semana das 9h às 23h. Quanto à limitação da quantidade de downloads, recorre-se ao módulo *quota*:

```
# iptables -A FORWARD -i eth0 -o eth1 -p tcp --sport 80 -m state --state ESTABLISHED -m quota --quota 52428800 -j ACCEPT
# iptables -A FORWARD -i eth0 -o eth1 -p tcp --sport 80 -m state --state ESTABLISHED -j DROP
```

Permite-se assim que, no *download* de páginas web (http), se descarreguem até 50MB (1024 *bytes* x 1024 x 1024 = 52428800 *bytes*).

Quanto à restrição de conteúdo, recorre-se ao módulo *string*:

```
# iptables -A FORWARD -i eth1 -o eth0 -m string --string 'google.com' -j DROP
```

Restringe-se assim a passagem de quaisquer pacotes que contenham a string "www.*google.com*", a não ser que se utilizem técnicas de evasão de *Network Intrusion Detection Systems* (NIDS), mesmo as mais básicas como a substituição de caracteres. Um exemplo disto é, ao invés de se tentar aceder a www.google.com, pode aceder a "%77%77%77%2E%67%6F%6F%67%6C%65%2E%63%6F%6D".

É importante que se note que a protecção contra esta e várias outras técnicas de evasão de NIDS, neste caso técnicas de substituição de caracteres, são impraticáveis para um *packet filtering firewall*, dado que ter-se-iam de inserir inúmeras regras na firewall para cada caso, o que em termos de eficiência é inaceitável. Se realmente se pretende fazer a detecção de *worms*, com base em assinaturas como é o caso, há que utilizar um NIDS (ex: Snort/Cisco Secure IDS) que está muito mais bem preparado para negar acesso a pacotes mesmo que utilizem estas técnicas que, para um NIDS, é relativamente básico. Outra vantagem dos NIDS, nestes casos, em relação às firewalls é o desempenho.

Muitos também tentam utilizar este módulo *string* para impedir acesso a certos métodos do HTTP, por exemplo, o POST. O autor deste módulo, Emmanuel Roger, salienta que isto é muito melhor feito por um *proxy* do protocolo em causa, neste caso dever-se-ia de utilizar por exemplo o *Squid* para o HTTP, e também realça o facto deste módulo ter sido criado apenas com o intuito de colocar "pacotes interessantes" numa *queue* para posterior análise por parte do administrador.

Cenário 3 - Balanceamento de Carga

Neste cenário pretende-se balancear a carga entre dois servidores DNS internos. Isto pode ser feito da seguinte forma:

```
# iptables -A PREROUTING -i eth0 -p udp --dport 53 -m nth --counter 0 --every 2 --packet 0 -j DNAT --to-destination 10.62.74.177:53
# iptables -A PREROUTING -i eth0 -p udp --dport 53 -m nth --counter 0 --every 2 --packet 1 -j DNAT --to-destination 10.62.74.178:53
```

Este módulo nth contém 16 contadores e está a utilizar o primeiro (--counter 0). O contador é incrementado até que atinja o valor 2, passando imediatamente a zero outra vez (--every 2). Cada regra é executada se o seu valor fixo (--packet x) corresponder ao mesmo do contador.

É importante notar que, caso haja comunicação cujos posteriores pacotes necessitem ser re-encaminhados da mesma forma, há que balancear apenas o primeiro pacote com NAT (-m state --state NEW) sendo que a tabela de NAT tratará de re-encaminhar os restantes pacotes para o mesmo servidor. Caso não se re-encaminhe apenas o primeiro pacote, corre-se um risco muito elevado de se terem pacotes a irem para um servidor e

outros a irem para servidores diferentes, o que não funcionaria por exemplo no caso de protocolos que usem o TCP ou mesmo de alguns UDP como é o caso do NFS ou TFTP.

Outra forma de fazer o balanceamento, para demonstrar a colocação de regras noutra tabela que não a por omissão, a tabela *filter*, é a seguinte:

```
# iptables -A PREROUTING -i eth0 -p udp --dport 53 -t mangle -m random --average 50 -j ROUTE --gw 10.62.74.177
# iptables -A PREROUTING -i eth0 -p udp --dport 53 -t mangle -j ROUTE --gw 10.62.74.178
```

Desta feita, na tabela *mangle* da cadeia PREROUTING, a primeira regra re-encaminha o DNS *query* para o servidor 10.62.74.177 com probabilidade 50% (--average 50) sendo que, as que não passem pela primeira, sejam re-encaminhados ao 10.62.74.178.

Cenário 4 - Falta de confiança

Este cenário reflecte uma situação onde tem-se um administrador sénior, e um júnior. Neste caso, o sénior não confia ainda o suficiente nas capacidades/habilidades do novo recruta, pelo que não tem intenção nenhuma de lhe dar permissão (ex: sudo) para executar o comando /sbin/iptables. Contudo em determinadas ocasiões o administrador sénior pode ter de sair e querer deixar a cargo do administrador júnior uma, ou mais, regras.

```
# iptables -A FORWARD -i eth0 -p tcp -d 10.62.74.177 --dport http -m condition --condition webdown -j REJECT --reject-with tcp-reset
# echo 1 > /proc/net/ipt_condition/webdown
```

Com estas regras o administrador sénior pode dar permissão de escrita ao júnior do ficheiro /proc/net/ipt_condition/webdown de forma a que este controle única e somente a regra que controla o acesso ao servidor HTTP interno 10.62.74.177. A pasta onde este ficheiro se encontra é a única pasta onde se podem colocar estes ficheiros. Neste caso, o sénior pode deixar a cargo do júnior uma operação de manutenção do servidor HTTP, tendo com estas regras a possibilidade de rejeitar (echo 1 > ...) pedidos de clientes enquanto efectua a manutenção, podendo logo de seguida voltar a dar acesso ao servidor (echo 0 > ...).

Desempenho da Firewall

Antes de terminar é importantes se notar que o *script* (anexo B código B.1) contém regras que poderão ser reposicionadas caso os seus *hit counts* demonstrem ser maiores que as regras que se encontram à sua frente com o mesmo *jail* (ex: ACCEPT). Aliás, o posicionamento das regras foi feito de acordo com o que se acredita ser uma lógica coerente com a dos *hit counts* já referida. Outra forma de aumentar o desempenho do Iptables é, à medida que as regras aumentam, agrupá-las de acordo com certas características em novas tabelas. Um exemplo disto é, ao invés de se terem N regras filtrando com base em dados TCP, obrigando um pacote UDP a percorrê-las todas, para só então encontrar o conjunto de regras UDP, pode-se colocar uma única regra que indique que, caso o pacote seja TCP que vá então percorrer uma cadeia tcp_filter por exemplo. Desta forma um pacote UDP não mais tem de percorrer as N regras TCP até chegar às suas, mas sim apenas percorrer uma regra.

Outro aspecto importante no desempenho consiste no armazenamento das regras da firewall. Uma opção inicialmente lógica seria a de automatizar, na arranque do sistema, a execução de um *script* (como é o caso do código B.1) de forma a activar a firewall nesse instante. Contudo, o problema reside no facto de, a cada chamada ao iptables, este extrair todo o conjunto de regras existentes no *Netfilter Kernel space*, efectuar a alteração pretendida (inserção,remoção, *append*), e finalmente voltar a transferir todo o conjunto da memória para o *Netfilter Kernel space* outra vez. Todo este processo, para cada chamada ao iptables, faz perder mais tempo quanto maior for o número de regras da firewall, o que a certo ponto torna-se inconcebível. É por esse motivo que existem duas ferramentas, iptables-save e iptables-restore, que servem para armazenar e restaurar o conjunto de regras num único pedido ao *kernel*.

Conclusão

Na tentativa de se proteger uma rede, deve-se implementar a chamada **defesa em camadas**. Basicamente isto significa que um atacante terá de ultrapassar várias camadas de segurança, até que atinja o seu objectivo. Esta estratégia dá maior consistência à segurança implementada, na medida em que aquilo que se pretende defender não fica vulnerável no caso de uma camada ser atacada com sucesso. No âmbito desta defesa em camadas, a firewall é efectivamente a primeira barreira com a qual são confrontados os atacantes.

É importante deixar claro que a firewall, por si só, não é uma solução completa de segurança de redes. Daí a importância de mais camadas de segurança por detrás desta primeira. Há uma enormidade de ataques, referidos nos parágrafos seguintes, que não são susceptíveis à detecção por parte de firewalls pois, apesar destas poderem suportar funcionalidades que analisem protocolos de camadas superiores, nomeadamente a camada de aplicação, estas funcionalidades atrasam consideravelmente o processo de encaminhamento dos pacotes, tornando-se funcionalidades impraticáveis em redes com grandes quantidades de tráfego. Normalmente, caso se pretenda examinar tráfego da camada de aplicação, recorrem-se a *proxies* especializados num determinado protocolo (ex: Squid para HTTP).

Como exemplo de um ataque que os *packet filtering firewalls* não detectam, temos a recepção, por parte de um utilizador da rede interna, de um e-mail com um From: conhecido por esse utilizador. Este simples facto retira todas as suspeitas deste utilizador. Contudo, o e-mail em causa pode ser forjado e, no campo Reply-To:, conter um endereço diferente do contido no From. Esta manipulação não pode ser detectada mesmo que se tenha um *proxy* firewall no meio. Isto porque é perfeitamente legítimo, e acontece muitas vezes, querer-se enviar e-mails de várias contas, mas ao mesmo tempo apenas se pretender receber numa única conta.

E há também que considerar o factor humano. Um utilizador da rede interna pode estar descontente com a QoS (ex: largura de banda disponível a um determinado serviço) fornecida pela rede em causa. Já foram relatados casos onde utilizadores ligaram-se à rede externa (internet) através de *modems* pessoais, não se apercebendo da criação de uma *backdoor* à rede interna.

Outros exemplos de ataques não detectáveis por firewalls packet filtering são o SPAM, o XSS, o sql injection, anexos de email com vírus, exploits específicos aos protocolos das camadas superiores, entre muitos outros. Estes e outros ataques justificam fortemente a existência das camadas seguintes de segurança. Nomeadamente: Network Intrusion Detection Systems (NIDS - ex:Snort); Host Intrusion Detection Systems (HIDS - ex:Tripwire); VLANs; Cifra de canais de comunicação (ex: kerberos, VPNs); Monitoring Analysis and Response System (Cisco MARS); a construção de políticas de segurança onde deverão ser sistematizados os passos que se deverão tomar em casos de emergência (*Incident Handling*); políticas de *backup*; segurança física dos servidores e dos dispositivos de redes(routers/switches); ACLs internas; treino dos utilizadores da rede para que tenham noções básicas de segurança e nomeadamente não agirem de acordo com o parágrafo anterior, nem divulgarem as suas passwords a terceiros, nem criarem passwords facilmente quebradas por ferramentas especializadas (ex: John the Ripper); etc.

Bibliografia

S. Vermeulen, G. Goodyear. “Gentoo Linux x86 Handbook”, 19/10/2009

Internet Systems Consortium (ISC). “Bind9 Administrator Reference Manual”, 2005

Quagga, “http://www.quagga.net/”

Ramon Hontanon, “Linux Security”, Julho 2001

http://www.linuxhomenetworking.com/wiki/index.php/Quick_HOWTO_:_Ch14_:_Linux_Firewalls_Using_iptables

http://www.netfilter.org/documentation/HOWTO//packet-filtering-HOWTO.txt

http://www.netfilter.org/documentation/HOWTO//netfilter-extensions-HOWTO.txt

http://www.netfilter.org/documentation/HOWTO//netfilter-hacking-HOWTO.txt

http://www.frozentux.net/iptables-tutorial/iptables-tutorial.html

http://www.portknocking.org/view/implementations

http://www.netfilter.org/projects/ulogd/index.html

http://insecure.org/sploits/land.ip.DOS.html

http://insecure.org/sploits/linux.fragmentation.teardrop.html

http://insecure.org/sploits/ping-o-death.html

Anexo A - DNS

NAMED.CONF	EXPLICAÇÃO

logging {
channel default_syslog {
file "/var/log/named/named.log" versions 3 size 5m;
severity debug; print-time yes; print-severity yes; print-category yes;
};
category default { default_syslog; };
};
	Suporte ao logging de interrogações ao servidor.

options {
directory "/var/bind";
listen-on-v6 { none; };
listen-on port 53 { 127.0.0.1; 10.62.75.132; };
pid-file "/var/run/named/named.pid";
};
	Directoria base que contém os restantes ficheiros de configuração.

zone "." IN {
type hint;
file "named.ca";
};
	Servidor de cache para dominios que desconhece (query recursivo aos root servers).

zone "localhost" IN {
type master;
file "pri/localhost.zone";
allow-update { none; };
notify no;
};
	Resolução do domínio localhost.

zone "127.in-addr.arpa" IN {
type master;
file "pri/127.zone";
allow-update { none; };
notify no;
};
	Suporte ao reverse resolution dos ips do localhost.

zone "g2.lrcd.local" IN {
type master;
file "pri/g2.lrcd.local.zone";
allow-update { none; };
notify no;
};
	Domínio atribuido ao grupo 2.

zone "2.31.172.in-addr.arpa" IN {
type master;
file "pri/g2.lrcd.local.rev";
};
	Suporte ao reverse resolution do ip atribuido ao grupo 2.

Tabela A.1 - named.conf

PRI/G2.LRCD.LOCAL.ZONE
$TTL 1D @ IN SOA g2.lrcd.local. alima.g2.lrcd.local. (2009110401 ; Serial 3H ; Refresh 15M ; Retry 1W ; Expire - 1 week 1D) ; Minimum IN NS server.g2.lrcd.local. IN MX 10 mail.g2.lrcd.local. server IN A 10.62.75.132 andre IN CNAME server mail IN CNAME server

Tabela A.2 - g2.lrcd.local.zone

PRI/G2.LRCD.LOCAL.REV

```
$TTL 1W
@               1D IN SOA       2.31.172.in-addr.arpa. alima.g2.lrcd.local. (
                                2008122601      ; serial
                                3H              ; refresh
                                15M             ; retry
                                1W              ; expiry
                                1D )            ; minimum

                NS      server.g2.lrcd.local.
132             PTR     server.g2.lrcd.local.
```

Tabela A.3 - g2.lrcd.local.rev

dig @10.62.75.132 g2.lrcd.local	; <<>> DiG 9.4.3-P3 <<>> @10.62.75.132 g2.lrcd.local ; (1 server found) ;; global options: printcmd ;; Got answer: ;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 8655 ;; flags: qr aa rd ra;QUERY: 1, ANSWER: 0, **AUTHORITY: 1**, ADDITIONAL: 0 ;; QUESTION SECTION: ;g2.lrcd.local. IN A **;; AUTHORITY SECTION:** **g2.lrcd.local. 86400 IN SOA g2.lrcd.local. alima.g2.lrcd.local. 2008122601 10800 900 604800 86400** ;; Query time: 18 msec ;; SERVER: 10.62.75.132#53(10.62.75.132) ;; WHEN: Fri Nov 6 20:55:17 2009 ;; MSG SIZE rcvd: 73
dig @10.62.75.132 g2.lrcd.local mx	; <<>> DiG 9.4.3-P3 <<>> @10.62.75.132 g2.lrcd.local mx ; (1 server found) ;; global options: printcmd ;; Got answer: ;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 58501 ;; flags: qr aa rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 1, ADDITIONAL: 1 ;; QUESTION SECTION: ;g2.lrcd.local. IN MX ;; ANSWER SECTION: **g2.lrcd.local. 604800 IN MX 10 mail.g2.lrcd.local.** ;; AUTHORITY SECTION: g2.lrcd.local. 604800 IN NS server.g2.lrcd.local. ;; ADDITIONAL SECTION: server.g2.lrcd.local. 604800 IN A 10.62.75.132 ;; Query time: 17 msec ;; SERVER: 10.62.75.132#53(10.62.75.132) ;; WHEN: Fri Nov 6 20:56:16 2009 ;; MSG SIZE rcvd: 89

dig @10.62.75.132 g2.lrcd.local ns	; <<>> DiG 9.4.3-P3 <<>> @10.62.75.132 g2.lrcd.local ns ; (1 server found) ;; global options: printcmd ;; Got answer: ;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 51198 ;; flags: qr aa rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1 ;; QUESTION SECTION: ;g2.lrcd.local. IN NS ;; ANSWER SECTION: **g2.lrcd.local. 604800 IN NS server.g2.lrcd.local.** ;; ADDITIONAL SECTION: server.g2.lrcd.local. 604800 IN A 10.62.75.132 ;; Query time: 15 msec ;; SERVER: 10.62.75.132#53(10.62.75.132) ;; WHEN: Fri Nov 6 20:57:05 2009 ;; MSG SIZE rcvd: 68
dig @10.62.75.132 server.g2.lrcd.local	; <<>> DiG 9.4.3-P3 <<>> @10.62.75.132 server.g2.lrcd.local ; (1 server found) ;; global options: printcmd ;; Got answer: ;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 2616 ;; flags: qr aa rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 1, ADDITIONAL: 0 ;; QUESTION SECTION: ;server.g2.lrcd.local. IN A ;; ANSWER SECTION: **server.g2.lrcd.local. 604800 IN A 10.62.75.132** ;; AUTHORITY SECTION: g2.lrcd.local. 604800 IN NS server.g2.lrcd.local. ;; Query time: 15 msec ;; SERVER: 10.62.75.132#53(10.62.75.132) ;; WHEN: Fri Nov 6 20:58:35 2009 ;; MSG SIZE rcvd: 68
dig @10.62.75.132 mail.g2.lrcd.local	; <<>> DiG 9.4.3-P3 <<>> @10.62.75.132 mail.g2.lrcd.local ; (1 server found) ;; global options: printcmd ;; Got answer: ;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 18753 ;; flags: qr aa rd ra; QUERY: 1, ANSWER: 2, AUTHORITY: 1, ADDITIONAL: 0 ;; QUESTION SECTION: ;mail.g2.lrcd.local. IN A ;; ANSWER SECTION: **mail.g2.lrcd.local. 604800 IN CNAME server.g2.lrcd.local.** **server.g2.lrcd.local. 604800 IN A 10.62.75.132** ;; AUTHORITY SECTION: g2.lrcd.local. 604800 IN NS server.g2.lrcd.local. ;; Query time: 15 msec ;; SERVER: 10.62.75.132#53(10.62.75.132) ;; WHEN: Fri Nov 6 21:00:16 2009 ;; MSG SIZE rcvd: 87

dig @10.62.75.132 www.google.com //testando o forwards	; <<>> DiG 9.4.3-P3 <<>> @10.62.75.132 www.google.com ; (1 server found) ;; global options: printcmd ;; Got answer: ;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 44185 ;; flags: qr rd ra; QUERY: 1, ANSWER: 7, AUTHORITY: 4, ADDITIONAL: 0 ;; QUESTION SECTION: ;www.google.com. IN A **;; ANSWER SECTION:** **www.google.com. 604743 IN CNAME www.l.google.com.** **www.l.google.com. 247 IN A 209.85.227.147** **www.l.google.com. 247 IN A 209.85.227.104** **www.l.google.com. 247 IN A 209.85.227.105** **www.l.google.com. 247 IN A 209.85.227.99** **www.l.google.com. 247 IN A 209.85.227.103** **www.l.google.com. 247 IN A 209.85.227.106** ;; AUTHORITY SECTION: google.com. 172741 IN NS ns3.google.com. google.com. 172741 IN NS ns1.google.com. google.com. 172741 IN NS ns2.google.com. google.com. 172741 IN NS ns4.google.com. ;; Query time: 19 msec ;; SERVER: 10.62.75.132#53(10.62.75.132) ;; WHEN: Mon Nov 9 20:47:42 2009 ;; MSG SIZE rcvd: 220
dig @10.62.75.132 -x 172.31.2.132 //reverse lookup	; <<>> DiG 9.4.3-P3 <<>> @10.62.75.132 -x 172.31.2.132 ; (1 server found) ;; global options: printcmd ;; Got answer: ;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 49519 ;; flags: qr aa rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 1, ADDITIONAL: 1 ;; QUESTION SECTION: ;132.2.31.172.in-addr.arpa. IN PTR **;; ANSWER SECTION:** **132.2.31.172.in-addr.arpa. 604800 IN PTR server.g2.lrcd.local.** ;; AUTHORITY SECTION: 2.31.172.in-addr.arpa. 604800 IN NS server.g2.lrcd.local. ;; ADDITIONAL SECTION: server.g2.lrcd.local. 604800 IN A 10.62.75.132 ;; Query time: 16 msec ;; SERVER: 10.62.75.132#53(10.62.75.132) ;; WHEN: Mon Nov 9 21:12:31 2009 ;; MSG SIZE rcvd: 107

ping server.g2.lrcd.local //ping ao servidor do ISEL que, por sua vez, faz a interrogação ao meu servidor, obtém a resposta e devolve ao terminal onde se executa o ping.	PING server.g2.lrcd.local (10.62.75.132): 56 data bytes 64 bytes from 10.62.75.132: icmp_seq=0 ttl=64 time=2.320 ms 64 bytes from 10.62.75.132: icmp_seq=1 ttl=64 time=2.073 ms 64 bytes from 10.62.75.132: icmp_seq=2 ttl=64 time=1.553 ms 64 bytes from 10.62.75.132: icmp_seq=3 ttl=64 time=1.537 ms --- server.g2.lrcd.local ping statistics --- 4 packets transmitted, 4 packets received, 0% packet loss round-trip min/avg/max/stddev = 1.537/1.871/2.320/0.337 ms

Tabela A.4 - output de testes realizados ao servidor DNS

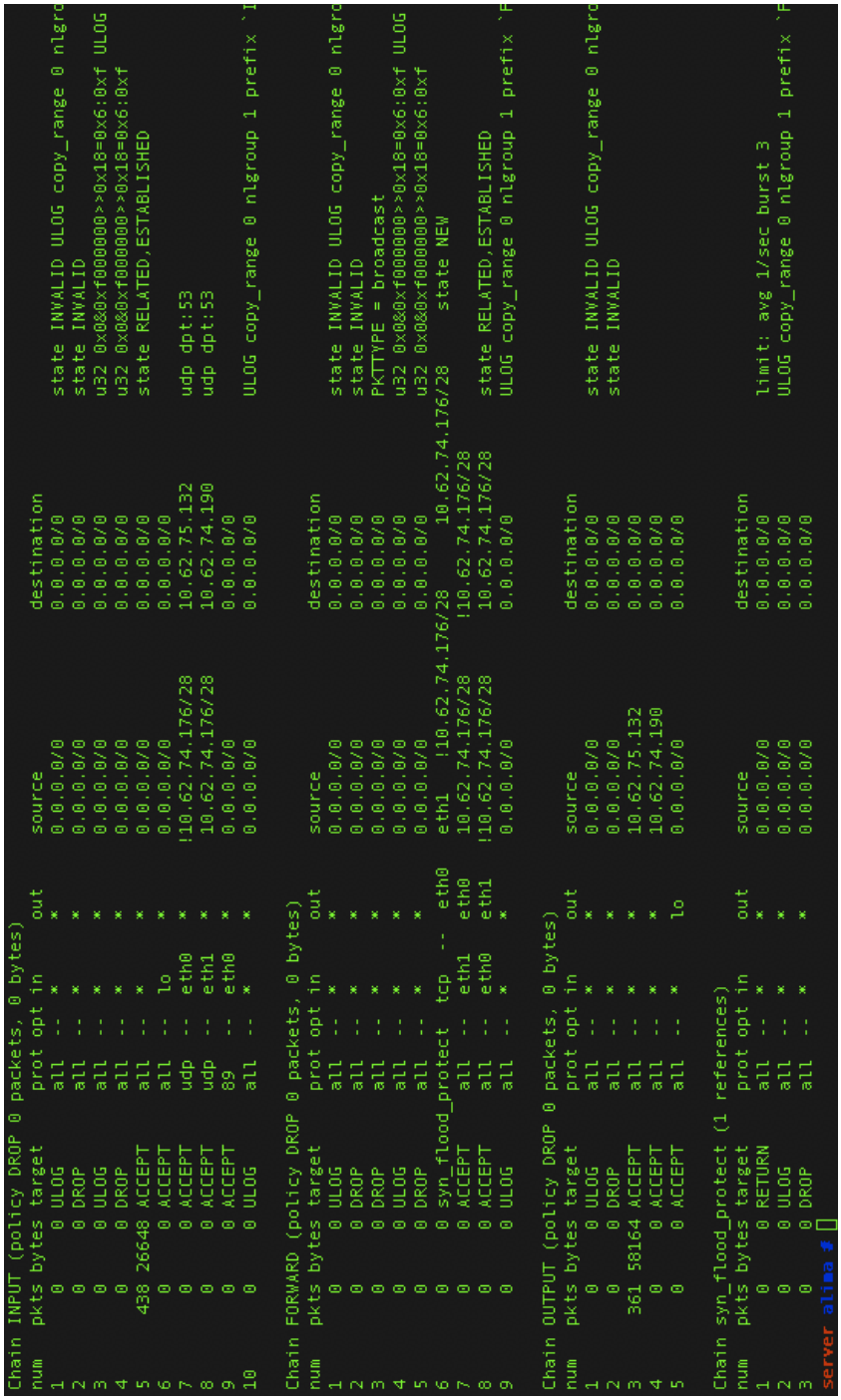
Anexo B - iptables

```
#!/bin/sh
clear

#para que o exit 1 provoque o fecho do script
set -e

echo -e "\nLoading Firewall by Andre Lima - 28838\n"

#executes a command sent as a parameter and if it can't execute it it stops the configuration
#and alerts the user.
execute(){
        ($*) || (echo "erro no comando: $*"; exit 1;)
}

IPTABLES=/sbin/iptables
IP6TABLES=/sbin/ip6tables
I0="eth0"
I1="eth1"
INTNET="10.62.74.176/28"
EXTNET="10.62.75.0/24"

#set $(/sbin/ifconfig eth0 | grep inet | cut -d : -f 2 | cut -d ' ' -f 1);
#SERVER_IP=$1;
EXTIP="10.62.75.132"
INTIP="10.62.74.190"

echo -e "Internal Interface: $I1 \n"
echo -e "External Interface:  $I0 \n\n"

#drop ip6 packets:
execute $IP6TABLES -P INPUT DROP; $IP6TABLES -F INPUT
execute $IP6TABLES -P OUTPUT DROP; $IP6TABLES -F OUTPUT
execute $IP6TABLES -P FORWARD DROP; $IP6TABLES -F FORWARD

#empty INPUT/OUTPUT/FROWARD chains and make them have a drop default policy:
execute $IPTABLES -P INPUT DROP; $IPTABLES -F INPUT
execute $IPTABLES -P OUTPUT DROP; $IPTABLES -F OUTPUT
execute $IPTABLES -P FORWARD DROP; $IPTABLES -F FORWARD

execute $IPTABLES -F syn_flood_protect
execute $IPTABLES -X syn_flood_protect

#####################################################################################################
#                                                                                                   #
#                                        OUTPUT TABLE CONFIGURATION                                 #
#                                                                                                   #
#####################################################################################################

#$IPTABLES -A OUTPUT -m state --state INVALID -j LOG --log-tcp-sequence --log-prefix "ALFirewall Output Invalid Packet Warning: "
execute $IPTABLES -A OUTPUT -m state --state INVALID -j ULOG --ulog-nlgroup 1 --ulog-prefix Output_Invalid:
execute $IPTABLES -A OUTPUT -m state --state INVALID -j DROP

#does not allow IP spoofing from this host
execute $IPTABLES -A OUTPUT -s $EXTIP -j ACCEPT
execute $IPTABLES -A OUTPUT -s $INTIP -j ACCEPT
execute $IPTABLES -A OUTPUT -o lo -j ACCEPT
```

```
#############################################################################################################
#                                                                                                           #
#                                          INPUT TABLE CONFIGURATION                                        #
#                                                                                                           #
#############################################################################################################

#$IPTABLES -A INPUT -m state --state INVALID -j LOG --log-tcp-sequence --log-prefix "ALFirewall Input Invalid Packet Warning: "
execute $IPTABLES -A INPUT -m state --state INVALID -j ULOG --ulog-nlgroup 1 --ulog-prefix Input_Invalid:
# default params for ULOG target:
# --ulog-cprange 0 //default value zero(copies entire pkt)
# --ulog-qthreshold 1 //defines how many pkts are first stored by the kernel before logging (helps protect DoS on loging file)
execute $IPTABLES -A INPUT -m state --state INVALID -j DROP

#drop ip which headers have options
#execute $IPTABLES -A INPUT -m ipv4options --any-opt -j DROP
execute $IPTABLES -A INPUT -m u32 --u32 "0&0x0F000000>>24=0x06:0x0F" -j ULOG --ulog-nlgroup 1 --ulog-prefix Input_Ipv4_Options:
execute $IPTABLES -A INPUT -m u32 --u32 "0&0x0F000000>>24=0x06:0x0F" -j DROP

#accept connections that originated from this station - ESTABLISHED
#  and conns that have something to do with other conns already
#  established (as in ftp active mode or an icmp response or a dhcp response) - RELATED.
#specifically: accept related and established connetions from all (icmp, tcp and udp):
execute $IPTABLES -A INPUT -m state --state ESTABLISHED,RELATED -j ACCEPT

#allow host to host comm (useful in the case of quagga terminal access for instance)
execute $IPTABLES -A INPUT -i lo -j ACCEPT

#disallow tcp ssh(22) - using SPA - and dns(53) - just answering to queries(udp)
#execute $IPTABLES -A INPUT -i $I0 ! -s $INTNET -d $EXTIP -p tcp -m multiport --dports ssh,domain -j ACCEPT
#execute $IPTABLES -A INPUT -i $I1 -s $INTNET -d $INTIP -p tcp -m multiport --dports ssh,domain -j ACCEPT

#allow udp dns
execute $IPTABLES -A INPUT -i $I0 -p udp ! -s $INTNET -d $EXTIP --dport domain -j ACCEPT
execute $IPTABLES -A INPUT -i $I1 -p udp -s $INTNET -d $INTIP --dport domain -j ACCEPT
#execute $IPTABLES -A INPUT -p udp --dport domain -j ACCEPT

#allows ping (echo-request)
#execute $IPTABLES -A INPUT -i $I0 ! -s $INTNET -d $EXTIP -p icmp --icmp-type echo-request -j ACCEPT
#execute $IPTABLES -A INPUT -i $I1 -s $INTNET -d $INTIP -p icmp --icmp-type echo-request -j ACCEPT

#allows ospf
#-d '224.0.0.5' '224.0.0.6' '10.62.75.132'
execute $IPTABLES -A INPUT -i $I0 -p ospf -j ACCEPT

#logs any other messages - these will all be dropped.
#execute $IPTABLES -A INPUT -j LOG --log-tcp-sequence --log-prefix Input_Drop:
execute $IPTABLES -A INPUT -j ULOG --ulog-nlgroup 1 --ulog-prefix Input_Drop:

#############################################################################################################
#                                                                                                           #
#                                          FORWARD TABLE CONFIGURATION                                      #
#                                                                                                           #
#############################################################################################################

#$IPTABLES -A FORWARD -m state --state INVALID -j LOG --log-tcp-sequence --log-prefix "ALFirewall Forward Invalid Packet Warning: "
execute $IPTABLES -A FORWARD -m state --state INVALID -j ULOG --ulog-nlgroup 1 --ulog-prefix FwdInvalidPkt:
execute $IPTABLES -A FORWARD -m state --state INVALID -j DROP

#drop broadcasts
execute $IPTABLES -A FORWARD -m pkttype --pkt-type broadcast -j DROP

#drop ip which headers have options:: if IHL>=6 then DROP
#execute $IPTABLES -A FORWARD -m ipv4options --any-opt -j DROP
execute $IPTABLES -A FORWARD -m u32 --u32 "0&0x0F000000>>24=0x06:0x0F" -j ULOG --ulog-nlgroup 1 --ulog-prefix Input_Ipv4_Options:
execute $IPTABLES -A FORWARD -m u32 --u32 "0&0x0F000000>>24=0x06:0x0F" -j DROP

#synflood defence and psd
execute $IPTABLES -N syn_flood_protect
execute $IPTABLES -A syn_flood_protect -m limit --limit 1/s --limit-burst 3 -j RETURN
execute $IPTABLES -A syn_flood_protect -j ULOG --ulog-nlgroup 1 --ulog-prefix Fwd_Possible_Syn_Flood:
execute $IPTABLES -A syn_flood_protect -j DROP
```

```
execute $IPTABLES -A FORWARD -i $I0 ! -s $INTNET -o $I1 -d $INTNET -p tcp -m state --state NEW -j syn_flood_protect

execute $IPTABLES -A FORWARD -i $I1 -s $INTNET -o $I0 ! -d $INTNET -j ACCEPT
execute $IPTABLES -A FORWARD -i $I0 ! -s $INTNET -o $I1 -d $INTNET -p ALL -m state --state ESTABLISHED,RELATED -j ACCEPT
#execute $IPTABLES -A FORWARD -j LOG --log-tcp-sequence --log-prefix Forward_Drop:
execute $IPTABLES -A FORWARD -j ULOG --ulog-nlgroup 1 --ulog-prefix Forward_Drop:

#############################################################################################################
#############################################################################################################

#lists the rules without dns lookup(--numeric) with hit count (--verbose)
$IPTABLES -L --verbose --numeric --line-numbers
#$IPTABLES -L -v -n --line-numbers

echo -e "\n\nAL Firewall up and running...\n\a\a\a\a"
```

Código B.1 - Firewall Base



Output 1 - “iptables -L -n -v --line-numbers”