FIREWALL IPTABLES agosto/2015

Firewall iptables e criação de regras.

1 – Introdução

A palavra firewall é traduzida como parede de fogo ou parede anti-chamas. Esse termo é empregado há muito tempo em veículos como automóveis ou aviões para designar uma parede isolante que separa uma parte quente [por exemplo, motor] de uma parte fria [por exemplo, mecanismo do trem de pouso de um avião].

Em computação, um firewall é uma barreira de proteção que controla o tráfego de dados entre redes. O objetivo do firewall é permitir apenas o tráfego de dados que for autorizado, bloqueando todo o resto.

Em alguns casos, não menos importantes, o firewall é apenas uma barreira de proteção entre o computador [estação de trabalho, servidor, etc.] e a rede.

No caso mais simples, o firewall é apenas um software que roda na máquina que precisa ser protegida, em outros casos é um equipamento completo com hardware, sistema operacional e software. No caso mais simples, muitas vezes esse tipo de firewall é classificado como firewall pessoal.

Por exemplo, a estação de trabalho pode [e deve] ter um firewall, que é um software local e a protege dos acessos indevidos. Mas para proteger a rede onde se encontra essa estação, é usado um equipamento também chamado de firewall. É esse equipamento que, a partir de suas regras de filtragem, decide qual tráfego está ou não autorizado nesta rede. Neste caso, para funcionar, todo o tráfego destinado ou originado nesta rede precisa passar pelo firewall para ocorrer a filtragem dos dados.

2 – Motivos para usar firewall

Existem muitas razões para usar firewall, as principais são:

- evitar acessos não autorizados a recursos computacionais ou dados. Isso se aplica tanto a um computador pessoal quanto a uma rede corporativa;
- bloqueio de portas usadas por vírus, vermes e cavalos de troia;
- impedir que o usuário acesse conteúdo indevido.

3 – Classificação

Os firewalls podem ser classificados como filtro de pacotes, gateway de aplicação ou filtros baseados em estado.

3.1 - Filtro de pacotes

O filtro de pacotes serve para filtrar o conteúdo. Essa filtragem está baseada em endereços IP e portas de acesso. Por exemplo, a partir de regras previamente definidas pode-se permitir que clientes em uma determinada rede tenham acesso à porta 25 de determinado serviço de e-mail em outra rede.

A filtragem de pacotes toma decisão se determinado pacote pode seguir em frente ou não pela verificação do endereço IP de origem, do endereço IP de destino, da porta TCP ou UDP de origem e da porta TCP ou UDP de destino.

O filtro de pacotes também pode fazer tradução de endereços de rede, para permitir ao cliente usar endereços IP privados [de uma rede interna, por exemplo 192.168.1.0/24] e mesmo assim navegar na internet, isso pela substituição do endereço privado por um endereço de internet toda vez que o pacote for roteado para a internet pelo firewall. Para isso pode ser usado NAT [Network Address Translation] ou então mascaramento [masquerading]. A vantagem é permitir a um grande número de clientes na rede interna navegarem na internet usando simultaneamente o mesmo IP público [de internet].

No caso do filtro de pacotes usado entre duas redes, este tem também a função de roteador.

3.2 - Gateway de aplicação [ou firewall de aplicação]

Esse tipo de firewall é normalmente instalado num servidor, funciona ao nível de aplicação e é mais conhecido como serviço proxy.

Além de isolar a rede do cliente da rede onde está o serviço, o proxy é que busca o conteúdo solicitado pelo cliente, impedindo o seu acesso direto à outra rede. Nesse caso, permite maior controle sobre o que o usuário pode acessar, e fornece melhores recursos de logs de acesso e regras de filtragem.

Pode funcionar executando também as funções de autenticação para acesso à internet, controle de acessos e monitoração do estado das conexões.

3.3 - Filtro baseado em estado

Trata-se de uma evolução do filtro de pacotes, com as mesmas funções daquele e com a

adição de tabelas de estado. As tabelas de estado auxiliam na tomada de decisão sobre o destino de um pacote, e estão associadas às tabelas de regras. Este firewall permite monitorar as conexões durante o tempo todo, desse modo o pacote somente pode ser roteado se fizer parte da tabela de estados.

4 – Comparação entre filtro de pacotes e gateway de aplicação

Se fizermos uma comparação entre o filtro de pacotes e o gateway de aplicação, veremos que as diferenças mais importantes são:

i - o filtro de pacotes inspeciona apenas os cabeçalhos [headers] dos pacotes, já o firewall de aplicação analisa todos os dados de aplicação dos pacotes. Isso significa que a análise do pacote por parte do firewall de aplicação é mais ampla;

ii – no filtro de pacotes, é o próprio pacote enviado pelo cliente que passa pelo firewall e acessa a outra rede [por exemplo, na internet], já no firewall de aplicação o pacote que é enviado pelo cliente só vai até o serviço proxy, que por sua vez gera um novo pacote para buscar na outra rede [na internet] o que o cliente está solicitando.

Outra coisa importante a notar é que o uso simultâneo de filtro de pacotes e firewall de aplicação aumenta em muito o nível de segurança no controle de acessos.

5 – Firewall iptables

Iptables é o nome da aplicação [ou executável] que permite a criação e manutenção de regras no firewall. Na prática, o iptables apenas controla o Netfilter, que é um módulo do kernel Linux responsável pelas funções de firewall, NAT e log de pacotes de dados.

É comum numa instalação padrão de Linux já instalar e habilitar automaticamente o firewall Iptables.

O iptables também permite filtragem baseada em estado [statefull], através do uso de regras NEW [nova], STABLESHID [estabelecida], RELATED [relacionada] ou INVALID [inválida] para identificar os estados das conexões.

6 – Instalação do iptables

Normalmente, na maioria das distribuições Linux, o Iptables já vem instalado. Além de instalado, também já trás uma configuração básica para proteger o sistema. Essa proteção básica poderia ser classificada como firewall pessoal.

Para saber se o Iptables está instalado, verificar se existe o script de inicialização do serviço firewall no diretório **/etc/init.d** com o comando **ls**:

```
shell# ls  /etc/init.d/iptables
```

Se não houvesse saída no comando acima, seria sinal de que não está instalado. E se não estivesse instalado, bastaria usar o comando **yum** para instalar:

```
shell# yum  install  iptables
```

NOTA:
Numa distribuição Debian [ou derivada desta, como é o caso do Ubuntu], o comando para instalar é "**apt-get install iptables**".

Uma vez instalado, deve haver o executável **/sbin/iptables-multi**. Isso é verificado como o comando file:

```
shell# file  /sbin/iptables-multi
/sbin/iptables-multi: ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV),
dynamically linked (uses shared libs), for GNU/Linux 2.6.18, stripped
```

7 – Tabelas [tables] e cadeias [chains] do Iptables

O iptables é composto de três tabelas:

i – **filter**, para filtrar pacotes baseado em regras pré-definidas;

ii – **nat** [Network Address Translation], para tradução de endereços IP;

iii – **mangle**, para alterar o conteúdo de pacotes.

E se nada em contrário for especificado na regra, será assumida a tabela **filter** por ser a padrão [default].

Cada uma dessas três tabelas é composta de algumas cadeias [chains].

7.1 - Tabela filter

A tabela **filter** é composta de três cadeias [chains]:

i - INPUT: trata de regras para pacotes destinados à máquina firewall;

ii - OUTPUT: trata de regras para pacotes originados na máquina firewall;

iii - FORWARD: trata de regras para pacotes que a máquina firewall roteia através da rede. Neste caso, precisa estar habilitado também o roteamento de pacotes no kernel Linux. O roteamento de pacotes é externo ao Iptables.

7.2 – Tabela nat

A tabela **nat** é utilizada para dados que geram outra conexão, isto é, são roteados. Como exemplos temos mascaramento [masquerading], NAT [destination NAT e source NAT], roteamento de portas [port forwarding] e proxy transparente. A tabela **nat** é composta de três cadeias:

i - PREROUTING: trata de regras para pacotes que precisam ser modificados logo que chegam no firewall, antes do roteamento. Este é o caso do DNAT [destination NAT ou NAT de destino];

ii - OUTPUT: trata de regras para pacotes gerados localmente que precisam ser modificados antes de serem roteados. Somente é usado em conexões originadas de endereços de interfaces de rede locais;

iii - POSTROUTING: trata de regras para pacotes que precisam ser modificados após a aplicação de regras de roteamento. A tradução dos pacotes ocorre quando eles estão saindo do sistema, depois de rotear. Este é o caso de SNAT [source NAT ou NAT de origem]

7.3 – Tabela mangle

A tabela **mangle** é que faz alterações em pacotes num nível maior de especialização. Essa tabela tem cinco cadeias:

i - OUTPUT: trata de regras para pacotes gerados localmente antes que sejam roteados, é usado quando os pacotes precisam ser modificados antes de serem enviados para a cadeia OUTPUT da tabela NAT;

ii - PREROUTING: trata de regras para pacotes que precisam ser modificados antes de serem roteados, é usado quando os pacotes precisam ser modificados antes de serem enviados para a cadeia PREROUTING da tabela NAT;

iii - FORWARD: trata de regras para pacotes que precisam ser roteados, é usado quando os pacotes precisam ser modificados antes de serem enviados para a cadeia FORWARD da tabela filter;

iv - INPUT: trata de regras para pacotes destinados à máquina firewall, é usado quando os pacotes precisam ser modificados antes de serem enviados para a cadeia INPUT da tabela filter;

v - POSTROUTING: trata de regras para pacotes que precisam ser modificados após a aplicação de regras de roteamento, é usado quando os pacotes precisam ser modificados antes de serem enviados para a cadeia POSTROUTING da tabela NAT.

8 – Regras do Iptables

As regras são impostas pelo comando iptables, e servem para realizar uma determinada ação.

Por exemplo, o comando:

```
shell# iptables -A INPUT -s 192.168.1.11 -j DROP
```

serve para impor uma regra no firewall, que vai excluir [drop] todos os pacotes originados no endereço IP 192.168.1.11 e que destinam à máquina firewall.

Ao aplicar uma regra, se esta for para a tabela **nat** ou **mangle** precisa usar a opção **-t** [table], mas se for para a tabela **filter** não precisa essa opção pois trata-se da tabela padrão [default].
Por exemplo:

```
iptables -I INPUT -i eth0 -s 1.2.3.4 -j DROP
iptables -t nat -A OUTPUT -d 1.2.3.4 -j DNAT --to 127.0.0.1
iptable -t mangle --list
```

No primeiro exemplo não existe a referência a **-t filter**, nem é necessário. No segundo existe a referência a **-t nat**, e no terceiro **-t mangle**.

As regras são armazenadas em cadeias. Cada cadeia é uma lista de regras as quais podem corresponder a um conjunto de pacotes. Cada regra especifica o que fazer com um pacote que corresponde a essas regras.

As tabelas são locais para guardar as cadeias com as regras.

Para a aplicação de regras, é preciso ter em mente que:

- a regra contém um critério e um alvo [target];
- se o critério é atingido, executa a regra especificada no alvo ou então executa os valores especiais mencionados no alvo;
- se o critério não é atingido, segue para a próxima regra.

8.1 - Valores do alvo [target values]

Os possíveis valores que podem ser especificados como alvos são:

- ACCEPT: para que o firewall aceite o pacote e conclua a cadeia;
- DROP: para que o firewall descarte o pacote [isto é, jogue fora] e conclua a cadeia;
- RETURN: para que o firewall pare a execução do próximo conjunto de regras da cadeia atual e retorne o controle à cadeia que chamou esta. Neste caso, não vai concluir o resto da cadeia atual;
- REJECT: faz a mesma coisa que o valor DROP, com o adicional de poder enviar uma mensagem icmp para a máquina que deu origem ao pacote. Essa mensagem poderia ser algo como “host-unreachable” [isto é, não foi possível acessar o host];
- LOG: apenas envia uma mensagem ao syslog para logar o evento, não para o processamento e continua com a próxima regra. Nesse caso, o pacote não é considerado ACCEPT, DROP, RETURN ou REJECT.

8.2 - Critérios

O critério é a especificação de um padrão a ser comparado com os pacotes de dados, e se corresponderem àquele padrão aplica-se o alvo.

Os critérios são aplicados através de opções do comando iptables. Essas opções se dividem em **comandos** [COMMANDS], **parâmetros** [PARAMETERS] e **outras opções**.

Comandos [COMMANDS] - Os principais comandos são:

-**A** [--append, anexar]: anexa uma ou mais regras ao final da cadeia selecionada;
-**D** [--delete, excluir]: exclui uma ou mais regras da cadeia selecionada;
-**I** [--insert, incluir]: inclui uma ou mais regras na cadeia selecionada. Essa inclusão vai com o número da regra indicado no comando;
-**R** [--replace, substituir]: substitui uma regra na cadeia selecionada;
-**L** [--list, listar]: lista todas as regras na cadeia selecionada. Como a lista pode ser extensa, é melhor sempre usar também **-n** [ver outras opções, abaixo] para não usar o DNS [e não resolver nomes, mostrar apenas endereços IP];
-**F** [--flush, descarregar]: descarrega [esvazia] as regras na cadeia selecionada. É equivalente a excluir todas as regras uma a uma;
-**P** [--policy, política]: configura a política para a cadeia para o destino especificado.

Parâmetros [PARAMETERS] - Os principais parâmetros são:

-**p** [--protocol, protocolo]: configura o protocolo da regra para verificação do pacote.

Alguns dos protocolos suportados são tcp, udp e icmp. Se for usado também uma exclamação "! -p", isso é para negar aquele protocolo;
-s [--source, origem]: especifica a origem do pacote, que pode ser um endereço IP, rede, nome de host, etc. Também pode usar o endereço seguido da máscara. Se for usado "! -s", isso é para negar determinado endereço;
-d [--destination, destino]: especifica o destino para o pacote, que pode ser um endereço conforme descrito acima para "-s";
-j [--jump, salto]: especifica o que fazer caso o pacote corresponda à regra;
-i [--in-interface, interface de entrada]: especifica o nome da interface pela qual o pacote é recebido, usado somente para pacotes entrando nas cadeias INPUT, FORWARD ou PREROUTING. Se for usado "! -i", isso é para negar determinada interface;
-o [--out-interface, interface de saída]: especifica o nome da interface pela qual o pacote é enviado, usado somente para pacotes entrando nas cadeias FORWARD, OUTPUT e POSTROUTING. Se for usado "! -o", isso é para negar determinada interface.

Outras opções [OTHER OPTIONS] - Os principais parâmetros são:

-v [--verbose, detalhado]: especifica que a saída deve mostrar detalhes do que tem configurado. Pode ser usado no comando "**--list**";
-n [--numeric, numérico]: especifica que endereços IP e portas serão apresentados no formato numérico. Essa opção é para não usar o DNS, não resolver os nomes dos host, redes, etc.;
--line-numbers, número de linhas: especifica que, ao listar regras, deve apresentar o número da linha no começo de cada regra, para corresponder ao número da posição da regra na cadeia.

9 – Exemplos de aplicação de regras

Alguns exemplos de aplicações de regras são apresentados aqui. O uso de scripts aqui é uma necessidade, pois normalmente um firewall envolve muitas regras e não seria prático incluir uma a uma. Então o melhor é escrever um script e, pela execução desse script, adicionar as regras todas em sequência.

Vamos ver aqui, através de alguns exemplos, como podem ser aplicadas regras no firewall. Essas regras estão sendo aplicadas na máquina com endereço IP 192.168.1.10, hostname lab10, que faz a função de firewall. Na maioria dos casos, para testar as regras os acessos são feitos de 192.168.1.12 e os serviços estão em 192.168.1.11.

É o comando **iptables** que impõe as regras de filtragem e redirecionamento de pacotes no Netfilter[1].

i) Para listar o status do firewall, usar o comando **iptables -L -n -v**:

1 Netfilter: é um framework [estrutura] que provê manipulação e interceptação de pacotes no kernel [núcleo] do Linux.

```
shell# iptables -L -n -v
Chain INPUT (policy ACCEPT 0 packets, 0 bytes)
pkts bytes target     prot opt in     out     source               destination
Chain FORWARD (policy ACCEPT 0 packets, 0 bytes)
pkts bytes target     prot opt in     out     source               destination
Chain OUTPUT (policy ACCEPT 0 packets, 0 bytes)
pkts bytes target     prot opt in     out     source               destination
```

Onde:
-L: lista as regras;
-v: saída detalhada [verbose];
-n: não resolve nomes no DNS, apenas mostra os endereços IP e redes.

Pode ser notado que no exemplo acima o firewall não está ativo, pois não existe nenhuma regra definida.

ii) Para inserir uma regra na tabela filter [que é a default], na cadeia INPUT, usar o comando:

```
shell# iptables -I INPUT -s 192.168.1.11 -j DROP
```

Onde:
-I: insere a regra na cadeia [no caso, INPUT];
-s: origem [source], no caso é o endereço IP 192.168.1.11;
-j: é o alvo, que no caso descarta o pacote [DROP].

Então, essa regra serve para descartar [drop] todos os pacotes originados no endereço IP 192.168.1.11 e que se destinam ao firewall [INPUT], em todas as suas interfaces de rede.

Neste caso, a listagem das regras vai mostrar:

```
shell# iptables -L -n -v
Chain INPUT (policy ACCEPT)
target     prot opt source               destination
DROP       all  --  192.168.1.11        0.0.0.0/0
Chain FORWARD (policy ACCEPT)
target     prot opt source               destination
Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination
```

Como nessa regra não foi feito referência a protocolo, ela atua em todos os protocolos suportados pelo iptables.

Também poderia ser o caso de usar uma rede inteira e não apenas um endereço IP, por exemplo: "**-s 192.168.1.0/24**".

Nesse ponto, é interessante notar que as regras normalmente são cumulativas, então para evitar conflitos é interessante sempre antes de cada exemplo esvaziar [flush] as regras antigas.

iii) Para esvaziar [descarregar; flush] todas as regras, usar:

```
shell# iptables -F
```

Onde:
-F: esvazia [flush] todas as regras da tabela.

NOTA:
Se fosse uma tabela específica e não a padrão, teria que informar no comando. Por exemplo: "**iptables -F -t nat**".

Mas o mais garantido mesmo para zerar o firewall é usar a sequência de comandos abaixo:

```
shell# iptables -F
shell# iptables -X
shell# iptables -P INPUT ACCEPT
shell# iptables -P OUTPUT ACCEPT
shell# iptables -P FORWARD ACCEPT
```

Onde:
-X: exclui uma cadeia pré-definida pelo usuário, caso haja;
-P: altera a política para ACCEPT, que permite o acesso. O default [padrão] é DROP.

Nesse ponto, o melhor mesmo é usar um script para executar a tarefa completa de zerar o firewall antes de definir uma nova regra.

=== clean.sh ====================================

```
#!/bin/bash
# esse script exclui todas as regras impostas pelo iptables

# esvazia todas as regras [flush]
iptables -F

# exclui as tabelas pre-definidas pelo usuario [caso houvesse]
iptables -X

# esvazia as regras de nat
```

```
iptables  -t  nat  -F
iptables  -t  nat  -X

# esvazia as regras de mangle
iptables  -t  mangle  -F
iptables  -t  mangle  -X

# define a politica como nao-restritiva
iptables  -P  INPUT  ACCEPT
iptables  -P  FORWARD  ACCEPT
iptables  -P  OUTPUT  ACCEPT

### final do script ###
```
==========

Desse modo, basta disparar o script **clean.sh** que ele já cumpre toda a tarefa de excluir as regras antigas e zerar o firewall.

NOTA:
O script **clean.sh** vem incluído no arquivo tar **squid26.tar.gz**, conforme **proxy-squid.pdf** (num Ubuntu, usar **squid31.tar.gz**).

iv) Para adicionar uma regra para guardar a tentativa de acesso em arquivos de log do sistema, e na sequência descartar os pacotes com essa tentativa, usar:

```
shell# iptables -A INPUT -i eth0 -s 192.168.1.0/24 -j LOG --log-prefix "AVISO: "
shell# iptables -A INPUT -i eth0 -s 192.168.1.0/24 -j DROP
```

Onde:
-**A**: adiciona [append] uma regra à cadeia;
-i: interface de entrada [no caso, eth0];
-**j LOG**: o alvo é o arquivo de log, portanto não finaliza a regra que prossegue para a segunda linha e que por sua vez faz um drop dos pacotes.

Nos arquivos de log do sistema, se alguém desse um ping para lab10 [192.168.1.10], o aviso apareceria como:

```
shell# tail -1 /var/log/messages
Sep 30 14:12:17 lab10 kernel: AVISO: IN=eth0 OUT=
MAC=00:23:5b:74:53:4a:01:15:af:94:70:5b:18:00 SRC=192.168.1.11
DST=192.168.1.10 LEN=84 TOS=0x00 PREC=0x00 TTL=64 ID=0 PROTO=ICMP
TYPE=8 CODE=0 ID=54026 SEQ=4
```

NOTA:

Num Ubuntu, loga no arquivo **/var/log/kern.log**.

v) Incluir uma regra para excluir pacotes originados na máquina firewall e que se destinem ao endereço IP 192.168.1.12:

```
shell# iptables -I OUTPUT -d 192.168.1.12 -j DROP
```

Onde:
-d: destino para o pacote [destination].

vi) Adicionar uma regra para bloquear apenas o ping:

```
shell# iptables -A INPUT -p icmp -j DROP
```

Onde:
-p: protocolo, no caso icmp.

vii) Adicionar uma regra para bloquear as requisições da rede 10.0.0.0/80 à porta de serviço 80 na máquina firewall, na interface de rede eth0:

```
shell# iptables -A INPUT -i eth0 -s 10.0.0.0/8 -p tcp --dport 80 -j DROP
```

Onde:
--dport: é a porta de destino, no caso porta 80.

viii) Adicionar regra para bloquear o acesso a determinado host ou rede a partir da interface eth0 da máquina firewall:

```
shell# iptables -A OUTPUT -o eth0 -d 192.168.1.12 -j DROP
```

Onde:
-o: interface de saída [out], no caso eth0.

NOTA:
A regra acima é importante, pois o firewall normalmente trabalha fazendo redirecionamento [forward] de pacotes de uma rede para outra, então se a saída de determinada interface de rede for bloqueada, nenhum pacote será enviado para a rede acessível através daquela interface.

ix) Adicionar regra para a máquina firewall aceitar acesso SSH de toda a rede 192.168.1.0/24:

```
shell# iptables -A OUTPUT -p tcp --sport 22 -m state --state
NEW,ESTABLISHED -j ACCEPT
shell# iptables -A INPUT -p tcp -s 192.168.1.0/24 --dport 22 -j ACCEPT
```

Onde:
-m state: corresponde [match], no caso corresponde a um estado [state] para a conexão;
-m state --state NEW,ESTABLISHED: verifica se de fato se trata de uma conexão nova ou então que já esteja estabelecida, e não uma tentativa de continuar uma conexão já expirada. Como sabemos, o iptables é um filtro de pacotes que também permite verificações baseadas em estado (statefull), portanto consegue verificar o estado das conexões;
-j ACCEPT: o alvo é aceito, aceita a conexão.

Um exemplo completo de script para aplicar a regra para aceitar acesso SSH é mostrado abaixo no script **accept_ssh.sh**:

```
=== accept_ssh.sh ======================================================
#!/bin/bash

# esvazia todas as regras [flush]
iptables -F

# impoem a politica [-P] restritiva: primeiro DROP, depois ACCEPT
iptables -P INPUT DROP
iptables -P OUTPUT DROP
iptables -P FORWARD DROP

# permite o trafego no servico ssh
iptables -A OUTPUT -p tcp --sport 22 -m state --state NEW,ESTABLISHED -j ACCEPT
iptables -A INPUT -p tcp -s 192.168.1.0/24 --dport 22 -j ACCEPT

# exclui [drop] todos os demais pacotes que nao sao contemplados pelas regras acima
iptables -A INPUT -j DROP
iptables -A OUTPUT -j DROP
iptables -A FORWARD -j DROP
=============================================================================
```

Uma vez imposta essas regras, a listagem com **iptables -L -n** vai mostrar:

```
shell# iptables -L -n
Chain INPUT (policy DROP)
target     prot opt source               destination
ACCEPT     tcp  --  192.168.1.0/24      0.0.0.0/0      tcp dpt:22
DROP       all  --  0.0.0.0/0           0.0.0.0/0

Chain FORWARD (policy DROP)
target     prot opt source               destination

Chain OUTPUT (policy DROP)
target     prot opt source               destination
ACCEPT     tcp  --  0.0.0.0/0           0.0.0.0/0      tcp spt:22 state
NEW,ESTABLISHED
DROP       all  --  0.0.0.0/0           0.0.0.0/0
```

NOTA 1:
Na regra acima, para aceitar acesso a outros serviços, basta alterar a porta de origem "--sport 22" e de destino "--dport 22" pela porta do outro serviço. Por exemplo, para permitir o acesso ao serviço web, basta fazer "--sport 80" e "--dport 80".

NOTA 2:
O script **accept_ssh.sh** vem incluído no arquivo tar **squid26.tar.gz**, conforme **proxy-squid.pdf** (num Ubuntu, usar **squid31.tar.gz**).

No exemplo acima o serviço SSH está na mesma máquina que o firewall [192.168.1.10], mas se estivesse em outra máquina teria que fazer redirecionamento de pacotes. O script abaixo, **accept_ssh2.sh**, mostra como deve ser esse redirecionamento.

=== accept_ssh2.sh ==

```
#!/bin/bash

# esvazia todas as regras [flush]
iptables -F
iptables -F -t nat

# impoem a politica [-P] restritiva: primeiro DROP, depois ACCEPT
iptables -P INPUT DROP
iptables -P OUTPUT DROP
iptables -P FORWARD DROP

# habilita roteamento de pacotes no kernel Linux
echo 1 > /proc/sys/net/ipv4/ip_forward

# faz o redirec. de pacote destinados a maquina local [192.168.1.10] para 192.168.1.11
iptables -t nat -A PREROUTING -p tcp --dport 22 -j DNAT --to-dest 192.168.1.11:22
iptables -A FORWARD -p tcp --sport 22 -j ACCEPT
iptables -A FORWARD -p tcp --dport 22 -j ACCEPT
```

```
iptables -t nat -A POSTROUTING -j MASQUERADE

# exclui [drop] todos os demais pacotes
iptables -A INPUT -j DROP
iptables -A OUTPUT -j DROP
iptables -A FORWARD -j DROP
```
==========

Onde:
-t nat -A PREROUTING: atua na chegada dos pacotes ao firewall na cadeia PREROUTING da tabela NAT;
-j DNAT --to-dest: o alvo é para *destination* NAT [nat de destino];
FORWARD: redireciona os pacotes;
POSTROUTING: atua na saída dos pacotes;
-j MASQUERADE: significa mascaramento, ou seja, o endereço de origem é "mascarado" pelo endereço da interface de rede onde é redirecionado. No exemplo acima, os pacotes com origem no cliente ssh em 192.168.1.12 chegam no serviço ssh em 192.168.1.11 como tendo origem em 192.168.1.10, que é a interface de rede do firewall.

Mas para funcionar o redirecionamento de pacotes e acessar o serviço ssh em outra máquina, precisa também habilitar o roteamento [forward] de pacotes no kernel, que é feito com o comando "**echo 1 > /proc/sys/net/ipv4/ip_forward**".

No caso acima, o cliente em 192.168.1.12 acessou o serviço ssh na máquina firewall [192.168.1.10], no entanto ele foi redirecionado para 192.168.1.11 que é onde de fato está o serviço ssh. Esse redirecionamento é transparente para o cliente.

Depois de disparado o script **accept_ssh2.sh** e impostas as regras acima, a listagem com "**iptables -L -n --line-number**" vai mostrar:

```
shell# iptables -L -n --line-number
Chain INPUT (policy DROP)
num  target     prot opt source               destination
1    DROP       all  --  0.0.0.0/0            0.0.0.0/0

Chain FORWARD (policy DROP)
num  target     prot opt source               destination
1    ACCEPT     tcp  --  0.0.0.0/0            0.0.0.0/0           tcp spt:22
2    ACCEPT     tcp  --  0.0.0.0/0            0.0.0.0/0           tcp dpt:22
3    DROP       all  --  0.0.0.0/0            0.0.0.0/0

Chain OUTPUT (policy DROP)
num  target     prot opt source               destination
1    DROP       all  --  0.0.0.0/0            0.0.0.0/0
```

Onde:

--line-number: mostra o número da linha da regra na cadeia. Isso possibilita tanto excluir uma regra dado o seu número quanto incluir uma nova regra em posição específica dentro da cadeia.

NOTA:
O script **accept_ssh2.sh** vem incluído no arquivo tar **squid26.tar.gz**, conforme **proxy-squid.pdf** (num Ubuntu, usar **squid27.tar.gz**).

x) Para incluir uma regra em determinada posição da cadeia, diferente da ordem sequencial, usar "**-I**" [de insert, inserir] e o número da linha da regra na cadeia. Por exemplo:

```
shell# iptables -I INPUT -s 192.168.1.0/24 -j ACCEPT
shell# iptables -I INPUT -s 10.0.0.0/24 -j ACCEPT
shell# iptables -L -n --line-number
Chain INPUT (policy ACCEPT)
num  target     prot opt source               destination
1    ACCEPT     all  --  10.0.0.0/24         0.0.0.0/0
2    ACCEPT     all  --  192.168.1.0/24       0.0.0.0/0

Chain FORWARD (policy ACCEPT)
num  target     prot opt source               destination

Chain OUTPUT (policy ACCEPT)
num  target     prot opt source               destination
```

NOTA:
Observar que as duas regras acima foram inseridas sequencialmente, portanto a primeira está na linha número 1 e a segunda na linha 2.

Agora, se for inserida uma regra na linha 2, a regra que antes ocupava esta posição vai para 3:

```
shell# iptables -I INPUT 2 -s 172.16.1.0/24 -j ACCEPT
shell# iptables -L -n --line-number
Chain INPUT (policy ACCEPT)
num  target     prot opt source               destination
1    ACCEPT     all  --  10.0.0.0/24          0.0.0.0/0
2    ACCEPT     all  --  172.16.1.0/24        0.0.0.0/0
3    ACCEPT     all  --  192.168.1.0/24       0.0.0.0/0

Chain FORWARD (policy ACCEPT)
num  target     prot opt source               destination

Chain OUTPUT (policy ACCEPT)
num  target     prot opt source               destination
```

NOTA:
Para excluir [delete] uma regra em determinada linha, usar [por exemplo] o comando "**iptables -D INPUT 3**", que exclui a terceira regra de cima para baixo na cadeia INPUT.

xi) Configurar um roteador NAT com duas interfaces de rede

Nesse caso, é necessário usar apelidos de interfaces, pois as máquinas do laboratório acadêmico só tem uma interface de rede. Na prática, isso funciona como se fossem duas interfaces, porém num caso de serviço firewall em produção nunca deveria ser usado por questões de segurança.

Neste exemplo, a interface **eth0** tem endereço IP **10.1.2.3** e a interface **eth0:0** tem IP **192.168.1.10**.

Nessa máquina, o comando **ifconfig** mostra a seguinte saída:

```
shell# ifconfig
eth0      Link encap:Ethernet  Endereço de HW 00:21:5B:77:53:4A
          inet end.: 10.1.2.3 Bcast:10.1.2.3.255  Masc:255.255.255.0
          endereço inet6: fe80::223:5aff:fe74:534a/64 Escopo:Link
          UP BROADCASTRUNNING MULTICAST  MTU:1500  Métrica:1
          RX packets:1043 errors:0 dropped:0 overruns:0 frame:0
          TX packets:985 errors:0 dropped:0 overruns:0 carrier:0
          colisões:0 txqueuelen:1000
          RX bytes:147834 (144.3 KiB)  TX bytes:144742 (141.3 KiB)
          IRQ:32 Endereço de E/S:0xe000

eth0:0    Link encap:Ethernet  Endereço de HW 00:21:5B:77:53:4A
          inet end.: 192.168.1.10  Bcast:192.168.1.255  Masc:255.255.255.0
          UP BROADCASTRUNNING MULTICAST  MTU:1500  Métrica:1
          IRQ:32 Endereço de E/S:0xe000
```

Repare que onde deveria ser eth1 está **eth0:0**, pois se trata de apelido para a interface eth0 e não outra interface de rede.

As regras a serem impostas devem contemplar ao cliente com IP **10.1.2.4** [na rede 10.1.2.0/24] acessar o serviço ssh em **192.168.1.11**, que está na rede 192.168.1.0/24. Para isso, deve ocorrer NAT na passagem dos pacotes da interface eth0 para eth0:0 na máquina roteador (IP **10.1.2.3**).

Essas regras estão no script **accept_ssh3.sh** abaixo:

```
=== accept_ssh3.sh ================================================
#!/bin/bash

# envolve 3 maquinas:
# 1 - maquina cliente ssh (para testar o acesso), na rede 10.1.2.0/24
# 2 - maquina firewall, tambem na rede 10.1.2.0/24
# 3 - maquina com o servico ssh, no IP 192.168.1.11

# na maquina roteador, que soh tem 1 interface de rede, eth0 estah na
# rede 10.1.2.0/24. Foi criado um apelido (alias) de rede eth0:0 com
# endereco IP 192.168.1.10. Deste modo, o firewall se comunica tanto
# com a maquina cliente quanto com a do servico ssh.

# NOTA: num caso real, o firewall teria 2 interfaces de rede (eth0 e eth1),
# e eth0 teria IP da rede 10.1.2.3 e eth1 teria IP 192.168.1.10.


# esvazia todas as regras [flush]
iptables -F
iptables -F -t nat
```

```
# impoem a politica [-P] restritiva: primeiro DROP, depois ACCEPT
iptables -P INPUT DROP
iptables -P OUTPUT DROP
iptables -P FORWARD DROP

# habilita roteamento de pacotes no kernel Linux
echo 1 > /proc/sys/net/ipv4/ip_forward

# faz o redirecionamento de pacotes destinados a maquina local no IP
# 10.1.2.3 [interface eth0] para 192.168.1.11 [rede da interface eth0:0]
# embora exista a interface eth0 e eth0:0, o iptables nao considera
# eth0:0 como interface de rede
iptables -t nat -A PREROUTING -p tcp --dport 22 -j DNAT --to-dest 192.168.1.11:22
iptables -A FORWARD -p tcp -o eth0 --sport 22 -j ACCEPT
iptables -A FORWARD -p tcp -i eth0 --dport 22 -j ACCEPT
iptables -t nat -A POSTROUTING -p tcp -o eth0 -j MASQUERADE

# exclui [drop] todos os demais pacotes
iptables -A INPUT -j DROP
iptables -A OUTPUT -j DROP
iptables -A FORWARD -j DROP

### final do script - 16/08/2015 ###
==========================================================================
```

Onde:
-j MASQUERADE: mascara os pacotes originados na rede 10.1.2.0/24 para que acessem a rede 192.168.1.0/24 com o endereço IP da interface eth0:0 do firewall, 192.168.1.10. Desse modo, o tráfego que chega a eth0 [10.1.2.3], passa para a interface eth0:0 [192.168.1.10] como se tivesse originado no endereço IP 192.168.1.10. O mascaramento pode ser útil para permitir aos clientes de uma rede interna acessar a internet no caso de não haver um serviço proxy.

Depois de disparado o script **accept_ssh3.sh** e impostas as regras acima, a listagem com **iptables -L -n -v** vai mostrar:

```
shell# iptables -L -n -v
Chain INPUT (policy DROP 0 packets, 0 bytes)
pkts bytes target     prot opt in     out     source               destination
 0     0    DROP       all  --  *      *       0.0.0.0/0            0.0.0.0/0

Chain FORWARD (policy DROP 0 packets, 0 bytes)
pkts bytes target     prot opt in     out     source               destination
 0     0    ACCEPT     tcp  --  *      eth0    0.0.0.0/0            0.0.0.0/0           tcp spt:22
 0     0    ACCEPT     tcp  --  eth0   *       0.0.0.0/0            0.0.0.0/0           tcp dpt:22
 0     0    DROP       all  --  *      *       0.0.0.0/0            0.0.0.0/0

Chain OUTPUT (policy DROP 0 packets, 0 bytes)
pkts bytes target     prot opt in     out     source               destination
 0     0    DROP       all  --  *      *       0.0.0.0/0            0.0.0.0/0
```

E a listagem com **iptables -L -n -v -t nat** vai mostrar:

```
shell# iptables -L -n -v -t nat
Chain PREROUTING (policy ACCEPT 0 packets, 0 bytes)
pkts bytes target     prot opt in     out     source               destination
 0     0    DNAT       tcp  --  *      *       0.0.0.0/0            0.0.0.0/0           tcp
dpt:22 to:192.168.1.11:22

Chain POSTROUTING (policy ACCEPT 0 packets, 0 bytes)
pkts bytes target     prot opt in     out     source               destination
 0     0    MASQUERADE tcp  --  *      eth0    0.0.0.0/0            0.0.0.0/0

Chain OUTPUT (policy ACCEPT 0 packets, 0 bytes)
pkts bytes target     prot opt in     out     source               destination
```

No caso acima, o cliente em 10.1.2.4 acessou o serviço ssh na máquina firewall [10.1.2.3], no entanto ele foi redirecionado para 192.168.1.11 que é onde de fato está o serviço ssh. E acessou o serviço ssh com o endereço IP da interface eth0:0 do firewall, que é 192.168.1.10.

10 – Proxy transparente

Proxy transparente é um conceito usado quando o cliente, para navegar na internet, é forçado a usar um serviço proxy mesmo que o seu navegador não esteja configurado para usar este serviço.

O cliente [navegador da internet] precisa ser instruído [ou configurado] para usar

determinado proxy. Ora, então um usuário poderia eventualmente modificar as configurações no seu navegador [por exemplo] e não usar mais o proxy. Desse modo, ficaria livre para acessar qualquer conteúdo, livre das regras de filtragem do serviço proxy.

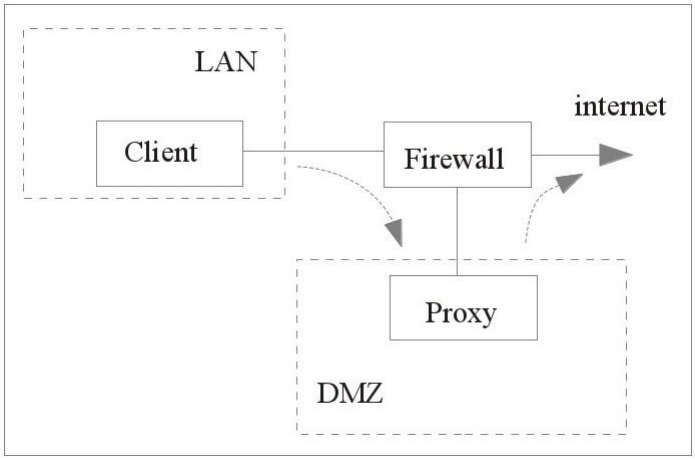
Também poderia ser o caso de todos os navegadores estarem configurados para usar o proxy, e este repentinamente ficar inacessível. Isso exigiria reconfigurar todos os navegadores para permitir aos usuários continuarem navegando, que pode ser uma tarefa bem demorada dependendo do número de máquinas envolvidas.

Para resolver os problemas exposto acima é quer existe o proxy transparente.

Mas para funcionar, o proxy transparente envolve configurar o firewall para redirecionar pacotes. Assim, toda vez que o cliente enviar pacotes de dados buscando conteúdo na internet, esses pacotes serão redirecionados para o proxy, que irá buscar esse conteúdo para o cliente. Evidentemente, também pode ser usado para filtrar o conteúdo solicitado pelo cliente.

No proxy transparente, se o serviço proxy ficasse inacessível bastaria modificar a regra de redirecionamento de pacotes no firewall que os clientes passariam a acessar a internet diretamente. Esta é uma solução rápida para garantir o acesso à internet em caso de falha do proxy.

Normalmente, o servidor que hospeda o serviço proxy fica na DMZ[2]. Na figura abaixo pode ser visto o esquema de uma rede interna [LAN] cujo tráfego tem que passar pelo firewall para acessar a internet.



Mas ao passar pelo firewall, as regras redirecionam os pacotes para o proxy, que é quem busca o conteúdo na internet para o cliente. Desse modo, o uso de regras apropriadas no firewall torna compulsivo o uso do proxy.

Para configurar um proxy transparente, como pré-requisito é necessário instalar e configurar o serviço proxy Squid conforme **proxy-squid.pdf**.

Nas configurações do proxy Squid, no arquivo **/etc/squid/squid.conf** é necessário retirar o comentário da linha **http_port 3128 intercept** e comentar a linha **http_port 3128**:

=== arquivo **/etc/squid/squid.conf** ==========
http_port 3128 intercept
http_port 3128
==========

NOTA:
Essa configuração só é válida a partir da versão 2.6 do Squid.

2 DMZ: Vem da expressão DeMilitarized Zone, ou zona desmilitarizada, pois pode ser acessada da internet. A LAN não pode ser acessada da internet. Na DMZ, além do proxy costumam ficar os serviços de e-mail, DNS e web, entre outros.

Depois disso, fazer um reload do serviço proxy Squid para esta configuração tomar efeito.

Vamos estudar duas topologias: firewall e proxy na mesma máquina e firewall e proxy em máquinas diferentes, sendo essa última a que de fato normalmente é usada num ambiente em produção.

i) Firewall e proxy na mesma máquina

No arquivo **squid26.tar.gz** (ou **squid31.tar.gz** para o Ubuntu) baixado de **www.jairo.pro.br**, além de **squid.conf**, também está incluído o script **transp.sh** que vai ser usado para aplicar as regras no proxy transparente.

Abaixo, segue o conteúdo do arquivo **transp1.sh**, usado no Ubuntu:

=== arquivo **transp1.sh** ==========

```
#!/bin/bash

# regras para proxy transparente e firewall na mesma maquina
# O proxy server e firewall estao na mesma maquina

# pre-requisitos:
# 1 - proxy Squid configurado para proxy transparente;
# 2 - os clientes que terao os pacotes redirecionados para o proxy precisam
#     ter como default gateway o endereco IP da maquina firewall. Eles
#     estao na rede 192.168.1.0/24

DNS_SERVER="186.251.39.194"
#DNS_SERVER="10.102.1.231"
LAN_ADDR="192.168.1.0/24"

# inicialmente, remover todas as regras anteriores [F: flush, X: delete].
iptables -F
iptables -t nat -F
iptables -t mangle -F
iptables -X
iptables -t nat -X
iptables -t mangle -X

# permitir o roteamento de pacotes no kernel Linux.
echo 1 > /proc/sys/net/ipv4/ip_forward

# para aceitar pacotes icmp (ping)
iptables -A INPUT  -p icmp -j ACCEPT
iptables -A OUTPUT -p icmp -j ACCEPT

# para aceitar query DNS (servico DNS nesta maquina)
iptables -A OUTPUT -p tcp --sport 53 -m state --state NEW,ESTABLISHED -j ACCEPT
```

```
iptables -A INPUT -p tcp --dport 53 -j ACCEPT

iptables -A OUTPUT -p udp --sport 53 -m state --state NEW,ESTABLISHED -j ACCEPT
iptables -A INPUT -p udp --dport 53 -j ACCEPT

# servico DNS em outra maquina: redirecionar pacotes para la
# redirecionamento de pacotes para o servico DNS na porta 53.
iptables -t nat -A PREROUTING  -s $LAN_ADDR -p tcp --dport 53 -j DNAT --to-dest
$DNS_SERVER:53
iptables -t nat -A PREROUTING  -s $LAN_ADDR -p udp --dport 53 -j DNAT --to-dest
$DNS_SERVER:53
iptables -A FORWARD  -p tcp --sport 53 -j ACCEPT
iptables -A FORWARD  -p tcp --dport 53 -j ACCEPT
iptables -A FORWARD  -p udp --sport 53 -j ACCEPT
iptables -A FORWARD  -p udp --dport 53 -j ACCEPT
iptables -t nat -A POSTROUTING -j MASQUERADE

# squid e firewall no mesmo server
iptables -t nat -A PREROUTING -s $LAN_ADDR -p tcp --dport 80 -j REDIRECT --to-port 3128
# como PREROUTING NAO atua no loopback, precisa tambem adicionar uma regra para
OUTPUT
iptables -t nat -I OUTPUT -p tcp -d 127.0.0.1 --dport 80 -j REDIRECT --to-ports 3128


#### final do script ####
==========================================================================
```

Nesse script, é necessário acertar:

- o IP do serviço DNS na variável **DNS_SERVER**;
- o endereço da rede onde estão firewall e cliente do serviço proxy, na variável **LAN_ADDR**.

Convém notar que esse script foi projetado para um caso de uma máquina com apenas uma interface de rede, que é o que ocorre nos laboratórios acadêmicos. Num caso mais realista, seria necessário duas interfaces de rede.

Antes de disparar o script **transp1.sh** para impor as regras de redirecionamento de pacotes, fazer um scan de portas na máquina firewall. Isso irá mostrar que a porta 80 está fechada.

Depois, executar o script **transp1.sh**:

```
shell# /tmp/squid31/transp.sh
```

Depois de disparado o script, fazer de novo o scan de portas na máquina firewall e observar que a porta 80 agora está aberta. Isso é devido à ação de redirecionamento de pacotes, e não que tenha sido instalado um serviço web nessa máquina.

Para o cliente testar o acesso ao proxy transparente, usar duas maneiras distintas:

a) apenas configurando a variável **http_proxy** para a porta 80;
b) alterando a rota *default* na máquina cliente.

No caso (a), carregar um novo valor na variável **http_proxy**, para usar a porta 80 e não mais a 3128. Ao carregar um novo valor numa variável, caso houvesse, o valor antigo seria perdido.

```
shell_cliente$ export  http_proxy=http://192.168.1.10:80
```

Onde:
O endereço IP da máquina firewall é 192.168.1.10 [o cliente está no IP 192.168.1.12].

Agora, o cliente testa o acesso com o comando **wget** e observa que o proxy busca a página. Ou seja, embora esteja usando a porta 80 e não 3128, mesmo assim o cliente foi atendido pelo proxy. Isso é devido ao redirecionamento de pacotes no firewall.

No caso (b), na máquina cliente não será mais usado configuração na variável **http_proxy** para acessar a internet através do proxy. E para esvaziar o valor que havia na variável **http_proxy**, usar o comando **unset**:

```
shell_cliente$ unset  http_proxy
```

Desse modo, a variável **http_proxy** vazia não instrui mais o cliente a acessar o proxy.

Para confirmar que a variável **http_proxy** está mesmo vazia, comandar:

```
shell_cliente$ echo  $http_proxy
```

Se não houver saída no comando acima, indica que a variável está mesmo vazia.

Mas, além disso, precisa remover a rota *default gateway* do cliente e incluir uma nova, que passará a apontar para o IP da máquina firewall.

Para isso, na máquina cliente [192.168.1.12], o usuário precisa virar root para excluir a rota *default* antiga e incluir uma nova rota *default*. Para virar root é usado o comando "**sudo su -**", e para excluir e incluir rota é usado o comando **route**.

```
shell_cliente$ sudo su -
[sudo] password for aluno:
shell_cliente_root# route  del  default
shell_cliente_root# route  add  default  gw  192.168.1.10  eth0
shell_cliente_root# exit
shell_cliente$
```

Por fim, testar o acesso com **wget**:

```
shell_cliente$ wget  www.jairo.pro.br
```

O comando acima baixa o arquivo através do proxy, pelo redirecionamento de pacotes imposto pelo script **transp.sh** na máquina firewall [que é a mesma que hospeda o serviço proxy].

Por fim, na máquina firewall [192.168.1.10], retirar as regras de redirecionamento de pacotes para demonstrar que a partir daí o cliente não irá mais acessar a internet. Para isso, usar o script **clean.sh**:

```
shell# ./clean.sh
```

E na máquina cliente [192.168.1.12], testar o acesso de novo com **wget**. Agora não deverá conseguir acessar, pois no firewall não existe mais redirecionamento de pacotes.

Convém notar que, num caso mais realista, todas as máquinas da LAN têm como *default gateway* o switch ou roteador na rede, e que a partir daí os roteadores levam os pacotes de dados para o firewall. Isso é o equivalente a ter configurado na máquina cliente o endereço IP da máquina firewall como *default gateway*.

ii) Firewall e proxy em máquinas diferentes

Até aqui, já sabemos que o firewall está no IP 192.168.1.10 e que o cliente do serviço proxy está no IP 192.168.1.12.

Agora, vamos disponibilizar o proxy transparente na máquina 192.168.1.11 e não mais em 192.168.1.10.

Para instalar e configurar o proxy transparente na máquina 192.168.1.11 basta seguir os mesmos procedimentos já descritos acima.

O que muda, é na máquina firewall editar o arquivo **transp.sh** e tirar o comentário da última linha de configuração:

```
=== arquivo transp.sh ================================================
...
# caso do servico proxy em maquina diferente do firewall, habilitar essa regra TAMBEM
 iptables -t nat -A PREROUTING -s $LAN_ADDR -p tcp --dport 80 -j DNAT
 --to 192.168.1.11:3128
=======================================================================
```

Aqui, foi configurado o endereço 192.168.1.11 para o serviço proxy transparente.

Feito isso, basta disparar novamente o script **transp.sh** que as novas regras tomam efeito.

Agora, no cliente [192.168.1.12], acessar novamente com **wget**:

```
shell_cliente$ wget  www.jairo.pro.br
```

O comando acima baixa o arquivo através do proxy em 192.168.1.11, pelo redirecionamento de pacotes imposto pelo script **transp.sh** na máquina firewall.