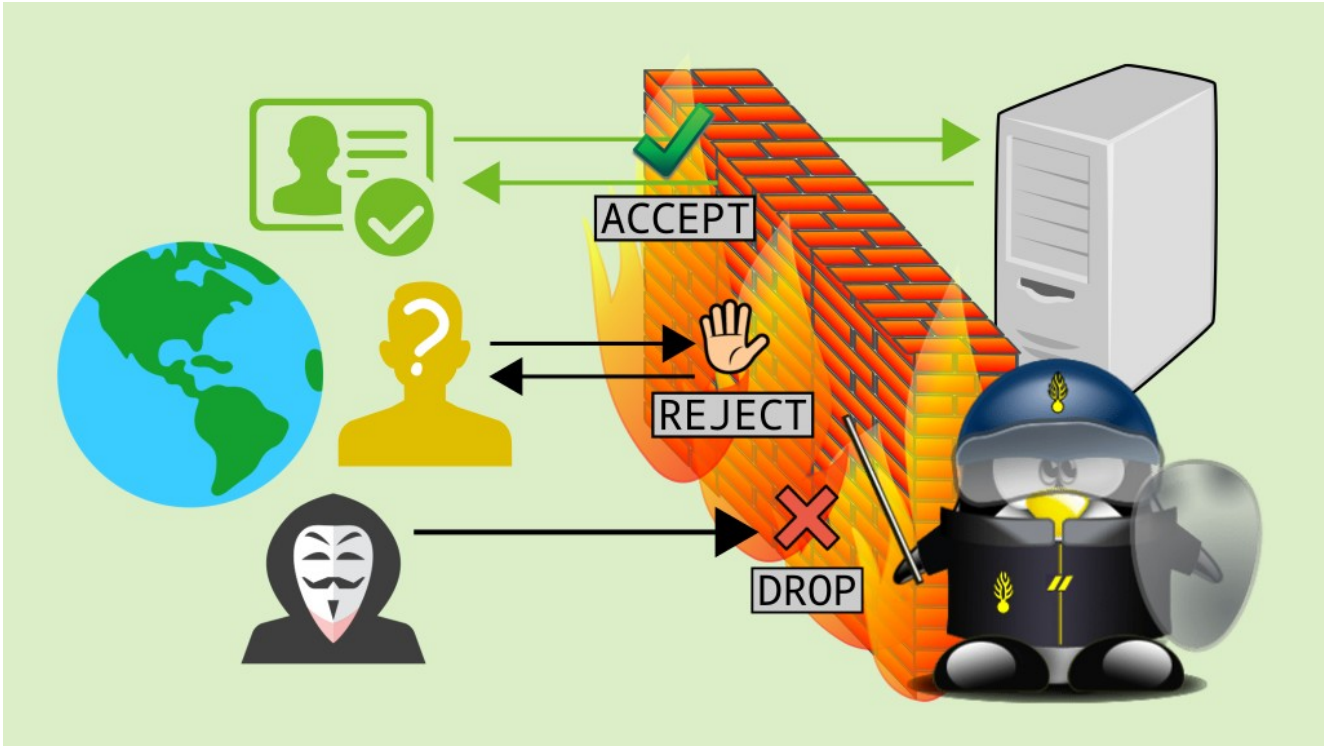
Proteja-se com o firewall iptables

18 de Novembro de 2019

53

Faz tempo que não falamos de servidores… Hoje vamos falar sobre **segurança** focando em servidores, mas serve também para *desktops*, seja em casa ou no trabalho. A Internet facilitou bastante acessar, enviar e compartilhar informações. Mas, como diz o ditado, "não há bônus sem ônus": junto das conveniências, a Internet trouxe também uma série de riscos e ameaças. Uma das soluções usadas para tentar prevenir ataques e invasões é o *firewall*.

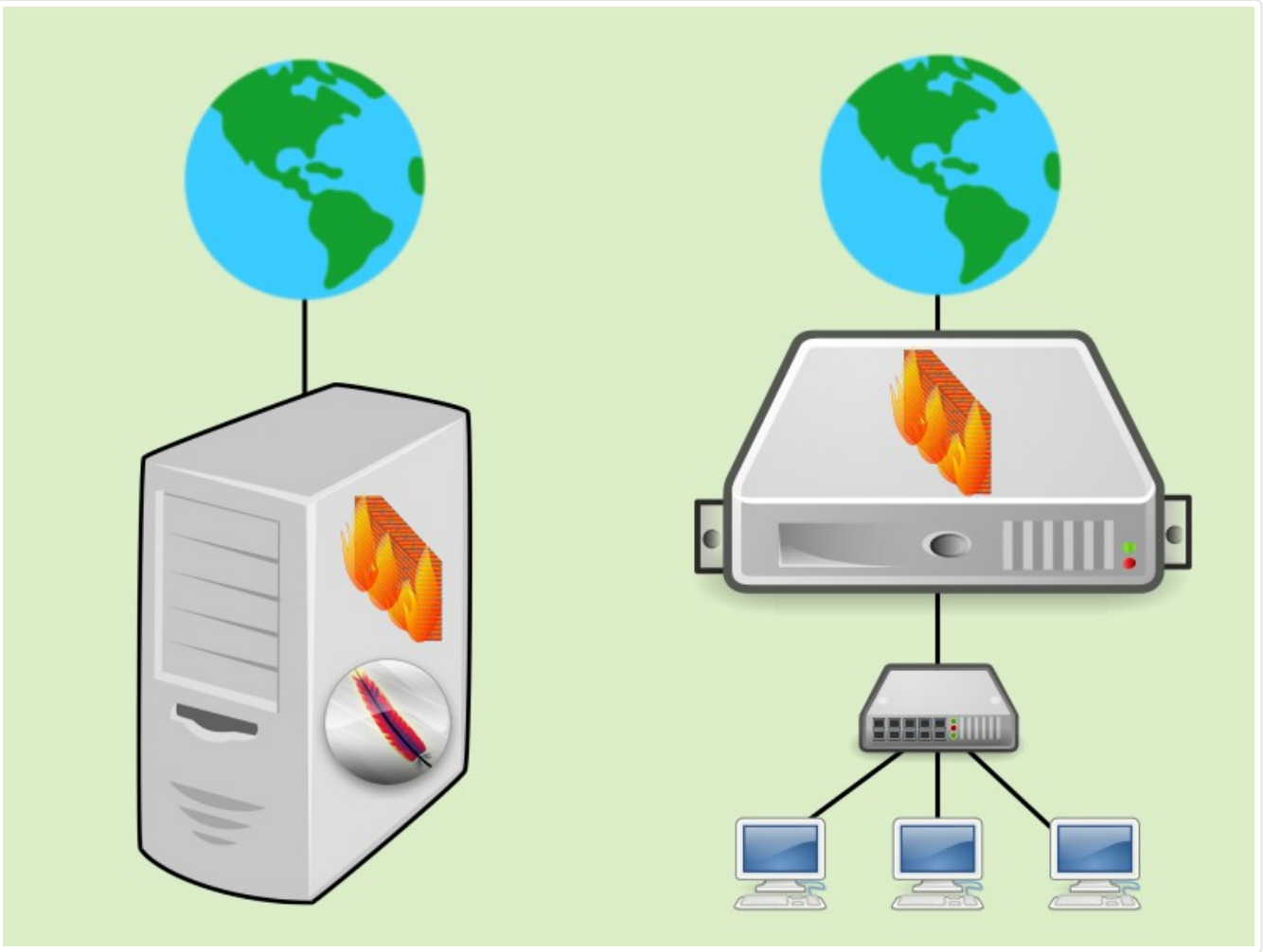
O ***firewall*** ("parede de fogo", em inglês) é um *software* que monitora os pacotes que trafegam entrando e saindo da rede e permite ou bloqueia pacotes específicos com base em regras de segurança predefinidas.



Também existem *firewalls* na forma de *hardware*, ou soluções mistas de *hardware* e *software*, mas comumente o *firewall* é um *software* instalado em um servidor ou *desktop*.

Firewalls podem ser instalados em diversos lugares com diferentes finalidades, como:

- no servidor (por exemplo, um servidor *web*) ou no *desktop* (por exemplo, uma estação de trabalho), para proteger o próprio servidor ou *desktop*;
- no servidor inserido entre a rede local e a Internet, para proteger a rede local; ou
- no servidor que faz roteamento entre diversas redes internas, para protegê-las.



Com base na camada do modelo TCP/IP em que atuam, *firewalls* podem ser classificados em:

- **filtro de pacotes** (*packet filter*): atua nas camadas 2 (rede) e 3 (transporte), permite ou bloqueia pacotes com base em suas características (endereço/porta/interface de origem/destino, protocolo usado — se é TCP, UDP, ICMP, etc). É o tipo de *firewall* mais antigo, simples e limitado, mas já oferece um nível de segurança significativo.
- ***firewall* de aplicação**, mais conhecido como ***proxy***: atua na camada 4 (aplicação) e é capaz de bloquear pacotes com base em seu conteúdo. Por exemplo, um *proxy* HTTP consegue bloquear o acesso a uma página se ela contiver determinada palavra no *link*, ou se ela é conhecida por apresentar pirataria, jogo, pornografia, *malware*, etc.

Como é o firewall no Linux

Normalmente, as pessoas simplificam e chamam tanto o módulo do *kernel* quanto a ferramenta de **iptables**, como se fossem a mesma coisa. Vamos fazer assim.

O **iptables** é, a princípio, um *firewall* em nível de pacotes, mas dispõe de módulos que o permitem atuar na camada de aplicação. Ele vem instalado por padrão na maioria das distribuições Linux, incluindo o Linux Kamarada e o openSUSE. Isso significa que podemos simplesmente começar a usar o **iptables**: não precisamos instalar nada antes.

A versão original do módulo `ip_tables` e da ferramenta **iptables** lida apenas com IPv4. Existem também os análogos `ip6_tables` e **ip6tables** para lidar com IPv6.

Muitas distribuições trazem ferramentas alternativas para configurar o *firewall*, como o **firewalld**, que pode ser usado tanto pela linha de comando quanto pela interface gráfica (que algumas pessoas podem achar mais fácil de usar). Por padrão, o openSUSE vem com o **iptables** e o **firewalld** quando instalado em *desktops*, mas apenas com o **iptables** (sem o **firewalld**) quando instalado em servidores. O Linux Kamarada, que é focado em *desktops* e usuários iniciantes de Linux, também traz o **iptables** e o **firewalld** instalados por padrão.

Hoje, vamos falar apenas do **iptables**. Oportunamente, podemos falar do **firewalld**, mas já mencionamos ele em um *post*, se você quiser ter uma ideia de como é a tela dele, veja:

- Transmitindo do Linux para a TV com Chromecast

Conceitos básicos

Para entender o funcionamento do **iptables**, você precisa saber o que são regras, *chains*, tabelas, *targets* e política padrão.

As **regras** descrevem tipos de pacotes (endereço/porta/interface de origem/destino, protocolo usado, etc.) e o que fazer com esses pacotes (como bloquear ou deixar passar).

Uma ***chain*** é uma lista de regras (traduções possíveis seriam cadeia, corrente, sequência, algo nesse sentido, mas comumente se usa o termo em inglês mesmo). O **iptables** possui *chains* predefinidas e o usuário também pode criar as suas próprias *chains*.

Já as **tabelas** são conjuntos de *chains*. O **iptables** tem 4 tabelas que são usadas em diferentes momentos, apresentadas a seguir: `filter`, `nat`, `mangle` e `raw`. Cada tabela contém *chains* predefinidas do **iptables** e pode conter também *chains* criadas pelo usuário. Hoje, vamos focar na tabela `filter`. Oportunamente, podemos falar mais sobre as outras tabelas.

- `filter`: essa é a tabela usada por padrão, contém 3 *chains* predefinidas do **iptables**:
  - `INPUT`: consultada quando pacotes chegam à máquina
  - `OUTPUT`: consultada quando pacotes devem sair da máquina
  - `FORWARD`: consultada quando pacotes devem ser roteados através da máquina (encaminhados de uma interface de rede para outra ou de uma máquina para outra, passando pela máquina atual)
- `nat`: essa tabela é usada para fazer NAT (*Network Address Translation*), contém 3 *chains* predefinidas do **iptables**: `PREROUTING`, `OUTPUT` e `POSTROUTING`
- `mangle`: essa tabela é usada para fazer alterações específicas em pacotes, como, por exemplo, modificar o tipo de serviço (ToS), contém 5 *chains* predefinidas do **iptables**: `PREROUTING`, `OUTPUT`, `INPUT`, `FORWARD` e `POSTROUTING`

Note que os nomes das *chains* são sensíveis à capitalização (*case sensitive*). Portanto, `INPUT`, `input` e `Input`, por exemplo, seriam 3 *chains* diferentes. As *chains* predefinidas do **iptables** são sempre escritas em caixa alta (por exemplo, `INPUT`).

Quando recebe um pacote, o **iptables** analisa as regras buscando uma regra que descreva aquele pacote. As regras são analisadas na ordem em que foram inseridas na *chain*. Quando o **iptables** analisa uma regra, se o pacote não corresponde à descrição, ele passa para a próxima regra. Se o pacote corresponde, então o **iptables** verifica na regra para onde enviar o pacote, que é o ***target*** (alvo).

O *target* pode ser o nome de uma *chain* do usuário — nesse caso, o **iptables** vai continuar analisando as regras dessa *chain* — ou pode ser um desses valores especiais, diante dos quais o **iptables** para de analisar as regras da *chain* atual e age sobre o pacote imediatamente:

- `ACCEPT`: aceitar o pacote, deixá-lo (permiti-lo) passar
- `DROP`: descartar (excluir, ignorar) o pacote — na prática, significa impedi-lo de passar
- `QUEUE`: passar o pacote para um programa do espaço de usuário (fora do *kernel*), que irá processar o pacote
- `RETURN`: interrompe o processamento das regras da *chain* atual e retorna o processamento das regras para a regra seguinte na *chain* anterior (a que chamou a *chain* atual)

Por fim, se o **iptables** já analisou todas as regras da *chain* e não encontrou uma regra que descreve o pacote, ele vai adotar a **política padrão** predefinida para aquela *chain*, que pode ser somente um dos quatro *targets* especiais acima.

Há ainda o *target* `REJECT`, que é semelhante ao `DROP`, mas só pode ser usado em regras das *chains* da tabela `filter`. Falaremos mais sobre o *target* `REJECT` adiante.

Parece confuso? Não se preocupe: vamos meter a mão na massa e tudo vai ficar mais claro.

Listando as regras em uso

Será que nosso *firewall* já possui regras antes mesmo de adicionarmos nossa primeira regra? Vejamos!

Para listar as regras atualmente em uso pelo **iptables**, execute o comando a seguir:

```
# iptables -L
```

Adianto que hoje só usaremos a interface de linha de comando e praticamente todos os comandos serão executados como administrador (usuário *root*). Suponho que você já possua certa familiaridade com o Linux e com o terminal.

A saída do comando acima deve ser algo parecido com:

```
Chain INPUT (policy ACCEPT)
target     prot opt source               destination

Chain FORWARD (policy ACCEPT)
target     prot opt source               destination

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

Se a saída para você é diferente, é possível que o administrador já tenha adicionado regras, que a distribuição que você usa tenha outra configuração padrão, ou que você esteja usando o **firewalld**, que cria suas próprias *chains*. A seguir, veremos como excluir essas regras.

Por padrão, são listadas as regras da tabela `filter`. Para listar as regras de outra tabela, adicione o parâmetro `-t` (ou `--table`) seguido do nome da tabela. Por exemplo:


```
# iptables -t nat -L
Chain PREROUTING (policy ACCEPT)
target     prot opt source               destination

Chain INPUT (policy ACCEPT)
target     prot opt source               destination

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination

Chain POSTROUTING (policy ACCEPT)
target     prot opt source               destination
```

Caso você queira limitar a exibição a uma *chain*, informe o seu nome no final. Exemplo:

```
# iptables -t mangle -L FORWARD
Chain FORWARD (policy ACCEPT)
target     prot opt source               destination
```

Excluindo todas as regras em uso

Como a ordem das regras importa para o **iptables**, é comum iniciar sua configuração excluindo quaisquer regras que estejam em uso.

> **Não** teste os comandos desse tutorial em um servidor que você acessa remotamente via SSH: você corre o risco de perder o acesso ao servidor.

> Para acompanhar este tutorial, recomendo que você use uma máquina virtual. Com isso, você não modifica as configurações do seu servidor ou *desktop*.
>
> Para mais informações, leia:
>
> - **VirtualBox: a forma mais fácil de conhecer o Linux sem precisar instalá-lo**

```
# iptables -F
```

Na verdade, por padrão, são excluídas as regras de todas as *chains* da tabela `filter`.

Nesse sentido, o comando `iptables -F` é semelhante ao comando `iptables -L`:

- para excluir as regras de outra tabela, adicione o parâmetro `-t` seguido do nome da tabela, e
- para excluir as regras apenas de uma *chain* específica, adicione seu nome ao final.

Exemplos:

```
# iptables -t nat -F
# iptables -t mangle -F POSTROUTING

```

Excluindo todas as chains do usuário

Além de excluir as regras, também é comum iniciar excluindo as *chains* do usuário.

Para excluir todas as *chains* do usuário presentes na tabela `filter`, use o comando:

```
# iptables -X
```

Ainda que você mesmo não tenha criado nenhuma *chain*, se sua distribuição usa o **firewalld**, com esse comando você exclui as *chains* criadas pelo **firewalld**.

Da mesma forma, para especificar uma tabela, use o parâmetro `-t`, e para excluir apenas uma *chain* específica, adicione seu nome ao final.

Note que não é possível excluir as *chains* do **iptables**, apenas as do usuário.

Resumindo, para fazer uma limpeza completa no **iptables**, execute os comandos:

```
# iptables -F
# iptables -X
# iptables -t nat -F
# iptables -t nat -X
# iptables -t mangle -F
# iptables -t mangle -X
# iptables -t raw -F
# iptables -t raw -X
```

Depois dessa limpeza, se você listar as regras (`iptables -L`), certamente verá a configuração padrão do **iptables**.

Adicionando regras

Agora sim vamos à parte mais interessante: vamos adicionar nossa primeira regra ao **iptables**. Como exemplo, vamos adicionar uma regra que bloqueia o acesso à própria máquina (`127.0.0.1` ou `localhost`).

Antes, execute um **ping** e certifique-se de que funciona (use **Ctrl + C** para interrompê-lo):

```
$ ping 127.0.0.1
PING 127.0.0.1 (127.0.0.1) 56(84) bytes of data.
```

```
--- 127.0.0.1 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1010ms
rtt min/avg/max/mdev = 0.036/0.071/0.106/0.035 ms
```

Agora, adicione a regra:

```
# iptables -A INPUT -d 127.0.0.1 -j DROP
```

Leiamos esse comando por partes:

- como nos comandos anteriores, poderíamos usar o parâmetro `-t` para especificar uma tabela, como não fizemos isso, usamos a tabela `filter` por padrão
- o parâmetro `-A` inicia a adição de uma nova regra ao final da *chain*
- `INPUT` é a *chain* onde a regra está sendo adicionada — nesse caso, estamos interessados nos pacotes que chegam à máquina
- o parâmetro `-d` (poderia ser também `--dst` ou `--destination`) indica o destino — nesse caso, filtramos pacotes com destino a `127.0.0.1` (a própria máquina)
- o parâmetro `-j` (poderia ser também `--jump`) indica o que fazer com o pacote (o *target*) — nesse caso, descartar (`DROP`)

Agora, depois de adicionada a regra, tente novamente usar o **ping**:

```
$ ping 127.0.0.1
PING 127.0.0.1 (127.0.0.1) 56(84) bytes of data.
^C
--- 127.0.0.1 ping statistics ---
2 packets transmitted, 0 received, 100% packet loss, time 1006ms
```

Viu só? Não funcionou. O **iptables** está bloqueando esses pacotes com base na regra que acabamos de adicionar.

> **Observação:** a regra acima foi usada apenas para fins didáticos. Nunca bloqueie o acesso à própria máquina, a menos que saiba o que está fazendo, pois muitos aplicativos utilizam soquetes TCP para realizar conexões.

Dito isso, vamos desfazer a adição da regra acima antes de continuar:

```
# iptables -F
```

Definindo a política padrão de uma chain

Entre a limpeza das tabelas e a adição das regras propriamente ditas, eu costumo definir a política padrão das *chains*. Usando o comando `iptables -L`, vimos que a configuração padrão do **iptables** é permitir todo o tráfego em todas as *chains* (política padrão de `ACCEPT`).

Normalmente, quando adotamos uma política padrão permissiva (`ACCEPT`) para uma *chain*, adicionamos regras restritivas (`DROP` ou `REJECT`) a essa *chain*, ou seja, as regras determinam o que bloquear, e por padrão, o que não é bloqueado, é permitido.

Para a maioria das *chains*, uma política padrão permissiva (`ACCEPT`) é considerada insegura. Ao menos para a *chain* `INPUT`, considere usar uma política padrão restritiva (`DROP`). Desse modo, você só precisa permitir os serviços considerados seguros e não corre o risco de esquecer de bloquear algum acesso não desejado.

Para mudar a política padrão de uma *chain*, use o comando `iptables -P` seguido do nome da *chain* e do *target* que deve ser a política padrão. Exemplos:

```
# iptables -P FORWARD DROP
# iptables -P INPUT DROP
# iptables -P OUTPUT ACCEPT
```

Se você executar os três comandos acima, talvez algo pare de funcionar. Por exemplo: o navegador não consegue acessar *sites*. Isso acontece porque, embora a requisição seja enviada (a política padrão de `OUTPUT` é `ACCEPT`), a resposta é descartada (a política padrão de `INPUT` é `DROP`). Para prevenir casos assim, eu costumo usar a seguinte regra:

```
# iptables -A INPUT -m state --state ESTABLISHED,RELATED -j ACCEPT
```

Trata-se de uma "receita pronta" que eu encontrei na Internet para aceitar conexões que já foram estabelecidas previamente. Com essa regra, ao notar que o pacote que chega é uma resposta a uma requisição feita anteriormente pela própria máquina (o pacote que chega não é uma tentativa de conexão iniciada por outra máquina), o **iptables** aceita o pacote.

Note que políticas padrão são configuradas apenas para *chains* predefinidas do **iptables** (*chains* definidas pelo usuário não possuem política padrão).

Descrevendo pacotes

Vejamos alguns parâmetros que podemos usar para descrever pacotes na hora de criar regras para o **iptables**, assim como exemplos de regras válidas.

> **Observação:** as regras a seguir são apenas exemplos, você não deve copiar e colar no seu servidor ou *desktop*, mas adaptar os comandos conforme a sua necessidade.

- **destino:** já vimos o parâmetro `-d` (ou `--dst`, ou `--destination`), que permite filtrar um pacote pelo seu destino, que pode ser expresso na forma de:
  - o endereço IP de um *host* (por exemplo, `157.240.12.35`);
  - o endereço IP de uma rede/sub-rede acompanhado da máscara de rede, em notação tradicional (por exemplo, `157.240.12.0/255.255.255.0`) ou CIDR (`157.240.12.0/24`);
  - nome de rede (*hostname*, por exemplo, `www`); ou
  - nome de domínio completo (do inglês *fully qualified domain name* ou FQDN, por exemplo, `www.facebook.com`).

```
# iptables -A OUTPUT -d 157.240.12.35 -j REJECT
```

(leia-se: todos os pacotes saindo desta máquina com destino a `157.240.12.35` devem ser rejeitados — é um dos endereços IP de www.facebook.com, note que na verdade bloquear o acesso ao Facebook não é tão simples assim, essa regra é apenas um exemplo)

- **origem:** de forma análoga, o parâmetro `-s` (ou `--src`, ou `--source`) permite filtrar um pacote pela sua origem, aceita os mesmos formatos do parâmetro `-d`.

```
# iptables -A INPUT -s 222.187.224.200 -j DROP
```

(todos os pacotes tentando entrar nesta máquina vindo de `222.187.224.200` devem ser descartados — uma vez, esse endereço IP estava tentando atacar um servidor que eu administrava, não sei se esse endereço ainda é usado para ataques)

- **interface de origem:** o parâmetro `-i` (ou `--in-interface`) permite filtrar um pacote pela interface de rede na qual ele é recebido (por exemplo, `eth0`).

```
# iptables -A INPUT -i lo -j ACCEPT
```

(todos os pacotes recebidos na interface de *loopback* — `lo` — devem ser aceitos)

- **interface de destino:** de forma análoga, o parâmetro `-o` (ou `--out-interface`) permite filtrar um pacote pela interface de rede pela qual ele deve ser enviado.

```
# iptables -A FORWARD -i eth0 -o eth1 -j DROP
```

(descartar todos os pacotes que tentam passar da interface `eth0` para a interface `eth1`)

- **protocolo:** o parâmetro `-p` (ou `--protocol`) permite filtrar um pacote pelo protocolo (TCP, UDP ou ICMP, pode ser escrito em maiúsculas ou minúsculas).

```
# iptables -A INPUT -p icmp -j ACCEPT
```

(aceitar todos os pacotes do protocolo ICMP — por exemplo, **ping**)

- **porta de origem:** o parâmetro `--sport` (ou `--source-port`) permite filtrar um pacote pela porta de origem, só pode ser usado após `-p tcp` ou `-p udp`.
- **porta de destino:** o parâmetro `--dport` (ou `--destination-port`) permite filtrar um pacote pela porta de destino, só pode ser usado após `-p tcp` ou `-p udp`.

```
# iptables -A INPUT -p tcp --dport 22 -i eth0 -j ACCEPT
# iptables -A INPUT -p udp --dport 22 -i eth0 -j ACCEPT
```

(aceitar todas as requisições TCP e UDP para a porta 22 recebidas na interface `eth0` — a porta 22 é a porta usada por padrão pelo serviço SSH)

Observação: todos os parâmetros acima aceitam negação usando exclamação (`!`) antes.

```
# iptables -A INPUT -p tcp --dport 22 ! -i eth0 -j DROP
# iptables -A INPUT -p udp --dport 22 ! -i eth0 -j DROP
```

(descartar todas as requisições TCP e UDP para a porta 22, **exceto** as recebidas na interface `eth0`)

```
$ man iptables
```

Ou, se preferir ler no navegador:

- iptables(8) - Linux man page

A seguinte lista de portas também pode ser útil enquanto estiver criando regras:

- Lista de portas dos protocolos TCP e UDP - Wikipédia

Qual é a diferença entre DROP e REJECT?

A essa altura, você pode estar se perguntando: qual é a diferença entre `DROP` e `REJECT`?

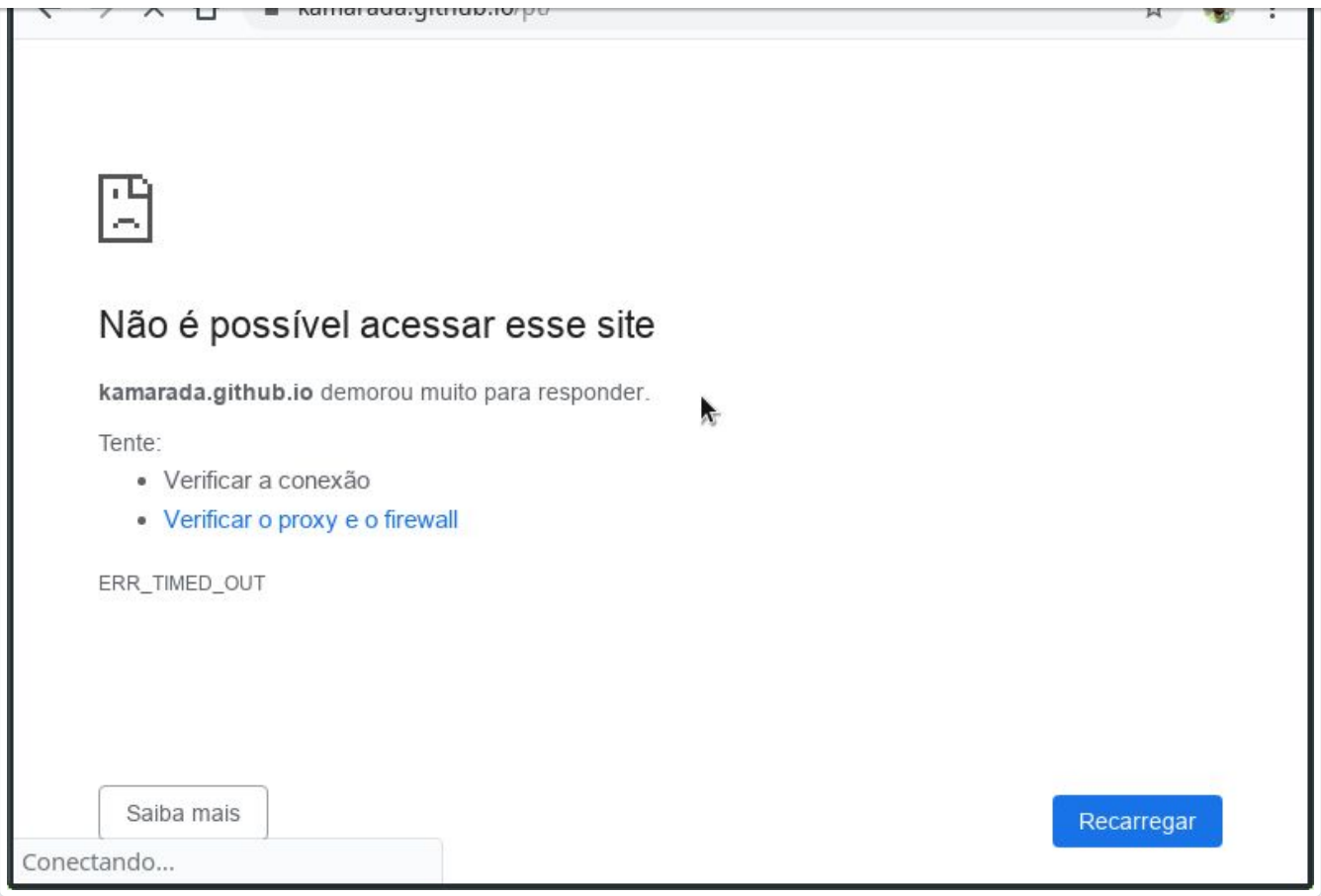
A semelhança é que em ambos os *targets* o pacote é bloqueado (o **iptables** não deixa o pacote passar). A diferença vai além da tradução (aqui, estou usando "descartar" para `DROP` e "rejeitar" para `REJECT`). A diferença entre `DROP` e `REJECT` é que no caso do `DROP`, o pacote é silenciosamente descartado pelo **iptables**, que não envia uma resposta ao remetente (que "se sente" ignorado), já no caso do `REJECT`, o **iptables** descarta o pacote, mas também envia uma resposta ao remetente (que pode compreender que aquela requisição não é permitida).

Você pode ver a diferença entre `DROP` e `REJECT` esquematizada na figura do início do *post*.

Façamos um teste prático. Adicione a seguinte regra, usando `DROP`:

```
# iptables -A OUTPUT -d kamarada.github.io -j DROP
```

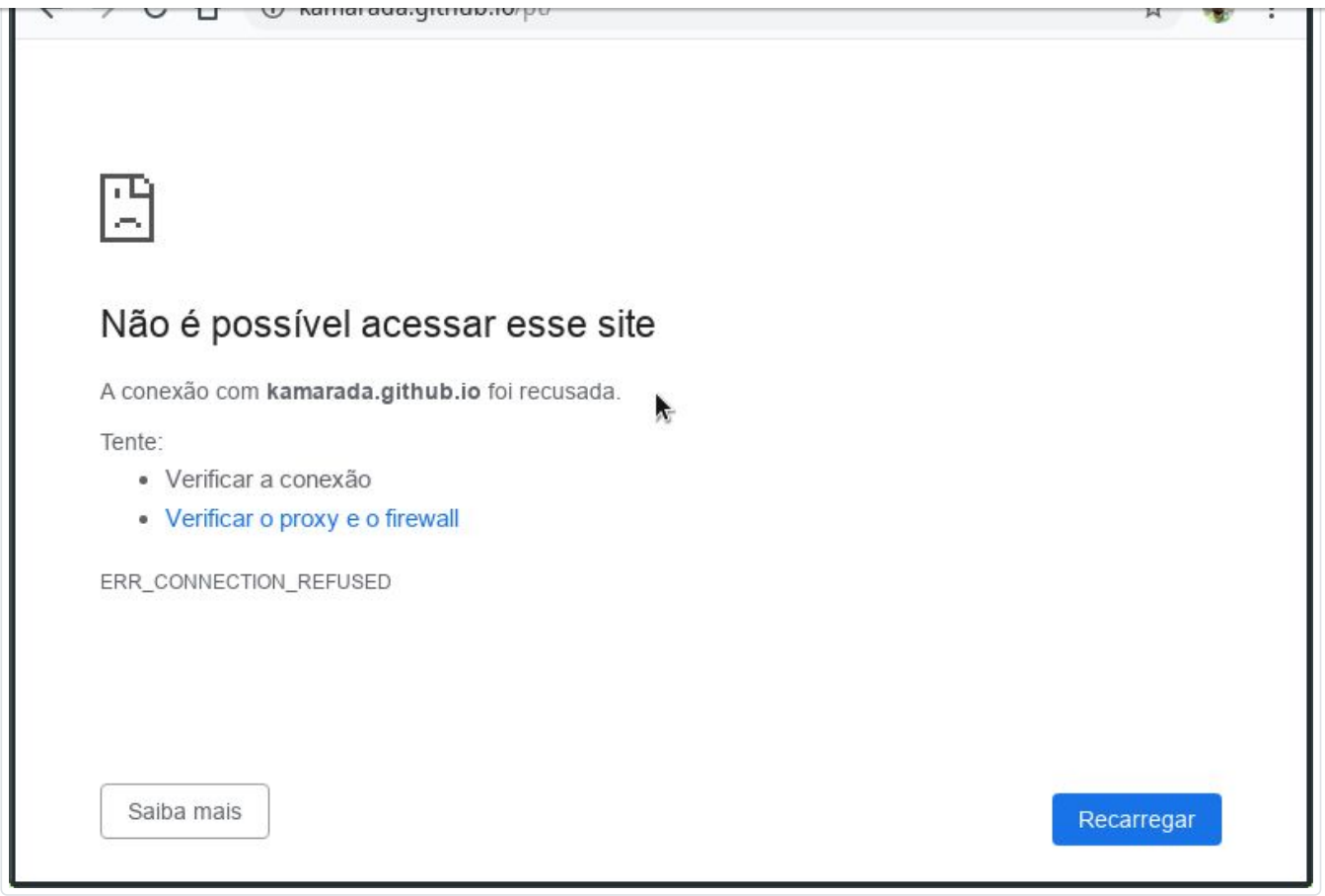
Agora tente acessar o *site* kamarada.github.io usando o navegador (abra esse *link* em uma nova aba). Após muito tempo tentando conectar, o navegador informa que não foi possível acessar o *site* porque ele demorou muito para responder:



Exclua essa regra (para excluir apenas uma regra específica, você pode repeti-la trocando `-A` por `-D`) e adicione outra, usando `REJECT`:

```
# iptables -D OUTPUT -d kamarada.github.io -j DROP
# iptables -A OUTPUT -d kamarada.github.io -j REJECT
```

Agora atualize aquela aba. O navegador responde de imediato, informando que a conexão foi **recusada**:



A depender da sua intenção, pode ser mais interessante usar um ou outro. Por exemplo, eu usaria o `REJECT` para bloquear o acesso a um *site* pela rede interna, porque para o usuário apareceria uma mensagem mais enfática, mas para bloquear um acesso indevido pelo mundo exterior eu usaria o `DROP`, porque um eventual atacante não receberia resposta e se sentiria "dando tiros no escuro".

Salvando e restaurando regras

As regras do **iptables** são armazenadas no *kernel* e começam a ser aplicadas assim que os comandos são executados, mas são perdidas se o sistema é reiniciado.

Dois comandos que você pode usar para salvar e restaurar as regras são o **iptables-save** e o **iptables-restore**, respectivamente.

Para salvar as regras atualmente em uso, execute:

```
1 # iptables-save > /caminho/nome_do_arquivo
```

Para restaurar as regras de um arquivo criado anteriormente, execute:

```
1 # iptables-restore < /caminho/nome_do_arquivo
```

Levantando o firewall com shell script

Embora os comandos **iptables-save** e **iptables-restore** sejam práticos, a forma mais comum de configurar o **iptables** (limpar as regras, definir a política padrão, adicionar regras) é criar um *shell script*. Fica mais legível e fácil de documentar e dar manutenção.

```
# nano /usr/local/sbin/meu-firewall.sh
```

Copie e cole o seguinte conteúdo (**dica:** para colar no terminal, use **Ctrl + Shift + V**):

```
#!/bin/bash
set -ex

# Limpar as regras
iptables -F
iptables -X
iptables -t nat -F
iptables -t nat -X
iptables -t mangle -F
iptables -t mangle -X
iptables -t raw -F
iptables -t raw -X

# Politica padrao: bloquear todas as conexoes de entrada
iptables -P FORWARD DROP
iptables -P INPUT DROP
iptables -P OUTPUT ACCEPT

# Aceitar conexoes estabelecidas previamente
iptables -A INPUT -m state --state ESTABLISHED,RELATED -j ACCEPT

# Aceitar tudo vindo da interface de loopback
iptables -A INPUT -i lo -j ACCEPT

# ICMP (ping)
iptables -A INPUT -p icmp -j ACCEPT

# SSH (porta 22/TCP,UDP)
iptables -A INPUT -p tcp --dport 22 -j ACCEPT
iptables -A INPUT -p udp --dport 22 -j ACCEPT

# HTTP (porta 80/TCP)
iptables -A INPUT -p tcp --dport 80 -j ACCEPT
```

> Esse *script* é um modelo bem simples, pensado para um servidor *web*. Lembre-se de adaptar os comandos conforme a sua necessidade.

Conceda permissão para executar o *script* apenas para o usuário *root*:

```
# chmod 744 /usr/local/sbin/meu-firewall.sh
```

Pronto! Agora, quando quiser levantar o *firewall*, é só chamar o *script*:

```
# /usr/local/sbin/meu-firewall.sh
```

Quando precisar mudar as regras do *firewall*, edite o *script* e execute-o de novo.

Levantando o firewall no boot

Em distribuições com **systemd**, como é o caso do openSUSE e do Linux Kamarada, a melhor forma de executar um *script* como o `meu-firewall.sh` durante o *boot* é criar um serviço.

Por exemplo, crie um serviço do **systemd** chamado `meu-firewall.service` na pasta `/etc/systemd/system/` usando seu editor de texto preferido:

```
# nano /etc/systemd/system/meu-firewall.service
```

Copie e cole o seguinte conteúdo:

```
[Unit]
After=network.target

[Service]
ExecStart=/usr/local/sbin/meu-firewall.sh

[Install]
WantedBy=default.target
```

Ajuste as permissões do arquivo:

```
# chmod 664 /etc/systemd/system/meu-firewall.service
```

Instale e habilite o serviço, para que ele seja iniciado no próximo *boot*:

```
# systemctl daemon-reload
# systemctl enable meu-firewall
```

Se quiser testar seu serviço antes de reiniciar, execute:

```
# systemctl start meu-firewall
# iptables -L
```

Pronto! Da próxima vez em que você reiniciar o sistema, o **systemd** se encarregará de iniciar o serviço que executa o *script* que configura o **iptables**.

Referências

Espero que esse texto tenha sido útil. Se você o leu todo até aqui, já dispõe de conhecimento suficiente para implantar o **iptables** em seus servidores. Se quiser saber mais, leia:

- Guia Foca GNU/Linux - Firewall iptables
- O que é firewall? - Conceito, tipos e arquiteturas - InfoWester
- IPTables - Desvendando o mistério - Viva o Linux
- iptables(8) - Linux man page
- Basic iptables Tutorial - SUSE Communities
- Linux Iptables block incoming access to selected or specific ip address - nixCraft
- DROP ou REJECT no iptables? - Viva o Linux
- How to automatically execute shell script at startup boot on systemd Linux - LinuxConfig.org

Gostou? Que tal compartilhar?