

ARQUITETURA DE **SERVIDORES DE REDE**

Esp. Geiza Caruline Costa

INICIAR

introdução

Introdução

Nesta unidade, aprenderemos os fundamentos da arquitetura cliente-servidor, os princípios relacionados ao uso de *sockets* em comunicação de dados, além da importância do *firewall* em uma rede corporativa.

Destacaremos, ainda, a arquitetura cliente-servidor, analisando quando um dado, enviado por um dispositivo, não chega ao seu destino final, e pode estar em trânsito ou em um computador servidor.

Abordaremos, também, o conteúdo sobre *sockets*, destacando o endereçamento das mensagens, a identificação correta das portas e o protocolo utilizado, que exercem um papel fundamental na comunicação de dados.

No final da unidade, veremos como fazer a ativação do *firewall* em um servidor, de modo a tornar a rede corporativa mais segura.

Apresentação do Modelo Cliente-Servidor

A arquitetura cliente-servidor – também conhecida por modelo cliente-servidor – é um paradigma relacionado à forma como os computadores se estruturam na rede de comunicação de dados, em termos de demanda e oferta de serviços. Significa, na prática, quando um usuário demanda um serviço, o acesso a uma base de dados, por exemplo, e a resposta é dada por um computador, que detém um sistema criado, especificamente, para atender às requisições dos usuários, tem-se, portanto, o modelo cliente-servidor em ação.

Quando um cliente usa um aplicativo no seu smartphone para localizar motoristas particulares, sabemos que, na verdade, essa solicitação é enviada do aplicativo do cliente para um *software* instalado em um computador central, que chamamos servidor – que gerencia as solicitações, localiza motoristas disponíveis na região e estabelece um preço. O programa instalado em um computador servidor pode ser chamado processo-servidor, enquanto o programa acessado diretamente pelo usuário é conhecido por processo-cliente.

Nesse contexto, o servidor pode ser um dispositivo, um serviço, ou uma

aplicação, cujo objetivo principal é atender às requisições dos clientes. O servidor fica o tempo todo aguardando as solicitações de usuários, no entanto, os processos-clientes são executados apenas quando necessário.

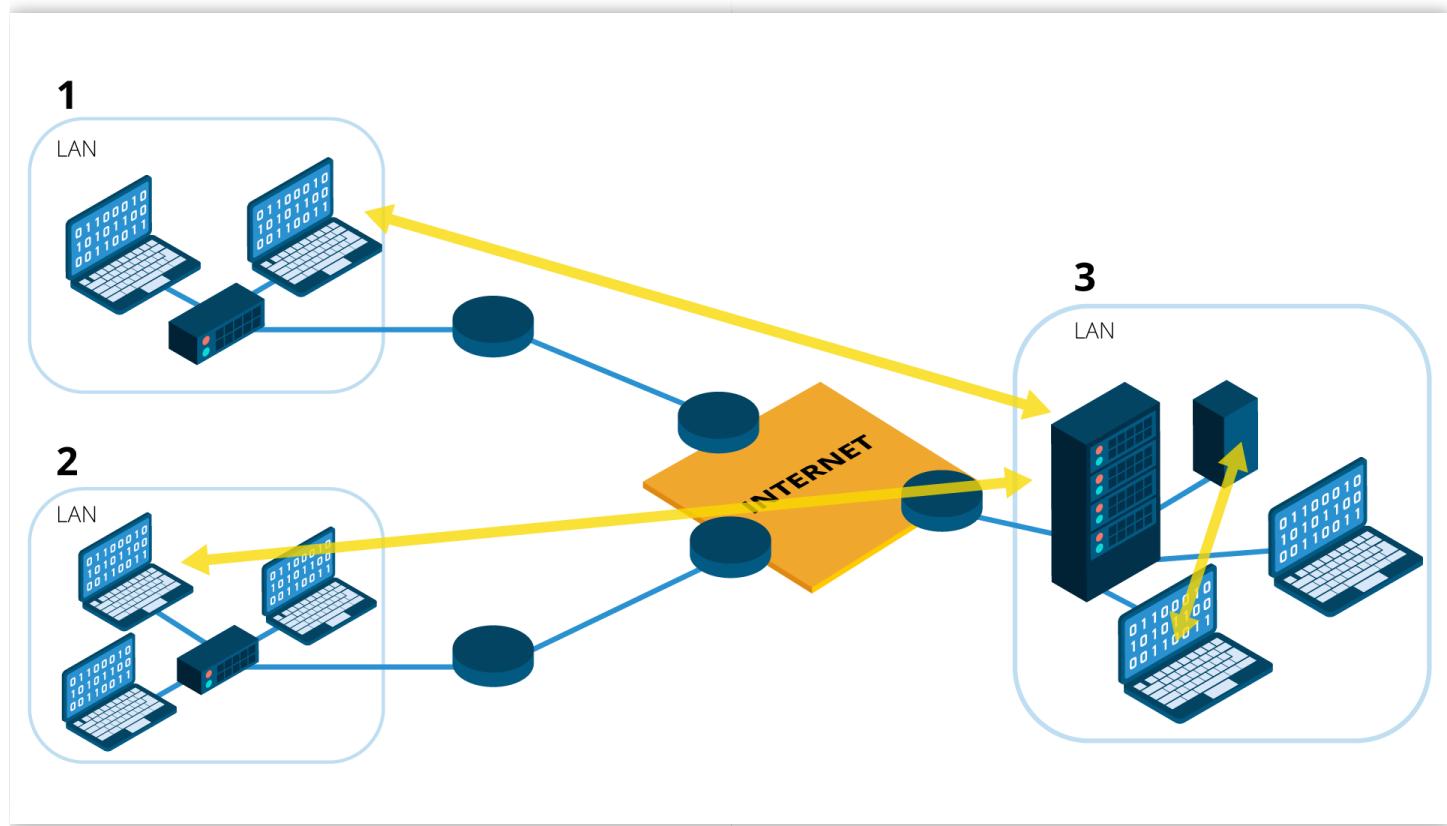


Figura 1.1 – Diagrama que exemplifica o modelo cliente-servidor

Fonte: Adaptada de Forouzan e Mosharraf (2013).

A Figura 1.1 apresenta um exemplo de troca de dados entre clientes e servidores. A LAN (Local Area Network) representada no quadro 1 da figura 1.1, ilustra dois notebooks interconectados, e um deles está conectado através da WAN (Wide Area Network) ao servidor da LAN 3. Por sua vez, o servidor da LAN 3 também está atendendo requisições de um notebook da LAN 3, e de outro, da LAN 2. Logo, temos que um servidor de rede, quando configurado, pode atender requisições de outros dispositivos distribuídos na LAN, ou geograficamente distantes.

Você pode ter identificado que, na Figura 1.1, clientes e servidores são representados por imagens diferentes: clientes como *notebooks* e servidores como torres/gabinetes. Isso ocorre, porque, geralmente, servidores requerem grande capacidade de processamento e memória. Para computadores com processadores de alto nível e funcionamento 24 horas por dia, são

necessárias fontes de energia e acesso à rede ininterruptos, sistema de refrigeração adequado, discos de armazenamento de grande volume. No caso dos clientes, basicamente é necessário ter acesso à *internet* ou a LAN para acesso aos serviços hospedados, já que a maior parte do processamento é feita no servidor.

Forouzan e Mosharraf (2013) apresentam um problema que administradores de rede precisam considerar quando adotam o modelo cliente-servidor:

Um problema com esse paradigma é que a carga de comunicação concentra-se no lado do servidor, o que significa que este deve ser um computador poderoso. Porém, mesmo um computador poderoso pode acabar sobrecarregado se um grande número de clientes tentar se conectar ao servidor ao mesmo tempo. Outro problema é que deve haver um provedor de serviços disposto a criar um servidor poderoso para um serviço específico e arcar com os custos envolvidos, o que significa que o serviço deve sempre gerar algum tipo de receita para o servidor de modo a incentivar tal empreendimento. (FOROUZAN; MOSHARRAF, 2013, p. 37).

Um servidor *firewall* , por exemplo, pode ser implantado numa rede corporativa local para gestão das políticas de segurança. Considerando a rede mundial de computadores, o modelo cliente-servidor pode ser implantado de forma distribuída. A Web é um exemplo de serviço cliente-servidor distribuído. O serviço é distribuído em diversos servidores, chamados *sites* , e os clientes são os navegadores dos usuários, que desejam acessar uma página (FOROUZAN; MOSHARRAF, 2013, p. 44).

atividade

atividade

A World Wide Web é um exemplo de serviço que funciona no modelo cliente-servidor. Segundo Forouzan e Mosharraf (2013), um “cliente envia um pedido, um servidor responde, e o relacionamento entre as duas partes acaba aí”, mas atualmente, a Web possui outras funcionalidades, e pode ser necessário recuperar algumas informações sobre os clientes, que foram salvas nos *cookies* do navegador.

FOROUZAN, B. A. MOSHARRAF, F. **Redes de computadores** : uma abordagem top-down. Porto Alegre: AMGH, 2013. p. 55.

Considerando o trecho citado, analise as sentenças a seguir e assinale a única verdadeira:

- a)** Um usuário acessa um site da Web, que salva a data da última visita no *cookie* em seu computador. Nesse tipo de comunicação, não é utilizado o modelo cliente-servidor, pois foi criada uma terceira instância de armazenamento, que não é no cliente nem no servidor.
- b)** Um desenvolvedor Web criou um sistema que armazena dados dos usuários na sessão do servidor, em vez de usar *cookies* para armazenar dados simples dos usuários. Dessa forma, fica preservado o modelo cliente-servidor.
- c)** A única forma de utilização do modelo cliente-servidor, na *internet*, é impedir que os *sites* da Web salvem *cookies* nos navegadores dos usuários.
- d)** Quando um servidor Web, onde está hospedado um site, requisita dados do computador do usuário, o modelo cliente-servidor não está mais sendo aplicado.
- e)** Um desenvolvedor Web criou um sistema que acessa dados de outros sites através de links. O usuário poderia acessar diretamente os sites de terceiros, mas, independentemente disso, o modelo cliente-servidor permanece em uso.



Sockets TCP



Agora que aprendemos os fundamentos do modelo cliente-servidor, podemos analisar como acontece a comunicação entre cliente e servidor em uma rede.

Os dispositivos interessados em estabelecer uma comunicação em rede têm de ter uma identificação. O servidor, por exemplo, precisa de um identificador, para que os clientes consigam encontrá-lo na rede, que pode ser um endereço IP (Internet Protocol) ou um nome (*hostname*). Os clientes podem não ter conhecimento do endereço IP do servidor, mas, pelo menos, seu nome deverá ser conhecido.

Para os dispositivos em uma rede não importam os nomes dos computadores, apenas seus endereços IP, mas, para os seres humanos, é mais conveniente decorar nomes em vez de números. Por essa razão, o sistema DNS é utilizado para fazer a resolução de nomes, ou seja, converter nomes de servidores aos respectivos endereços IP.

O Domain Name System (DNS) fornece um serviço que mapeia nomes simbólicos legíveis por seres humanos para endereços de computadores. Navegadores, softwares de e-mail e a maioria dos

outros aplicativos da Internet usam o DNS. O sistema fornece um interessante exemplo de interação cliente-servidor, pois o mapeamento não é executado por um simples servidor (COMER, 2016, p. 63).

Uma vez que o servidor foi identificado, é preciso identificar o serviço que pode oferecer. Nesse sentido, podemos considerar que um servidor pode atender diversos usuários, bem como oferecer variados serviços. De modo análogo, uma empresa pode estar instalada em um prédio que abriga, ainda, muitas outras empresas. Como um carteiro poderia saber, precisamente, onde entregar uma correspondência se no envelope da carta estivesse escrito apenas os dados que identificam o prédio?

A identificação do serviço é dada pelo número de porta, um código de 16 bits , utilizado pelo sistema operacional para endereçar as requisições dos processos-clientes aos serviços adequados em um servidor.

Uma requisição com a correta identificação do servidor segue a estrutura apresentada no quadro 1.1:

Descrição: **protocolo :// endereço_ip : número_porta**

Exemplo 1: http://192.168.1.120:25

Exemplo 2: http://mailserver.com:25

Quadro 1.1 – Endereço de socket

Fonte: Elaborada pela autora.

O quadro 1.1 apresenta dois exemplos de identificação de servidor e serviço: na primeira linha, temos um descritivo, que apresenta o protocolo, seguido pelo endereço IP e a porta. No primeiro exemplo, o protocolo utilizado é o HTTP (Hypertext Transfer Protocol), seguido de dois pontos e barra; endereço IP vem em seguida, em um campo que pode ser substituído pelo nome do host correspondente. Por fim, o número de porta, que nesses exemplos, corresponde à porta número 25.

O conjunto de endereço IP (ou *hostname*) mais o número de porta podem ser denominados endereço de *socket*. Em programação de computadores para redes são usadas APIs (Application Program Interface) para o desenvolvimento de aplicações que se comunicam na *internet*, ou até em uma LAN.

Além do endereço de *socket*, o desenvolvedor de *software* deverá especificar se a aplicação será um processo-servidor ou um processo-cliente. Isso porque as APIs de *sockets* possuem funções que, na aplicação, podem ser usadas tanto em processos do tipo cliente, quanto em processos do tipo servidor, mas há outras funções exclusivas para um ou para outro.

	<i>accept</i>	Aceita uma chamada entrante
Servidor	<i>listen</i>	Prepara o <i>socket</i> para uso do servidor
	<i>getpeername</i>	Obtém o endereço do cliente

Quadro 1.2 – Funções de uso exclusivo no processo-servidor em programação com *sockets*

Fonte: Adaptada de Comer (2016).

No quadro 1.3 é possível observar a programação socket voltada para funções do tipo cliente.

Cliente	<i>connect</i>	Conecta com a aplicação remota
----------------	----------------	--------------------------------

Quadro 1.3 – Funções de uso exclusivo no processo-cliente em programação com *sockets*

Fonte: Adaptada de Comer (2016).

A seguir, podem-se verificar informações relacionadas às funções, tanto de processo servidor como de clientes em sockets.

Servidor e cliente	<i>socket</i>	Cria o <i>socket</i>
	<i>recv</i>	Recebe dados ou mensagens entrantes
	<i>close</i>	Termina a comunicação
	<i>send</i>	Envia dados

Quadro 1.4 – Funções de uso no processo-servidor ou no processo-cliente em programação com *sockets*

Fonte: Adaptada de Comer (2016).

Um serviço confiável é aquele que oferece a garantia de recebimento dos dados, ou seja, em caso de falhas na transmissão, automaticamente é feito o reenvio dos dados perdidos. Caso seja preciso prover um serviço confiável, é indicada a utilização de sockets TCP (Transmission Control Protocol).

TCP é um protocolo da camada de transporte, orientado à conexão, cuja principal característica é prover a entrega de dados livre de erros e de forma ordenada, ao destinatário.

Para a comunicação entre processos do tipo cliente e servidor sob o protocolo TCP, o processo-cliente deve contatar o servidor. Desse modo, o processo-servidor já deverá estar em execução e aceitando conexões. A dinâmica de criação de sockets TCP é a seguinte:

- **cliente** cria um *socket* TCP local, especificando IP e porta do processo-servidor;
- **servidor** recebe solicitação de conexão;
- **cliente** estabelece comunicação;
- **servidor** cria um novo *socket* TCP, com o número de porta do processo-cliente – isso permite que múltiplas conexões sejam criadas com clientes diferentes.

Exemplo de Codificação

As Figuras 2 e 3 exemplificam um sistema escrito com a linguagem Java, criado para prover uma comunicação confiável entre um processo-servidor (Figura 2) e processo-cliente (Figura 3).

```
1 import java.io.*;
2 import java.net.*;
3 ServerSocket welcomeSocket = new ServerSocket(numero_da_porta);
4 while(true) {
5     ...
6 }
7 Socket connectionSocket = welcomeSocket.accept();
8 BufferedReader infoDoCliente =
9     new BufferedReader(new InputStreamReader(connectionSocket.getInputStream()));
10 String mensagemDoCliente = infoDoCliente.readLine();
11 DataOutputStream infoParaCliente =
12     new DataOutputStream(connectionSocket.getOutputStream());
13 infoParaCliente.writeBytes(mensagem_para_cliente);
```

Figura 1.2 – Exemplo de codificação do processo-servidor em linguagem Java, utilizando socket TCP

Fonte: Adaptada de WikiLivros (2019).

Na Figura 1.2 podemos observar, na linha 3, que é criado um objeto *socket* do lado do servidor, em que o **número_da_porta** deverá ser trocado pelo número de porta pela qual o processo-cliente usará para conectar ao servidor. O laço de repetição infinito da linha 4 permite que o processo-servidor seja executado continuamente. Na linha 7, vemos que, quando o processo-cliente se conectar ao processo-servidor, será criada uma conexão virtual entre os envolvidos na comunicação. Daí em diante, é feita a troca de dados de *input* (recebimento no servidor) e *output* (envio pelo servidor).

reflita
reflita

Os endereços de socket, numa conexão entre cliente e servidor, são diferentes para cada lado da comunicação, ou seja, existe uma porta e um endereço IP para o cliente, e outra porta e endereço IP para o servidor.

Eventualmente, endereços de porta podem ser os mesmos, mas endereço IP não deve se repetir, visto que representa um identificador único da interface de rede de um computador.

Pense nas formas de verificar quais conexões estão abertas no seu computador neste momento. Considere, ainda, que mesmo com seu navegador fechado, pode existir comunicação do seu computador com diversos outros sistemas da Web.

```
1 import java.io.*;
2 import java.net.*;
3 Socket clientSocket =
4     new Socket("nome_do_servidor", numero_da_porta);
5 DataOutputStream infoParaServidor =
6     new DataOutputStream(clientSocket.getOutputStream());
7 BufferedReader infoDoServidor =
8     new BufferedReader(new InputStreamReader(client.getInputStream()));
9 infoParaServidor.writeBytes(mensagem_do_cliente);
10 String mensagem_para_cliente = infoDoServidor.readLine();
11 clientSocket.close();
```

Figura 1.3 - Exemplo de codificação do processo-cliente em linguagem Java, utilizando socket TCP

Fonte: Adaptada de WikiLivros (2019).

A Figura 1.3 exemplifica uma aplicação que executa o processo-cliente. Podemos verificar que a linha 3 cria um *socket*, em que está escrito “*nome_do_servidor*”, e deve ser substituído pelo *hostname* ou pelo endereço IP do servidor. Os objetos *infoDoServidor* e *infoParaServidor*, das linhas 5 e 7, serão ligados ao *socket* para enviar e receber cadeias de dados. A função *close*, que foi vista no quadro 4, foi implementada na linha 11, e expressa a intenção, no cliente, em encerrar a comunicação.

atividade

atividade

Sobre identificação de portas e *hosts* em comunicação usando *sockets TCP*, assinale a única alternativa correta.

- a)** Em um sistema-cliente, para criar um *socket* não é preciso identificar o servidor, mas é preciso identificar o cliente.
- b)** Em um sistema-cliente, para criar um *socket* é preciso identificar o servidor e a porta.
- c)** Em um sistema-servidor, não é necessário identificar número de porta.
- d)** Em um sistema-cliente, a função *accept* significa que o cliente está aceitando chamadas de servidores.
- e)** Usando *sockets TCP* não é possível estabelecer uma comunicação segura.



Sockets UDP



Iniciamos este tópico destacando que, segundo Comer (2016), na *internet*, todos os serviços são fornecidos por aplicações que utilizam ou o paradigma de fluxo, como é o caso de *sockets TCP*, ou o paradigma de mensagem. Este último oferece um serviço sem garantias de entrega dos dados ao destinatário, ou seja, um serviço de comunicação não confiável. A implementação da troca de dados entre programas sem garantia de recebimento é feita usando *sockets UDP*.

UDP (User Datagram Protocol) é um protocolo da camada de transporte não orientado à conexão, indicado para transmissões de dados não sensíveis, ou seja, dados que podem ser perdidos sem prejuízo para a comunicação.

Assim como *sockets TCP*, os *sockets UDP* também contam com a identificação de servidor e serviço (IP ou *hostname* e número de porta). No entanto, a principal diferença do socket UDP, em relação ao anterior, é que o processo-cliente UDP pode criar um *socket* e enviar dados para processos-servidores diferentes.



Figura 1.4 - Câmera de vídeo com transmissão ao vivo

Fonte: Freepik.com.

A utilização de sockets UDP é mais indicada para transmissões cuja perda de dados não é um problema, mas o envio de dados atrasados ou o reenvio de dados que talvez tivessem chegado com erro resulta em uma comunicação prejudicada.

Nesse sentido, podemos considerar uma transmissão ao vivo de uma partida de futebol pela televisão. Uma série de imagens estão sendo enviadas de modo contínuo aos clientes, de maneira a produzir um vídeo. Considere que o processo-servidor identificou que uma parte da sequência de imagens não foi enviada corretamente, e, após alguns milissegundos, fizesse o reenvio. Sabe-se que a imagem seria reproduzida pelo sistema-cliente, após o tempo de processamento mais o tempo de tráfego. Nessa situação, na apresentação de uma imagem, considerando um cenário bastante dinâmico como uma partida de futebol, pode ser mais relevante descartar as imagens atrasadas e continuar a reproduzir o conteúdo que vem sendo recebido no processo-cliente.

Por essas razões, e pelo fato de o protocolo UDP ser mais simples que o do

TCP, sockets UDP são mais indicados para aplicações com transmissão de fluxo de áudio e vídeo.

Exemplo de Codificação

As Figuras 1.5 e 1.6 exemplificam a codificação dos processos executados no servidor e no cliente, com a implementação de *sockets UDP*.

```
1 import java.net.*;
2 import java.io.*;
3
4 DatagramSocket s = new DatagramSocket(numero_da_porta);
5 while (true) {
6     ...
7 }
8 byte[] buffer = new byte[1000];
9
10 DatagramPacket infoDoCliente =
11     new DatagramPacket(buffer, buffer.length);
12 s.receive(infoDoCliente);
```

Figura 1.5 - Exemplo de codificação do processo-servidor em linguagem Java, utilizando socket UDP Fonte: Adaptada de Oracle (2017).

Na Figura 1.5, foi criado um socket UDP de nome *s* na linha 4; **numero_da_porta** indica a porta que o servidor receberá dados. Na linha 10, foi criado um objeto DatagramPacket, nomeado *infoDoCliente*, que permitirá capturar dados que possam identificar a origem dos dados (como IP e porta do remetente) e o tamanho do que foi recebido.

```
1 import java.net.*;
2 import java.io.*;
3
4 DatagramSocket socket = new DatagramSocket();
5
6 String hostname = "hostname_server.com";
7 int port = numero_da_porta;
8
9 InetAddress address = InetAddress.getByName(hostname);
10
11 byte[] buffer = new byte[512];
12
13 DatagramPacket request = new DatagramPacket(buffer,
14     buffer.length, address, port);
15 socket.send(request);
16
17 DatagramPacket response = new DatagramPacket(buffer,
18     buffer.length);
19 socket.receive(response);
20 socket.close();
```

Figura 1.6 - Exemplo de codificação do processo-cliente em linguagem Java, utilizando socket UDP

Fonte: Adaptada de WikiLivros (2019).

Na linha 6, foi criada uma String, para a qual foi atribuída o nome do servidor. Podemos observar que, na linha 9, o endereço IP do servidor é obtido e atribuído a variável *address*. Isso é útil, pois, mesmo que o servidor mude de endereço IP, a aplicação continua funcionando. A função *send* , na linha 15,

envia uma requisição ao servidor, cuja resposta é recebida através da função *receive* , na linha 18.

Quando se compara o uso de *sockets* TCP e UDP, pode-se identificar desempenho superior na utilização do protocolo UDP, pois não há perda de tempo com verificações e retransmissões. Entretanto, como no UDP não há controle de fluxo, pode haver um gargalo na transmissão, quando a capacidade de envio de dados pelo processo-cliente for muito superior à capacidade de recepção do processo-servidor.

Saiba mais

Saiba mais

Visite o *site* a seguir e veja mais detalhes sobre desenvolvimento de *sockets* em aplicações Java, tanto em rede local como com acesso à Internet. Atente-se também aos conceitos de datagramas, a utilidade desse tipo de recurso e quais são suas respectivas classes em Java.

[ACESSAR](#)

atividade

atividade

Assinale a alternativa que indica uma aplicação para a qual o protocolo TCP é mais indicado:

- a)** Chamada de voz via Skype®
- b)** Chamada de voz via WhatsApp®
- c)** Transmissão de vídeo pelo Vimeo®
- d)** Transmissão de vídeo pelo YouTube®
- e)** Transmissão de mensagens de e-mail via Outlook®



Firewall



Quanto maior for o tráfego de rede, maior será o interesse de possíveis invasores na obtenção dos dados dos usuários, sobretudo em uma rede corporativa. A fragilidade dos sistemas de segurança expõe, além dos usuários de uma rede corporativa, os clientes e os fornecedores.



Figura 1.7 - Invasores podem estar interessados no tráfego de sua rede
Fonte: Freepik.com

A segurança da informação depende, não somente dos investimentos do setor de Tecnologia da Informação em recursos físicos, mas envolve conscientização e capacitação dos usuários de uma rede de computadores.

Em termos de sistemas, algumas tecnologias podem contribuir para a segurança na rede, como o *firewall*. Esse sistema é projetado para proteger os computadores da LAN contra invasores e tráfego indesejado. Um *firewall* é colocado entre uma organização e o resto da Internet, e todos os pacotes que entram ou saem da rede corporativa passam por ele (COMER, 2016).

A análise da necessidade de implantação do *firewall* em uma empresa passa pelo aval da diretoria ou de instâncias hierarquicamente superiores, que poderão avaliar quais as implicações administrativas que o uso desse sistema pode acarretar.

A implantação começa por um protótipo pelo qual seja possível estabelecer e testar as regras de segurança. A contratação de um *hacker* experiente contribui para verificar as falhas restantes e aprimorar a configuração e eficácia do sistema (KOLBE JÚNIOR, 2017).

Classificação

Os sistemas de *firewall* podem ser classificados pelo seu nível de atuação.

Nível de rede

Tomam decisões com base nos dados contidos nos cabeçalhos dos pacotes, como IP de origem, IP de destino, protocolo utilizado, números de portas.
Exemplo: Netfilter.

Saiba mais
Saiba mais

O Netfilter é um *framework* gratuito, incluso no kernel Linux, capaz de fazer a filtragem de pacotes, tradução de endereços, redirecionamento de portas, entre outras funcionalidades.

[ACESSAR](#)

4.1.2 Nível de Aplicação

Além de analisar os cabeçalhos, os níveis de aplicação podem verificar as cargas de dados úteis dos pacotes, ou seja, o conteúdo em si, que está sendo transmitido na rede. Exemplo: Squid.

Saiba mais
Saiba mais

O Squid é um servidor de proxy, que estabelece controle de acesso, autenticação de usuários, e outras funções, incluindo a personalização de páginas de erros aos usuários.

ACESSAR

Ativação do Netfilter

Iptables é o módulo do Netfilter que permite acesso aos administradores rede para efetuar a configuração de regras, ou seja, funciona como uma interface. Por ser muito popular, eventualmente, as pessoas confundem o Netfilter com o *iptables* .

O *iptables* está disponível em, praticamente, todas as distribuições mais populares do sistema operacional Linux, como Ubuntu, SuSe, CentOS, Debian e Fedora, sem a necessidade de instalações adicionais.

Para começar a utilizar o *iptables* e configurar o Netfilter, é preciso diferenciar um fluxo de entrada de um de saída da rede de computadores. O tráfego de entrada é originário em outro computador com destino à máquina local; o de saída é originário na máquina local com destino a outro *host* .

Podemos considerar que o administrador de uma rede verificou que determinado servidor apresentou, repentinamente, um desempenho inferior ao habitual. Isso pode acontecer em decorrência de uma sobrecarga. Esse servidor pode estar recebendo uma quantidade de requisições excessiva de processos-clientes em determinada porta. Caso essa porta seja conhecida e o administrador queira bloquear as requisições ao processo-servidor, basta digitar o comando abaixo, no terminal do sistema operacional, substituindo a expressão NÚMERO-DA-PORTE.

```
iptables -A INPUT -p tcp --destination-port {NÚMERO-DA-PORTA} -j  
DROP
```

Quadro 1.5 – Exemplo de definição de regra para bloqueio de porta de entrada

Fonte: Adaptada de Brito (2017).

No quadro 1.5 é dado um exemplo da definição de uma regra de bloqueio para o *firewall*, que começa a valer imediatamente, para todos os pacotes que trafegam pela placa de rede nesse computador.

Nesse comando, o parâmetro `-p` indica o protocolo, no caso, TCP. O parâmetro `--destination-port` significa que a regra se aplica aos pacotes destinados à porta indicada. O parâmetro `-j` indica que todos os pacotes serão tratados segundo essa regra. Por fim, o parâmetro `DROP` indica que, de acordo com essa regra, os pacotes serão bloqueados.

atividade

atividade

Segundo Brito (2017), as pessoas costumam utilizar o firewall, somente nas bordas das redes corporativas, em que existe uma clara separação entre a rede interna e as redes externas, no contexto da Internet. Contudo, um sistema de firewall também pode ser utilizado para filtragem de pacotes, dentro de uma LAN.

BRITO, S. H. B. **Serviços de redes em Servidores Linux**. São Paulo: Novatec, 2017.

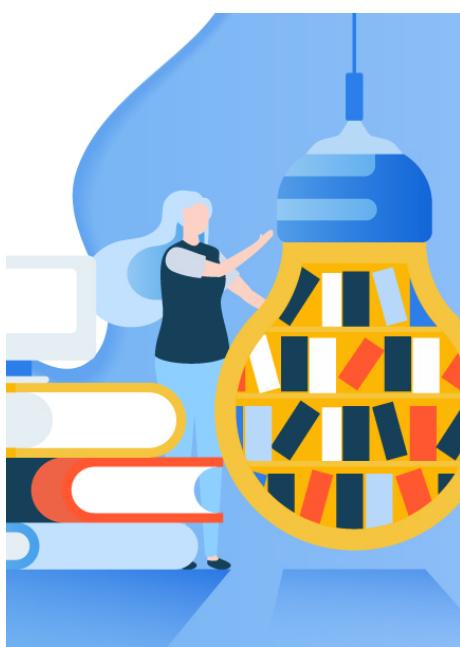
Assinale uma justificativa adequada para a análise de pacotes de tráfego na rede local.

- a)** Para viabilizar o controle de acesso entre as várias sub-redes, quando houver.
- b)** Para impedir que usuários mal-intencionados, dentro da rede, promovam acesso indevido a *sites* externos.
- c)** Para impedir que *hackers* fora da rede promovam acesso indevido aos sistemas corporativos.
- d)** Para impedir que os clientes internos tenham acesso físico aos servidores da empresa.
- e)** Para dificultar o acesso dos visitantes à LAN.

indicações

Material

Complementar

**LIVRO**

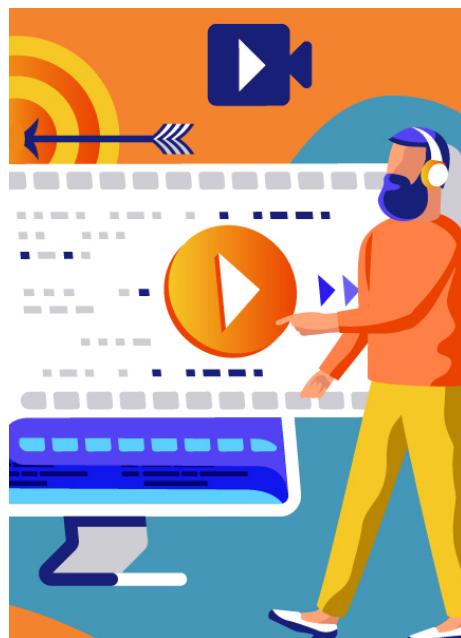
Programação de Sistemas Distribuídos em Java

Editora : FCA

Autor : Jorge Cardoso

ISBN : 9789727226016

Comentário : para aprofundar a aprendizagem em programação de computadores voltada para redes usando *sockets* e outras tecnologias, consulte uma das obras mais conhecidas na área. Nesse livro, o autor apresenta uma visão geral acerca de computação, desde os primeiros computadores, tipos de redes e componentes até detalhes sobre programação com *sockets* usando a linguagem Java.

**FILME****Firewall - Segurança em Risco****Ano :** 2006

Comentário : assista a este filme, e reflita acerca das principais vulnerabilidades de segurança exploradas e como preservar os dados do protagonista.

TRAILER

conclusão

Conclusão

Nesta unidade, estudamos a utilização do modelo cliente-servidor, que implica a forma como os sistemas são desenvolvidos, como é feita a troca de dados por meio de uma rede e até de que maneira o tráfego pode ser monitorado, por questões de segurança.

Destacamos, também, que a identificação dos serviços, dos servidores, dos clientes e utilização de *sockets* também sofrem grande influência dos protocolos de comunicação, como UDP e TCP.

Por fim, destacamos que a administração de uma rede de computadores requer conhecimentos técnicos, do ambiente de rede, além das ameaças à segurança da rede e como lidar com elas.

referências

Referências Bibliográficas

BRITO, S. H. B. **Serviços de redes em servidores Linux** . São Paulo: Novatec, 2017.

COMER, D. E. **Redes de computadores e internet** [recurso eletrônico]. 6. ed.

Porto Alegre: Bookman, 2016.

FOROUZAN, B. A.; MOSHARRAF, F. **Redes de Computadores** : uma abordagem top-down. Porto Alegre: AMGH, 2013.

KOLBE JÚNIOR, A. **Sistemas de segurança da informação na era do conhecimento** [livro eletrônico]. Curitiba: InterSaber, 2017.

IMPRIMIR