Aplicações e Serviços de Rede

Prof. Ricardo Mesquita

Noções Básicas de Serviços

- Os **serviços TCP** estão entre os mais fáceis de entender porque são construídos sobre fluxos de dados bidirecionais simples e ininterruptos.
- Talvez a melhor maneira de ver como eles funcionam seja falar diretamente com um servidor Web não criptografado na porta TCP 80 para se observar e como os dados se movem pela conexão.
- Vamos executar um exemplo.

Noções Básicas de Serviços

- Por exemplo, execute o seguinte comando para conectar-se ao servidor Web de exemplo da documentação da IANA:

```
$ telnet example.org 80
```

- Você deve obter uma resposta como esta, indicando *uma conexão bem-sucedida* com o servidor:

```
Trying some address...
Connected to example.org.
Escape character is '^]'.
```

Noções Básicas de Serviços

- Agora insira estas duas linhas:

```
GET / HTTP/1.1
Host: example.org
```

- Pressione ENTER *duas vezes* após a última linha.
- O servidor deve enviar um texto HTML como resposta.
- Para encerrar a conexão, pressione Ctrl-D.

Noções Básicas de Serviços

- Esse exercício demonstra que:
  - O host remoto tem um processo servidor Web "escutando" na porta TCP 80.
  - O telnet foi o cliente que iniciou a conexão.
- O motivo pelo qual você tem que encerrar a conexão com Ctrl-D é que, como a maioria das páginas da Web precisa de várias solicitações para carregar, faz sentido *manter a conexão aberta*.
- Se você explorar servidores Web no nível do protocolo, poderá descobrir que esse comportamento varia.
  - Por exemplo, muitos servidores se desconectam rapidamente se não receberem uma solicitação logo após a abertura de uma conexão.

Observações

- O telnet foi originalmente criado para habilitar logins em hosts remotos.
- O programa cliente pode não ser instalado na sua distribuição Linux por padrão, mas é facilmente instalado como um pacote extra.
- Embora o servidor de login remoto telnet seja *completamente inseguro*, o cliente telnet pode ser útil para depurar serviços remotos.
- O telnet *não funciona* com UDP ou qualquer protocolo de transporte diferente de TCP.

Uma Análise mais Detalhada

- No exemplo anterior, interagimos manualmente com um servidor Web na rede com telnet, usando o protocolo de camada de aplicação HTTP.
- Vamos, agora, usar o utilitário curl com uma opção para registrar detalhes sobre a comunicação:

```
$ curl --trace-ascii trace_file http://www.example.org/
```

Uma Análise mais Detalhada

- Você obterá muita saída HTML. Ignore-a (ou redirecione-a para /dev/null *[lembra como se faz?)* e, em vez disso, olhe para o arquivo recém-criado trace_file.
- Se a conexão foi bem-sucedida, a primeira parte do arquivo deve ser parecida com a seguinte, no ponto em que curl tenta estabelecer a conexão TCP com o servidor:

```
== Info: Trying 93.184.216.34...
== Info: TCP_NODELAY set
== Info: Connected to www.example.org (93.184.216.34) port 80 (#0)
```

- Tudo o que você viu até agora acontece na camada de transporte ou abaixo.

Uma Análise mais Detalhada

- No entanto, se essa conexão for bem-sucedida, o curl tenta enviar a solicitação (o "cabeçalho"); é aqui que a camada de aplicação começa:

```
=> Send header, 79 bytes (0x4f)
0000: GET / HTTP/1.1
0010: Host: www.example.org
0027: User-Agent: curl/7.58.0
0040: Accept: */*
004d:
```

Note:
- A linha 1 é a saída de depuração do curl informando o que ele fará em seguida.
- As linhas restantes mostram o que o curl envia para o servidor.
- O texto em negrito é o que vai para o servidor; os números hexadecimais no início são apenas offsets de depuração

Uma Análise mais Detalhada

- Em seguida, o servidor envia uma resposta, primeiro com seu próprio cabeçalho, mostrado aqui em negrito:

```
<= Recv header, 17 bytes (0x11)
0000: HTTP/1.1 200 OK
<= Recv header, 22 bytes (0x16)
0000: Accept-Ranges: bytes
<= Recv header, 12 bytes (0xc)
0000: Age: 17629
--snip--
```

- As linhas <= são saídas de depuração, e 0000: precede as linhas de saída para informar os deslocamentos.
  - No curl, o cabeçalho não contará para o deslocamento; é por isso que todas essas linhas começam com 0.

Uma Análise mais Detalhada

- O cabeçalho da resposta do servidor pode ser bastante longo, mas em algum momento o servidor faz a transição da transmissão de cabeçalhos para o envio do documento solicitado. Algo assim:

```
<= Recv header, 22 bytes (0x16)
0000: Content-Length: 1256
<= Recv header, 2 bytes (0x2)
0000:
<= Recv data, 1256 bytes (0x4e8)
0000: <!doctype html>.<html>.<head>.    <title>Example Domain</title>.
0040: .    <meta charset="utf-8" />.    <meta http-equiv="Content-type
--snip--
```

O curl "entende" que quando recebe uma linha em branco (0000) - o fim dos cabeçalhos HTTP - ele deve interpretar qualquer coisa que se siga como o documento solicitado.

Servidores de Rede

- **httpd**, **apache**, **apache2**, **nginx**: Web servers
- **sshd**: Secure shell daemon
- **postfix**, **qmail**, **sendmail**: Mail servers
- **cupsd**: Print server
- **nfsd**, **mountd**: Network filesystem (file-sharing) daemons
- **smbd**, **nmbd**: Windows file-sharing daemons
- **rpcbind**: Remote procedure call (RPC) portmap service daemon

Servidores de Rede

- Uma característica comum à maioria dos servidores de rede é que eles geralmente operam como múltiplos processos.
  - Pelo menos um processo escuta em uma porta de rede e, ao receber uma nova conexão de entrada, o processo de escuta usa **fork()** para criar um processo filho, que é então responsável pela nova conexão.
  - O filho, geralmente chamado de processo de trabalho *(worker process)*, termina quando a conexão é fechada.
  - Enquanto isso, o processo de escuta original continua a escutar na porta de rede.
  - Esse processo permite que um servidor manipule facilmente muitas conexões sem muitos problemas.

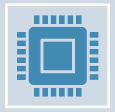
Servidores de Rede

Note:

- Chamar **fork()** adiciona uma quantidade significativa de sobrecarga do sistema.
- Para evitar isso, servidores TCP de alto desempenho, como o servidor Web Apache, podem criar vários processos de trabalho na inicialização para que estejam disponíveis para manipular conexões conforme necessário.
- Servidores que aceitam pacotes UDP não precisam do fork, pois não têm conexões para escutar; eles simplesmente recebem dados e reagem a eles.

Secure Shell

Conforme já mencionado, o SSH foi projetado para permitir logins de shell seguro, execução remota de programas, compartilhamento simples de arquivos etc. — substituindo os antigos telnet e rlogin.



A maioria dos ISPs e provedores de nuvem exigem SSH para acesso de shell aos seus serviços, e muitos dispositivos de rede baseados em Linux (como dispositivos de armazenamento conectado à rede ou NAS) também fornecem acesso via SSH.

O OpenSSH (http://www.openssh.com/) é uma implementação SSH gratuita popular para Unix, e quase todas as distribuições Linux vêm com ele pré-instalado.

O programa cliente OpenSSH é ssh, e o servidor é sshd.

Secure Shell

- Entre seus muitos recursos e capacidades úteis, o SSH faz o seguinte:
  - Criptografa senhas e todos os outros dados da sessão.
  - Tunela outras conexões de rede.
  - Oferece clientes para quase qualquer sistema operacional.
  - Usa chaves para autenticação de host.

- Uma desvantagem do SSH é que, para configurar uma conexão SSH, é preciso de uma chave pública do host remoto, e ela não é necessariamente obtida de forma segura (embora você possa verificá-la).

O Servidor sshd

- Executar o servidor **sshd** para permitir conexões remotas ao seu sistema requer um arquivo de configuração e chaves de host.
- A maioria das distribuições mantém as configurações no diretório de configuração **/etc/ssh** e tenta configurar tudo corretamente para você se você instalar o pacote **sshd**.
  - O nome do arquivo de configuração do servidor **sshd_config** é fácil de confundir com o arquivo de configuração **ssh_config** do cliente, então, tome cuidado.

O Servidor sshd

- Você não deve precisar alterar nada em sshd_config, mas nunca é demais verificar.
- O arquivo consiste em pares de chave-valor, como mostrado neste fragmento.

```
Port 22
#AddressFamily any

#ListenAddress 0.0.0.0
#ListenAddress ::
#HostKey /etc/ssh/ssh_host_rsa_key
#HostKey /etc/ssh/ssh_host_ecdsa_key
#HostKey /etc/ssh/ssh_host_ed25519_key
```

As linhas que começam com # são *comentários*, e muitos comentários no **sshd_config** indicam valores padrões para vários parâmetros, como se pode ver neste trecho.

O Servidor sshd

- A página do manual sshd_config(5) contém descrições dos parâmetros e valores possíveis, mas estes estão entre os mais importantes:
  - **HostKey**: Usa um arquivo como uma chave de host.
  - **PermitRootLogin**: Permite que o superusuário faça login com SSH se o valor estiver definido como **yes**. Defina o valor como não para evitar isso.
  - **LogLevel**: Registra mensagens com *syslog level* (default: INFO).
  - **SyslogFacility**: Registra mensagens com o nome de instalação do *syslog* (default: AUTH).
  - **X11Forwarding**: Habilita o tunelamento do cliente do **X Window System** se o valor estiver definido como **yes**.
  - **XAuthLocation**: Especifica o local do utilitário **xauth** no seu sistema.
    - O tunelamento X não funcionará sem esse caminho.
    - Se **xauth** não estiver em **/usr/bin**, defina o **path** como o nome completo do caminho para **xauth**.

Criação de Host keys

- O OpenSSH tem vários conjuntos de chaves de host. Cada conjunto tem uma chave pública (com uma extensão de arquivo .pub) e uma chave privada (sem extensão).
- A versão 2 do SSH tem chaves RSA e DSA. RSA e DSA são algoritmos de criptografia de chave pública.
- Os nomes de arquivo de chave são fornecidos na Tabela a seguir.

Filename	Key type
ssh_host_rsa_key	Private RSA key
ssh_host_rsa_key.pub	Public RSA key
ssh_host_dsa_key	Private DSA key
ssh_host_dsa_key.pub	Public DSA key

Criação de Host keys

- Criar chaves envolve uma computação numérica que gera chaves públicas e privadas.
- Normalmente, não é preciso criar as chaves porque o programa de instalação do OpenSSH ou o script de instalação da sua distribuição fazem isso.
- Mas é preciso saber como fazer isso, caso planeje usar programas como o ssh-agent que fornece serviços de autenticação sem uma senha.

Criação de Host keys

- Para criar chaves do protocolo SSH versão 2, use o programa ssh-keygen que vem com o OpenSSH:

```
# ssh-keygen -t rsa -N '' -f /etc/ssh/ssh_host_rsa_key
# ssh-keygen -t dsa -N '' -f /etc/ssh/ssh_host_dsa_key
```

- O servidor SSH e os clientes também usam um arquivo de chave, chamado **ssh_known_hosts**, para armazenar chaves públicas de outros hosts.
- Se pretende-se usar autenticação com base na identidade de um cliente remoto, o arquivo **ssh_known_hosts** do servidor deve conter as chaves públicas de host de todos os clientes confiáveis.

Criação de Host keys

Observação:

- Saber sobre os arquivos de chave é útil, caso se esteja substituindo uma máquina.
- Ao instalar uma nova máquina do zero, pode-se importar os arquivos de chave da máquina antiga para garantir que os usuários não tenham incompatibilidades de chave ao se conectar à nova.

Iniciando o Servidor SSH

- Embora a maioria das distribuições seja fornecida com SSH, elas geralmente não iniciam o servidor sshd por padrão.
- No Ubuntu e Debian, o servidor SSH não é instalado em um novo sistema; instalar seu pacote cria as chaves, inicia o servidor e *adiciona a inicialização do servidor à configuração de inicialização*.
- No Fedora, o sshd é instalado por padrão, mas desativado.
  - Para iniciar o sshd na inicialização, use systemctl assim:

    ```
    # systemctl enable sshd
    ```
  - Se você quiser iniciar o servidor imediatamente sem reinicializar, use:

    ```
    # systemctl start sshd
    ```

fail2ban

- Se você configurar um servidor SSH na sua máquina e abri-lo para a internet, você descobrirá rapidamente tentativas constantes de intrusão.
- Esses ataques de força bruta não terão sucesso se o seu sistema estiver configurado corretamente e você não tiver escolhido senhas estúpidas.
- No entanto, eles serão irritantes, consumirão tempo de CPU e bagunçarão, desnecessariamente, seus logs.
- Para evitar isso, você deve configurar um mecanismo para bloquear tentativas repetidas de login.
- O pacote **fail2ban** é, simplesmente, um *script* que observa mensagens de log.
  - Ao detectar um certo número de solicitações com falha de um *host* dentro de um certo período de tempo, o **fail2ban** usa o **iptables** para criar uma regra para negar tráfego daquele host.
  - Após um período especificado, durante o qual o *host* provavelmente desistiu de tentar se conectar, o **fail2ban** remove a regra.

O Cliente SSH

- Para efetuar login em um host remoto, execute:

```
$ ssh remote_username@remote_host
```

- Pode-se omitir o **remote_username@** se o nome de usuário local for o mesmo de remote_host.
- Pode-se também executar pipelines de e para um comando **ssh**, como mostrado no exemplo a seguir, que copia um diretório **dir** para outro host:

```
$ tar zcvf - dir | ssh remote_host tar zxvf –
```

- O arquivo de configuração global do cliente SSH **ssh_config** deve estar em **/etc/ssh**, o mesmo local do arquivo **sshd_config**.
- Assim como o arquivo de configuração do servidor, o arquivo de configuração do cliente tem pares de chave-valor. (Normalmente, não é preciso alterá-los.)

O Cliente SSH

- O problema mais frequente com o uso de clientes SSH ocorre quando uma chave pública SSH no seu arquivo local ssh_known_hosts ou .ssh/known_hosts não corresponde à chave no host remoto.
- Chaves ruins causam erros ou avisos como este:

```
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@    WARNING: REMOTE HOST IDENTIFICATION HAS CHANGED!     @
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
IT IS POSSIBLE THAT SOMEONE IS DOING SOMETHING NASTY!
Someone could be eavesdropping on you right now (man-in-the-middle attack)!
It is also possible that the RSA host key has just been changed.
The fingerprint for the RSA key sent by the remote host is
38:c2:f6:0d:0d:49:d4:05:55:68:54:2a:2f:83:06:11.
Please contact your system administrator.
Add correct host key in /home/user/.ssh/known_hosts to get rid of this message.
Offending key in /home/user/.ssh/known_hosts:12
RSA host key for host has changed and you have requested
strict checking.
Host key verification failed.
```

O Cliente SSH

- Isso geralmente significa apenas que o administrador do host remoto alterou as chaves (o que geralmente acontece em uma atualização de hardware ou servidor em nuvem), mas nunca é demais verificar com o administrador se você não tiver certeza.
- Em qualquer caso, a mensagem informa que a chave incorreta está na linha 12 do arquivo **known_hosts** de um usuário.
- Se você não suspeitar de algo, apenas remova a linha problemática ou substitua-a pela chave pública correta.

Clientes SSH (*file transfer*)

- O **OpenSSH** inclui os programas de transferência de arquivos **scp** e **sftp**, que substituem os programas mais antigos (e inseguros) **rcp** e **ftp**.
- Pode-se usar o **scp** para transferir arquivos para ou de uma máquina remota para sua máquina ou de um host para outro.
- Ele funciona como o comando **cp**. Exemplos:
  - Copie um arquivo de um host remoto para o diretório atual:

    ```
    $ scp user@host:file .
    ```
  - Copie um arquivo da máquina local para um host remoto:

    ```
    $ scp file user@host:dir
    ```
  - Copie um arquivo de um host remoto para um segundo host remoto:

    ```
    $ scp user1@host1:file user2@host2:dir
    ```

Clientes SSH (*file transfer*)

- O programa **sftp** funciona como o obsoleto cliente **ftp** de linha de comando, usando comandos **get** e **put**.
- O host remoto deve ter um programa **sftp-server** instalado, o que se pode esperar, caso o host remoto também usar **OpenSSH**.

Dúvidas?