Acesse remotamente máquinas com portas SSH bloqueadas

Passagem remota

O acesso público a partir da Internet geralmente é protegido por firewalls restritivos, e não se tem qualquer chance de usar o SSH. Porém HTTPS pela porta 443 normalmente é permitido. O Ajaxterm permite que usuários móveis entrem em seus servidores remotos.

por Udo Wolter

Seja em uma viagem de negócios ou simplesmente a turismo, diversos usuários costumam entrar em cybercafés para verificar seus emails e os logs de seus servidores web, ou simplesmente para atualizar algum software remotamente. Uma ferramenta de webmail consegue resolver a primeira dessas tarefas, mas usuários de Linux freqüentemente preferem ferramentas mais leves e baseadas em texto, como o *Mutt*.

Antigamente, era possível instalar os softwares que desejassem nos cybercafés (por exemplo, usando o *Putty*[1] como cliente SSH para Windows®). No entanto, devido ao crescente problema dos vírus, hoje é altamente improvável encontrar PCs abertos nesses estabelecimentos.

Applets Java que usam SSH para se conectar a seu servidor corporativo ou doméstico (como o *Mindterm*[2], por exemplo), poderiam ser uma alternativa, mas geralmente há firewalls bloqueando a porta 22, usada pelo SSH.

Nem mesmo o repasse da porta SSH para a porta HTTPS (443) funciona mais em vários casos, pois os analisadores de protocolo impedem a conexão. Se o cliente na porta HTTPS falar algum protocolo diferente de HTTPS, o analisador simplesmente bloqueará a conexão.

Fuga do firewall

Durante anos faltou ao Linux uma ferramenta que suportasse serviços de terminal em uma sessão HTTPS, que é necessária para se fugir de um sistema trancado e entrar de forma segura em seu próprio servidor. Finalmente chegou a tecnologia *AJAX* [3] com uma nova solução baseada nela, o *Ajaxterm* [4].

Esse programa de terminal compatível com *VT100* é baseado no *Anyterm*[5], porém mais fácil de instalar e usar. Os comandos do **exemplo 1** permitem que se teste o Ajaxterm.

Script python

O Ajaxterm é um script em *Python*. Ele abre a porta 8022 na interface *localhost* e pode ser executado imediatamente, porém ainda não permitirá

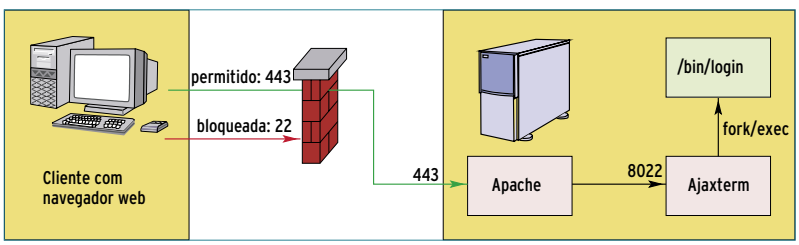

Figura 1 Mesmo que um *firewall* esteja bloqueando acessos através da porta 22 – e mesmo que ele bloqueie todas as portas à exceção da 80 (*HTTP*) e da 443 (*HTTPS*) – o *Ajaxterm* ainda permitirá o acesso remoto.

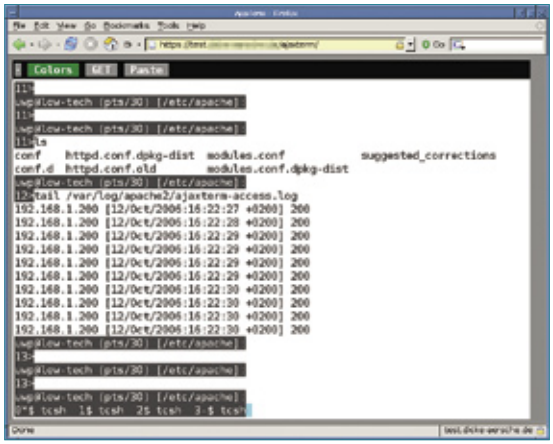

Figura 2 Essa sessão do terminal usa *SSL* através de um servidor *Apache*, que atua como um *proxy*, repassando as requisições para o *Ajaxterm*. O programa então chama `/bin/login`, e o usuário pode entrar no sistema como se estivesse em um console local.

Exemplo 1: Começando no Ajaxterm

```
01 kdir -p /var/www/test; cd /var/www/test
02 wget http://antony.lesuisse.org/qweb/files/Ajaxterm-0.9.tar.gz
03 tar zxvf Ajaxterm-0.9.tar.gz
04 mv Ajaxterm-0.9 ajaxterm
05 cd ajaxterm
06 ./ajaxterm.py
```

conexões através da porta 443 (que é nosso objetivo). Um *login* local via `http://localhost:8022` é possível. Para fazer isso, o Ajaxterm executa `/bin/login` no servidor, o que abre uma sessão de terminal na janela do navegador.

Três outros botões na janela oferecem outras funções: *Color* liga e desliga as cores; *GET* alterna entre os modos *get* e *post* (*post* é mais seguro, e portanto preferível); e *Paste* suporta ações tipo copiar-e-colar da área de transferência para a sessão atual do Ajaxterm. Isso só funcionará se o acesso a *Javascript* estiver ativado.

A direção contrária sempre é possível, pois a tela do terminal no navegador é baseada em caracteres. Clique para alternar entre os modos de operação; um botão verde mostra quais modos estão ativos.

Javascript contra copiar-e-colar

O recurso de copiar-e-colar pode ajudar a simplificar a vida do usuário, mas também pode representar uma falha de segurança em aplicativos Javascript. Quando se ativa essa opção, recebe-se um link para um tutorial[6] a respeito de como ativar o recurso nas opções de segurança do *Firefox*.

Em vez de chamar o `/bin/login`, pode-se passar a opção `-c` para o Ajaxterm iniciar um programa diferente, permitindo assim o repasse por SSH para outra máquina, por exemplo. A porta (o padrão é 8022) é configurável, e o Ajaxterm registrará a atividade, caso deseje (para a saída de erro padrão, caso a opção `-l` seja usada).

O Ajaxterm suporta apenas conexões através da interface *localhost*, então é necessário ter um servidor web para conseguir acesso remoto; os exemplos deste artigo descrevem como configurar o *Apache* 2 ou mais recente.

Ajax e Apache

Para acessar o Ajaxterm pela porta HTTPS, a documentação do programa recomenda o redirecionamento externo usando as funções de proxy do Apache (**exemplo 2**, **linhas 24 e 25**). A **figura 1** mostra como os três componentes cooperam: o navegador web no lado do cliente, e o Apache e o Ajaxterm no servidor.

Se você usa o Apache, a maior velocidade proporcionada pelo modo *GET* pode ser uma faca de dois gumes: o servidor web registra todas as URLs e, conseqüentemente, também os pressionamentos de teclas. Para evitar esse risco, deve-se ao menos alternar para o modo *POST* durante a entrada no sistema. Se você configurar uma conexão com uma terceira máquina após entrar no sistema por SSH, você não estará a salvo das entradas de log do Apache, pois o servidor Apache continuará recebendo entrada do teclado em texto puro.

Certificado SSL

Em servidores Apache que não possuem um certificado SSL (no arquivo de configuração `apache.pem`) é necessário executar o `apache2-ssl-certificate` para criar um. Ao rodar o script com o parâmetro `-new`, a ferramenta pede vários detalhes, incluindo seu país, código, Estado, cidade, nome da empresa e assim por diante. Apesar de ser permitido aceitar os valores padrão, é interessante preencher essa seção com dados realmente úteis.

O nome correto do servidor é essencial: se você se enganar, o navegador vai reclamar do circuito ao tentar carregar algo, e poderia até recusar-se a cooperar com o servidor. Certificados assinados pelo próprio servidor possuem algumas desvantagens em relação àqueles emitidos por uma autoridade certificadora (AC). Para o acesso ao terminal em sua própria máquina, pedir que o navegador verifique as impressões digitais deveria ser suficiente. Se isso for pouco prático, talvez seja necessário comprar um certificado.

Também será preciso convencer o Apache a usar o módulo de proxy. Os comandos do **exemplo 3** ativam os módulos para SSL e proxy no *Debian Sarge*. Após completar os passos necessários para que o servidor responda requisições na porta SSL, o acesso ao Ajaxterm através de uma URL como `https://teste.exemplo.com/ajaxterm/` já deve funcionar. A **figura 2** mostra um exemplo de sessão que utiliza o *screen*.

O log faz mal à sessão

O log do Apache tem a tendência de crescer rapidamente, e é esse o motivo para que ele esteja restrito aos itens críticos no exemplo de configuração (**linha 20** do **exemplo 2**); o servidor registrará somente o IP de origem, hora e estado da requisição. Talvez seja interessante

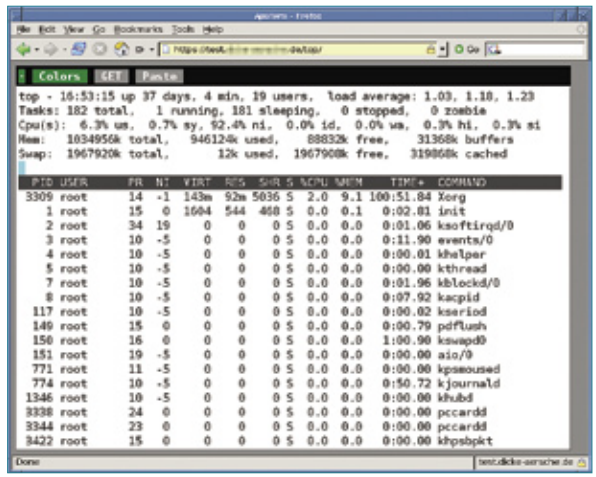

Figura 3 Login desnecessário: entrar na URL correta, após criar entradas adicionais na configuração do *Apache* e iniciar outros processos do *Ajaxterm*, mostrará a saída do *top* na janela do navegador.

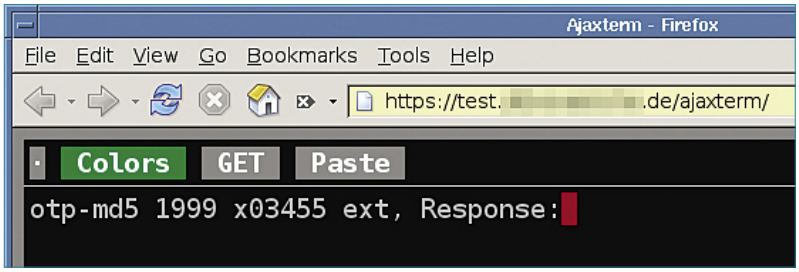

Figura 4: Senhas usadas uma única vez através do *OPIE* protegem o acesso ao terminal contra ataques de *keyloggers*; a figura mostra uma rotina de login esperando o usuário digitar a senha para o número 1999.

enviar uma requisição *keep-alive* junto com os eventos do teclado a cada dois segundos, para evitar que a sessão seja terminada; nesse caso, restrinja ao máximo o nível do registro: `LogLevel emerg` em vez de `LogLevel warn` faz exatamente isso. No exemplo anterior, o nível do log (*Loglevel*) ainda está configurado para `warn`, que pode ser útil para solucionar problemas enquanto se está configurando o serviço.

A atividade do log afeta tanto a velocidade da sessão do terminal a ponto de gerar confusão entre os modos *GET* e *POST*. Embora o `ls` e o `cat` gerem saídas com rapidez suficiente, há uma diferença em comparação com uma conexão SSH. Trabalhar com o *vi_* torna-se lento, mas ainda aceitável. Mesmo que se restrinja o Apache a eventos de pane com `Loglevel emerg`, existe uma diferença bem clara entre os modos *GET* e *POST*; o segundo funciona mais rápido.

Acelerando

A medição da velocidade de transmissão entre o navegador e o servidor web revelou os seguintes resultados: quando a conexão se torna ociosa (ou seja, quando o usuário não está pressionando qualquer tecla), a taxa de transferência entre o cliente e o servidor fica entre 1 e 3 kbps, de acordo com o *ifstat*.

Quanto à saída em grande escala (do `cat` ou `ls`), a taxa de saída para dados transferidos no modo *GET* é de aproximadamente 1:5, isso é, o volume de dados que precisa atravessar a conexão é o quíntuplo daquele gerado.

No modo *POST*, esse valor cai para 1:1,5 ou 1:2, ou seja, há muito menos dados transitando entre o servidor e o navegador, chegando, no máximo, ao dobro do volume de dados mostrados no terminal. Adicionar um proxy no Apache parece afetar o Ajaxterm, geralmente. O aplicativo responde mais rápido a uma porta localhost sem o ambiente Apache/SSL.

Exemplo 2: Configuração do Apache

```
01 Listen 443
02 NameVirtualHost *:443
03 <VirtualHost *:443>
04 ServerName test.Domain.de
05 SSLEngine On
06 SSLCertificateKeyFile ssl/apache.pem
07 SSLCertificateFile ssl/apache.pem
08 # Diretório principal deste host virtual
09 DocumentRoot /var/www/teste
10 # Desativa o comportamento normal de proxy do
11 # módulo de proxy para evitar que agressores
12 # façam mau uso do servidor web como um proxy aberto!
13 ProxyRequests Off
14 # Loglevel normalmente “warn”; ou seja, guarda
15 # dados demais. Para guardar menos, ou nada,
16 # use “emerg” no lugar de “warn”.
17 LogLevel warn
18 # Mesmo que você queira registrar eventos, não se deve registrar demais;
19 # apenas o IP de origem, a hora e o estado são registrados aqui
20 CustomLog /var/log/apache2/ajaxterm-access.log “%a %t %s”
21 ErrorLog /var/log/apache2/ajaxterm-error.log
22 # Repassando agora para os
23 # aplicativos que rodam internamente
24 ProxyPass /ajaxterm/ http://localhost:8022/
25 ProxyPassReverse /ajaxterm/ http://localhost:8022/
26 </VirtualHost>
```

Mais que apenas um terminal

A capacidade de utilizar múltiplas entradas de proxy para fazer o Ajaxterm rodar diferentes comandos abre uma série de opções. Para isso, basta iniciar o Ajaxterm diversas vezes (em portas diferentes) e acrescentar os parâmetros necessários.

Para usar a URL `https://servidor.com.br/top/` para mostrar a saída do *top*, serão necessárias duas linhas a mais na configuração do Apache:

```
ProxyPass /top/ http://localhost:8023/
ProxyPassReverse /top/ http://
➥localhost:8023/
```

Adicione uma linha de comando do Ajaxterm de acordo (`./ajaxterm.py -ctop -p8023`). A **figura 3** mostra a saída do top em um navegador. O parâmetro `-c` especifica o nome do programa a ser executado no terminal; `-p` determina a porta. A opção `-d` coloca o Ajaxterm em segundo plano.

Acesso restrito

Um link direto para um programa (como o top, em nosso exemplo) pode ser arriscado; será preciso, no mínimo, pedir o nome e a senha do usuário, para evitar o risco de hackers usarem seqüências de escape da *shell* para acessar seu servidor (veja o **quadro 1**). Como sempre, adicionar mais obstáculos faz bem.

Além disso, deve-se rodar o Ajaxterm em alguma conta sem muitos privilégios criada pelo administrador (`groupadd ajaxterm; useradd -g ajaxterm ajaxterm`). O script *Ajaxterm* suporta o parâmetro `-u`, que espera uma ID de usuário como argumento. Iniciar o programa com o comando `su ajaxterm caminho/do/ajaxterm` é ainda melhor.

Seguro com uma única senha

Apesar de o SSL criptografar a sessão para aumentar a segurança, ele não oferece qualquer proteção con-

tra *keyloggers* no nível do sistema operacional. O perigo de cavalos de tróia e outras pragas atinge seu máximo em cybercafés, embora seja problema geral e não específico do Ajaxterm.

O venerável *Java MindTerm* é tão vulnerável quando o Ajaxterm, quando se trata de keyloggers. As senhas usadas uma única vez (*One-Time Passwords*, ou *OTPs*), implementadas através do *OPIE*, podem ser de grande ajuda nesse caso – o OPIE cria listas de OTPs e é compatível com o *PAM*; deve-se simplesmente acrescentar a seguinte linha no início de `/etc/pam.d/ssh`:

```
auth sufficient pam_opie.so
```

Teoricamente, isso também deveria funcionar com `/bin/login`, pois `/etc/pam.d/login` possui uma estrutura semelhante. No entanto, o SSH mostra como os parâmetros são passados. A forma mais fácil de se fazer isso é chamando o Ajaxterm assim:

```
./ajaxterm.py -c'ssh usu&#225;rio@localhost'
```

Não se deve esquecer das aspas simples, pois elas são necessárias para se escapar o espaço.

O OPIE então pedirá uma senha com uma ID específica (1999, nesse caso; veja a **figura 4**). Pode levar algum tempo até que se localize as senhas em uma lista, mas isso certamente é mais seguro que usar a mesma senha repetitivamente em máquinas pouco confiáveis nos Internet cafés. Para ter êxito, um keylogger precisaria "grampear" a senha e fazer login imediatamente.

Incidentalmente, essa abordagem para o acesso por SSH também possui a vantagem de não precisar de uma porta SSH aberta na interface externa do servidor. Um agressor potencial precisaria utilizar a mesma abordagem (através da interface web), o que ao menos evitaria ataques simples com scripts. Além disso, proteger a sessão de login com um simples pedido de senha do Apache (com nome de usuário e senha diferentes) poderia dificultar ainda mais a vida do agressor.

Processos de longo prazo

Os programas que não terminam automaticamente após certo tempo (como o cliente top, por exemplo) continuarão sua execução no servidor mesmo após o cliente fechar a janela do navegador. O alvo detecta o fim da conexão quando o navegador é fechado.

Como um maior número de processos inúteis em execução poderia afetar o desempenho – e também causar uma situação de bagunça – os administradores devem executar uma tarefa no cron regularmente para buscar processos órfãos.

Útil, porém um pouco arriscado

Graças à emulação do VT100, programas como o Screen e o Mutt funcionam, embora não perfeitamente. A versão atual (0.9) possui o problema do Mutt não atualizar a tela quando se pressiona [Ctrl]+[L]. Além disso, o Ajaxterm não roda em navegadores antigos: não tivemos problemas para rodá-lo no *Firefox* ou no *Internet Explorer*; o *Konqueror* e o *Opera* mostram o terminal, mas ocorrem erros. Esse não parece ser um problema específico do Ajaxterm, pois vários programas em AJAX possuem defeitos na compatibilidade com navegadores.

É importante ter em mente que erros de programação no script Ajaxterm podem levar à conquista do acesso à shell por um agressor, na pior das hipóteses. Afinal, o Ajaxterm é um aplicativo em Python, e escapar de um script para a shell costuma ser mais fácil que escapar de um binário. Portanto, considere os riscos potenciais antes de executar o Ajaxterm em seu servidor. ■

Exemplo 3: Ativação do Proxy e do SSL

```
01 cd /etc/apache2/mods-enabled/
02 for i in proxy.load proxy.conf ssl.load ssl.conf; do
03 ln -s ../mods-available/$i .;
04 done
05 /etc/init.d/apache2 restart
```

Quadro 1: Protegendo o Apache por senha

Para proteger com senha a execução de programas como o *top*, pode ser interessante usar o mecanismo simples de autenticação do *Apache*. Se você não protegeu o servidor inteiro, é possível configurar um subdiretório separado para o Ajaxterm.

É necessário acrescentar o seguinte à configuração do servidor (no bloco `<VirtualHost>`):

```
<Location "/top">
   AuthName       "Ajaxterm"
   AuthType       Basic
   AuthUserFile   /etc/apache2/security/.htpasswd
   Require user   Nome_do_usuário
</Location>
```

De forma semelhante, será necessário um bloco `<Location>` para cada conexão ao Ajaxterm. Para criar o arquivo `.htpasswd`, use o comando `htpasswd` da seguinte forma:

```
htpasswd /etc/apache2/security/.htpasswd Nome_do_usu&#225;rio
```

Se o arquivo já existir, o `htpasswd` acrescentará uma entrada para o nome de usuário passado para ele.

Mais Informações

[1] Putty: http://www.chiark.greenend.org.uk/~sgtatham/putty

[2] MindTerm: http://www.appgate.com/products/80_MindTerm

[3] Página do AJAX na Wikipédia: http://pt.wikipedia.org/wiki/Ajax_(programação)

[4] Ajaxterm: http://antony.lesuisse.org/qweb/trac/wiki/AjaxTerm

[5] Anyterm, antecessor do Ajaxterm: http://anyterm.org

[6] Tutorial para habilitar cópia e colagem com Javascript no Firefox: http://kb.mozillazine.org/Granting_JavaScript_access_to_the_clipboard