从入门到精通：构建SSH安全通道

2011-03-06 09:11出处：计世网作者：佚名【我要评论】 [导读]

SSH命令行实用程序是远程系统的工作人员安全访问的通道。SSH代表着“安全外壳”，所以你可能认为其最常用的使用方法就是用作一个远程外壳。也许这是其最常见的应用，但并非使用SSH的唯一的有趣方法。

　　一、连接、执行远程命令并使用授权密钥

　　1、创建一个连接

　　为通过SSH进行通信，你首先需要建立一个到达远程服务器的连接。使用SSH命令行实用程序时，我们有很多参数选择。最基本的命令行参数是：

　　1、SSH IP地址

　　在这里，IP地址就是你想要连接的服务器的IP地址。下面是一个简单的例子：

　　abc:~ jmjones$ ssh 192.168.1.20

　　The authenticity of host '192.168.1.20 (192.168.1.20)' can't be established.

　　RSA key fingerprint is 24:1e:2e:7c:3d:a5:cd:a3:3d:71:1f:6d:08:3b:8c:93.

　　Are you sure you want to continue connecting (yes/no)? yes

　　Warning: Permanently added '192.168.1.20' (RSA) to the list of known hosts.

　　注意，上面的消息中有这样一行“The authenticity of host '192.168.1.20 (192.168.1.20)' can't be established.”。这条消息意味着笔者的SSH 客户端并不知道远程服务器。笔者在这里用了“客户端”，因为SSH命令行实用程序初始化了网络并使其成为网络客户端。

　　在上面的消息中，SSH程序还询问作者是否希望继续连接(Are you sure you want to continue connecting (yes/no)? )笔者的回答是“yes”，因为笔者知道这台服务器正是笔者想连接的服务器。一般而言，对此问题回答“yes”是比较安全的。(但如果不怀好意的家伙假冒了你想连接的服务器，这样做就危险了。)在回答“yes”继续连接后，笔者的客户端用下面的内容更新了文件$HOME/.ssh/known\_hosts：

　　192.168.1.20 ssh-rsa

　　^4rsa5jmjones6cd7jmjones8^/^9cd10^+9^11yc12yc13rsa14AAAAB15^+^16r

　　sa17AAAAB18^99u2^19oT20oT21^7N7^22AAAAB23^+^24cd25^5f+^26ykuwQcXI

　　27EAAAABIwAAAQEAvb28jmjones29oT30commandline31^2Ax3J88^32commandl

　　ine33yc34^+rOB+gOdRaD+NTkuzrB/^38oT39^50L6^40oT41AAAAB42^61rq+9v+4^

　　44AAAAB45rsa46ykuwQcXI47^5q1P11^48AAAAB49gcgPr50^==

　　在笔者再次连接到同样的服务器时，笔者的SSH客户端将检查“known\_hosts”文件，检查是否与上次的服务器相同。如果服务器传回的信息与“known\_hosts”文件中的信息不匹配，就会看到类似于下面的消息：

　　abc:~ jmjones$ ssh 192.168.1.20

　　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

　　24:1e:2e:7c:3d:a5:cd:a3:3d:71:1f:6d:08:3b:8c:93.

　　Please contact your system administrator.

　　Add correct host key in /Users/jmjones/.ssh/known\_hosts to get rid of this message.

　　Offending key in /Users/jmjones/.ssh/known\_hosts:1

　　RSA host key for 192.168.1.20 has changed and you have requested strict checking.

　　Host key verification failed.

　　2、口令验证

　　继续上面的例子。在回答了“yes”后，程序要求笔者输入口令。下面是此次交互的部分信息：

　　jmjones@192.168.1.20's password:

　　Be careful.

　　No mail.

　　Last login: Tue Dec 30 06:36:20 2008 from abc

　　jmjones@oksir:~$

　　笔者键入了口令，然后ssh客户端与远程服务器建立了一个交互连接。在此，可以看出登录到一个Linux服务器的证据，如上次登录的时间。

　　3、授权密钥

　　如果我们不想在每次登录时都键入口令该怎么办？或者，笔者是一个系统管理员，希望让服务器更强健，让不法之徒难以猜测，又该如何呢？我们可以使用一个公钥/私钥对来让登录到服务器的过程更安全更简单。

　　为使用公钥/ 私钥对,我们必须创建它。可通过在命令行使用ssh –keygen程序来达此目的。其实，该命令还是有许多参数的，如密钥的类型、想要创建的文件名、密钥文件的注释等，不过我们完全只使用默认选项。下面是笔者在没有使用任何参数时的结果：

　　abc:~ jmjones$ ssh-keygen

　　Generating public/private rsa key pair.

　　Enter file in which to save the key (/Users/jmjones/.ssh/id\_rsa):

　　Created directory '/Users/jmjones/.ssh'.

　　Enter passphrase (empty for no passphrase):

　　Enter same passphrase again:

　　Your identification has been saved in /Users/jmjones/.ssh/id\_rsa.

　　Your public key has been saved in /Users/jmjones/.ssh/id\_rsa.pub.

　　The key fingerprint is:

　　fe:e9:fa:f5:e2:4e:a1:6c:9e:9e:20:a4:cc:ec:4f:62 jmjones@abc

　　The key's randomart image is:

　　+--[ RSA 2048]----+

　　| |

　　| |

　　| |

　　| |

　　| . S . |

　　| + o . . . . |

　　| E o o + o |

　　| o o . = \*.. |

　　| ... .=Xoo.. |

　　+-----------------+

　　笔者接受了“id\_rsa”作为密钥文件，还接受了不输入任何口令短语的默认选项。如果选择了给文件添加一个口令短语，那么在每次使用时都得输入此短语。运行ssh-keygen的结果是在$HOME/.ssh文件中生成了两个文件：

　　abc:~ jmjones$ ls -l ~/.ssh/

　　total 16

　　-rw------- 1 jmjones staff 1675 Dec 30 17:37 id\_rsa

　　-rw-r--r-- 1 jmjones staff 400 Dec 30 17:37 id\_rsa.pub

　　“id\_rsa”是笔者的私钥。笔者并不希望任何人可以访问这个文件，以防止其他人冒充自己。注意，对“id\_rsa”的限制强于对“id\_rsa.pub”的限制。笔者可以将此文件传送给希望连接的任何人。不用担心，任何人都无法猜测我们的私钥是什么。

如果笔者希望将此密钥用于前面例子中的服务器，就可将公钥的内容放到远程服务器的“$HOME/.ssh/authorized\_keys”文件中。为了设置正确，我们一般需要以SSH 方式连接到远程服务器并将本地的“id\_rsa.pub”文件复制到远程“authorized\_keys”文件中，如下所示：

　　jmjones@oksir:~$ echo "ssh-rsa

　　AAAAB3NzaC1yc2EAAAABIwAAAQEAw4DTUeLXZbjjNhR+AaW9^102rsa103^+Pg2+Q

　　8M+gK/IGDbPjsAV4KwulqDWS+ChlIiq0wXj/bQKQwZacbghXud/YBI7FfYOkF1R9p

　　FZ7O9B7zJGAnAtcOEDLfyDhYF2Cl5/1HFolIUuUSCGPJy3bbIK5s6yNwQV6cW6yEF

　　UuqE8DHlGKf9jwDFgiXrhtuThH2EFGBCxELaumworegMD39Jb9^123rsa124^1zWF

　　qP2qHX/SzItHm1JrKJdnbsOn5h+KMTeztpn1AExOx1lxSFLk9lp4JAMk8NTURYmBc

　　AE6yASaQApw5jDw/JpSAdFaQR/Vl6Kpzf9MD1KAEpyd8RaxLa+RQ== jmjones@abc" > ~/.ssh/authorized\_keys

　　jmjones@oksir:~$ ls -l ~/.ssh/

　　total 4

　　-rw-r--r-- 1 jmjones jmjones 400 2008-12-30 17:48 authorized\_keys

　　jmjones@oksir:~$

　　此后，在登录时，程序就不再提示输入口令。这里，登出服务器，然后再次以SSH方式进入：

　　jmjones@oksir:~$ logout

　　Connection to 192.168.1.20 closed.

　　abc:~ jmjones$ ssh 192.168.1.20

　　Be careful.

　　No mail.

　　Last login: Tue Dec 30 17:50:26 2008 from abc

　　请注意，笔者的客户端并没有提示要求输入口令。现在，只要想连接到此服务器，只要想以SSH方式登录，便会立即连接。

　　4、执行远程命令

　　前面提到，在SSH连接到远程服务器后，默认地我们就处于外壳提示符下，但这并不是我们可做的唯一事情。使用SSH客户端的另外一个有用方法是在远程服务器上执行命令，而不用键入到远程服务器上的交互外壳中。也就是说，在本地系统上执行SSH程序时，你可以指定在远程系统上运行什么命令。例如，如果希望看到某个进程是否正在远程系统的25号端口上监听，可像如下一样操作：

　　abc:~ jmjones$ ssh 192.168.1.20 netstat -ltpn | grep 25

　　(Not all processes could be identified, non-owned process info

　　will not be shown, you would have to be root to see it all.)

　　tcp 0 0 127.0.0.1:25 0.0.0.0:\* LISTEN -

　　在第一行中，其语法是“ssh 地址 命令”，我们可以用此法检查磁盘的利用率，查看有哪些进程正在运行，或复制文件等。

　　为什么不直接登录然后以交互方式运行命令呢？因为这会丧失脚本的好处。从现在起，在远程系统上执行命令成为外壳脚本的一部分。

　　可以看出，SSH是一个重要的工具。就其一般的使用情况，它准许我们在远程服务器上以交互方式管理外壳。这对于远程系统管理来说当然是很有必要的。不过，通过使用授权密钥，我们还可以增强认证过程的安全性。它准许我们在远程系统上运行外壳脚本而无需处于交互式外壳中。

　　二、实用安全：构建SSH安全通道

　　下面我们讨论一些比远程外壳访问或远程执行更为有趣的内容：创建隧道。

　　1、构建隧道的理由

　　先谈谈什么是隧道。

　　SSH隧道也就是用加密的SSH协议将网络通信包装起来的过程。隧道包括一个连接到SSH服务器的SSH客户端。不过，在SSH客户端连接到服务器时，客户端指定了隧道的源和目的地。

　　所谓的源也就是其它的进程可以连接的绑定网络端口。这个端口要么由SSH客户端，要么由SSH服务器管理。

　　目的地是另外一个绑定的网络端口，不过，这里指的是另外的一个网络服务器，它是SSH隧道的另外一端可以与之通信的服务器。下面笔者将用一个例子更详细地阐述之。读者尽可以将SSH隧道看作是如同安全的端口转发一样的数据传输通道。

　　使用SSH隧道的一个重要理由是连接两个彼此并不能直接访问的两个网络。举一个例子，假如你拥有一台笔记本电脑，你想不管身在何处都能够连接到家里的IMAP服务器。你可以打开家里的IMAP服务器的访问，不过这并不是一个好主意。你可以在路由器上建立一个VPN隧道，不过，这样也许有点太过于奢侈。当然，在你想访问IMAP服务时，你还可以创建一个从笔记本电脑到家里网络的SSH隧道。后文将有一个例子。

　　使用SSH隧道的第二个理由是它可以加密网络通信。在IMAP的例子中，使用SSH的另外一个好处是电子邮件数据是加密的。在你与亲朋好友通信时，这些通信数据在通过互联网上的隧道传输时是很安全的。不过，在这种通信进入隧道之前和离开隧道后它并没有经过加密。

　　使用SSH时，有两种安全的端口转发：本地转发和远程转发。对于本地转发而言，SSH客户端管理源端口。对于远程转发而言，SSH服务器管理源端口。你选择的是本地或远程转发都依赖于哪个系统发起连接，哪个系统在其上拥有SSH服务器，你在哪里需要隧道的源地址。

　　下面给出本地转发的一个例子。继续以上面的IMAP连接为例，假设我有一台名称为“abc”的笔记本电脑，我想访问“oksir”机器上的一个IMAP服务器。为什么不直接连接到“oksir”呢？如果“oksir”位于防火墙之后而你无法连接到143端口，那么使用ssh隧道就是一个很好的选择。

　　abc:~ jmjones$ ssh -L 8143:localhost:143 oksir

　　注意，这里的“-L”参数指明了这是一种本地转发。而“8143”指明了笔者希望将127.0.0.1:8143绑定为隧道源。“localhost:143”指明了要将通信转发到何处。虽然在远程端上指明为“localhost”，但也可以指定远程系统可以与之通信的任何地址和端口。最后，“oksir”指明了笔者希望以SSH方式进入的机器。

　　在运行此命令之前，笔者运行了netstat命令，查看在笔记本电脑“abc”上是否有什么东西正在8143端口上监听：

　　abc:~ jmjones$ netstat -an | grep 8143

　　abc:~ jmjones$

　　根据运行的结果可知没有什么正在监听，在创建隧道之后，再次运行netstat命令，这时看到了不同的结果：

　　abc:~ jmjones$ netstat -an | grep 8143

　　tcp4 0 0 127.0.0.1.8143 \*.\* LISTEN

　　tcp6 0 0 ::1.8143

　　从上面的netstat命令的运行结果可以看出，SSH客户端上的某程序正在127.0.0.1(本地的循环网络设备)的8143端口上监听。任何与127.0.0.1:8143的连接都将被转发给“oksir”上的localhost：143端口上。在创建隧道后，需要在笔记本电脑的电子邮件客户端上配置一个账户，在locahos：8143上找一个IMAP服务器，并开始在“oksir”上读邮件。

　　现在，我们假定服务器“oksir”使用fetchmail，并从ISP的pop3服务器上读取邮件消息。但问题是笔者的ISP仅允许连接到其网络机器可以访问其pop3服务器。因为笔者的笔记本电脑“abc”直接连接到ISP和服务器：

　　2、可创建类似于下面的一个隧道：

　　abc:~ jmjones$ ssh -R 8110:mail.myisp.com:110 oksir

　　在上面的命令中，“-R”指明这是一个反向转发，“8110”指明8110端口上监听并绑定之。“mail.myisp.com:110”指定笔记本电脑将会转发从隧道接收的任何通信。“oksir”是以ssh方式连接的机器。

　　在运行SSH之前，在服务器上运行了netstat命令，显示出没有什么程序正在8110端口上监听：

　　jmjones@oksir:~$ netstat -an | grep 8110

　　jmjones@oksir:~$

　　笔者并没有看到什么程序在监听。在笔记本电脑上运行了ssh命令后，在服务器上运行了同样的命令：

　　jmjones@oksir:~$ netstat -an | grep 8110

　　tcp 0 0 127.0.0.1:8110 0.0.0.0:\* LISTEN

　　tcp6 0 0 ::1:8110 :::\* LISTEN

　　在创建隧道后，fetchmail可在localhost:8110上运行，并将请求转发给ISP。当然，在笔记本电脑连接到服务器和ISP的邮件服务器时，这个隧道才是活动的。

　　总之，通过SSH建立隧道是实现安全的数据通信的一种简单方法。同时，它还是连接两个并不直接相连的两个网络的一种简便方法。在找出使用它的方法后，它就成为一个不可替换的好工具。

原文出自【比特网】，转载请保留原文链接：http://sec.chinabyte.com/208/11844708.shtml