1 简介
2 作用
3 层次
4 用法
4.1 基本操作
4.2 口令登录
4.3 公钥登录
4.4 authorized_keys文件
4.5 绑定本地端口
4.6 本地端口转发
4.7 远程端口转发
4.8 SSH的其他参数

1 简介

SSH（安全外壳协议，Secure Shell 的缩写）由 IETF 的网络小组（Network Working Group）所制定，是建立在应用层基础上的安全协议。SSH 是目前较可靠，专为远程登录会话和其他网络服务提供安全性的协议，利用 SSH 协议可以有效防止远程管理过程中的信息泄露问题。SSH 最初是 UNIX 系统上的一个程序，后来又迅速扩展到其他操作平台。SSH 客户端适用于多种平台，几乎所有 UNIX 平台都可运行 SSH。

2 作用

传统的网络服务程序，如：FTP、POP 和 Telnet 在本质上都是不安全的，因因为它们在网络上用明文传送口令和数据，别有用心的人非常容易就可以截获这些口令和数据。而且，这些服务程序的安全验证方式也是有其弱点的， 就是很容易受到“中间人攻击”（Man-in-the-middle attack）。

所谓“中间人攻击”， 就是“中间人”冒充真正的服务器接收你传给服务器的数据，然后再冒充你把数据传给真正的服务器。服务器和你之间的数据传送被“中间人”一转手做了手脚之后，就会出现很严重的问题。通过使用 SSH，你可以把所有传输的数据进行加密，这样”中间人”这种攻击方式就不可能实现了，而且也能够防止 DNS 欺骗和 IP 欺骗。使用 SSH，还有一个额外的好处就是传输的数据是经过压缩的，所以可以加快传输的速度。

3 层次

SSH 主要由三部分组成：传输层协议 [SSH-TRANS]、用户认证协议 [SSH-USERAUTH] 和连接协议 [SSH-CONNECT] 组成。

- **传输层协议 [SSH-TRANS]**：提供了服务器认证，保密性及完整性。此外它有时还提供压缩功能。SSH-TRANS 通常运行在 TCP/IP 连接上，也可能用于其它可靠数据流上。 SSH-TRANS 提供了强力的加密技术、密码主机认证及完整性保护。该协议中的认证基于主机，并且该协议不执行用户认证。更高层的用户认证协议可以设计为在此协议之上。

- **用户认证协议 [SSH-USERAUTH]**：用于向服务器提供客户端用户鉴别功能，它运行在传输层协议 SSH-TRANS 上面。当 SSH-USERAUTH 开始后，它从低层协议那里接收会话标识符，会话标识符唯一标识此会话并且适用于标记以证明私钥的所有权。 SSH-USERAUTH 也需要知道低层协议是否提供保密性保护。
- **连接协议 [SSH-CONNECT]**：将多个加密隧道分成逻辑通道。它运行在用户认证协议上。它提供了交互式登录话路、远程命令执行、转发 TCP/IP 连接和转发 X11 连接。

4 用法

4.1 基本操作

SSH 主要用于远程登录。假定我们要以用户名 `user`，登录远程主机 `host`，只要一条简单命令就可以啦！

```
$ ssh user@host
```

如果本地用户名与远程用户名一致，登录时可以省略用户名。

```
$ ssh host
```

SSH 的默认端口是 `22`，也就是说，我们的登录请求会送进远程主机的 `22` 端口。使用 `p` 参数，可以修改这个端口。

```
$ ssh -p 2222 user@host
```

上面这条命令表示，SSH 直接连接远程主机的 `2222` 端口。

4.2 口令登录

如果我们是第一次登录对方主机，系统会出现下面的提示：

```
$ ssh user@host
The authenticity of host 'host (12.18.81.21)' can't be established.
RSA key fingerprint is 98:2e:d7:e0:de:9f:ac:67:28:c2:42:2d:37:16:58:4d.
Are you sure you want to continue connecting (yes/no)?
```

这段话的意思是，无法确认 host 主机的真实性，只知道它的公钥指纹，问你还想继续连接吗？

所谓"公钥指纹"，是指公钥长度较长（这里采用 RSA 算法，长达 1024 位），很难比对，所以对其进行 MD5 计算，将它变成一个 128 位的指纹。上例中是 `98:2e:d7:e0:de:9f:ac:67:28:c2:42:2d:37:16:58:4d`，再进行比较，就容易多啦！很自然想到的一个问题就是，用户怎么知道远程主机的公钥指纹应该是多少？回答是没有好办法，远程主机必须在自己的网站上贴出公钥指纹，以便用户自行核对。假定经过风险衡量以后，用户决定接受这个远程主机的公钥。

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

系统会出现一句提示，表示 host 主机已经得到认可。

```
Warning: Permanently added 'host,12.18.429.21' (RSA) to the list of known hosts.
```

然后，会要求输入密码。

```
Password: (enter password)
```

如果密码正确，就可以登录了。

当远程主机的公钥被接受以后，它就会被保存在文件 `$HOME/.ssh/known_hosts` 之中。下次再连接这台主机，系统就会认出它的公钥已经保存在本地了，从而跳过警告部分，直接提示输入密码。每个 SSH 用户都有自己的 `known_hosts` 文件，此外系统也有一个这样的文件，通常是 `/etc/ssh/ssh_known_hosts`，保存一些对所有用户都可信赖的远程主机的公钥。

4.3 公钥登录

使用密码登录，每次都必须输入密码，非常麻烦。好在 SSH 还提供了公钥登录，可以省去输入密码的步骤。

所谓"公钥登录"，原理很简单，就是用户将自己的公钥储存在远程主机上。登录的时候，远程主机会向用户发送一段随机字符串，用户用自己的私钥加密后，再发回来。远程主机用事先储存的公钥进行解密，如果成功，就证明用户是可信的，直接允许登录 shell，不再要求密码。这种方法要求用户必须提供自己的公钥。如果没有现成的，可以直接用 `ssh-keygen` 生成一个：

```
$ ssh-keygen
```

运行上面的命令以后，系统会出现一系列提示，可以一路回车。其中有一个问题是，要不要对私钥设置口令（passphrase），如果担心私钥的安全，这里可以设置一个。运行结束以后，在 `$HOME/.ssh/` 目录下，会新生成两个文件：`id_rsa.pub` 和 `id_rsa`。前者是你的公钥，后者是你的私钥。这时再输入下面的命令，将公钥传送到远程主机 host 上面：

```
$ ssh-copy-id user@host
```

好了，从此我们再登录，就不需要输入密码了。如果还是不行，就打开远程主机的 `/etc/ssh/sshd_config` 这个文件，检查下面几行前面 `#` 注释是否取掉。

```
RSAAuthentication yes
PubkeyAuthentication yes
AuthorizedKeysFile .ssh/authorized_keys
```

然后，重启远程主机的 SSH 服务。

```
/* ubuntu系统 */
service ssh restart
/* debian系统 */
/etc/init.d/ssh restart
```

4.4 authorized_keys文件

远程主机将用户的公钥，保存在登录后的用户主目录的 `$HOME/.ssh/authorized_keys` 文件中。公钥就是一段字符串，只要把它追加在 `authorized_keys` 文件的末尾就行了。这里不使用上面的 `ssh-copy-id` 命令，改用下面的命令，解释公钥的保存过程：

```
$ ssh user@host 'mkdir -p .ssh && cat >> .ssh/authorized_keys' < ~/.ssh/id_rsa.pub
```

这条命令由多个语句组成，依次分解开来看：

- `$ ssh user@host`，表示登录远程主机；
- 单引号中的 `mkdir .ssh && cat >> .ssh/authorized_keys`，表示登录后在远程 Shell 上执行的命令：
- `$ mkdir -p .ssh"的作用是，如果用户主目录中的` .ssh`目录不存在，就创建一个；
- `cat >> .ssh/authorized_keys < ~/.ssh/id_rsa.pub` 的作用是，将本地的公钥文件 `~/.ssh/id_rsa.pub`，重定向追加到远程文件 `authorized_keys` 的末尾。

写入 `authorized_keys` 文件后，公钥登录的设置就完成啦！

4.5 绑定本地端口

既然 SSH 可以传送数据，那么我们可以让那些不加密的网络连接，全部改走 SSH 连接，从而提高安全性。假定我们要让 `8080` 端口的数据，都通过 SSH 传向远程主机，命令就这样写：

```
$ ssh -D 8080 user@host
```

SSH 会建立一个 Socket，去监听本地的 `8080` 端口。一旦有数据传向那个端口，就自动把它转移到 SSH 连接上面，发往远程主机。可以想象，如果 `8080` 端口原来是一个不加密端口，现在将变成一个加密端口。

4.6 本地端口转发

有时，绑定本地端口还不够，还必须指定数据传送的目标主机，从而形成点对点的”端口转发”。为了区别后文的”远程端口转发”，我们把这种情况称为“本地端口转发（Local forwarding）”。

假定 host1 是本地主机，host2 是远程主机。由于种种原因，这两台主机之间无法连通。但是，另外还有一台 host3，可以同时连通前面两台主机。因此，很自然的想法就是，通过 host3，将 host1 连上 host2。我们在 host1 执行下面的命令：

```
$ ssh -L 2121:host2:21 host3
```

命令中的 `L` 参数一共接受三个值，分别是 `本地端口:目标主机:目标主机端口` ，它们之间用冒号分隔。这条命令的意思，就是指定 SSH 绑定本地端口 `2121` ，然后指定 host3 将所有的数据，转发到目标主机 host2 的 `21` 端口（假定 host2 运行 FTP，默认端口为 `21` ）。这样一来，我们只要连接 host1 的 `2121` 端口，就等于连上了 host2 的 `21` 端口。

```
$ ftp localhost:2121
```

“本地端口转发”使得 host1 和 host3 之间仿佛形成一个数据传输的秘密隧道，因此又被称为”SSH 隧道”。下面是一个比较有趣的例子。

```
$ ssh -L 5900:localhost:5900 host3
```

它表示将本机的 `5900` 端口绑定 host3 的 `5900` 端口（这里的 localhost 指的是 host3，因为目标主机是相对 host3 而言的）。另一个例子是通过 host3 的端口转发，SSH 登录 host2。

```
$ ssh -L 9001:host2:22 host3
```

这时，只要 SSH 登录本机的 `9001` 端口，就相当于登录 host2 了。

```
```

```
$ ssh -p 9001 localhost
```

上面的 `-p` 参数表示指定登录端口。

4.7 远程端口转发

既然"本地端口转发"是指绑定本地端口的转发，那么"远程端口转发（Remote forwarding）"当然是指绑定远程端口的转发。

还是接着看上面那个例子，host1 与 host2 之间无法连通，必须借助 host3 转发。但是，特殊情况出现了，host3 是一台内网机器，它可以连接外网的 host1，但是反过来就不行，外网的 host1 连不上内网的 host3。这时，本地端口转发就不能用了，怎么办?

解决办法是，既然 host3 可以连 host1，那么就从 host3 上建立与 host1 的 SSH 连接，然后在 host1 上使用这条连接就可以了。我们在 host3 执行下面的命令：

```
$ ssh -R 2121:host2:21 host1
```

`R` 参数也是接受三个值，分别是 `远程主机端口:目标主机:目标主机端口` 。这条命令的意思，就是让 host1 监听它自己的 `2121` 端口，然后将所有数据经由 host3，转发到 host2 的 `21` 端口。由于对于 host3 来说，host1 是远程主机，所以这种情况就被称为远程端口绑定。绑定之后，我们在host1就可以连接 host2 了：

```
$ ftp localhost:2121
```

这里必须指出，远程端口转发的前提条件是，host1 和 host3 两台主机都有 sshD 和 SSH 客户端。

4.8 SSH的其他参数

SSH 还有一些别的参数，也值得介绍。`N` 参数，表示只连接远程主机，不打开远程 Shell；`T` 参数，表示不为这个连接分配 TTY。这个两个参数可以放在一起用，代表这个 SSH 连接只用来传数据，不执行远程操作。

```
$ ssh -NT -D 8080 host
```

`f` 参数，表示 SSH 连接成功后，转入后台运行。这样一来，你就可以在不中断 SSH 连接的情况下，在本地 Shell 中执行其他操作。

```
$ ssh -f -D 8080 host
```

要关闭这个后台连接，只能用 `kill` 命令去杀掉进程。