三、中间人攻击

SSH之所以能够保证安全，原因在于它采用了公钥加密。

整个过程是这样的：

（1）远程主机收到用户的登录请求，把自己的公钥发给用户。

（2）用户使用这个公钥，将登录密码加密后，发送回来。

（3）远程主机用自己的私钥，解密登录密码，如果密码正确，就同意用户登录。

这个过程本身是安全的，但是实施的时候存在一个风险：如果有人截获了登录请求，然后冒充远程主机，将伪造的公钥发给用户，那么用户很难辨别真伪。

因为不像https协议，SSH协议的公钥是没有证书中心（CA）公证的，也就是说，都是自己签发的。

如果攻击者插在用户与远程主机之间（比如在公共的wifi区域），用伪造的公钥，获取用户的登录密码。再用这个密码登录远程主机。那么SSH的安全机制就荡然无存了。这种风险就是著名的[中间人攻击](http://en.wikipedia.org/wiki/Man-in-the-middle_attack)（Man-in-the-middle attack）。

SSH协议是如何应对的呢？

四、口令登录

如果你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 (12.18.429.21)' 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，就容易多了。

很自然的一个问题就是，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。

五、公钥登录(有待考证)

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。

```
When a client connects to the host, wishing to use SSH key authentication, it will inform the server of this intent and will tell the server which public key to use. The server then check its authorized_keys file for the public key, generate a random string and encrypts it using the public key. This encrypted message can only be decrypted with the associated private key. The server will send this encrypted message to the client to test whether they actually have the associated private key.

Upon receipt of this message, the client will decrypt it using the private key and combine the random string that is revealed with a previously negotiated session ID. It then generates an MD5 hash of this value and transmits it back to the server. The server already had the original message and the session ID, so it can compare an MD5 hash generated by those values and determine that the client must have the private key.
参考:https://www.digitalocean.com/community/tutorials/ssh-essentials-working-with-ssh-servers-clients-and-keys
```

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。从此你再登录，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
```

六、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：（1）"$ ssh user@host"，表示登录远程主机；（2）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；（3）"$ 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；（4）'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了。
本例中用本机(192.168.2.6)连接目标主机(192.168.2.110)

```
[root@local .ssh]# pwd
/root/.ssh
[root@local .ssh]# ll
total 0
[root@local .ssh]# ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key (/root/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /root/.ssh/id_rsa.
Your public key has been saved in /root/.ssh/id_rsa.pub.
The key fingerprint is:
f1:3e:73:15:06:e4:21:46:e2:9c:9e:94:0e:0d:dc:5c root@local
The key's randomart image is:
+--[ RSA 2048]----+
|     ..o.oE.+    |
|     .=0= o o    |
|      . O   .o   |
|       = +   . . |
|        S .   .  |
|         .       |
|          + .    |
|           +     |
|                 |
+-----------------+
[root@local .ssh]# ll
total 8
-rw-------. 1 root root 1675 Mar 19 17:20 id_rsa
-rw-r--r--. 1 root root  392 Mar 19 17:20 id_rsa.pub
[root@local .ssh]# ssh-copy-id root@192.168.2.110
The authenticity of host '192.168.2.110 (192.168.2.110)' can't be established.
RSA key fingerprint is 53:d4:0c:cb:db:66:b8:75:65:f0:58:a1:41:d2:1f:e1.
Are you sure you want to continue connecting (yes/no)? yes
/usr/bin/ssh-copy-id: INFO: attempting to log in with the new key(s), to filter out any that are already installed
/usr/bin/ssh-copy-id: INFO: 1 key(s) remain to be installed -- if you are prompted now it is to install the new keys
root@192.168.2.110's password:

Number of key(s) added: 1

Now try logging into the machine, with:   "ssh '192.168.2.110'"
and check to make sure that only the key(s) you wanted were added.

[root@local .ssh]# ll
total 12
-rw-------. 1 root root 1675 Mar 19 17:27 id_rsa
-rw-r--r--. 1 root root  392 Mar 19 17:27 id_rsa.pub
-rw-r--r--. 1 root root  395 Mar 19 17:29 known_hosts
[root@local .ssh]# ssh root@192.168.2.110
Last login: Sat Mar 19 17:30:51 2016 from 192.168.2.6
[root@cent5 ~]#
```

注意：

```
每一次执行ssh-keygen后都会生成id_rsa和id_rsa.pub且每次里面的内容都不同
每次用ssh/ssh-copy-id连接新的主机时都会生成known_hosts文件，用以记录，删除后会再生成
当使用ssh-copy-id时相当于登录到目标主机(192.168.2.110)，并把id_rsa.pub中的内容追加到目标主机的~/.ssh/authorized_keys中
遇到"Agent admitted failure to sign using the key."时只需要在本机执行ssh-add 即可
本机的id_rsa（私钥）和目标主机的authorized_keys（公钥）共同决定无密码登录成功
```

如何防止SSH会话超时：
如果你使用的是Mac或Linux，则可以编辑用户目录下的~/.ssh/config并添加以下行：

```
ServerAliveInterval 120.  // ssh -o serveraliveinterval=60 username@host
```

这将在您的SSH连接上每120秒发送一个空数据包以使它们保持活动状态。
更改服务器上/etc/ssh/sshd_config的SSH配置文件，以防止客户端超时，因此不必修改SSH客户端配置：

```
ClientAliveInterval 120
ClientAliveCountMax 720
```

如果客户端处于非活动状态120秒，这将使服务器向客户端发送一个空数据包，共发送720次，如果服务端向客户端发送消息达到此阈值，sshd将断开客户端的连接
以上的两种方法设置哪一个都可以。