1.2 SSH RSA Key Based Authentication

> This article is in draft. The tone of the article does not fall in line with the rest of the site. The examples are not yet made clear.
>
> This also needs a followup article about how to use keys across multiple systems and moving keys across systems.

Introduction

If your system is available through ssh on the Internet key based authentication must be used.

Even with preventative software such as fail2ban we have observed honeypot system being compromised within 3 days of being set up.

For the choice of keys to use, RSA is often selected over DSA because it has a stronger key length of 2048 and 4096. DSA can only be 1024.

It is unlikely you will run into issues if the versions of OpenSSH are different from client and server. However, just in case, you might want to determine the version of Open SSH installed,

```
ssh -V # Determine SSH client version
OpenSSH_5.2p1, OpenSSL 0.9.8l 5 Nov 2009

sshd -v #Determine SSH Server version (ignore the error message)
sshd: illegal option -- v
OpenSSH_5.2p1, OpenSSL 0.9.8l 5 Nov 2009
usage: sshd [-46DdeiqTt] [-b bits] [-C connection_spec] [-f config_file]
            [-g login_grace_time] [-h host_key_file] [-k key_gen_time]
            [-o option] [-p port] [-u len]
```

As long as the major number (the first digit) is close you should have no issues.

Installing SSH

This installs the SSH server and client,

```
sudo apt-get install ssh
```

At this point we have installed ssh, and it is time to create the keys for the users who require SSH access.

Note if you find that connecting via SSH is slow you might want to disable DNS lookup.

Generate Public and Private Keys on Client Machine

In principle, the generation of the Public and Private keys are done by user themselves on their own machine. This is because even the Unix Administrator should not have the user's private key.

> Scratching your head on why keys should be generated by users? Think passwords. Any enterprise grade environment will ask **you** to define your **own** password. Your password is then hashed and never revealed to the Administrator. In the case where an Administrator sets your initial password, the password will be "one time" where the system will prompt you to set a new password upon successful authentication.

There are many variations of keys you can generate but currently industry standard is below,

Encryption Key Type = RSA
Key Length = 2048

With a Unix based system, this can be accomplished with the command line (below). Windows does not have a native way of doing this, but most Windows ssh client programs will provide a means of key generation.

Windows Client

If you are on a Windows machine, make sure to store your private key on a protected location. Usually this would be your Windows desktop or home directory.

Putty is one of the most popular Windows SSH Clients and your keys can be setup using puttyGen. However, Putty also has not been updated in years and I've found the generated keys to be problematic (for example will not work on my Mac). Instead, I recommend BitVise SSH Client Tunnelier for the key generation.

For console work, I still use Putty (actually Kitty) for normal console work, but still keep BitVise for its superior interface for file uploads and port tunnelling.

Unix Client

On Ubuntu it's super easy and your generated private key also work with Windows SSH clients.

ssh-keygen without parameters generates 2048 RSA public and private keys,

- Private key kept on the client machine = **id_rsa**
- Public key put on the target server machine = i**d_rsa.pub** which will then be added into ~/.ssh/authorized_keys

```
ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key (/Users/tinpham/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /Users/tinpham/.ssh/id_rsa.
Your public key has been saved in /Users/tinpham/.ssh/id_rsa.pub.
The key fingerprint is:
c7:6c:3e:87:4a:09:90:ef:6d:a9:88:f8:f0:89:d2:13
tinpham@Tin-Phams-iMac.local
The key's randomart image is:
+--[ RSA 2048]----+
|        . oo.    |
|         s ..  . |
|          ...++ .|
|       T . +.=...|
|        F o + *. |
|         + o + . |
|          C .    |
|         . +     |
|                 |
+-----------------+
```

On a Unix system file permissions should automatically be set to protect your key files from other accounts.

Storing Your Private Key

You private key is to never be shared. It's the equivalent of giving away your password.

Windows Client

If you are on a Windows machine, make sure to store your private key on a protected location. Be default, usually the safest place is your Windows desktop or home directory.

Unix Client

If you used the commands provides, your keys will be generated in your protected home folder with further restrictions placed on your directory and files.

Place Public Key on Server

Ubuntu Shortcut

If you happen to using a Linux client and your Linux server still allows username password authentication, there is a shortcut to getting everything up and running on the server,

```
ssh-copy-id username@remotehost
```

It accomplishes in one command,

(Roderick you should fill this in)

Manually Copy Over Public Key to the Target Server

Transfer Over Public Key

Since I happen to be using Mac OS X I do this manually,

```
scp .ssh/id_rsa.pub bhitch@krypton.com:~
```

Setup .ssh Directory

Log into the **server** using your existing authentication method.

First check in your home folder that you have a .ssh directory and an authorized_keys file. The folder and file are generated if you had ever run the ssh client command,

```
ls -al # which will show hidden directories
ls -al .ssh # given the .ssh directory exists
```

If the directory did not exist, no problem, we can create ourselves,

```
mkdir ~/.ssh
chmod 700 ~/.ssh
touch ~/.ssh/authorized_keys
chmod 600 ~/.ssh/authorized_keys
```

Add the Public key added to the authorized_keys file,

```
cd ~/.ssh
cat ~/id_rsa.pub >> authorized_keys # appends the contents of the your
public key to the authorized_keys file

cd ~
rm id_rsa.pub # no need to keep this file around
```

Here is how everything should look in terms of permissions,

```
ls -al
drwx------ 2 tin.pham staff 4096 Jan 23 19:35 .
drwxr-xr-x 5 tin.pham staff 4096 Jan 15 20:41 ..
-rw------- 1 tin.pham staff  410 Jan 15 20:41 authorized_keys
-rw------- 1 tin.pham staff 1671 Jan 23 19:35 id_rsa
```

If you have been using SSH before, you might also see a file called known_hosts (will link and explain to this later).

TroubleShoot and Test Key Based Authentication

In the very first attempt, I like to trouble-shoot with a Unix or Linux command line client using the -v verbose option. Here's what a proper connection looks like coming form my Mac to the bonsaiframework,

Working ssh with verbose command.

```
ssh -v www.bonsaiframework.com
OpenSSH_7.4p1, LibreSSL 2.5.0
debug1: Reading configuration data /Users/tin.pham/.ssh/config
debug1: Reading configuration data /etc/ssh/ssh_config
debug1: Connecting to www.bonsaiframework.com [52.184.187.123] port 22.
debug1: Connection established.
debug1: identity file /Users/tin.pham/.ssh/id_rsa type 1
debug1: key_load_public: No such file or directory
debug1: identity file /Users/tin.pham/.ssh/id_rsa-cert type -1
debug1: key_load_public: No such file or directory
debug1: identity file /Users/tin.pham/.ssh/id_dsa type -1
debug1: key_load_public: No such file or directory
debug1: identity file /Users/tin.pham/.ssh/id_dsa-cert type -1
debug1: key_load_public: No such file or directory
debug1: identity file /Users/tin.pham/.ssh/id_ecdsa type -1
debug1: key_load_public: No such file or directory
debug1: identity file /Users/tin.pham/.ssh/id_ecdsa-cert type -1
debug1: key_load_public: No such file or directory
debug1: identity file /Users/tin.pham/.ssh/id_ed25519 type -1
debug1: key_load_public: No such file or directory
debug1: identity file /Users/tin.pham/.ssh/id_ed25519-cert type -1
debug1: Enabling compatibility mode for protocol 2.0
debug1: Local version string SSH-2.0-OpenSSH_7.4
debug1: Remote protocol version 2.0, remote software version OpenSSH_7.2p2
Ubuntu-4ubuntu2.1
debug1: match: OpenSSH_7.2p2 Ubuntu-4ubuntu2.1 pat OpenSSH* compat
0x04000000
debug1: Authenticating to www.bonsaiframework.com:22 as 'tin.pham'
debug1: SSH2_MSG_KEXINIT sent
debug1: SSH2_MSG_KEXINIT received
debug1: kex: algorithm: curve25519-sha256@libssh.org
debug1: kex: host key algorithm: ecdsa-sha2-nistp256
debug1: kex: server->client cipher: chacha20-poly1305@openssh.com MAC:
<implicit> compression: none
debug1: kex: client->server cipher: chacha20-poly1305@openssh.com MAC:
<implicit> compression: none
debug1: expecting SSH2_MSG_KEX_ECDH_REPLY
debug1: Server host key: ecdsa-sha2-nistp256
SHA256:8MQQ0Cd0AG3WEQdOhnOalK0z7XmGW6PE6s+1sa7Gb0Y
debug1: Host 'www.bonsaiframework.com' is known and matches the ECDSA host
key.
debug1: Found key in /Users/tin.pham/.ssh/known_hosts:20
debug1: rekey after 134217728 blocks
debug1: SSH2_MSG_NEWKEYS sent
debug1: expecting SSH2_MSG_NEWKEYS
debug1: SSH2_MSG_NEWKEYS received
```

```
debug1: rekey after 134217728 blocks
debug1: SSH2_MSG_EXT_INFO received
debug1: kex_input_ext_info: server-sig-algs=<rsa-sha2-256,rsa-sha2-512>
debug1: SSH2_MSG_SERVICE_ACCEPT received
debug1: Authentications that can continue: publickey
debug1: Next authentication method: publickey
debug1: Offering RSA public key: /Users/tin.pham/.ssh/id_rsa
debug1: Server accepts key: pkalg rsa-sha2-512 blen 277
debug1: Authentication succeeded (publickey).
Authenticated to www.bonsaiframework.com ([52.184.187.123]:22).
debug1: channel 0: new [client-session]
debug1: Requesting no-more-sessions@openssh.com
debug1: Entering interactive session.
```

```
debug1: pledge: network
debug1: client_input_global_request: rtype hostkeys-00@openssh.com
want_reply 0
```

The key values to notice are the loading of the proper key files, that RSA is being used and the "offering" of the key completes successfully. You should not need to type in your account password if your keys are working.

Disable Password Authentication

Just Enabling Key Based Authentication is not enough to secure your system. If you do not have a key, you will still be prompted for a password which will still work. The final step is to disable password authentication altogether.

Modify the sshd_config file to disable password authentication,

```
# Since this is a critical file, back it up first.
sudo cp /etc/ssh/sshd_config
/etc/ssh/sshd_config.2011-02-12.v0.0.tinpham_about_to_disable_password_aut
h.bck
# Load the file in your favourite editor.
sudo vi /etc/ssh/sshd_config
```

We can modify **sshd_config** quickly using sed,

```
sudo sed -i 's/#PasswordAuthentication yes/PasswordAuthentication no/g'
/etc/ssh/sshd_config
```

Changes the following,

```
# Change to no to disable tunnelled clear text passwords
#PasswordAuthentication yes
```

Uncomment and change yes to no. It should look like this,

```
# Change to no to disable tunnelled clear text passwords
PasswordAuthentication no
```

Finally restart ssh for the change to take effect,

```
sudo service ssh restart
```

In older versions of Ubuntu (to determine) where Upstart is not available use,

```
sudo /etc/init.d/ssh reload
 * Reloading OpenBSD Secure Shell server's configuration sshd
   ...done.
```

Now go to another machine other than the server and try to authenticate using ssh,

```
ssh tpham@lemonbistro.com
Permission denied (publickey).
```

The **Permission denied** indicates that password authentication is now disabled.

Reusing Public Keys Across Machines

You can actually reuse public keys across machines. With this approach, you only need to keep track of one private key per user. Of course, this also means if your private key is compromised all your systems are accessible with the one key.

Key Compromise

- ... revoking keys
- ... strategies for centralizing key management and then also pitfalls
- ... is it possible to force password protected private keys

Resources

http://www.ibm.com/developerworks/library/l-keyc.html - pretty good article, I think I can improve it, shorter, clearly show when running on client or server.

http://serverfault.com/questions/40071/ssh-keypair-generation-rsa-or-dsa - talks about key length.

https://help.ubuntu.com/10.10/serverguide/C/openssh-server.html - Ubuntu version of docs.

http://www.howtoforge.com/ssh_key_based_logins_putty - instructions on using Putty, found the Auto-login tip useful.

http://stackoverflow.com/questions/2419566/best-way-to-use-multiple-ssh-private-keys-on-one-client - how to use multiple ssh keys using the config file.

http://www.freetutorialssubmit.com/convert-ssh-private-key-with-putty-keygen/1400 - sometimes you need to use different formats of keys.