Passwordless SSH with encrypted homedir in Ubuntu

Quite recently I came across a very interesting issue: while configuring **passwordless SSH** (it's public key based, so depending on you have it configured it may not be completely passwordless) access to some of my VPS servers, I found that the same keypair just wouldn't work on one of the servers.

Not only that, but the behaviour was quite bizzare: upon my first attempt to connect the public key would get rejected and a regular password would be requested by the ssh session. But once I successfully logged in with my password, any subsequent ssh connections would happily authenticate by my public key and would let me in without a problem.

Those of you using home dir encrypiton in Ubuntu are probably smiling right now! But becase I have never consciously configured or used this feature, it took me a good few hours to troubleshoot the issue and come up with the fix.

Why public-key based SSH doesn't work with encrypted home directories

The answer is quite simple: before your server can decide whether you are providing a valid and trusted SSH key, it must read your public key stored in your homedir. But if your homedir is encrypted, this becomes a classical chicken-and-egg scenario – until you log in and therefore decrypt your homedir the server won't gain access to your public key. Only you wouldn't be needing the public key by then, would you?

Store your authorized SSH keys outside your encrypted home directory

If you happen to like your homedir encryption AND would like

to use public/private key SSH authentication, there is a way out: you need to store your authorized keys outside of your encrypted homedir.

The usual access restrictions and directory/file permissions still apply, so the only thing you're changing is moving your authorized keys outside of the encrypted homedir on your server. This way things will work exactly as you expect: you authenticate with your private key and this results in your automatically mounted and decrypted homedir.

Here are the steps to make this happen. You're going to need superuser privileges for my scenario because it caters for all the users on your Ubuntu server, not just one account that belongs to you (use **sudo to become root**).

Step 1: create a directory structure for your authorized keys.

First, the main directory, I created it under **/var** – seems quite a safe choice since this directory is unlikely to grow and is equally unlikely to get removed by accident.

```
# mkdir /var/openssh
```

Perfect! Now we need to create user-specific directories, just to keep this dir really tidy. My username is "**greys**", so here is the directory:

```
# mkdir /var/openssh/greys
# chown greys /var/openssh/greys
```

Step 2: copy existing authorized keys file into new location

(you must log in as your username for this, otherwise the homedir will stay encrypted)

```
$ cp /home/greys/.ssh/authorized_keys /var/openssh/greys
```

Step 3: update SSHd config with new location for authorized_keys file

You're going to do this as root once again:

 # vi /etc/ssh/sshd_config

update the value of the AuthorizedKeysFile so that it looks like this:

AuthorizedKeysFile %h/.ssh/authorized_keys /var/openssh/%u/authorized_keys

Step 4: Restart SSH service

 # service ssh restart
ssh start/running, process 3708

That's it! Give it a try and let me know how it worked out.

Recommended books:

[AMAZONPRODUCTS asin="1590594762"]

See also

- ssh – secure shell
- **How To Enable Secure Shell service in Ubuntu**
- Passwordless SSH
- Changing passphrase to your SSH key