요약

client의 public key를 server에 등록하면 끝.

client 작업 (ssh key pair 만들기)

```
cd ~/.ssh    # 권한이 0700 이어야함
ssh-keygen
# 이후 그냥 다 엔터.

#그럼 id_rsa랑 id_rsa.pub가 생김
```

server 작업 (ssh public key 넣기

```
cd ~/.ssh

# client의 id_rsa.pub를 복사해서 넣기
# (파일명을 mac_local.pub로 했다고 가정)

cat mac_local.pub >> authorized_keys   # client의 public key를 append 해주는거 임.
```

지금은 그냥 ubuntu server접속을 예시로 들었지만

ssh 통신을 하는 다른 곳에 많이 활용할 수 있다.

ex) public key를 github 서버에 등록하면 github 로그인이 필요 없어진다.

번외

ssh private key 비밀번호

위 Client 작업에서 `ssh-keygen` 이후에 그냥 다 엔터를 쳐서 private key에 대한 비밀번호를 생성 안했는데

만약 그 비밀번호를 설정하면 server접속시 server 계정 비밀번호 대신 그 ssh private key 비밀번호를 쳐야 함.

근데 이 비밀번호 치는 작업이 귀찮으면 `ssh-agent` 라는 프로그램(데몬)을 백그라운드로 돌리고

`ssh-add` 명령으로 해당 private key 와 그 비밀번호를 등록해놓으면

ssh private key의 비밀번호를 치지않아도 자동으로 로그인 가능하게 해준다.

하는 법

```
eval "$(ssh-agent)"
ssh-add ~/.ssh/{private key}

# 이후 private key의 비밀번호를 등록
```

`eval` 명령을 하는 이유

1 Answer

Active | Oldest | Votes

113

`ssh-agent` outputs the environment variables you need to have to connect to it:

```
shadur@proteus:~$ ssh-agent
SSH_AUTH_SOCK=/tmp/ssh-492P67qzMeGA/agent.7948; export SSH_AUTH_SOCK;
SSH_AGENT_PID=7949; export SSH_AGENT_PID;
echo Agent pid 7949;
shadur@proteus:~$
```

By calling `eval` you immediately load those variables into your environment.

As to why `ssh-agent` can't do that itself... Note the word choice. Not "won't", "**can't**". In Unix, a process can only modify its own environment variables, and pass them on to children. It **can not** modify its parent process' environment because the system won't allow it. This is pretty basic security design.

You could get around the `eval` by using `ssh-agent utility` where `utility` is your login shell, your window manager or whatever other thing needs to have the SSH environment variables set. This is also mentioned in the manual.

위 stackoverflow 링크

`ssh-agent` 명령을 치면 daemon이 실행되면서 `환경변수` 를 설정하는 shell script를 출력하는데

ssh-agent를 실행시키는 동시에 환경변수까지 설정해주기 위해서 eval을 사용하는 것이다.

```
ubuntu@ip-172-31-45-39:~/.ssh$ ssh-agent
SSH_AUTH_SOCK=/tmp/ssh-yGnt5JqqfwQ0/agent.29632; export SSH_AUTH_SOCK;
SSH_AGENT_PID=29633; export SSH_AGENT_PID;
echo Agent pid 29633;
```

ssh-agent 명령만 친 모습

```
ubuntu@ip-172-31-45-39:~/.ssh$ ps -ef | grep 29633
ubuntu   29633     1  0 17:48 ?        00:00:00 ssh-agent
ubuntu   29656 29124  0 17:49 pts/1    00:00:00 grep --color=auto 29633
```

daemon은 돌아가고 있다!

ssh-agent 명령만 쳤더니 프로세스는 돌아가지만 환경변수 설정이 안 된 것을 볼 수 있다.

정리

known_hosts와 authorized_keys

ssh 통신을 할때 key pair를 두 개 이용해서 한다.

"**서버의 pair(public, private), 클라이언트의 pair(public, private)**"

~/.ssh 폴더에 보면 `known_hosts` 와 `authorized_keys` 라는 public key가 등록된 파일이 두 개 있는데

`known_hosts` 에는 `내가 다른 서버에 접속` 할 때, 그 서버의 public key가 등록되고

`authorized_keys` 는 `다른 컴퓨터가 client로서 내 서버에 접속` 하려고 할 때, 그 client의 public key 다.

`known_hosts` 는 해당 서버 접속 시 자동으로 등록이 되고

`authorized_keys` 는 직접 등록을 해야 해당 클라이언트가 이 컴퓨터로 접속이 가능해진다.

정리2

ssh접속시 전체적인 진행상황

1. 클라이언트에서 private, public 키를 생성한다. ssh-keygen
2. 타겟서버에 public key를 복사해 놓는다. public key는 자유롭게 공유하여도 된다.
3. private key는 클라이언트만 가지고 있어야하며, 누구에게도 복사해 주거나 공개하면 안된다.
4. 공유된 public key로 메세지를 암호화 하고, 그 암호화된 메세지는 그 쌍이 되는 private key로만 해석할수 있다.
5. 클라이언트에서 타겟서버로 public key를 공유한다 ssh-copy-id -i 공개키경로 사용자@타겟서버ip
6. 클라이언트에서 타겟서버로 ssh접속 ssh 사용자@타겟서버ip
7. 클라이언트에서 타겟서버 최초 접속시 RSA key fingerprint로 접속여부를 확인하는 차원으로 (yes/no)를 물어본다.
8. 여기서 yes를 할시에는 ~/.ssh/known_hosts 파일에 해단 RSA key정보가 등록되며, 다음접속 부터는 물어보지 않는다.
9. no를 할시에는 접속이 불가능해진다.