

SSH and ssh-agent

Brian Hatch 2004-11-23

No one likes typing passwords. If people had their way, computers would simply know who they were and what they should have access to without us proving it at every turn.^[1] In my [last article](#) I showed you how to create SSH Identities/Pubkeys, which can be used as an alternative to password authentication. However, I then went right back and told you to passphrase protect them, so now you were substituting one password for another, seemingly gaining nothing.

This week we have the payoff. We'll take the Identity/Pubkey trust we created last time, and learn how to use the `ssh-agent` program as our keymaster. We'll decrypt our keys once, put them in into the agent, and have it handle all our authentication needs thereafter.

Starting up the Agent

To start up the agent you can simply run it on the command line:

```
$ ssh-agent
SSH_AUTH_SOCK=/tmp/ssh-OqdW7921/agent.7921; export SSH_AUTH_SOCK;
SSH_AGENT_PID=7922; export SSH_AGENT_PID;
echo Agent pid 7922;
```

When the agent starts, it writes some information to your screen that you can use to set up your shell's environment variables. In the above example, it is using Bourne shell syntax. If you were in a C-shell, say `/bin/csh` or `/bin/tcsh`, then it would have generated the variables differently. If `ssh-agent` can't determine which shell you are using correctly, you can use the `-s` or `-c` arguments to force it to provide Bourne or C-shell syntax, respectively.

The `/usr/bin/ssh` program uses the `SSH_AUTH_SOCK` environment variable to know how to contact the `ssh-agent` you're running, so once you run the agent, you should set the variables it provided. The easiest way to do this, and the reason it outputs those variables ready-to-go, is that you can use the shell `eval` function and backticks (```) to run the agent and the commands it creates, all in one fell swoop:

```
# Note: no ssh agent related variables yet
$ set | grep SSH_

# Run it inside backticks, which will capture the output and
# pass it to 'eval' which will run it in your current shell.
```

```
$ eval `ssh-agent`
Agent pid 7943

# And now those variables are in your shell, ready to use.
$ set | grep SSH_
SSH_AUTH_SOCK=/tmp/ssh-xoGi7942/agent.7942
SSH_AGENT_PID=7943
```

If you have the `SSH_AGENT_PID` variable set, you can kill off the agent using `ssh-agent -k`. You can always kill the daemon manually with the `kill` command as well.

Putting keys into the agent

The agent isn't very useful until you've actually put keys into it. All your agent key management is handled by the `ssh-add` command. If you run it without arguments, it will add any of the 'standard' keys `$HOME/.ssh/identity`, `$HOME/.ssh/id_rsa`, and `$HOME/.ssh/id_dsa`. If your keys are passphrase protected (and they should be!) then it will ask you for the passphrase to decode them. If the keys use the same passphrase, it will only ask you once, which can be convenient. [\[2\]](#)

So, let's actually put our keys into the agent:

```
$ ps -fp $SSH_AGENT_PID
UID          PID    PPID  C  STIME TTY          TIME CMD
laine     7943      1   0  15:52 ?           00:00:00 ssh-agent

# Are there any keys in there currently?
# 'ssh-add -l' (list) will show us.
$ ssh-add -l
The agent has no identities.

# Let's import the default keys.  In our case, we have
# each key protected with the same passphrase, which is
# why it only asks once.
$ ssh-add
Enter passphrase for /home/laine/.ssh/id_rsa:
Identity added: /home/laine/.ssh/id_rsa (/home/laine/.ssh/id_rsa)
Identity added: /home/laine/.ssh/id_dsa (/home/laine/.ssh/id_dsa)
Identity added: /home/laine/.ssh/identity (laine@desktop)
```

```
# What's in our agent now?
$ ssh-add -l
1024 79:e9:6f:99:a3:2d:ae:f3:bd:3a:87:6c:ed:4e:bb:ad laineedesktop (RSA1)
1024 23:d5:2b:20:02:a4:1a:c2:d0:d8:66:8f:a9:67:db:c0 id_dsa (DSA)
1024 e8:17:67:cf:9c:24:2b:59:ad:48:1d:e6:ea:d6:d9:3d id_rsa(RSA)
```

```
# And let's add a few one-off keys also
$ ssh-add ssh-keys/id*
Enter passphrase for id_dsa.webrooters:
Identity added: id_dsa.webrooters (id_dsa.webrooters)
Enter passphrase for identity.webrooters:
Identity added: identity.webrooters (webrooters@my_company.com)
```

```
# What's in our agent now?
$ ssh-add -l
1024 79:e9:6f:99:a3:2d:ae:f3:bd:3a:87:6c:ed:4e:bb:ad laineedesktop (RSA1)
1024 23:d5:2b:20:02:a4:1a:a9:67:db:c0:c2:d0:d8:66:8f id_dsa (DSA)
1024 e8:17:67:cf:9c:24:2b:59:ad:48:1d:e6:ea:d6:d9:3d id_rsa(RSA)
1024 1a:c2:d0:d8:66:23:d5:2b:20:02:a4:8f:a9:67:db:c0 id_dsa.webrooters (DSA)
1024 ae:f3:bd:3a:87:79:e9:6f:99:4e:bb:ad:a3:2d:6c:ed webrooters@my_company.com (RSA1)
```

Above we used `ssh-add` to add the default keys, `ssh-add -l` to list the keys in the agent, and `ssh-add filename` to add other keys explicitly.

Deleting keys from the agent

You can use the `ssh-agent -d` command to delete keys from the agent, as seen here:

```
# List keys
$ ssh-add -l
1024 79:e9:6f:99:a3:2d:ae:f3:bd:3a:87:6c:ed:4e:bb:ad laineedesktop (RSA1)
1024 23:d5:2b:20:02:a4:1a:a9:67:db:c0:c2:d0:d8:66:8f id_dsa (DSA)
1024 e8:17:67:cf:9c:24:2b:59:ad:48:1d:e6:ea:d6:d9:3d id_rsa(RSA)
1024 1a:c2:d0:d8:66:23:d5:2b:20:02:a4:8f:a9:67:db:c0 id_dsa.webrooters (DSA)
1024 ae:f3:bd:3a:87:79:e9:6f:99:4e:bb:ad:a3:2d:6c:ed webrooters@my_company.com (RSA1)
```

```
# Remove the key that came from the file ~/.ssh/id_dsa.webrooters
# from the agent. (Does not remove the file from the directory.)
$ ssh-add -d ~/.ssh/id_dsa.webrooters
Identity removed: id_dsa.webrooters (id_dsa.webrooters.pub)

# List keys again
$ ssh-add -l
1024 79:e9:6f:99:a3:2d:ae:f3:bd:3a:87:6c:ed:4e:bb:ad laineedesktop (RSA1)
1024 23:d5:2b:20:02:a4:1a:a9:67:db:c0:c2:d0:d8:66:8f id_dsa (DSA)
1024 e8:17:67:cf:9c:24:2b:59:ad:48:1d:e6:ea:d6:d9:3d id_rsa(RSA)
1024 ae:f3:bd:3a:87:79:e9:6f:99:4e:bb:ad:a3:2d:6c:ed webrooters@my_company.com (RSA1)
```

Why might you want to delete keys from the agent? The most common reasons are:

- You want to temporarily add a key that you want to use a lot for the next hour, but don't want it to stick around after you're done with it for security/paranoia reasons. (See the "Agent Security Concerns" section later.)
- You no longer use the keys, for example if you've upgraded all your servers to support SSHv2, and your RSA1 keys aren't used any more.
- You have too many keys in your agent, so you remove the keys that are least necessary. See the next section for why this may occur.

Too Many Agent Keys?

SSH servers only allow you to attempt to authenticate a certain number of times. Each failed password attempt, each failed pubkey/identity that is offered, etc, take up one of these attempts. If you have a lot of SSH keys in your agent, you may find that an SSH server may kick you out before allowing you to attempt password authentication at all. If this is the case, there are a few different workarounds.

- If you have keys in your agent that you don't need any more (for instance old obsolete RSA1 keys) then you can delete them from the agent using `ssh-agent -d filename`.
- If you know you want to use password authentication, you can prevent SSH from offering keys at all by temporarily disabling your `SSH_AUTH_SOCK` environment variable:

```
$ SSH_AUTH_SOCK='' ssh user@myserver ...
```

Or you can use put configuration options into `~/.ssh/options`, or supply them manually on the

command line:

```
# Using the configuration file:
```

```
$ head ~/.ssh/config
```

```
Host myserver
```

```
    # Allow SSHv1 Identity authentication?
```

```
    RSAAuthentication    no
```

```
    # Allow SSHv2 Pubkey authentication?
```

```
    PubkeyAuthentication no
```

```
$ ssh myserver
```

or

```
# Put it all on the command line.
```

```
# Or better yet, put it in a shell script or an alias...
```

```
$ ssh -o'RSAAuthentication=no' -o 'PubkeyAuthentication=no' user@myserver ...
```

- If you want to use a specific key, but it's too far down in the list in the agent (the SSH server kicks you out before it's offered) then you are out of luck. While I wish there were a way to suggest an order for the agent to offer keys, I don't know of one. If anyone has an idea, let me know. I'd love to just be able to supply a `-i filename` option on the command line, but that doesn't work.

Agent Security Concerns

The `ssh-agent` creates a unix domain socket, and then listens for connections from `/usr/bin/ssh` on this socket. It relies on simple unix permissions to prevent access to this socket, which means that any keys you put into your agent are available to anyone who can connect to this socket.

When the agent starts, it creates a new directory in `/tmp/` with restrictive permissions (0700), and creates it's socket therein with similarly restrictive permissions (0600). However, the `root` user on this machine still has the ability to override access restrictions on all local files, so `root` can access your agent's keys!

```
root# set | grep SSH_
```

```
root# ssh-add -l
```

```
Cannot connect to your agent.
```

```
root# ls -l /tmp/ssh-*/*
```

```
srwx----- 1 lainee alandra 0 Jan 21 11:51 /tmp/ssh-OqdW7921/agent.7921
```

```
root# SSH_AUTH_SOCK=/tmp/ssh-OqdW7921/agent.7921
```

```
root# export SSH_AUTH_SOCK
```

```
root# ssh-add -l
```

```
1024 79:e9:6f:99:a3:2d:ae:f3:bd:3a:87:6c:ed:4e:bb:ad lainee@desktop (RSA1)
```

```
1024 23:d5:2b:20:02:a4:1a:a9:67:db:c0:c2:d0:d8:66:8f id_dsa (DSA)
```

```
1024 e8:17:67:cf:9c:24:2b:59:ad:48:1d:e6:ea:d6:d9:3d id_rsa(RSA)
```

```
1024 ae:f3:bd:3a:87:79:e9:6f:99:4e:bb:ad:a3:2d:6c:ed webrooters@my_company.com (RSA1)
```

So the bad news is that your agent keys are usable by the `root` user. The good news, however, is that they are only usable while the agent is running -- `root` could use your agent to authenticate to your accounts on other systems, but it doesn't provide direct access to the keys themselves. This means that the keys can't be taken off the machine and used from other locations indefinitely.

Is there any way to keep `root` from using your agent, even though it can subvert unix file permissions? Yes, you can. If you supply the `-c` option when you import your keys into the agent, then the agent will not allow them to be used without confirmation. When someone attempts to use your agent to authenticate to a server, the `ssh-agent` will run the `ssh-askpass` program. This program will pop up on your X11 desktop and ask for confirmation before proceeding to use the key.

At this point you're probably going to realize that we're still fighting a losing battle. The local `root` account can access your X11 desktop, all your processes, you name it. If you can't trust the `root` user, you're in trouble.

However this will prevent `root` on machines to which you've forwarded the agent from accessing your agent.

Agent forwarding

One of the nice things about the agent is that it can follow you as you SSH from machine to machine. The default in newer versions of OpenSSH is to disable agent forwarding by default, so you'll need to decide when it's correct for you to use and specify it appropriately.

How does the agent forwarding actually work? In short, the agent is running on one machine, and

each time you SSH with agent forwarding, the server creates a 'tunnel' back through the SSH connection to the agent so it's available for any further SSH connections.

Let's say we're on our desktop, we SSH to a management server with agent forwarding, and from the management server SSH to our mail server. Here's what happens:

- `/usr/bin/ssh` on your desktop connects to the management server, authenticates, and requests agent forwarding.
- `/usr/sbin/sshd` on the management server creates a socket in `/tmp/ssh-XXXXXXX/agent.#####` and sets the `SSH_AUTH_SOCK` environment variable to match.
- The SSH daemon then starts up your shell, and you begin doing your work on the management server.
- When you decide to SSH out to the mail server, the `/usr/bin/ssh` program (here on the management server) sees the `SSH_AUTH_SOCK` environment variable and connects to that local socket file.
- The SSH daemon, who is the other end of the local socket `/tmp/ssh-XXXXXXX/agent.#####`, simply transfers data from `/usr/bin/ssh` on the management server to and from the `ssh-agent` running **on your desktop**. All the key mathematics are handled on the actual agent, which is running on your desktop, not on any of the intervening machines.
- The agent authenticates you to the mail server, and you're in.

Using agent forwarding can save you a lot of time and typing.

Also note that since your agent is available to any machine to which you forward it, it's also accessible by `root` on those systems, so don't forward it unless you trust those systems with your authentication credentials!

Turn off agent forwarding globally

Unless you have a good reason to forward the agent by default, you should verify that the agent forwarding is disabled by default. Locate the global `ssh_config` file, which typically lives in `/etc/` or `/etc/ssh/` and make sure you have the following:

```
Host *
    ForwardAgent no
```

This will disable `ssh-agent` forwarding unless explicitly requested.

Agent forwarding on the command line

To forward your agent via the command line, just include a `-A` flag:

```
desktop$ ssh -A user@remotehost
```

The `-a` option disables agent forwarding, which is the default.

Agent forwarding via the config file

If you have a host to which you always wish to forward your agent, without the trouble of supplying the `-A` flag, you can create entries in `~/.ssh/config` to turn it on for these hosts:

```
$ cat ~/.ssh/config
Host shellserver
    ForwardAgent yes

Host management-server
    ForwardAgent yes

Host *
```

Although the restrictive `Host *` section should be already contained in the global `ssh_config` file, I prefer to have it in my personal copy regardless.

Other Useful Features

There are several other command line flags and features of `ssh-add` and `ssh-agent`.

```
ssh-add -L
```

When called with a capital "L" argument, `ssh-add` will output the entire key, not just the fingerprint. Useful for concatenating into one's `~/.ssh/authorized_keys` file.

```
ssh-add -D
```

Delete **all** keys from the agent

```
ssh-add -x
```

Lock the agent with a password - it will be unusable until you unlock it again. A good thing to do if you want to leave your keys in the agent when you go home at night - you'll need to unlock it when you return, and no one can abuse it while it's locked. Unlock using `ssh-add -x` and supplying the password again.

```
ssh-add -t seconds filename
```

The `-t` tells the agent to discard the key after a specified amount of time. A good way to temporarily have keys in your agent, or feed your own paranoia.

```
ssh-agent -t seconds
```

Select a default lifetime for keys when the agent starts up. Lifetimes specified on the `ssh-add` command line, if present, override this default.

Conclusion

We've seen how `ssh-agent` can save a great deal of time and typing when used with SSH [Identity/ Pubkey authentication](#). While we're still some ways away from computers simply knowing who we are and what we have access to, tools like `ssh-agent` go a long way to keeping authentication strong yet making it easy-to-use.

Notes

[1] Unfortunately, many of those users are the same ones that choose their sweethearts's name for their password, and then stick it on their monitor in case they forget it.

[2] You may choose to have different passphrases for the keys to prefer security over convenience, but if you use a strong passphrase and each key has equivalent access, then a compromise of one is no worse than a compromise of all anyway.

About the author

Brian Hatch is the author of [Hacking Linux Exposed, 2nd Edition](#), [Building Linux VPNs](#), and of the [Linux Security: Tips, Tricks, and Hackery Newsletter](#). In order to exit an xterm, he frequently needs to log out of ten or more SSH connections, each with cascaded port forwards, to get back to his desktop shell. And that's not even including all the virtual `/usr/bin/screen` TTYS.

More articles by this author

View [more articles](#) by Brian Hatch on SecurityFocus.

[Privacy Statement](#)

Copyright 2006, SecurityFocus