Using Rsync and SSH

Keys, Validating, and Automation

This document covers using cron, ssh, and rsync to backup files over a local network or the Internet. Part of my goal is to ensure no user intervention is required when the computer is restarted (for passwords, keys, or key managers).

I like to backup some logging, mail, and configuration information sometimes on hosts across the network and Internet, and here is a way I have found to do it. You'll need these packages installed:

- rsync
- openssh
- cron (or vixie-cron)

Please note these instructions may be specific to Red Hat Linux versions 7.3, 9, and Fedora Core 3, but I hope they won't be too hard to adapt to almost any *NIX type OS. The man pages for 'ssh' and 'rsync' should be helpful to you if you need to change some things (use the "man ssh" and "man rsync" commands).

First, I'll define some variables. In my explanation, I will be synchronizing files (copying only new or changed files) one way, and I will be starting this process from the host I want to copy things to. In other words, I will be syncing files from /remote/dir/ on *remotehost*, as *remoteuser*, to /this/dir/ on *thishost*, as *thisuser*.

I want to make sure that 'rsync' over 'ssh' works at all before I begin to automate the process, so I test it first as *thisuser*:

```
$ rsync -avz -e ssh remoteuser@remotehost:/remote/dir /this/dir/
```

and type in *remoteuser@remotehost*'s password when prompted. I do need to make sure that *remoteuser* has read permissions to /remote/dir/ on *remotehost*, and that *thisuser* has write permissions to /this/dir/ on *thishost*. Also, 'rsync' and 'ssh' should be in *thisuser*'s path (use "which ssh" and "which rsync"), 'rsync' should be in *remoteuser*'s path, and 'sshd' should be running on *remotehost*.

Configuring *thishost*

If that all worked out, or I eventually made it work, I am ready for the next step. I need to generate a private/public pair of keys to allow a 'ssh' connection without asking for a password. This may sound dangerous, and it is, but it is better than storing a user password (or key password) as clear text in the script [0]. I can also put limitations on where connections made with this key can come from, and on what they can do when connected. Anyway, I generate the key I will use on *thishost* (as *thisuser*):

```
$ ssh-keygen -t rsa -b 2048 -f /home/thisuser/cron/thishost-rsync-key
Generating public/private rsa key pair.
Enter passphrase (empty for no passphrase): [press enter here]
Enter same passphrase again: [press enter here]
Your identification has been saved in /home/thisuser/cron/thishost-rsync-key.
Your public key has been saved in /home/thisuser/cron/thishost-rsync-key.pub.
The key fingerprint is:
2e:28:d9:ec:85:21:e7:ff:73:df:2e:07:78:f0:d0:a0 thisuser@thishost
```

and now we have a key with no password in the two files mentioned above [1]. Make sure that no other unauthorized user can read the private key file (the one without the '.pub' extension).

This key serves no purpose until we put the public portion into the 'authorized_keys' file [2] on *remotehost*, specifically the one for *remoteuser*:

```
/home/remoteuser/.ssh/authorized_keys
```

I use scp to get the file over to *remotehost*:

```
$ scp /home/thisuser/cron/thishost-rsync-key.pub remoteuser@remotehost:/home/remoteuser/
```

and then I can prepare things on *remotehost*.

Configuring *remotehost*

I 'ssh' over to *remotehost*:

```
$ ssh remoteuser@remotehost
remoteuser@remotehost's password: [type correct password here]
$ echo I am now $USER at $HOSTNAME
I am now remoteuser at remotehost
```

to do some work.

I need to make sure I have the directory and files I need to authorize connections with this key [3]:

```
$ if [ ! -d .ssh ]; then mkdir .ssh ; chmod 700 .ssh ; fi
$ mv thishost-rsync-key.pub .ssh/
$ cd .ssh/
$ if [ ! -f authorized_keys ]; then touch authorized_keys ; chmod 600 authorized_keys ; fi
$ cat thishost-rsync-key.pub >> authorized_keys
```

Now the key can be used to make connections to this host, but these connections can be from anywhere (that the ssh daemon on *remotehost* allows connections from) and they can do anything (that *remoteuser* can do), and I don't want that. I edit the 'authorized_keys' file (with vi) and modify the line with 'thishost-rsync-key.pub' information on it. I will only be adding a few things in front of what is already there, changing the line (and what follows is just one line with badly similated line wrapping) from this:

```
ssh-dss AAAAB3NzaC1kc3MAAAEBAKYJenaYvMG3nHwWxKwlWLjHb77CT2hXwmC8Ap+fG8wjlaY/9t4u
A+2qx9JNorgdrWKhHSKHokFFlWRj+qk3q+lGHS+hsXuvta44W0yD0y0sW62wrEVegz+JVmntxeYc0nDz
5tVGfZe6ydlgomzj1bhfdpYe+BAwop8L+EMqKLS4iSacNjoPlHsmqHMnbibn3tBqJEq2QJjEPaiYj1iP
5IaCuYBhuTKQGa+oyH3mXEif5CKdsIKBj46B0tCy0/GC7oWcUN92QdLrUyTeRJZsTWsxKpRbMliD2pBh
4oyX/aXEf8+HZBrO5vQjDBCfTFQA+35Xrd3eTVEjkGkncI0SAeUAAAAVAMZSASmQ9Pi38mdm6oiVXD55
Kk2rAAABAE/bA402VuCsOLg9YS0NKxugT+o4UuIjyl6b2/cMmBVWO39lWAjcsKK/zEdJbrOdt/sKsxIK
1/ZIvtl92DLlMhci5c4tBjCODey4yjLhApjWgvX9D5OPp89qhah4zu509uNX7uH58Zw/+m6ZOLHN28mV
5KLUl7FTL2KZ583KrcWkUA0Id4ptUa9CAkcqn/gWkHMptgVwaZKlqZ+QtEa0V2IwUDWS097p3SlLvozw
46+ucWxwTJttCHLzUmNN7w1cIv0w/OHh5IGh+wWjV9pbO0VT3/r2jxkzqksKOYAb5CYzSNRyEwp+NIKr
Y+aJz7myu4Unn9de4cYsuXoAB6FQ5I8AAAEBAJSmDndXJCm7G66qdu3ElsLT0Jlz/es9F27r+xrg5pZ5
GjfBCRvHNo2DF4YW9MKdUQiv+ILMY8OISduTeu32nyA7dwx7z5M8b+DtasRAa1U03EfpvRQps6ovu79m
bt1OE8LS9ql8trx8qyIpYmJxmzIdBQ+kzkY+9ZlaXsaU0Ssuda7xPrX4405CbnKcpvM6q6okMP86Ejjn
75Cfzhv65hJkCjbiF7FZxosCRIuYbhEEKu2Z9Dgh+ZbsZ+9FETZVzKBs4fySA6dIw6zmGINd+KY6umMW
yJNej2Sia70fu3XLHj2yBgN5cy8arlZ80q1Mcy763RjYGkR/FkLJ611HWIA= thisuser@thishost
```

to this [4]:

```
from="10.1.1.1",command="/home/remoteuser/cron/validate-rsync" ssh-dss AAAAB3Nza
C1kc3MAAAEBAKYJenaYvMG3nHwWxKwlWLjHb77CT2hXwmC8Ap+fG8wjlaY/9t4uA+2qx9JNorgdrWKhH
SKHokFFlWRj+qk3q+lGHS+hsXuvta44W0yD0y0sW62wrEVegz+JVmntxeYc0nDz5tVGfZe6ydlgomzj1
bhfdpYe+BAwop8L+EMqKLS4iSacNjoPlHsmqHMnbibn3tBqJEq2QJjEPaiYj1iP5IaCuYBhuTKQGa+oy
H3mXEif5CKdsIKBj46B0tCy0/GC7oWcUN92QdLrUyTeRJZsTWsxKpRbMliD2pBh4oyX/aXEf8+HZBrO5
vQjDBCfTFQA+35Xrd3eTVEjkGkncI0SAeUAAAAVAMZSASmQ9Pi38mdm6oiVXD55Kk2rAAABAE/bA402V
uCsOLg9YS0NKxugT+o4UuIjyl6b2/cMmBVWO39lWAjcsKK/zEdJbrOdt/sKsxIK1/ZIvtl92DLlMhci5
c4tBjCODey4yjLhApjWgvX9D5OPp89qhah4zu509uNX7uH58Zw/+m6ZOLHN28mV5KLUl7FTL2KZ583Kr
cWkUA0Id4ptUa9CAkcqn/gWkHMptgVwaZKlqZ+QtEa0V2IwUDWS097p3SlLvozw46+ucWxwTJttCHLzU
mNN7w1cIv0w/OHh5IGh+wWjV9pbO0VT3/r2jxkzqksKOYAb5CYzSNRyEwp+NIKrY+aJz7myu4Unn9de4
cYsuXoAB6FQ5I8AAAEBAJSmDndXJCm7G66qdu3ElsLT0Jlz/es9F27r+xrg5pZ5GjfBCRvHNo2DF4YW9
MKdUQiv+ILMY8OISduTeu32nyA7dwx7z5M8b+DtasRAa1U03EfpvRQps6ovu79mbt1OE8LS9ql8trx8q
yIpYmJxmzIdBQ+kzkY+9ZlaXsaU0Ssuda7xPrX4405CbnKcpvM6q6okMP86Ejjn75Cfzhv65hJkCjbiF
7FZxosCRIuYbhEEKu2Z9Dgh+ZbsZ+9FETZVzKBs4fySA6dIw6zmGINd+KY6umMWyJNej2Sia70fu3XLH
j2yBgN5cy8arlZ80q1Mcy763RjYGkR/FkLJ611HWIA= thisuser@thishost
```

where "10.1.1.1" is the IP (version 4 [5]) address of *thishost*, and "/home/remoteuser/cron/validate-rsync" (which is just one of a few options [6], including customization [7] to enhance security) is a script that looks something like this :

```
#!/bin/sh

case "$SSH_ORIGINAL_COMMAND" in
*\&*)
echo "Rejected"
;;
*\(*)
echo "Rejected"
;;
*\{*)
echo "Rejected"
;;
*\;*)
echo "Rejected"
;;
*\<*)
echo "Rejected"
;;
*\>*)
echo "Rejected"
;;
*\`*)
```

```
echo "Rejected"
;;
*\|*)
echo "Rejected"
;;
rsync\ --server*)
$SSH_ORIGINAL_COMMAND
;;
*)
echo "Rejected"
;;
esac
```

If *thishost* has a variable address, or shares its address (via NAT or something similar) with hosts you do not trust, omit the 'from="10.1.1.1",' part of the line (including the comma), but leave the 'command' portion. This way, only the 'rsync' will be possible from connections using this key. Make **certain** that the 'validate-rsync' script is executable by *remoteuser* on *remotehost* and test it.

PLEASE NOTE: The private key, though now somewhat limited in what it can do (and hopefully where it can be done from), allows the possessor to copy **any** file from *remotehost* that *remoteuser* has access to. This is dangerous, and I should take whatever precautions I deem necessary to maintain the security and secrecy of this key. Some possibilities would be ensuring proper file permissions are assigned [8], consider using a key caching daemon, and consider if I really need this process automated verses the risk.

ALSO NOTE: Another security detail to consider is the SSH daemon configuration on *remotehost*. This example focuses on a user (*remoteuser*) who is not *root*. I recommend not using *root* as the remote user because *root* has access to every file on *remotehost*. That capability alone is very dangerous, and the penalties for a mistake or misconfiguration can be far steeper than those for a 'normal' user. If you do not use *root* as your remote user (ever), and you make security decisions for *remotehost*, I recommend either:

```
PermitRootLogin no
```

or:

```
PermitRootLogin forced-commands-only
```

be included in the '/etc/ssh/sshd_config' file on *remotehost*. These are global settings, not just related to this connection, so be sure you do not need the capability these configuration options prohibit. [9].

The 'AllowUsers', 'AllowGroups', 'DenyUsers', and 'DenyGroups' key words can be used to restrict SSH access to particular users and groups. They are documented in the man page for "sshd_config", but I will mention that they all can use '*' and '?' as wildcards to allow and deny access to users and groups that match patterns. 'AllowUsers' and 'DenyUsers' can also restrict by host when the pattern is in USER@HOST form.

Troubleshooting

Now that I have the key with no password in place and configured, I need to test it out before putting it in a cron job (which has its own small set of baggage). I exit from the ssh session to *remotehost* and try [10]:

```
$ rsync -avz -e "ssh -i /home/thisuser/cron/thishost-rsync-key"
remoteuser@remotehost:/remote/dir /this/dir/
```

If this doesn't work, I will take off the "command" restriction on the key and try again. If it asks for a password, I will check permissions on the private key file (on *thishost*, should be 600), on 'authorized_keys' and (on *remotehost*, should be 600), on the '~/.ssh/' directory (on both hosts, should be 700), and on the home directory ('~/') itself (on both hosts, should not be writeable by anyone but the user). If some cryptic 'rsync' protocol error occurs mentioning the 'validate-rsync' script, I will make sure the permissions on 'validate-rsync' (on *remotehost*, may be 755 if every *remotehost* user is trusted) allow *remoteuser* to read and execute it.

If things still aren't working out, some useful information may be found in log files. Log files usually found in the **/var/log/** directory on most linux hosts, and in the **/var/log/secure** log file on Red Hat-ish linux hosts. The most useful logfiles in this instance will be found on *remotehost*, but *localhost* may provide some client side information in its logs [11] . If you can't get to the logs, or are just impatient, you can tell the 'ssh' executable to provide some logging with the 'verbose' commands: '-v', '-vv', '-vvv'. The more v's, the more verbose the output. One is in the command above, but the one below should provide much more output:

```
$ rsync -avvvz -e "ssh -i /home/thisuser/cron/thishost-rsync-key"
remoteuser@remotehost:/remote/dir /this/dir/
```

Hopefully, it will always just work flawlessly so I never have to extend the troubleshooting information listed here [12] .

Cron Job Setup

The last step is the cron script. I use something like this:

```
#!/bin/sh

RSYNC=/usr/bin/rsync
SSH=/usr/bin/ssh
KEY=/home/thisuser/cron/thishost-rsync-key
RUSER=remoteuser
RHOST=remotehost
RPATH=/remote/dir
LPATH=/this/dir/

$RSYNC -az -e "$SSH -i $KEY" $RUSER@$RHOST:$RPATH $LPATH
```

because it is easy to modify the bits and pieces of the command line for different hosts and paths. I will usually call it something like 'rsync-remotehost-backups' if it contains backups. I test the script too, just in case I carefully inserted an error somewhere.

When I get the script running successfully, I use 'crontab -e' to insert a line for this new cron job:

```
0 5 * * * /home/thisuser/cron/rsync-remotehost-backups
```

for a daily 5 AM sync, or:

```
0 5 * * 5 /home/thisuser/cron/rsync-remotehost-backups
```

for a weekly (5 AM on Fridays). Monthly and yearly ones are rarer for me, so look at "man crontab" or here for advice on those.

Alright! Except for the everyday "keeping up with patches" thing, the insidious "hidden configuration flaws" part, and the unforgettable "total failure of human logic" set of problems, my work here is done. Enjoy!

Notes:

[0] The reason behind choosing a SSH key with no password, over options like ssh-agent or keychain , is that the automated process will survive a reboot of the host machine and execute at the next scheduled time without any intervention on my part (not all machines so automated are always accessable). If you do not have those requirements, these other options may lend your implementation more security.
[1] If *remotehost* only has SSH1 installed, you may need to use another key type. Instead of 'rsa' you will need to use 'rsa1'. You can use 'dsa' instead of 'rsa', but it will still only be useful for a SSH2 connection (and key length may be an issue as noted here -- thank you @avenjamin). SSH2 connections are more secure than SSH1 connections, but you'll have to look elsewhere for the details on that ("man ssh-keygen" and Google). Also, the key creation can be done with the command (ssh-keygen -b 2048 -f keyfile -t rsa -N '') to automate the "no key password part", or (ssh-keygen -b 2048 -f keyfile -q -t rsa -N '') to eliminate any output from the command.
[2] Some configurations use the file 'authorized_keys2' instead of 'authorized_keys'. Look for "AuthorizedKeysFile" in '/etc/ssh/sshd_config'.
[3] If you use a shell other than 'bash' (or other bourne compatible shell), like 'csh' or 'tcsh', the commands listed may not work. Before executing them, start up a 'bash' (or 'sh', or 'ksh', or 'zsh') shell using the 'bash' (or 'sh', or 'ksh', or 'zsh') command. After completing the commands, you will have to exit the 'bash' shell, and then exit the shell your host spawns normally.
[4] Remember not to insert any newlines into the "authorized_keys" file. The key information, and the inserted commands associated with that key, should all be on one line. The key you generate (the nonsensical stuff on the key line) will be different from the one here. Choosing an editor that doens't automatically "wrap" (insert newlines) the text may be pivotal here.
[5] I have seen one host ignore a properly presented IPv4 address and instead see the incoming connection as a IPv6-ish sort of address ("::fff:10.1.1.1"). I found the address in '/var/log/messages' on a Fedora Core 3 Linux host, and it does allow connections from that host with the IPv6-ish version in the 'authorized_keys' file.
[6] Another option for validation (and more) is the Perl script located here: http://www.inwap.com/mybin/miscunix/?rrsync, though it is more complicated. A version of this Perl script is now bundled with the rsync source here: http://www.samba.org/ftp/unpacked/rsync/support/rrsync (with improvements). If you are writing a custom script, in whatever language you find comfortable, look inside this one for suggestions.
[7] By the time the 'validate-rsync' script runs, a SSH connection has been made with the SSH key you associated with this command in the 'authorized_keys' file. This example script basically tries to return 'Rejected' to anything other than a command that starts with "rsync --server", which is what rsync over ssh does on the other end of the connection. I found this out by running 'ps auxw | grep rsync' on the remote end of the connection after initializing a long running rsync job, but an rsync pro said you can add '-v -v -n' to your command line options for rsync and it will display the command it will use on the server end, so use that to make your script command more specific if you wish. The first six 'Rejected' lines try to elimate shell symbols that will allow a person to execute more than one command within a session (for example, a short rsync and some naughty command you don't want running remotely). You can also force the transfer to be read only by changing the command to "rsync --server --sender*" (thanks David Fred).
[8] By proper file permissions, I mean secure file permissions. In this you are essentially defending remotehost from remoteuser, so

that remoteuser would not be able to modify this setup in any way. That means that remoteuser will not own, or being able to write, the validation script or even remoteusers authorized_keys file. At this point, though, you may want to consider using "Match User" in /etc/ssh/sshd_config and use ChrootDirectory and/or ForceCommand to contain them if security is very important to you. Go as far as you see a need to go. (Thank you Yanek Martinson).

[9] "PermitRootLogin no" does what it says: the *root* user is not allowed to login via SSH. "PermitRootLogin forced-commands-only" requires that all connections, via SSH as *root*, need to use public key authentication (with a key like 'thishost-rsync-key.pub') and that a command be associated with that key (like 'validate-rsync'). For more explanation, use the "man sshd_config" command. If you are using Ubuntu, please make sure the package 'openssh-server' is installed (it is not installed by default).

[10] All kinds of SSH command line switches can be included (quoted) within the Rsync '-e' command line switch, like non-standard SSH server port connections (for example: "-p 2222" if SSH listens on port 2222), in addition to the private key ("-i identity_file") switch. (Per Funke suggested this and referenced http://mike-hostetler.com/blog/2007/12/rsync-non-standard-ssh-port).

[11] You can find out what log file SSH will be writing to by looking in two files: '/etc/ssh/sshd_config' and '/etc/syslog.conf'. 'sshd_config' contains the parameter "SyslogFacility", which by default is set to "AUTH", but Red Hat typically sets it to "AUTHPRIV". Whichever it is, remember the setting and look for it in the 'syslog.conf' file. Usually you will find a line with 'authpriv.*' followed by some tabs and then the log file you are searching for. Pay no attention to lines with 'authpriv.none' in them, as they are probaby taking in a many kinds of messages, but disallowing those from the 'authpriv' syslog facility.

[12] Not likely.

Links:

- Rsync
- Rsync Tutorial
- OpenSSH
- SSH, The Definitive Guide
- OpenSSH Key Management, Part 1 Part 2 Part 3
- Securing SSH
- Rsync + Stunnel 4.x (for another way to secure rsync)
- Using Rsnapshot and SSH
- Bulgarian translation by Albert Ward
- Polish translation by Katia Osipova
- French translation by Kate Bondareva