pam_mount and sshfs with password authentication

Posted by johns on Wed 26 Mar 2008 at 16:19

pam_mount is "a Pluggable Authentication Module that can mount volumes for a user session". It is used to automatically mount a network share or volume when a user logs in, and unmount it when the user logs out *sshfs* is a FUSE filesystem that allows mounting a directory using the SSH sftp subsystem.

pam_mount and sshfs work if SSH keys are set up, but will fail if one tries to use password authentication. In most cases it is better to use SSH keys, but sometimes there may be too many users (or too many computers) for this to be feasible.

If you have tried to use ssh in a script you might have noticed the following:

```
 $ echo password | ssh root@192.168.26.139
Pseudo-terminal will not be allocated because stdin is not a terminal.
root@192.168.26.139's password:
```

ssh never reads the password from stdin (which is what pam_mount expects), instead it opens the terminal directly. To get around this one can either use expect (which allocates a PTY and emulates a terminal), or one can use $SSH_ASKPASS. $SSH_ASKPASS is normally used to specify a X11 program which prompts the user for a password.

Below is a wrapper for ssh/sshfs which prompts for a password on stdin and sets things up so that ssh will call the wrapper to get the password, which will be read from the parent process using a pipe. The idea comes from the python version of LDM, the LTSP Display Manager, which passes the password to ssh this way.

```
#!/bin/bash
# Copyright (C) 2008 John S. Skogtvedt <jss at bzz.no>
# Licence: GNU GPL v3 or later at your option

if [ -n "$SSH_ASKPASS_FD" ]
then
         read password <&$SSH_ASKPASS_FD
        echo "$password"
       exit 0
elif [ $# -lt 1 ]
then
         echo "Usage: echo password | $0 <ssh command line>" >&2
       exit 1
fi

export SSH_ASKPASS=$0
export SSH_ASKPASS_FD=4
[ "$DISPLAY" ] || export DISPLAY=dummy:0
read password

exec 3<&0
# write password 100 times to make repeated ssh connections work
for x in $(seq 100)
do
   echo "$password"
done | exec setsid "$@" 4<&0 0<&3
```

Install and test the script:

```
$ sudo install sshaskpass.sh /usr/local/bin
$ stty -echo; read pw; stty echo
$ echo "$pw" | sshaskpass.sh ssh fileserver date
```

You should add the remote host to the global ssh_known_hosts file. Otherwise sshfs will fail unless the user has ssh'ed to the host before.

```
# ssh-keyscan -t rsa,dsa fileserver >> /etc/ssh/ssh_known_hosts
```

Configure pam_mount.

```
# vim /etc/security/pam_mount.conf.xml
```

These are only the changes necessary to get pam_mount to work with sshfs. PAM setup is fairly simple (see /etc/pam.d/common-pammount), and covered elsewhere.

The version of libpam-mount in testing has a new XML configuration format, but the equivalent changes for older versions should hopefully be obvious (I'm not sure it works though, I have only tested with 0.33 in testing).

Find the line that defines the mount command to be used for FUSE. Change it so it looks like this:

```
<fusemount>sshaskpass.sh mount.fuse %(VOLUME) %(MNTPT) -o %(OPTIONS)</fusemount>
```

Note that I also changed the *OPTIONS* part of the line. It is necessary because the way it's defined in the original file, -o and the options are sent in the same argument, breaking mount.fuse's argument parsing.

Add a volume/directory to be mounted. With user set to root and invert set to 1, the line is active for every user other than root.

```
<volume user="root" invert="1" fstype="fuse" path="sshfs#%(USER)@fileserver:" mountpoint="~/fileserver" options="reconnec
```

That should be it. Log in and test.

To mount directly on the home directory instead:

```
<volume user="root" invert="1" fstype="fuse" path="sshfs#%(USER)@fileserver:" mountpoint="~/" options="reconnect,nonempty
```

reconnect tells sshfs to reconnect if the connection is lost. *nonempty* allows mounting on a non-empty mountpoint (home directories are rarely empty). *allow_root* is necessary to avoid problems with pam_mkhomedir and display managers like GDM, which make sure the user's homedirectory exist on login.

If pam_mount doesn't unmount the sshfs filesystem at logout, saying *user seems to have other remaining open sessions* (if debugging is enabled in pam_mount.conf.xml), it is probably because you're also using pam_mount when logging in with ssh. To work around this, either don't use pam_mount for ssh logins, or set *UsePrivilegeSeparation no* in */etc/ssh/sshd_config*. The problem is that a file isn't deleted on logout. To delete it manually:

```
#  rm  /var/run/pam_mount/user
```

-- John S. Skogtvedt, BzzWare AS.
