

SSH Password Tripwire

Contributed by Mr-Oss
 Saturday, 08 November 2008
 Last Updated Saturday, 08 November 2008

SSH Password Tripwire

This article will demonstrate an easy to use method that will help you defend your system from ssh brute forcing and unauthorized access. Through the use of bash scripting inside of the system profile we will be adding a second level of passwording to remote ssh connections. Even if someone happens to successfully log into our system remotely via ssh, then they will have to have the second password in order to gain shell access.

This article will be using slackware linux and assumes the default user shell of bash. The changes we will be making take place in the global profile which should ensure that every user on our system will be required to successfully enter our defined password. The code to create this second layer authentication is as follows.

```
#####
# PROFILE TRIPWIRE
#####
MYTTY=`tty | awk -F"/" ' { print $3 } ' | tr [:upper:] [:lower:]`
REMOTEIP=`env | grep SSH_CONNECTION | cut -d '=' -f2 | cut -d ' ' -f1`
if [ "$REMOTEIP" != "" ]
then
if [ "$MYTTY" == "pts" ]
then
stty susp ""
trap `echo "Trapping CTRL-C, TERM and HUP";` HUP TERM INT
echo
PASSTHROUGH='mysecret'
echo "YOU ARE LOGGING INTO THIS SERVER FROM A REMOTE LOCATION"
echo "YOUR REMOTE IP ADDRESS IS $REMOTEIP"
echo "YOU ARE NOW REQUIRED TO ENTER THE SECOND LEVEL PASSWORD"
echo "PLEASE ENTER THE PASSWORD NOW: "
read PASSATTEMPT
if [ ! "$PASSATTEMPT" == "$PASSTHROUGH" ]
then
exit
fi
fi
fi
stty susp "^Z"
#####
# END OF PROFILE TRIPWIRE
#####
```

This code is found at the very end of our /etc/profile file. Lets break it down a little further so you understand how it works.

```
MYTTY=`tty | awk -F"/" ' { print $3 } ' | tr [:upper:] [:lower:]`
```

The line above will setup a variable that contains the terminal device which we will check later.

```
REMOTEIP=`env | grep SSH_CONNECTION | cut -d '=' -f2 | cut -d ' ' -f1`
```

The REMOTEIP line checks to see if an IP address value has been setup for the ssh connection.

```
if [ "$REMOTEIP" != "" ]
then
```

This line checks to see if the REMOTEIP variable has been set. If the variable has been set, it is safe to assume we are dealing with a remote ssh session which will need to comply to our second password standard.

```
if [ "$MYTTY" == "pts" ]  
then
```

This line checks the MYTTY variable against a pts terminal type. Most ssh sessions will be using a psuedo terminal which will be a /dev/pts. This line could probably be removed but just incase I have included it for good measure

```
stty susp ""  
trap 'echo "Trapping CTRL-C, TERM and HUP";' HUP TERM INT  
echo
```

The three lines above provide our script with some resilliance to very simple attacks such as the control+z suspending and control+c interrupt. Without these lines our super secure secondary password could be easily defeated with a control+z suspend or a control+c interrupt which would then result in the remote user gaining access to an interactive ssh session.

```
PASSTHROUGH='mysecret'
```

The PASSTHROUGH variable is our password. Change this to fit your own needs. For the sake of simplicity I have set it to mysecret.

```
echo "YOU ARE LOGGING INTO THIS SERVER FROM A REMOTE LOCATION"  
echo "YOUR REMOTE IP ADDRESS IS $REMOTEIP"  
echo "YOU ARE NOW REQUIRED TO ENTER THE SECOND LEVEL PASSWORD"  
echo "PLEASE ENTER THE PASSWORD NOW: "
```

The four lines above will be printed out on the terminal window of the remote ssh client. It will include the REMOTEIP address assuming it was correctly set. The last line then asks the user to input the secondary system password.

```
read PASSATTEMPT
```

The read PASSATTEMPT line will read the users input of a password and set it to a variable named PASSATTEMPT

```
if [ ! "$PASSATTEMPT" == "$PASSTHROUGH" ]  
then  
exit  
fi
```

The above if statement will take the user supplied password \$PASSATTEMPT and check it against our defined password variable \$PASSTHROUGH. If the two variables do not match then we execute the exit command which will boot out the unauthorized user from our system.

```
fi  
fi  
stty susp "^Z"
```

The lines above here close our 2 earlier defined if statements and restore our ability to suspend jobs because at this point we know the secondary password and have successfully been authenticated onto the system.

HERE IS AN EXAMPLE OF AN AUTHENTICATION ATTEMPT WHICH TRIES TO BREAK OUT USING ^C

```
login as: user  
user@192.168.0.11's password:  
Last login: Sat Nov 8 15:43:24 2008 from 192.168.0.25
```

YOU ARE LOGGING INTO THIS SERVER FROM A REMOTE LOCATION

YOUR REMOTE IP ADDRESS IS 192.168.0.25
YOU ARE NOW REQUIRED TO ENTER THE SECOND LEVEL PASSWORD
PLEASE ENTER THE PASSWORD NOW:

we enter control + c to attempt to interrupt the password prompt and get greeted with the following

Trapping CTRL-C, TERM and HUP

Next the session is closed and we are terminated

HERE IS AN EXAMPLE OF AN AUTHENTICATION ATTEMPT WHICH TRIES TO BREAK OUT USING ^Z

```
login as: user
user@192.168.0.11's password:
Last login: Sat Nov  8 15:43:24 2008 from 192.168.0.25
```

YOU ARE LOGGING INTO THIS SERVER FROM A REMOTE LOCATION
YOUR REMOTE IP ADDRESS IS 192.168.0.25
YOU ARE NOW REQUIRED TO ENTER THE SECOND LEVEL PASSWORD
PLEASE ENTER THE PASSWORD NOW:

^Z

control + z is entered and echoed to the screen at which time our session is then disconnected upon pressing the enter key

Invalid password entries are along the same lines. The incorrect password is entered and checked and then the session is disconnected and the user is booted from the system.

HERE IS AN EXAMPLE OF A SUCCESSFUL LOGIN

```
login as: user
user@192.168.0.11's password:
Last login: Sat Nov  8 15:19:02 2008 from 192.168.0.25
```

YOU ARE LOGGING INTO THIS SERVER FROM A REMOTE LOCATION
YOUR REMOTE IP ADDRESS IS 192.168.0.25
YOU ARE NOW REQUIRED TO ENTER THE SECOND LEVEL PASSWORD
PLEASE ENTER THE PASSWORD NOW:

```
mysecret
user@slackware:~$
```

At this point we have gained access to the system by entering our second password correctly. We also have had the control+c and control+z functionality restored to our session.

This is a simple but effective way to setup a secondary level of authentication on your systems. This could help defend you against brute force attacks that might guess your system password correctly but will then get denied by your secondary password which gets enforced when the system wide profile in /etc/profile is executed.

I hope you enjoyed this tutorial, look for more in the near future. Help us out if this helped you by clicking on our google ads.

-

MrOss