ROOT LEVEL ACTIVITY LOGGING - TAKE TWO

Recently I have had to relook at the concept of doing logging of the activities of the root user on *NIX systems. I have briefly touched on something like this in the past (see here) but because I have now had to cover a multitude of systems and shells I figured that an update would be useful. So lets start with the premise - you want to log what the root user is doing. Lets ignore the implications of needing to do this or even if you should, lets look at the how.

The first thing you need to realize is that on *NIX systems, the easiest logging is going to happen if you make use of the syslog (or syslog-like) facility. And if you want your root activity logs to mean anything, these logs need to be sent to a central logging server. It is useless to log the root activity on the same box that the root user can delete the logs on after all. Assuming you have these pre-requisites, lets move on..

AIX and the KSH shell
When I needed to do here looked something like this:

```
function log
{
        typeset -i stat=$?
        typeset x
        x=$(fc -ln -0)
        logger -p daemon.notice -t "ksh $LOGNAME $$" Status $stat PWD $PWD \'${x#
}\'
}
trap log DEBUG
```

This creates syslog entries that look something like this:

```
Jun 11 10:13:23 birdie ksh root 4008: [ID 702911 daemon.notice] Status 0 PWD /etc ' cd
/etc'
Jun 11 10:13:23 birdie ksh root 4008: [ID 702911 daemon.notice] Status 0 PWD /etc ' ls'
Jun 11 10:13:28 birdie ksh root 4008: [ID 702911 daemon.notice] Status 0 PWD /etc ' cat
/etc/passwd'
```

Now, this function needs to be placed where it would be used whenever the shell was used. This means you need to put it into the profile files used by the root shell.

SOLARIS and the BASH and KSH shell
Now the above did not work when we tried to use it on more recent solaris versions using the ksh shell, it kept core dumping and that is not fun. So we had to ask the team to change the shell used to bash and then we modified the function to look like:

```
function log
{
 typeset x
 x=$(history 1 | cut -f 5-)
 logger -p daemon.info -t "$LOGNAME" $PWD ${x#        }
}
trap log DEBUG
```

This gave us these syslog entries:

```
Aug 25 09:40:50 doggie root: [ID 702911 daemon.info] / 501 pwd
Aug 25 09:40:59 doggie root: [ID 702911 daemon.info] / 502 ps -ef|grep bob|grep sh
```

And again this needed to be placed into the profile files used by the root shell.

LINUX and the BASH shell
For the linux server we used the above function, just tweaked a little bit:

```
function log
{
 typeset x
 x=$(history 1 | cut -f 5-)
 logger -p daemon.notice -t "$LOGNAME" $PWD "${x#        }"
}
trap log DEBUG
```

And got the same output as above. Now you could also use the variable $BASH_COMMAND instead of history to get your last command listing, but the main problem with the variable is that it also includes whatever you have in your $PROMPT settings, which you need to then grep out, and that can get messy. Your call.

OLD SOLARIS and the KSH shell
Astute readers will have noted that in the above examples, the whole setup hinges around using functions in the shell. What do you do when you have to do this on a OLD solaris box that only has shells that have no function capability. This is where it got messy. I put together a work-around, but it was messy. This is what it looks like. First we need to setup the profile files used by root, we added a line to run our logging script:

```
nohup /bin/ksh /usr/local/bin/log.sh &
```

..and a line to handle our shell exits:

```
trap '. /.sh_logout; exit' 0
```

Now the logging script called looks like this:

```
BSE=/tmp
touch $BSE/lck.file

cat /.sh_history | strings -n 1 > $BSE/one

while [ -f $BSE/lck.file ]
 do
  sleep 10
  cat /.sh_history | strings -n 1 > $BSE/two
  diff $BSE/one $BSE/two | grep ">" | cut -c 2- > $BSE/gosyslog
  logger -p daemon.notice -t "root_profile" -f $BSE/gosyslog
  mv $BSE/two $BSE/one
 done
```

..simply put this takes the commands used by root in the last 10 seconds and sends them to syslog. Then to handle the cleanups when the shell exists:

```
clear
if [ -f /tmp/lck.file ]
then
    rm -rf /tmp/lck.file
    rm -rf /nohup.out
  fi
```

..just to make sure the background job dies and that all temporary files are cleared up.

Now all of the above solutions do work. The shell activity is captured and sent to syslog and from there can be sent off-server. Mission accomplished.

But.... You see, as root user -or any user- these logging methods are trivial to bypass. Because they all hinge on the fact that they run in the same privilege level as the user being logged. So when a function is used, you just overwrite the function:

```
function log { echo ""; } ; trap log DEBUG
```

..and in my messy 'old shell' kludge you can just:

```
export HISTSIZE=0
```

Now you could get creative with chattr and some other tricks to help with this, but fundamentally, root can do whatever they want on their server. And while I understand the need to log activities you need to be aware of 2 factors - (1) such logging should never be the only thing being done, look at using selinux, rotate staff, do periodic audits, etc and (2) if you really do not trust an administrator, then that staff member should not be working for you. There are certain roles that are all about trust first, skill level second and once the trust is gone, that person cannot fulfill that role properly. And an administrator is one of these roles.

Final Words

Fell free to use whatever you want to from this but please bear in mind the possible shortcomings and implications. For example, many people may take the idea of doing this type of logging as a sign of distrust. Dealing with administrator access and accountability is always going to be a issue, so be sure to do it with forethought and understanding. Have fun, play and learn.