Global Information Assurance Certification Paper

Sudo and SSH: A Scheme for Controlling Administrator Privileges and System Account Access

By Liam Forbes
GSEC Practical Assignment Ver 1.2e
June 11, 2001

1. Introduction

In an environment with many UNIX systems, several system administrators, and a plethora of users, it is easy to lose track of who has what privileges and access to system accounts. Controlling privilege allocation and system account access can become a management nightmare, or a gaping security hole. Handing out passwords to system accounts provides opportunity for those accounts to become backdoors into the systems. Having those account passwords on the system are opportunities for crackers to practice their craft. There are two tools that can improve site security, and simplify security management, sudo and SSH.

"Sudo (superuser do) allows a system administrator to give certain users (or groups of users) the ability to run some (or all) commands as root or another user while logging the commands and arguments." [10] The ability to parcel out privileged commands and not reveal system passwords greatly reduces the chance that an unauthorized user will learn those passwords. An administrator can go one step further and eliminate system passwords, including root's, on non-server systems (and probably a few servers as well). Then, even a cracker cannot learn a system password, because there is no encrypted string to crack.

In a networked environment, an administrator can securely execute commands across the network as various system accounts from a single central, trusted host. Using a one-way, encrypted, trust relationship from an administrative server to a client workstation or server, administrators can execute privileged commands without exposing any passwords. Again, going one step further, by using sudo and SSH's alternate authentication methods, an administrator can remove UNIX passwords from most of the system accounts across the entire network if they so desire.

2. Using Sudo to Assign Privileges and Eliminate System Account Passwords

In multi-user mode, sudo replaces su when administrators need to work as root, or other system accounts. This is especially useful in environments that use NIS and cannot use shadow password files. Sudo performs several actions before executing a command. It logs the command and arguments to a file, syslog, or both. It resets the PATH variable to a predefined setting. It eliminates other potentially dangerous variables from the environment (IFS, ENV, BASH_ENV, LD_*, & RLD_*). It checks that the user is in the sudoers file, and warns the administrators if he or she is not. The user is prompted for their password, or other authentication mechanism defined at compile time. If successfully authenticated, the user's effective and real UID and GID are set to 0, or another account ID if the -u option is used.

Many of these actions can be modified by compile time options. For example the authentication mechanism, the logging facility, the environment variables, and the PATH setting are all fixed when sudo is compiled. These compile time options control the behavior of sudo. The 'sudoers' file, controls the privileges.

Configuring Sudo

The sudo configuration file, /etc/sudoers, does not configure how sudo behaves, it only configures the privileges assigned to users. Via the 'sudoers' file, an administrator defines which users can use sudo, which commands they can execute, and where (which systems) the commands can be executed. Figure 1 provides an example sudoers file.

```
# /usr/local/etc/sudoers
# only edit this file with visudo

# User Aliases
User_AliasADMINS=evert,applebee
User_AliasSTUDENTS=bob,mary,will
User_AliasDATABASE=eli,avery

# Host Aliases
Host_AliasLAB=wkstn1,wkstn2,wkstn3
Host_AliasDATABASE=database.univ.edu

# Runas Aliases
Runas_Alias    DATABASE=dbmgr

# Command Aliases
Cmnd_AliasDATABASE=/usr/local/db/bin
Cmnd_AliasREBOOT=/sbin/reboot, \
            /sbin/shutdown -r *
Cmnd_AliasPRINTER=/usr/sbin/lpc,/usr/bin/lprm
Cmnd_AliasVIEWING=/bin/cat, /bin/more, \
            /usr/bin/head, /usr/bin/tail, \
            /bin/grep, /usr/bin/diff

# User Privilege Specification
ADMINS    ALL=(ALL) ALL, NOPASSWD: VIEWING
STUDENTS  LAB=REBOOT,PRINTER
DATABASE  DATABASE=(DATABASE) DATABASE
```

Figure 1. A sample sudoers file

There are several sections to the sudoers file. Most of the sections define aliases which are then used to define the user privileges. All of the aliases are comma separated lists. When the lists get long, a "\" indicates that the list continues on the next line. Aliases are used to group items (hostnames, usernames, commands) into symbols that will be used in

the final section. Wildcard characters can be used to simplify the items into patterns. It is important to be as specific as possible though, so that privileges are not accidentally assigned to unauthorized users.

User Aliases are lists of users who have sudo privileges. By grouping users into categories, an administrator can assign only those privileges necessary for the user to do their work. The sample file shows three types of users, administrators who require full root access, student lab managers who only require the ability to reboot systems and manage printers, and database users who manage a network database.

Host Aliases allow the administrator to assign privileges only on certain systems. A host alias can be a list of hosts, a list of netgroups, or a combination of the two. These aliases allow the administrator to group hosts based on location, or function. In the sample file, the LAB alias lists the client workstations and the DABASE alias lists a specific system, the database server.

Runas Aliases allow the administrator to restrict who a command can be executed as. By default, sudo commands are executed as root. However, it is also possible to execute the commands as other system accounts, or even other users. The '-u' option to sudo specifies what UID and GID are used to execute the command, but only if the user has been authorized in the sudoers file to execute the command as that account. The only runas alias in the sample file allows authorized users to execute commands as the dbmgr account.

Command Aliases define the commands that will be assigned to users. Pattern matching in this section is especially important. If a pattern is written too losely, then the user may be able to perform unauthorized functions. If a pattern is too strict, the user may not receive the intended privileges. By specifying just the command name, a user can use any of the arguments that go with that command. However, if the entire command line is specified, then only that function can be used. Do not assign shells or commands that can invoke a shell. If a user can get to a shell from a sudo command, then they have full root privileges, with no restrictions or logging. For example, vi should not be a sudo command because the user can break out to a shell using the ":!" key sequence.

The sample file has four command aliases. The DATABASE alias is a directory containing the commands used to manage a database. As long as an executable is located in that directory, it can be run with sudo. The REBOOT alias has two commands. The second command, `shutdown -r *`, is a pattern allowing the user to specify when the shutdown should occur. In order to allow only an immediate shutdown, the "*" could be replaced with the string "now". The PRINTER and VIEWING aliases are just command lists.

The User Privilege Specification combines all the aliases together to define "who gets what privileges on which hosts."[8] Here, all of the aliases are used to establish the users' root and system privileges. The syntax of a privilege specification is:

```
user host = [(runas)] [NOPASSWD:] [op]cmnd [: host = …]
```

This is a little difficult to read, however it's really just a build up of the aliases and a few extra pieces.

The "user" is either a username, or a user alias. The "host" is a hostname, or a host alias. The "runas" is a username, or a runas alias. The "cmnd" is a command, a list of commands, or a command alias. A single specification can define the user privileges available on different system groups by separating them with a ":".

The sample file has three specifications. The first gives the ADMINS group permission to execute any command as root. The need for su, and logging into a system as root, has been removed by this line. The specification goes a little further and gives the ADMINS permission to execute a subset of commands, the VIEWING commands, without having to authenticate themselves. This specification is the most permissive, therefore the ADMINS alias should be as small as possible with only those people who really need full root access.

The second and third specifications assign a very limited set of privileges to the students and the database managers. The STUDENTS group is given the permission to execute a system reboot. The DATABASE group is given the permission to execute any command in the /usr/local/db/bin as the dbmgr account. As seen in the DATABASE example, the same name can be used repeatedly for different aliases.

It is very important that the sudoers file not contain any mistakes. If it does, then sudo will not work for anyone. To avoid this, the sudo package comes with the command `visudo`, a tool for editing the sudoers file. This command scans the sudoers for any mistakes after editing is finished. If any mistakes are detected, it prompts the editor to fix them before finishing the edit session. The command, `visudo`, itself needs to be executed with root permissions, i.e. `sudo`, and invokes a `vi` editing session. Therefore this permission should be restricted to just those people with full root access.

Using Sudo

A user with sudo privileges enters commands from their own account, but prefixes them with the command `sudo`. Sudo prompts for the user's password and then executes the command as root, or some other user. Example: `% sudo rm /core`
Commands entered this way are logged, either to a separate sudo log, or to the `syslog(8)` daemon. The administrator can control which commands the user can execute, and the ownership under which they are executed. For example an administrator could give a user permission to execute a command as user 'adm'. Example: `% sudo -u adm acctcom`

After all of the privileges are assigned, and tested, the administrator can remove the password from the root account on most, or all of the systems where `sudo` is installed. Replacing the encrypted string with an asterisks, but leaving a valid home directory and shell, means the root account can no longer be logged into from the network or have its password cracked. There is no danger that the root password will be misplaced, misused,

given to the wrong person, or passed over the network, because the root password does not exist.

Once the root password is removed, all access to root is via sudo commands. Each command is logged which creates an audit trail of actions performed as root. For example, if one of the students rebooted the system, the following would appear in syslog:

```
May  2 07:16:57 5E:wkstn1 sudo:  mary : TTY=ttyq9 ;
PWD=/home/studentadmin/mary ; USER=root ;
COMMAND=/sbin/reboot
```

There are times when multiple commands have to be executed at once. Sudo does not maintain any state information from command to command, other than the last successful authentication time. Except for specific envionment settings, all other state information is taken from the user's own environment. For example the current working directory, and the display setting are based upon the user's settings. Multiple commands can be placed in double quotes and separated by semi-colons. This allows environment settings to be reset as part of a single sudo command. For example:

```
% sudo "setenv DISPLAY localhost.net:0.0; xclock"
```

If even the quotes are too restricting, or too many metacharacters have to be escaped, then sudo can be used to execute the `su` command. Since this means root actions will not be logge, it is a good idea to add some kind of explanation to the logs with the `echo` command. For example:

```
% sudo echo "Becoming root to restore backups from library"
% sudo su -
```

For administrators that are not ready to completely remove the root password, or for production servers that need a root password to enter single user mode, the root password can be stored in a safe place and no longer needs be circulated.

3. Using Sudo With SSH to Eliminate System Account Passwords On the Network

Configuring SSH

For administrators who still have to access client workstation's root account over the network though, there is a good tool for doing so - SSH. SSH has the obvious benefit of encrypting everything that passes over the network. There are several papers, and a book, on setting up SSH in general. The List of References in Appendix A lists some of those writings. Please refer to them for how to compile and configure SSH in general.

SSH also has "nonpassword authentication schemes"[8] which are more secure than standard UNIX passwords. Using symmetric AND asymmetric encryption schemes, an administrator can create a one-way trust relationship between system accounts on a central, trusted administrative server and the equivalent accounts on all the client

systems. A one-way relationship is preferable so that compromising the client system does not allow access back to the administrative server.

If SSH is only going to be used for administrative purposes, then configuration is simple. However, if SSH will be used for regular user logins, and system administration, it is better to run two instances of sshd, one for users, and one for administration. Seperating the use of ssh allows finer grained control fo the SSH daemon configuration. Figure 2 shows two examples of an sshd configuration file. One file is for SSH 1.2.27, the other is for OpenSSH 2.2.0.

```
# /etc/sshd-config (1.2.27)
# daemon configuration
Port                    30
ListenAddress         0.0.0.0
HostKey       /etc/sshhostkey
RandomSeed       /etc/sshseed
ServerKeyBits           1024
LoginGraceTime          180
KeyRegenerationInterval 3600
X11Forwarding           no
KeepAlive               yes
PrintMotd               yes
SyslogFacility         DAEMON
FascistLogging          yes
QuiteMode               no
StrictModes             yes
Umask                   0077

# authentication
PasswordAuthentication  no
PermitEmptyPassword     no
RhostsAuthentication    no
IgnoreRhosts            yes
RSAAuthentication       yes
RhostsRSAAuthentication yes

# permissions
AllowUsers          root dbmgr
AllowHosts       admin.u.edu
PermitRootLogin         yes
```

```
# /etc/sshd-config (2.2.0)
# daemon configuration
Port                    30
ListenAddress         0.0.0.0
HostKey       /etc/sshhostkey

ServerKeyBits           1024
LoginGraceTime          180
KeyRegenerationInterval 3600
X11Forwarding           no
KeepAlive               yes
PrintMotd               yes
SyslogFacility         DAEMON

StrictModes             yes

# authentication
PasswordAuthentication  no
PermitEmptyPassword     no
RhostsAuthentication    no
IgnoreRhosts            yes
RSAAuthentication       yes
RhostsRSAAuthentication yes

# permissions
AllowUsers          root dbmgr

PermitRootLogin
        forced-commands-only
```

Figure 2. SSH 1.2.27 (left column) and OpenSSH 2.2.0 (right column) sshd-config file

First an administrator has to create a set of keys for the central, trusted, administrative server. Then a set of keys has to be created for each system account that will be accessed from the administrative server. If commands will be executed from the crontab, across the network, then no passphrase should be associated with the keys. Otherwise, each account should have it's own unique passphrase. Once the keys are created, they have to be securely distributed to each system.

Once the keys are created, the daemon configuration file (figure 2) has to be copied to each system and the daemon started. The daemon listens on port 30, and will only accept connections by root or dbmgr. This corresponds to the accounts that users can execute sudo commands as (the runas aliases). Password login is disallowed, as is standard R-services authentication. However, public/private key logins are enabled along with RSA host authentication. By disabling the standard password login, but using RSA keys with no passphrase, the accounts can be logged into automatically by scripts running on the administrative server.

Using Sudo With SSH

Once SSH is configured, and privileges have been assigned on the administrative server, executing commands across the network, is just like issuing commands locally with sudo. The only change is the invocation of ssh to a workstation. Example:

```
% sudo ssh wkstn1 /sbin/reboot
```

This also works when connecting to other system accounts. Example:

```
% sudo -u dbmgr ssh database /usr/local/db/bin/restart
```

If the accounts have passphrases, or even passwords, SSH ensures they are not passed over the network in the clear. At the same time, sudo logs the entire command. The combination of the two tools provides a secure system for the centralized management of many hosts.

Conclusion

When su and the standard R-services were developed, they were intended for simple and efficient system administration and network connectivity. Over time, those programs worked well, but they are limited and insecure. Now, open source tools that extend the features of the original tools, and secure them are freely available. Provided they are configured correctly, sudo and SSH help system administrators get their work done, without giving away the "keys to the kingdom."

Appendix A - References

1. Barrett, Daniel and Richard Silverman. SSH, The Secure Shell: The Definitive Guide, Sebastopol, CA: O'Reilly & Associates, Inc., 2001.
2. Bhosle, Chandrashekhar. "Psudo root!", FreeOS.com, 14 March 2001. URL: http://www.freeos.com/articles/3799 (11 June 2001).
3. Crawford, Chuck. " SSH, Secure Shell", SANS GIAC Security Essentials Practicals, 12 October, 2000. URL: http://www.sans.org/infosecFAQ/authentic/SSH.htm (11 June 2001).
4. "FreeSSH", URL: http://www.freessh.org (11 June 2001).
5. Haszlakiewicz, Eric and Thor Lancelot Simon. "FreSSH", FreSSH.org, 15 February 2001. URL: http://www.fressh.org (11 June 2001).
6. Komarnitsky, Alek. "Sudo A Method of Controlling/Auditing Root Access; How it is used at a Large Aerospace Company", October 1998. URL: http://www.komar.org/pres/sudo (11 June 2001).
7. Koppel, Paul. "Using SSH2 for UNIX and Windows", SANS GIAC Security Essentials Practicals, 25 November, 2000. URL: http://www.sans.org/infosecFAQ/encryption/SSH2.htm (11 June 2001).
8. Mann, Scott and Ellen Mitchell. Linux System Security, The Administrator's Guide to Open Source Security Tools, Upper Saddle River, NJ: Prentice Hall PTR, 2000, pp 173 - 192, 257 - 312.
9. "OpenSSH", OpenBSD Project, 8 April 2001, URL: http://www.openssh.com (11 June 2001)
10. Reed, Jeremy C. "Delegating superuser tasks with sudo", BSD Today. June 2000, URL: http://www.bsdtoday.com/2000/June/Features192.html (11 June 2001).
11. Shama, Kapil. "Delegating Limited Superuser Access with Sudo", Linux.com. 29 June 2000. URL: http://www.linux.com/security/newsitem.phtml?sid=11&aid=9814 (11 June 2001).
12. "Sudo", URL: http://www.courtesan.com/sudo (11 June 2001)
13. "SSH" URL: http://www.ssh.com (11 June 2001)
14. "SSH FAQ" URL: http://www.employees.org/~satch/ssh/faq (11 June 2001)\

Last Updated: July 28th, 2017

Upcoming Training

SANS Prague 2017	Prague, Czech Republic	Aug 07, 2017 - Aug 12, 2017	Live Event
SANS Boston 2017	Boston, MA	Aug 07, 2017 - Aug 12, 2017	Live Event
SANS Salt Lake City 2017	Salt Lake City, UT	Aug 14, 2017 - Aug 19, 2017	Live Event
SANS New York City 2017	New York City, NY	Aug 14, 2017 - Aug 19, 2017	Live Event
Virginia Beach 2017 - SEC401: Security Essentials Bootcamp Style	Virginia Beach, VA	Aug 21, 2017 - Aug 26, 2017	vLive
SANS Chicago 2017	Chicago, IL	Aug 21, 2017 - Aug 26, 2017	Live Event
SANS Virginia Beach 2017	Virginia Beach, VA	Aug 21, 2017 - Sep 01, 2017	Live Event
SANS Adelaide 2017	Adelaide, Australia	Aug 21, 2017 - Aug 26, 2017	Live Event
Community SANS Pasadena SEC401 @ NASA	Pasadena, CA	Aug 23, 2017 - Aug 30, 2017	Community SANS
Mentor Session - SEC401	Minneapolis, MN	Aug 29, 2017 - Oct 10, 2017	Mentor
SANS Tampa - Clearwater 2017	Clearwater, FL	Sep 05, 2017 - Sep 10, 2017	Live Event
SANS San Francisco Fall 2017	San Francisco, CA	Sep 05, 2017 - Sep 10, 2017	Live Event
Mentor Session - SEC401	Edmonton, AB	Sep 06, 2017 - Oct 18, 2017	Mentor
SANS Network Security 2017	Las Vegas, NV	Sep 10, 2017 - Sep 17, 2017	Live Event
Mentor Session - SEC401	Ventura, CA	Sep 11, 2017 - Oct 12, 2017	Mentor
Community SANS Albany SEC401	Albany, NY	Sep 11, 2017 - Sep 16, 2017	Community SANS
Community SANS Dallas SEC401	Dallas, TX	Sep 18, 2017 - Sep 23, 2017	Community SANS
Community SANS Columbia SEC401	Columbia, MD	Sep 18, 2017 - Sep 23, 2017	Community SANS
SANS Baltimore Fall 2017	Baltimore, MD	Sep 25, 2017 - Sep 30, 2017	Live Event
SANS Copenhagen 2017	Copenhagen, Denmark	Sep 25, 2017 - Sep 30, 2017	Live Event
Community SANS Boise SEC401	Boise, ID	Sep 25, 2017 - Sep 30, 2017	Community SANS
Baltimore Fall 2017 - SEC401: Security Essentials Bootcamp Style	Baltimore, MD	Sep 25, 2017 - Sep 30, 2017	vLive
Community SANS New York SEC401	New York, NY	Sep 25, 2017 - Sep 30, 2017	Community SANS
Rocky Mountain Fall 2017	Denver, CO	Sep 25, 2017 - Sep 30, 2017	Live Event
SANS London September 2017	London, United Kingdom	Sep 25, 2017 - Sep 30, 2017	Live Event
SANS DFIR Prague 2017	Prague, Czech Republic	Oct 02, 2017 - Oct 08, 2017	Live Event
Community SANS Charleston SEC401	Charleston, SC	Oct 02, 2017 - Oct 07, 2017	Community SANS
Community SANS Sacramento SEC401	Sacramento, CA	Oct 02, 2017 - Oct 07, 2017	Community SANS
Mentor Session - SEC401	Arlington, VA	Oct 04, 2017 - Nov 15, 2017	Mentor
SANS October Singapore 2017	Singapore, Singapore	Oct 09, 2017 - Oct 28, 2017	Live Event
Community SANS Indianapolis SEC401	Indianapolis, IN	Oct 09, 2017 - Oct 14, 2017	Community SANS