Best Practices in UNIX Access Control with SUDO

Jun 30, 2015 | BSD Magazine Article |

by Leonardo Neves Bernardo

This article will discuss about security related issues at sudo environments. Will be evaluated advantages and disadvantages of to centralize sudo with LDAP back-end. Another issue summarized in this article is about taking care with content of sudo registers.

In the early days of UNIX, there were only two kinds of users: administrators and common users. Until now, this structure remained in the same model. Nevertheless, in our day by day activity, it is very common to meet some situations where it is necessary to delegate some responsibilities to operational groups and the others, who are not administrators nor common users. Some administrators do some insecure techniques like: sharing of root passwords, creation of users with uid 0, changes in file permission, and so on. These techniques are a solution for the immediate problem, but don't follow least privilege principle.

Around 1972, the notable Dennis Ritchie invented the setuid bit. The setuid bit allows users to run an executable with the permissions of the executable's owner. The most common situation is when an executable is owned by root. Programs must be carefully designed when the setuid bit permission is enabled, because vulnerable applications allow an attacker to execute arbitrary code under the rights of the process being exploited. After setuid bit creation, the division between root and other users starts to be broken. Unfortunately, to take advantage of this feature, it is necessary to rewrite the programs.

Around 1980, Bob Coggeshall and Cliff Spencer wrote Substitute User DO, or SUDO, one setuid program to run other programs without the necessity of these programs being rewritten. Sudo became the most used tool for privilege escalation in the UNIX environment. Sudo is under constant development. Security concerns are very important in sudo and sometimes some vulnerabilities are discovered and corrected immediately.

Basics about /etc/sudoers

The sudoers file is composed of three sections: defaults, aliases and user specifications.

Defaults

Defaults defines options to be used in every sudo entry. It's possible to overwrite options in each entry. We will discuss a little about some options ahead in this article.

Aliases

Aliases are variables used to group names. There are four types of aliases: `User_Alias`, `Runas_Alias`, `Host_Alias` and `Cmnd_Alias`. The name of an alias must start with an uppercase letter. Let's explain a little about each alias:

`User_Alias`

Is used to define group of users, for example:

```
User_Alias WEBMASTERS = user1, user2
```

You've probably realized UNIX has groups of users stored in the UNIX group of users (NSS group database) and there is no need to redefine those groups again. To use a UNIX group inside sudo, you need to append % in the register. In the following example, the UNIX group webmasters can be used inside sudoers as WEBMASTERS:

```
User_Alias WEBMASTERS = %webmasters
```

`Runas_Alias`

Is used to define group target users. Not always root is the target user, it's possible to use another users. `xRunas_Alias` is used to group them. Example:

```
Runas_Alias OPERATORS = operator1, operator2
```

`Host_Alias`

`/etc/sudoers` is prepared to be distributed among hosts. Hostnames, IP addresses and other kind of addresses are grouped in `Host_Alias`. Like `User_Alias`, it's possible to use a UNIX group of hosts, called netgroup (NSS netgroup database). Netgroup is not very common, but is useful for big environments. To use UNIX netgroup inside sudo, you need to append + in the register. In the following example, a UNIX netgroup webservers can be used inside sudoers as WEBSERVERS:

```
Host_Alias WEBSERVERS = +webservers
```

There are others possibilities to use `Host _ Alias`, like lists of hostnames or ip addresses:

```
Host_Alias WEBSERVER = host1, host2
Host_Alias WEBSERVER = 192.168.0.1, 172.16.0.0/16
```

Cmnd_Alias

`Cmnd_Alias` groups commands inside lists. Example of `Cmnd_Alias`:

```
CmnaAlias PRINTING= /usr/sbin/lpc, /usr/bin/lprm
```

For each one type of alias, there is one name built-in called ALL. It's possible to use sudo without any aliases, but aliases are recommended if you intend to use /etc/sudoers.

User Specifications

In the end of the sudoers file, there are user specifications entries. The sudoers user specification is in following form:

```
user host = (runas) command [,command,..]
user can be user, UNIX group prepending with % or User_Alias
host can be host, netgroup prepending with + or Host_Alias
runas can be user or group of user and unix group
```

command can be a command, list of commands divided by comma or Cmnd_Alias. command support wildcards.

Let's see an example of user specification:

```
root ALL = (ALL) ALL
```

In the above example, it is shown one user entry which permits the root user to run all commands (last ALL), in all hosts (first ALL), becoming all users (ALL inside parenthesis) when running a command.

The following example is more restrictive than the first example:

```
neves neves-laptop = (root) /usr/sbin/useradd
```

In this case, the user neves has permission to run the command `/usr/sbin/useradd` as user root in host neveslaptop only. As you can see, the second example is more adapted to the least privilege principle.

Let's go to see the result when user neves runs a command directly:

```
$ /usr/sbin/useradd neves2
 useradd: cannot lock /etc/passwd; try again later.
```

User neves doesn't have access to add user directly, but with sudo it could be possible:

```
$ sudo /usr/sbin/useradd neves2
 [sudo] password for neves:
 $
```

Well, it is a typical use of sudo and now it is possible to delegate some activities for operators group. By default, sudo requests the user password and maintains user password in cache for 5 minutes.

Let's see a little more complex example using aliases:

```
User_Alias OPERATORS = neves, neves2
 Host_Alias DESKTOPS = neves-laptop, neves-laptop2
 Cmnd_Alias MNGUSERSCMDS = /usr/sbin/userdel, /usr/sbin/
 useradd, /usr/sbin/usermod
 OPERATORS DESKTOPS=(ALL) MNGUSERSCMDS
```

Now, beyond `useradd` command, user neves is allowed to run `usermod` and `useradd` commands and sudoers is organized with aliases.

To manage `/etc/sudoers`, it is strongly recommended to use the visudo command. The advantage of the use of visudo is that it assures sudo syntax is correct before allowing one to save the sudoers file. We've seen a little about file `/etc/sudoers`. Almost all environments use this way to control sudo and it is okay for standalone servers or small environments. We will see that file sudoers is not the best configuration for big and medium size networks.

Common situations about sudoers distribution

Although it's possible to use `/etc/sudoers` setup in a perhost basis, sudo doesn't have any built-in way to distribute `/etc/sudoers` file among servers. It's very common in some companies that some team is in charge of operating and distributing `/etc/sudoers`. In another companies, there are scripts using version control (cvs, svn, etc), transfer commands (rsync, rdist, rcp, scp, ftp, wget, curl, etc.) or file share (nfs, netbios, etc.) to distribute `/etc/sudoers`. Although the use of scripts is better than manual operation, there are a lot of security issues to be considered in this case. There are some questions that need to be answered:

Are Changes in /etc/sudoers audited?

Imagine one attacker using sudo to get root access in your environment. It's important to think about which information you have in your log when something like that happens.

Do operators or scripts need root access to change `/etc/sudoers`?

If you are using push strategy to distribute `/etc/sudoers`, then probably the source will have rights to change destination servers, as the usual, with root access. In the worst case, with push strategy, you probably created one unique point where it is possible to get root access to entire environment.

Is the source of `/etc/sudoers` trusted?

Instead push strategy, perhaps you are using pull strategy. In this case, all servers are getting `/etc/sudoers` from one central point. There are two major concerns in pull strategy, first it's necessary to protect from man in middle attacks and second is to raise security level of central point. In general, pull is the best strategy to deploy sudoers files, because security problems don't compromise the entire environment. If you use one software of configuration management like puppet or cfengine to distribute sudoers and protect the configuration management server, your environment probably has a reasonable level of security. Even so, the pull strategy with configuration management lacks real time updates and sometimes lacks an auditing of changes in sudoers files.

Using back-end LDAP

Now let's discuss about the current best way to use sudo. With an LDAP back-end, sudo becomes a client-server service. For each use of sudo, the LDAP server will be consulted. We join the best advantages of LDAP and the best advantages of sudo to create one authorization system for UNIX environment.

Advantages of LDAP

Some advantages to use LDAP as sudo back-end are:

- LDAP protocol is standards-based
- If well structured with replication servers, you will have a high availability service
- There are access control lists (ACLs)
- It's possible to audit all changes and all consults
- LDAP is cross-platform, it's possible even to change from one server to another completely different one (e.g.: from openldap to Microsoft active directory)
- LDAP is very fast for search operations (almost all commands in sudo service)
- It's possible to use cryptography/TLS as requirement

Beyond these advantages, maybe the most important security consideration is that it is not necessary to open some security breach to distribute the sudoers file.

I don’t think it’s necessary restate about the importance of protecting your LDAP server(s). Some basic actions like to use firewall, TLS and put LDAP servers in segregated network are outside the scope of this article. If you have a non protected LDAP environment, it is probably better to use another strategy.

Creating LDAP structure

We will explain about how to build one basic LDAP server (OpenLDAP) to store sudo information. We will use OpenLDAP software, because OpenLDAP is the most widely known LDAP server distributed as open software. The procedures are about compilation of OpenLDAP, but if you prefer, you could install by package manager and achieve the same results. If you have one OpenLDAP server running, it is possible for you to jump to next topic. You could use another LDAP server instead openldap, but we won’t explain about this, please look for information in sudo documentation.

First of all, download the latest release of the Berkeley DB from the Oracle site (www.oracle.com/technetwork/database/berkeleydb) and latest version of OpenLDAP from the OpenLDAP site (www.openldap.org).

Compiling and installing Berkeley DB:

```
# tar -zxvf db-4.8.30.NC.tar.gz
 # cd db-4.8.30.NC/build_unix/
 # ../dist/configure && make && make install
```

OpenLDAP needs to find berkeley DB before compilation:

```
# export CFLAGS=”-I/usr/local/BerkeleyDB.4.8/include”
 # export CPPFLAGS=”-I/usr/local/BerkeleyDB.4.8/include”
 # export LDFLAGS=”-L/usr/local/BerkeleyDB.4.8/lib”
 # export LD_LIBRARY_PATH=”/usr/local/BerkeleyDB.4.8/lib”
```

Compiling and installing OpenLDAP:

```
# tar -zxvf openldap-2.4.26.tgz
 # cd openldap-2.4.26
 # ./configure && make depend && make install
```

Let’s start with a minimal OpenLDAP configuration file. Create a `/usr/local/etc/openldap/slapd.conf` with Listing 1 content.

And finally, start LDAP server with the command:

```
# /usr/local/libexec/slapd
```

OpenLDAP will bind TCP port 389, verify with netstat command:

```
# netstat -an | grep 389
 tcp 0 0 0.0.0.0:389 0.0.0.0:* LISTEN
 tcp6 0 0 :::389 :::* LISTEN
```

The next step is to create the root of your LDAP tree. Create one file named base.ldif with Listing 2 content.

And add content with the command ldapadd:

```
# ldapadd -D"cn=admin,dc=example,dc=com" -w"secret" -f
 base.ldif
 adding new entry „dc=example,dc=com"
 adding new entry „cn=admin,dc=example,dc=com"
```

Listing 1. ***Minimal slapd.conf***

```
#slapd.conf file
include /usr/local/etc/openldap/schema/core.schema
pidfile /usr/local/var/run/slapd.pid
argsfile /usr/local/var/run/slapd.args
database bdb
suffix "dc=example,dc=com"
rootdn "cn=admin,dc=example,dc=com"
rootpw secret
directory /var/lib/ldap
index objectClass eq
```

Listing 2. ***Base ldif***

```
#base.ldif
dn: dc=example,dc=com
objectClass: dcObject
objectClass: organization
dc: example
o: example
dn: cn=admin,dc=example,dc=com
objectClass: organizationalRole
cn: admin
```

Use ldapsearch to verify funcionality of your LDAP directory, as showed in Listing 3.

If the results are like Listing 3, your OpenLDAP is okay.

Remember that there are no security concerns in this server example. Your LDAP base is dc=example,dc=com, your admin user is cn=admin,dc=example,dc=com and your password of admin user is secret.

Listing 3. ***Test with ldapsearch***

```
# ldapsearch -x -b "dc=example,dc=com" -LLL
dn: dc=example,dc=com
objectClass: dcObject
objectClass: organization
dc: example
o: example
dn: cn=admin,dc=example,dc=com
objectClass: organizationalRole
cn: admin
```

Listing 4. ***Slapd.conf with sudo structure***

```
#slapd.conf file
include /usr/local/etc/openldap/schema/core.schema
include /usr/local/etc/openldap/schema/sudo.schema
pidfile /usr/local/var/run/slapd.pid
argsfile /usr/local/var/run/slapd.args
database bdb
suffix "dc=example,dc=com"
rootdn "cn=admin,dc=example,dc=com"
rootpw secret
index objectClass eq
index sudoUser eq
```

Creating sudo container

Now it's necessary to prepare your OpenLDAP to accept sudo information. First step is to include the sudo.schema. Download the latest stable sudo release source from the sudo site (www.sudo.ws) and copy the sudo.schema to the openldap schema directory:

```
# tar -zxvf sudo-1.8.2.tar.gz
 # cp sudo-1.8.2/doc/schema.OpenLDAP /usr/local/etc/
 openldap/schema/sudo.schema
```

Edit slapd.conf to include the sudo.schema and index to sudoUser attribute. Listing 4 shows slapd.conf with information related to sudo. Restart slapd to reread the new configuration:

```
# killall slapd
 # /usr/local/libexec/slapd
```

Create the file ldif sudo container, with the following content:

```
dn: ou=SUDOers,dc=example,dc=com
 objectClass: top
 objectClass: organizationalUnit
 ou: SUDOers
```

Add to the directory with ldapadd:

```
# ldapadd -D"cn=admin,dc=example,dc=com" -w"secret" -f
 sudo.ldif
 adding new entry „ou=SUDOers,dc=example,dc=com"
```

Your OpenLDAP is okay to control access with sudo. You have two possibilities at this moment, migrate your `/etc/sudoers` or start from zero.

Migrating sudoers content

Usually the easiest way to migrate sudoers information to LDAP is using a script `sudoers2ldif`. `sudoers2ldif` is located at `plugins/sudoers`, from the sudo source. To generate ldif file from `/etc/sudoers`, use the following commands:

```
# SUDOERS_BASE=ou=SUDOers,dc=example,dc=com
 # export SUDOERS_BASE
 # /usr/src/sudo-1.8.2/plugins/sudoers/sudoers2ldif /etc/
 sudoers > sudoers.ldif
```

And importing sudoers.ldif content to LDAP server:

```
# ldapadd -D"cn=admin,dc=example,dc=com" -w"secret" -f
 sudoers.ldif

 adding new entry „cn=defaults,ou=SUDOers,dc=example,dc=com"

 adding new entry „cn=root,ou=SUDOers,dc=example,dc=com"
 adding new entry „cn=OPERATORS,ou=SUDOers,dc=example,dc=com”
```

The script sudoers2ldif creates one register called defaults containing the default options and creating one LDAP register for each `/etc/sudoers` entry. Sometimes it's necessary to correct resulting ldif file before importing to LDAP. It happens because, depending your sudoers file, it sometimes creates more than one LDAP entry with the same DN (distinguished name). Duplicate DNs aren't supported by LDAP protocol.

LDAP sudoers registers

First, the difference between `/etc/sudoers` and sudoers inside LDAP is that, inside LDAP there are no aliases.

First of all, sudo looks for the register `cn=defaults` and parses it like a Defaults section in `/etc/sudoers`. The `cn=defaults` is a list of `sudoOptions`.

Other sudo registers, in general, are formed by combination of attributes `sudoHost`, `sudoUser` and `sudoCommand`. It's possible to use multiple values in each these attributes.

Listing 5 shows one example of sudo LDAP entry. In Listing 5, there is a sudo LDAP register with multiples of sudoUser, multiples of sudoHost and multiples of sudoCommand. It's possible to use attributes `sudoRunAs`, `sudoOption`, `sudoRunAsUser`, `sudoRunAsGroup`, `sudoNotBefore`, `sudoNotAfter`, `sudoOrder`, `sudoNotBefore` and `sudoNotAfter` are very interessing, because it's possible to define the time that permission is valid in sudo.

Listing 5. ***sudo LDAP entry***

```
# OPERATORS, SUDOers, example.com
dn: cn=OPERATORS,ou=SUDOers,dc=example,dc=com
objectClass: top
objectClass: sudoRole
cn: OPERATORS
sudoUser: neves
sudoUser: neves2
sudoHost: neves-laptop
sudoHost: neves-laptop2
sudoRunAsUser: ALL
sudoCommand: /usr/sbin/userdel
sudoCommand: /usr/sbin/useradd
```