System Security, Solution Access Control

George Cojocar

November 1, 2015

1 Unix Access Control

a. (Exercise 4.5 from book [1]) Unix treats file directories in the same fashion as files; that is, both are defined by the same type of data structure, called an inode. As with files, directories include a nine-bit protection string. Before you answer the following questions, make sure you know what each of these nine bits represent and what their effect on files and directories is. If care is not taken, this can create access control problems.

For example, consider a file owned by user A with protection mode 644 (octal) contained in a directory with protection mode 777. How can user B compromise this file in a somewhat unexpected way? Name a scenario that user B could use the compromise for.

Solution:

> The user B can remove or rename the file using the commands rm and respectively mv. This is possible because these file operations need the write and the execute permissions only on the parent directory. The permissions of the file itself are irrelevant.

b. Given the following table of UNIX directory/file permissions, and assuming that in what

Directory	Permissions	Owner	Group
/	rwx,r-x,r-x	root	root
/home	rwx,r-x,r-x	root	root
/home/capkun	rwx,r-x,–x	capkun	faculty
/home/capkun/syssec	rwx,rwx,–x	capkun	syssec_admin
/home/capkun/syssec/exercises	rwx,rwx,r-x	rmasti	syssec_admin
/home/capkun/syssec/exams	rwx,rwx,—	capkun	syssec_admin
/home/capkun/syssec/grades	rwx,—,—	capkun	nobody
/home/capkun/syssec/team1	rwx,–x,–x	rmasti	nobody
/home/capkun/syssec/team1/exercise1	r-x,rwx,–x	thomas	team1
/home/capkun/syssec/team1/exercise1/results	r-x,rwx,—	markus	team1

follows a string *group.username* denotes a user with username *username* and group identity *group*:

(a) Would nobody.thomas be able to create a directory “exercise2” under /home/capkun/syssec/team1/? What can you infer about the creation of directory “exercise1” ?

(b) Given that a user *group*.markus created "results" without a problem, what can you infer about *group* ?

(c) Which of the following commands will succeed when executed individually?(True/False) Please, briefly justify your answers for the ones that fail.

1. rmasti: echo "grades 2012" > /home/capkun/syssec/grades/grades2012.txt
2. nobody.matthias: ls /home/capkun/syssec/exercises
3. syssec_admin.ellia: echo "exercise1">/home/capkun/syssec/exercises/ex1.txt
4. syssec_admin.ellia: chmod 007 /home/capkun/syssec/exercises/
5. nobody.rmasti: chmod 007 /home/capkun/syssec/exercises/
6. team2.matthias: ls /home/capkun/syssec/team1/exercise1/results
7. nobody.rmasti: rm /home/capkun/syssec/team1/exercise1/results/*
8. nobody.christina: touch /home/capkun/syssec/exams/exam2012.txt
9. faculty.johnson: ls /home/capkun/syssec/exams

Given that thomas has read access to /home/capkun/syssec/exercises/*:

10. nobody.thomas: cp /home/capkun/syssec/exercises/* /home/capkun/syssec/team1/exercise1/
11. team1.thomas: cp /home/capkun/syssec/exercises/* /home/capkun/syssec/team1/exercise1/

Solution:

(a) "nobody.thomas" would not be able to create any directory within /home/capkun/syssec/team1/. The execute permission for group "nobody" only allows him to enter the folder using the command *cd*.

(b) The *group* is "team1", because only members of group "team1" have write permission on directory "exercise1".

(c)
1. False: Only user "capkun" has write permission on directory "grades".
2. True
3. True
4. False: Only the owner "rmasti" of the directory "exercises", or the root user, may change the permissions.
5. True
6. False: "team2.matthias" belongs to the *other* category, which has no permissions on folder "results".
7. False: Only members of group "team1" can remove the content of directory "results".

8. False: Only user "capkun" or members of group "syssec_admin" can create a file in directory "exams".
9. False: Only user "capkun" or members of group "syssec_admin" have read permission on folder "exams".
10. False: The user "nobody.thomas" does not have write permission to folder "exercise1", and therefore he cannot copy any files into this folder.
11. False: The owner "thomas" does not have write permission even though his *group* "team1" has. The Linux's ACL uses only one set of *rwx* bits when evaluating the permissions. The *owner* permissions take precedence over *group* permissions which take precedence over *other*. This scenario is rather a corner case than a security measure, because the owner "thomas" can change anyhow the permissions of directory "exercise1" using the command *chmod*.

2 Permission Delegation: setuid, sudo, su

(a) What is the difference between the sudo and su commands in Linux? Which would you use to enable temporary root access and why?

Solution:

The *su* command switches to the root user account by requiring the root account's password when executed without additional arguments. It offers also the possibility to switch to any user account, if a user name is provided as argument *su - username*, it prompts for user's password.

The *sudo* command runs a single command with root privileges, on behalf of authorized users. The users need to provide only their own password in order to be able to execute any command with the same privileges as root.

The sudo command is more suitable for temporary root access because does not require the root's password. Another advantage is that it discourage users from logging in as root in order to get a root's shell to do their normal work. Running fewer commands as root increase security and prevents system-wide changes.

(b) Which users can "sudo" on a Linux system?

Solution:

Any user defined in `/etc/sudoers` file gains full root privileges when preceding a command with sudo. For instance, the following entry grants to "user" the root's rights:

```
user    ALL=(ALL) ALL
```

Another option is to allow all members of the group *sudo* to execute commands with root privileges:

```
%sudo ALL=(ALL) ALL
```

(c) In the VM we have provided a script to setup the next test. Please run `acsetup.sh` as **root** (e.g. `sudo ./acsetup.sh`). It will create a secret directory with a secret file:

```
mkdir -p /secret/root-only
chmod go-rwx /secret/root-only
echo "topsecret" > /secret/root-only/secret-file
```

It creates a shell script /secret/script.sh (owner:root, group:root) with the content:

```
#!/bin/sh
echo "I am not root but can see this";
ls -al /secret/root-only
```

It makes script.sh readable and executable by all (`chmod o+rx /secret/script.sh`) and makes it setuid root (`chmod +s /secret/script.sh`).

Now try to run script.sh without root access (e.g., using sudo) from a user account with sudo permissions, i.e., the user can run sudo in general but you do not use it here. What do you expect to see? What was the obtained output? Can you explain the result?

Solution:

I would expect to see the content of *root-only* directory, but the output returned by the script is the following:

```
% /secret/script.sh

I am not root but can see this
ls: cannot open directory /secret/root-only: Permission denied
```

An explanation of this behavior is that Linux normally ignores [3] the setuid bit on all executable files starting with a shebang (`#!`) symbol, because it presents a high security risk.

(d) As a sysadmin, think of a setup that allows a normal user to list the contents of `/secret/root-only` without changing the file permissions. The setuid-bit should be used somewhere. How do you get such a setup? Try to follow the least privilege principle.

Solution:

One can configure the setuid-bit on `ls` command and then list the content of `/secret/root-only` directory as follows:

```
% sudo chmod +s /bin/ls

% /secret/script.sh
```

```
I am not root but can see this
total 12
drwx------ 2 root root 4096 Nov  1 16:05 .
drwxr-xr-x 3 root root 4096 Nov  1 16:05 ..
-rw-r--r-- 1 root root   10 Nov  1 16:05 secret-file
```

References

[1] Computer Security: Principles and Practice. William Stallings and Laurie Brown, Prentice Hall, 2008

[2] How can I get setuid shell scripts to work, http://www.faqs.org/faqs/unix-faq/faq/part4/section-7.html

[3] Quick introduction to SUID, http://www.techrepublic.com/blog/linux-and-open-source/quick-introduction-to-suid-what-you-need-to-know/