Lab - 1 - Linux files' access control

Practical activities- Linux files' access control

Introduction

With this work we intend to study the elementary security mechanisms in common Linux file systems. Only one Linux system should be used, either natively installed on a machine or running on a virtual machine. You can use a Linux Live distribution.

Linux file systems

Linux supports several file system types. A file system is a file storage structure that provides a set of functionalities for managing files and directories. The default Linux file systems are named `ext2`, `ext3` and `ext4`. These file system types support a set of optional features that depend of their drivers installed in a Linux kernel. One of those options is the POSIX Access Control Lists (ACLs), that can be used to complement, or extend, the so-called minimal Linux ACLs.

Linux file access rights

Files have a protection ACL that allows you to indicate whether they can be read, modified or executed (`rwx`). The latter is only of interest for files that have applications directly executable on the microprocessor (so-called binary executables) or for files that have commands, usually called scripts, that can be interpreted by a script interpreters (e.g. `shell`, `awk`, `perl`, `python`, etc.).

A note on file execution

There are binary executables files that usually are not handled as applications (e.g. dynamically loadable libraries, which are mapped in virtual memory by applications to run the programs in those libraries) and you can have scripts that can be provided as parameter to an interpreter. In both cases, the files only need to be readable, either by applications or by script interpreters. The executable attribute is only relevant when a file is the program container provided in an `exec` system call (which loads a program from a file to the virtual memory of the current process and starts its execution). This system call rejects the execution of a program provided in a not-executable file.

The way Linux handles the execution of a program stored in a file depends on the so-called file's magic number. This is a value in the first bytes of the file that provides some indication about its contents (file extensions are disregarded by the Linux kernel). More information about magic numbers can be obtained with the command `man magic`

The command `type` gives an explanation of the contents of a given file given its magic number. Note, however, that many files do not need to have magic numbers.

Executable scripts need to provide to the Linux kernel an indication about which script interpreter should be used to run their program. To do so, script must have an initial line starting with `#!`, followed by the path to script interpreter (which can be any binary executable file) and some command line parameters. The script file will be provided to the interpreter as its standard input.

Linux minimal ACL

The Linux minimal file ACL is the one that was initially designed for UNIX systems. It contains 3 parts, each one specifying the access rights for a given entity.

Each file has a owner, and its User Identifier (UID) is the entity for which the first set of access rights are specified.

Then, the ACL specifies the access rights for a (owner) group, specified by its Group Identifier (GID).

There is no need to have any relationshipt between this group and the file owner, they can be set arbitrarily. You can even set the owner and the group of a file to numbers not assigned to anyone in the system.

Finally, the ACL specifies the access rights for others than the owner and the group.

| — | – | – | | owner (UID) | group (GID) | others | | `rwx` | `rwx` | `rwx` |

The command `ls -la` allows observing minimal file ACLs of a given file or of a set of files in a directory. Access rights absent from each of the 3 ACL parts is indicated by a hyphen (`-`). For example, `rwxr-xr--` means that the file owner has read, write and execute rights, the file group has read and execute rights, and the others have only read rights.

Access granting decision

Processes are contexts to running application. Those applications act on behalf of a given user (provided by the process' effective UID) and on behalf of a given set of groups (provided by the process' list of groups). When an application requires the Linux kernel a specific kind of access to a file, that access requires first opening the file with a set of required access rights. The file's ACL is then checked as follows:

- If the process effective UID is equal to the file owner, its ACL part is checked for the required access right. If present, the file opening proceeds; otherwise, is denied.
- If on of the process' GIDs is equal to the file's (owner) group, its ACL part is checked for the required access right. If present, the file opening proceeds; otherwise, is denied.
- the ACL part for others is checked for the required access right. If present, the file opening proceeds; otherwise, is denied.

Note that this is a `if ... elseif ... else` decision flow.

ACL manipulation

Usually file access rights can only be modified by its owner, using the command `chmod`. New access rights can be set, added or removed.

Setting access rights can be performed with a textual description. For example, the command

```
chmod u=rx,g=r file
```

sets the file protection to read and execute for the owner, read for the (owner) group, and leaves the currebt protection for the others, while the command

```
chmod u+rx,g+r file
```

adds those exact same access rights to the ACL, and the command

```
chmod u-rx,g-r file
```

removes them from the ACL.

A legacy mode using octal numbers (!) is also an alternative, but it sets all the ACL access rights. In this case, a number is formed by adding 4 for `r`, 2 for `w` and 1 for `x`. Thus, `754` is equivalent to `u=rwx,g=rx,o=r`. A leading octal number can also be used to specify the Set-UID bit (with value 4), the Set-GID bit (with value 2) and the Sticky bit (with value 1).

Linux POSIX file ACLs

Linux file systems usually support the so-called POSIX ACLs. These ACLs allow files(or directories) to have more entries than the 3 present in the minimal ACL.

POSIX ACLs use exactely the same set of access rights (read, write and execute) but allow files to specify access rights for more users and more groups.

POSIX ACLS allow file ACLs to specify an arbitrary set of user entries and group entries. The former complement the owner entry, while the latter complement the (owner) group entry. These are called named entries.

However, these ACLs contain an extra entry that has no parallel in the old, minimal ACL: a right's mask. This mask can be used to deny arbitrary users and groups to have more rights than the file owner and its (owner) group. For instance, if the mask entry specifies read and execute rights, no individual user or group specified in the extended ACL can benefit from a write right.

Access granting decision

The access checking algorithm follows the logic behind the algorithm used with the minimal ACL:

- First is checked the ACL's owner UID entry with the process' effective UID. On a match, that entry is selected.
- Then are successively checked the ACL's named user entries with the process' effective UID. On a match, that entry is selected.
- Then is checked the ACL's (owner) group entry with all the process' GID. On a match, and provided that the entry grants the intended access, that entry is selected
- Then is checked the ACL's named group entries with all the process' GID. On a match, and provided that the entry grants the intended access, that entry is selected
- If one of the process' GIDs matches a group entry in the ACL, either the owner or any named one, the access is denied.
- Finally, the others' entry is selected.

Upon selecting an entry, that entry will then be used to check if the intended access is granted or not. If the entry belongs to a named entity (user or group), that entry is first processed with the mask, in order no to give more rights than the ones specified in that mask. An absent mask means that this processing is skipped.

Extended ACL manipulation

Extended ACLs can be manipulated with the command `setfacl` and observed with the command `getfacl`. When a directory or file is listed with commands such as `ls`, which only show the minimal ACL, a plus (`+`) sign at the end of the minimal ACL indicates the presence of an extended one.

Default ACLs

File system directories can be used to set default ACLs for files which (first) name is stored in that directory. In that case, if a directory contains default ACL entries, those are used as the starting point for defining the extended ACL of new files. However, extended ACL entries specifically crafted for a file override default ACL entries for the same subject (user or group). In other words, a specific ACL entry for a subject overrides a default ACL entry for that same subject.

Experiments with the minimal ACL

Create a file in the current directory using the command `cat > stuff` where `stuff` is the name of the file you want to create. Write the file contents (don't edit!) and finish with `Ctr-D` (Unix code end of file).

Ownership and protections

Check the protection of the file `stuff` with the command `ls -la` or `ls -la stuff` Check which user owns the file, what group owns it, and what file protections for these two entities and for others.

Effect of read protection

Print the contents of the file with the command `cat stuff`

Remove the `read` right from the user who owns the file `stuff` using the command `chmod u-r stuff`

Print the contents of the file again with `cat stuff`

and check the impossibility of doing so. Explain why.

Restore the `read` rights of the file owner with `chmod u+r stuff`

Effect of write protection

Append content to the `stuff` file with the command `cat >> stuff`

Type the extra contents of the file (don't edit!) and finish with `Ctr-D`. Print the contents of the file with

`cat stuff`

Remove the `write` rights from the user who owns that file with `chmod u-w stuff`

Reappend content to that file with

`cat >> stuff`

and check the impossibility of doing so. Explain why.

Restore the `write` rights of the file owner with `chmod u+w stuff`

Execution protection effect

Copy the `ls` command to the current directory with `cp /bin/ls myls` The copy was named `myls`.

Run the application `myls` with `./myls -la`

Check that its result is equal to what happens with the command `ls`. Remove the execute right from the user that owns `myls` with `chmod u-x myls`

Run `myls` again by doing `./myls -la` Check the impossibility of doing so. Explain why.

Restore the execution rights of the `myls` owner with `chmod u+x myls` and run it again.

Experiments with extended ACLs

Default extended ACLs

Print the default extended ACL of the current directory with `getfacl .`

You can see that you only have entries corresponding to the minimal ACL.

Create a default extended ACL entry in the current directory for a specific and existing user with `setfacl -m d:u:user:rwx` where the `-m` flag means an ACL modification, `d:u` mean a default user entry, `user` is the intended user entry and `rwx` are the intended rights.

Print again the directory's ACL and check what as changed. As you can see, now there is a default ACL formed by 5 entries: 3 derived from the minimal ACL, the named one that you just added and a mask entry.

Create a new file (e.g. `new_stuff`) and check its ACL. See that it inherited the default extended ACL from its directory. Check as well the effect of the mask in the effective entries displayed.

Remove the default mask entry from the file's extended ACL with the command

`setfacl -x d:m new_stuff`

and check the file's extended ACL again. See that a new mask (not a default one) is now in place. Now, if you change the current directory's ACL default mask, it will not influence the file's mask entry.

Specific file extended ACL

Remove the new file and flush the current directory extended ACL with

`setfacl -b .`

Check that it was cleared. Create again the same file and verify that it has no extended ACL.

Add new extended ACL entries, both with existing users/groups and unknown ones (providing their number instead of their name). Use a command such as

`setfacl -m u:user:rights filename`

to add a `user` entry or

`setfacl -m g:group:rights filename`

to add a `group` entry.

Check the presence and the effect of the mask entry. Change the mask to different values and check its effect in the effective access rights granted.

Bibliography

- Andreas Grünbacher, POSIX Access Control Lists on Linux

2024

PREVIOUS

Lecture 2 - OAuth 2.0

NEXT

Project - Flexible, Risk Aware Authentication System

Last updated on 16 Feb 2024