Operating Systems Security – Assignment 4

Version 1.0 – 2016/2017
Due Date: 09 Dec 2016 (23:59 CET)

Institute for Computing and Information Sciences,
Radboud University, The Netherlands.

1 Access Control Lists (ACLs) in Linux

Background[1]

The default Linux file permission system only associates a single group with a file. This poses some limitations as there are only three possibilities for the permissions that can be assigned to a file. In most situations this is sufficient, however, it can be a limitation. It also means that should a file need to made accessible to various group of users which already exist, it may be required to create a new group specifically to grant these rights.

Alternatives, such as Access Control Lists (ACLs) offer more granularity. The ACL functionality gives a user the ability to, among other things, grant file permissions on a user-by-user basis. So, for example, you can create a file that is readable by *joeuser* and *janeuser* but only writable by *janeuser*. ACLs provide a much higher degree of control over permissions than standard Unix groups. In addition, they are completely under the control of the owner of the file. You don't need the system administrator to create and maintain groups for you.

Basic ACL Commands and Operations

The two main commands you will use to manipulate ACLs are **`setfacl`** and **`getfacl`**. For example, use the **`setfacl`** command to grant read-only access to user *joe* to the file named **`hello.c`**:
$ **`setfacl -m user:joe:r-- hello.c`** and read-write access to user *jane* with:
$ **`setfacl -m user:jane:rw- hello.c`**
After setting the ACL on the file, note that **`ls`** shows a '**`+`**' after the normal permission list:

```
$ ls -l hello.c
-rw-r--r--+ 1 root root 0 Nov 25 10:00 hello.c
```

The '**`+`**' signifies that there is an ACL set for the file. You can then use **`getfacl`** to display the ACL for the file:

```
$ getfacl hello.c
# file: hello.c
# owner: root
# group: root
user::rw-
user:joe:r--
user:jane:rw-
group::r--
mask::r--
other::r--
```

[1] http://svn.gna.org/svn/thepilots/trunk/Bronze2Wkbk.odt

The ACL shows that user *joe* has read access and user *jane* has read/write access. Once you have an ACL set on one file, you can duplicate this ACL for other files by creating **`rules.acl`**
$ **`getfacl hello.c > rules.acl`** and use this file to set the ACL of other files:
$ **`setfacl -M rules.acl goodbye.c`**
For information about ACL for more advanced configurations like user and group restrictions on directories, please refer one of the following websites[2][3][4]

Objectives

Add five test users (**`test1...test5`**), two directories (**`dir1,dir2`**) and six files (**`file1...file6`**) to your (Kali) Linux system. Put three files into each of the directories and define the following rules.

a) **`test1`** has read and write access to all directories and files.
b) **`test2`** has only read access to all **`dir1`** and the files in there.
c) **`test3`** has the same restrictions as user **`test2`**, but can write to one file and execute another file in **`dir1`**
d) **`test4`** has only write access to **`dir2`** (can create new files), but cannot write to files initially stored in **`dir2`**.
e) **`test5`** cannot list the directories **`dir1`** and **`dir2`** but can read, write and execute all files stored in them (assume he knows their exact storage paths).

Hand in the rules you have to compose during this exercise.

2 Bell-LaPadula & Biba Model

Consider a reference monitor which uses Mandatory Access Control (MAC) to implement the Bell-LaPadula and the Biba model. The Bell-LaPadula model uses levels **unclassified** $\leq$ **confidential** $\leq$ **secret** $\leq$ **top secret**. The Biba model uses levels **untrusted** $\leq$ **user** $\leq$ **application** $\leq$ **system** $\leq$ **trusted**. You may assume **weak tranquility** is used. Consider further the following objects with corresponding secrecy and trust levels:

- `/home/me/mydata` (confidential, user),
- `/etc/fstab` (confidential, trusted)
- `/usr/bin/myprog` (unclassified, application)
- `/usr/lib/mylib.so` (unclassified, system)
- Network socket to 190.93.240.15, port 80 (unclassified, untrusted)

For each of the following consecutive steps determine whether the reference monitor will allow the action and explain why or why not (if there are multiple reasons, state all). If an action is not allowed, assume that the process simply continues.

1. User `me` logs in with clearance (secret, application) and tries to run `/usr/bin/myprog`.
2. The process dynamically loads (reads) `/usr/lib/mylib.so`.
3. The process reads `/home/me/mydata`.
4. The process writes data to the network socket.
5. The process reads `/etc/fstab`.

[2] http://www.cs.indiana.edu/csg/FAQ/General/ACL.html
[3] https://wiki.archlinux.org/index.php/Access_Control_Lists
[4] http://linuxcommando.blogspot.nl/2008/01/part-2-how-to-work-with-access-control.html