Operating Systems Security – Assignment 2

Version 1.0.0 – 2014/2015

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

1 Introduction to Access Control Lists (ACLs) in Linux

Background[1]

The default Linux file permission system only associates a single group with a file. This has limitations in that there are only three possibilities for the permissions for 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 groups of user 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 14 08:01 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 directorie and define the following rules.

a) **test1** has a access to all directories and files.
b) **test2** has a 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** can not list the directories **dir1** and **dir2** but can read, write and execute all files stored in them (suggest he knows their exact storage path).

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

2 Play around with suid bit

Login to your (Kali) Linux system as a **non**-root user and download the program **showdate** from http://cryptojedi.org/peter/teaching/ossec2014/showdate.
Then, change the owner
$ **sudo chown root:root showdate**
and set the suid bit
$ **sudo chmod u+s showdate**
Execute the program and verify it prints the date correctly
$ **./showdate**
`Tue Nov 18 07:44:23 EST 2014`

Objectives

a) Find out what the program does internally. What system calls does it use?
b) Assume the role of a non-privileged attacker. Use the program `showdate` to obtain a root shell. You can verify if you succeeded by looking at the output of **id**, it should be something like:
$ **/usr/bin/id**
`uid=0(root) gid=0(root) groups=0(root),27(sudo),1001(test1)`
Hand in the exact console commands you used to get this working.
c) Explain what a developer could do to overcome this issue. What explicit actions should a developer take when writing software that is intended to be used with setuid-root to avoid these kinds of problems?

[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

3 Compile and load your own Linux kernel module

Login to your (Kali) Linux system as a **root** user and compile the program **cr4.c**:

```
#include <stdio.h>

void main() {
   unsigned long long result;
   __asm__("movq %%cr4, %%rax\n" : "=a"(result));
   printf("Value of CR4 = %llx\n", result);
}
```

with the command line:
$ **gcc -o cr4 cr4.c**

Notice that executing will result in a exception:
$ **./cr4**
`Segmentation fault`

Using a debugger we can quickly pinpoint what the problem is. Start debugger in assembly mode
$ **gdb -ex "layout asm" ./cr4**
and execute it using the following **GDB** instruction
$ **run**

Objectives

a) Figure out where the register **CR4** is used for and report back why you think it should not be accessible in user mode[5].

b) Figure out which exact assembly instruction of **cr4.c** triggers the segmentation fault and briefly write down what it tries to do.

c) Follow the *"How to Write Your Own Linux Kernel Module with a Simple Example"* guide hosted at this website[6] and try to reproduce their results. You should be able to see your kernel module output with the following command:
$ **dmesg | tail -10**

If your kernel module is working correctly, try to adjust the kernel module to read out the exact same **CR4** register. Hand in the source-code of your kernel module together with a Makefile to build it and report back which value the **CR4** in your (Kali) Linux system has.

[5] http://en.wikipedia.org/wiki/Control_register
[6] http://www.thegeekstuff.com/2013/07/write-linux-kernel-module/