Operating Systems Security – Assignment 2

Version 1.2 – 2015/2016

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

1 Play around with `suid` bit

Login to your (Kali) Linux system as a **non**-root user and download the program **`showdate`** from `https://cryptojedi.org/peter/teaching/ossec2015/showdate` (for 64-bit OS) and `https://cryptojedi.org/peter/teaching/ossec2015/showdate32` (for 32-bit OS).
Then, change the owner

```
$ sudo chown root:root showdate
```

set the suid bit and make it executable

```
$ sudo chmod u+s,a+x showdate
```

Execute the program and verify it prints the date correctly

```
$ ./showdate
Wed Nov 18 xx:xx:xx EST 2015
```

Install the tool **`strace`**

```
$ sudo apt-get install strace
```

and run it to see system calls used by **`showdate`**

```
$ strace -f ./showdate
```

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 types of problems?

2 Compile and load your own Linux kernel module

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

```c
#include <stdio.h>

void main() {
   unsigned long long result;
   /*unsigned long result; (for 32-bit OS)*/
   __asm__("movq %%cr4, %%rax\n" : "=a"(result));
   /*__asm__("mov %%cr4, %%eax\n" : "=a"(result)); (for 32-bit OS)*/
   printf("Value of CR4 = %llx\n", result);
}
```

with the command line:

```
# gcc -o cr4 cr4.c
```

Notice that executing will result in an 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[1].

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[2] and try to reproduce their results. You should be able to see your kernel module output with the following command:

```
$ dmesg | tail -10
```

d) 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.

3 Write your own PAM module

In Assignment 1, you learned about Pluggable Authentication Modules (PAM). In this section, you are required to write a basic custom PAM module which asks a user 1 out of 5 questions randomly and the user is required to provide the correct answer. You are free to be as creative as you like with these 5 questions.
We advise you to execute `sudo apt-get install libpam0g-dev` and test your module using `su` (and not `login` or `ssh`) .
You need to hand the source code of the module together with a Makefile to build it and a config file `/etc/pam.d/su` that uses the module for authentication.
Note: For additional background knowledge about PAM, please refer to the following websites [3][4][5]

Additional Exercise: Buffer-overflow attack

This is **not** a mandatory exercise for the 'Operating Systems Security' course and is only meant to serve as a refresher for those who have not done the 'Software and Web Security' course. You are strongly recommended to complete this task as it serves as a prerequisite to better understand the lecture on *Memory* (Lecture 3).
The exercise can be found here:
http://www.cs.ru.nl/~erikpoll/sws1/exercises/assignment5b.pdf

[1] http://en.wikipedia.org/wiki/Control_register
[2] http://www.thegeekstuff.com/2013/07/write-linux-kernel-module/
[3] http://www.linux-pam.org/Linux-PAM-html/Linux-PAM_SAG.html
[4] http://www.rkeene.org/projects/info/wiki/222
[5] http://www.wpollock.com/AUnix2/PAM-Help.htm