Linux Capability Exploration Lab

The development of this document is funded by the National Science Foundation's Course, Curriculum, and Laboratory Improvement (CCLI) program under Award No. 0618680 and 0231122.

1 Lab Description

The learning objective of this lab is for students to gain first-hand experiences on capability, to appreciate the advantage of capabilities in access control, to master how to use capability in to achieve the principle of least privileges, and to analyze the design of the capability-based access control in `Linux`. This lab is based on POSIX 1.e capability, which is implemented in recent versions of `Linux` kernel.

2 Lab Setup

The lab was developed based on Ubuntu 9, which uses `Linux` kernel version `2.6.28`. Some of the features involved in this lab are not available before the kernel version `2.6.24`.

2.1 Install Libcap

There are several ways for user-level programs to interact with the capability features in `Linux`; the most convenient way is to use the `libcap` library, which is now the standard library for the capability-related programming. This library does not come with some `Linux` distributions, so you need to download and install it. If you already have the file `/usr/include/sys/capability.h`, then the `libcap` library has already been installed. If the header file is not there, install the library using the following commands:

```
# apt-get install wget (use "yum install wget" for Fedora)
# cd dir_name  (assume you want to put the libcap library in dir_name)
# wget http://www.kernel.org/pub/linux/libs/security/linux-privs/
              libcap2/libcap-2.16.tar.gz
# tar xvf libcap-2.16.tar.gz
# cd libcap-2.16
# make   (this will compile libcap)
# make install
```

Note: If you are using our pre-built Ubuntu Virtual Machine, `libcap 2.16` is already installed. If you use Fedora 9 for this lab you may want to use older version of `libcap`
For this lab, you need to get familiar with the following commands that come with `libcap`:

- `setcap`: assign capabilities to a file.
- `getcap`: display the capabilities that carried by a file.
- `getpcaps`: display the capabilities carried by a process.

2.2 Put SELinux in Permissive Mode

Ubuntu 9 doesn't come with SELinux. Skip this section if your Linux doesn't have SELinux. However, recent versions of Fedora come with SELinux. Unfortunately, SELinux will be in our way, preventing us from doing some of the activities in this lab. We need to put SELinux to permissive mode for this lab.

To temporarily put SELinux to permissive mode, issue 'setenforce 0' as root. To make permissive mode as a startup mode, you need to modify /etc/selinux/config by changing the line 'SELINUX=enforcing' to 'SELINUX=permissive'. Note: do not disable SELinux (only temporarily put it in the permissive mode), otherwise when you enable the SELinux next time, the OS will take time to re-lable the file system for the SELinux context during the boot time.

3 Lab Tasks

In a capability system, when a program is executed, its corresponding process is initialized with a list of capabilities (tokens). When the process tries to access an object, the operating system check the process' capabilities, and decides whether to grant the access or not.

3.1 Task 1: Experiencing Capabilities

In operating systems, there are many privileged operations that can only be conducted by privileged users. Examples of privilegd operations include configuring network interface card, backing up all the user files, shutting down the computers, etc. Without capabilities, these operations can only be carried out by superusers, who often have many more privileges than what are needed for the intended tasks. Therefore, letting superusers to conduct these privileged operations is a violation of the *Least-Privilege Principle*.

Privileged operations are very necessary in operating systems. All Set-UID programs invole privileged operations that cannot be performed by normal users. To allow normal users to run these programs, Set-UID programs turn normal users into powerful users (e.g. root) temporarily, even though that the involved privileged operations do not need all the power. This is dangerous: if the program is compromised, adversaries might get the root privilege.

Capabilities divide the powerful root privilege into a set of less powerful privileges. Each of these privileges is called a capability. With capabilities, we do not need to be a superuser to conduct privileged operations. All we need is to have the capabilities that are needed for the privileged operations. Therefore, even if a privileged program is compromised, adversaries can only get limited power. This way, risk of privileged program can be lowered quite significantly.

Capabilities has been implemented in Linux for quite some time, but they could only be assigned to processes. Since kernel version 2.6.24, capabilities can be assigned to files (i.e., programs) and turn those programs into privileged programs. When a privileged program is executed, the running process will carry those capabilities that are assigned to the program. In some sense, this is similar to the Set-UID files, but the major difference is the amount of privileged carried by the running processes.

We will use an example to show how capabilities can be used to remove unnecessary power assigned to certain privileged programs. First, let us login as a normal user, and run the following command:

```
% ping www.google.com
```

The program should run successfully. If you look at the file attribute of the program /bin/ping, you will find out that ping is actually a Set-UID program with the owner being root, i.e., when you execute ping, your effective user id becomes root, and the running process is very powerful. If there are

vulnerabilities in `ping`, the entire system can be compromised. The question is whether we can remove these privileged from `ping`.

Let us turn `/bin/ping` into a non-`Set-UID` program. This can be done via the following command (you need to login as the root):

```
# chmod u-s /bin/ping
```

Note: Binary files like `ping` may locate in different places in different distribution of Linux, use `'which ping'` to locate your `ping` program.

Now, run `'ping www.google.com'`, and see what happens. Interestingly, the command will not work. This is because `ping` needs to open RAW socket, which is a privileged operation that can only be conducted by root (before capabilities are implemented). That is why `ping` has to be a `Set-UID` program. With capability, we do not need to give too much power to `ping`. Let us only assign the `cap_net_raw` capability to `ping`, and see what happens:

```
$ su root
# setcap cap_net_raw=ep /bin/ping
# su normal_user
$ ping www.google.com
```

Question 1: Please turn the following `Set-UID` programs into non-`Set-UID` programs, without affecting the behaviors of these programs.

- `/usr/bin/passwd`

Question 2: You have seen what we can do with the `cap_net_raw` capability. We would like you to get familiar with several other capabilities. For each of the following capabilities, do the following: (1) explain the purpose of this capability; (2) find a program to demonstrate the effect of these capabilities (you can run the application with and without the capability, and explain the difference in the results). You can also write your own applications if you prefer, as long as they can demonstrate the effect of the capability. Here is the list of capabilities that you need to work on (read `include/linux/capability.h` to learn about the capabilities).

- cap_dac_read_search
- cap_dac_override
- cap_fowner
- cap_chown
- cap_fsetid
- cap_sys_module
- cap_kill
- cap_net_admin

- cap_net_raw
- cap_sys_nice
- cap_sys_time

3.2 Task 2: Adjusting Privileges

Compared to the access control using ACL (Access Control List), capabilities has another advantage: it is quite convenient to dynamically adjust the amount of privileges a process has, which is essential for achieve the principle of least privilege. For example, when a privilege is no longer needed in a process, we should allow the process to permanently remove the capabilities relevant to this privilege. Therefore, even if the process is compromised, attackers will not be able to gain these deleted capabilities. Adjusting privileges can be achieved using the following capability management operations.

1. *Deleting:* A process can permanently delete a capability.
2. *Disabling:* A process can temporarily disable a capability. Unlike deleting, disabling is only temporary; the process can later enable it.
3. *Enabling:* A process can enable a capability that is temporarily disabled. A deleted capability cannot be enabled.

Without capabilities, a privileged `Set-UID` program can also delete/disable/enable its own privileged. This is done via the `setuid()` and `seteuid()` system calls; namely, a process can change its effective user id during the run time. The granularity is quite coarse using these system calls, because you can either be the privileged users (e.g. root) or a non-privileged users. With capabilities, the privileges can be adjusted in a much finer fashion, because each capability can be independently adjusted.

To support dynamic capability adjustment, `Linux` uses a mechanism similar to the `Set-UID` mechanism, i.e., a process carries three capability sets: permitted (P), inheritable (I), and effective (E). The permitted set consists of the capabilities that the process is permitted to use; however, this set of capabilities might not be active. The effective set consists of those capabilities that the process can currently use (this is like the effective user uid in the `Set-UID` mechanism). The effective set must always be a subset of the permitted set. The process can change the contents of the effective set at any time as long as the effective set does not exceed the permitted set. The inheritable set is used only for calculating the new capability sets after `exec()`, i.e., which capabilities can be inherited by the children processes.

When a process forks, the child's capability sets are copied from the parent. When a process executes a new program, its new capability sets are calculated according to the following formula:

```
pI_new = pI
pP_new = fP | (fI & pI)
pE_new = pP_new   if fE = true
pE_new = empty    if fE = false
```

A value ending with "new" indicates the newly calculated value. A value beginning with a p indicates a process capability. A value beginning with an f indicates a file capability.

To make it convenient for programs to disable/enable/delete their capabilities, please add the following three functions to `libcap-2.16/libcap/cap_proc.c` (`libcap-2.16` is the directory created when you run `'tar xvf libcap-2.16.tar.gz'` to extract the libcap package).

```
int cap_disable(cap_value_t capflag)
{
        cap_t mycaps;

        mycaps = cap_get_proc();
        if (mycaps == NULL)
                return -1;
        if (cap_set_flag(mycaps, CAP_EFFECTIVE, 1, &capflag, CAP_CLEAR) != 0)
                return -1;
        if (cap_set_proc(mycaps) != 0)
                return -1;
        return 0;
}

int cap_enable(cap_value_t capflag)
{
        cap_t mycaps;

        mycaps = cap_get_proc();
        if (mycaps == NULL)
                return -1;
        if (cap_set_flag(mycaps, CAP_EFFECTIVE, 1, &capflag, CAP_SET) != 0)
                return -1;
        if (cap_set_proc(mycaps) != 0)
                return -1;
        return 0;
}

int cap_drop(cap_value_t capflag)
{
        cap_t mycaps;

        mycaps = cap_get_proc();
        if (mycaps == NULL)
                return -1;
        if (cap_set_flag(mycaps, CAP_EFFECTIVE, 1, &capflag, CAP_CLEAR) != 0)
                return -1;
        if (cap_set_flag(mycaps, CAP_PERMITTED, 1, &capflag, CAP_CLEAR) != 0)
                return -1;
        if (cap_set_proc(mycaps) != 0)
                return -1;
        return 0;
}
```

Run the following command to compile and install the updated `libcap`. After the library is installed, programs can use these three library functions that we have just added.

```
# cd libcap_directory
# make
# make install
```

Question 3: Compile the following program, and assign the `cap_dac_read_search` capability to the executable. Login as a normal user and run the program. Describe and explain your observations.

```
/* use_cap.c */
#include <fcntl.h>
#include <sys/types.h>
#include <errno.h>
#include <stdlib.h>
#include <stdio.h>
#include <linux/capability.h>
#include <sys/capability.h>
int main(void)
{
        if (open ("/etc/shadow", O_RDONLY) < 0)
            printf("(a) Open failed\n");
        /* Question (a): is the above open sucessful? why? */

        if (cap_disable(CAP_DAC_READ_SEARCH) < 0) return -1;

        if (open ("/etc/shadow", O_RDONLY) < 0)
            printf("(b) Open failed\n");
        /* Question (b): is the above open sucessful? why? */

        if (cap_enable(CAP_DAC_READ_SEARCH) < 0) return -1;

        if (open ("/etc/shadow", O_RDONLY) < 0)
            printf("(c) Open failed\n");
        /* Question (c): is the above open sucessful? why?*/

        if (cap_drop(CAP_DAC_READ_SEARCH) < 0) return -1;

        if (open ("/etc/shadow", O_RDONLY) < 0)
            printf("(d) Open failed\n");
        /* Question (d): is the above open sucessful? why?*/

        if (cap_enable(CAP_DAC_READ_SEARCH) == 0) return -1;

        if (open ("/etc/shadow", O_RDONLY) < 0)
            printf("(e) Open failed\n");
        /* Question (e): is the above open sucessful? why?*/
}
```

The program can be compiled using the following command (note in the second command, the second character in "-lcap" is ell, not one; it means linking the `libcap` library):

```
$ gcc -c use_cap.c
$ gcc -o use_cap use_cap.o -lcap
```

After you finish the above task, please answer the following questions:

- **Question 4:** If we want to dynamically adjust the amount of privileges in ACL-based access control, what should we do? Compared to capabilities, which access control is more convenient to do so?
- **Question 5:** After a program (running as normal user) disables a capability A, it is compromised by a buffer-overflow attack. The attacker successfully injectes his malicious code into this program's stack space and starts to run it. Can this attacker use the capability A? What if the process deleted the capability, can the attacker uses the capability?

- **Question 6:** The same as the previous question, except replacing the buffer-overflow attack with the race condition attack. Namely, if the attacker exploites the race condition in this program, can he use the capability A if the capability is disabled? What if the capability is deleted?

4 Submission

You need to submit a detailed lab report to describe what you have done and what you have observed; you also need to provide explanation to the observations that are interesting or surprising. In your report, you need to answer all the questions listed in this lab.