Linux Capability Exploration Lab

The development of this document is/was funded by the following grants from the US National Science Foundation: No. 1303306 and 1318814. This lab was imported into the Labtainer framework by the Naval Postgraduate School, Center for Cybersecurity and Cyber Operations under National Science Foundation Award No. 1438893.

1 Lab Description

The learning objective of this lab is for students to gain first-hand experiences on the use of capabilities to achieve the principle of least privilege. This lab is based on POSIX 1.e capabilities, which is implemented in recent versions of `Linux` kernel.

Capability based systems are sometimes promoted as an access control strategy in contrast to the use of Access Control Lists (ACLs) or Unix file permissions. In practice, Linux systems typically use capabilities to limit program privilege rather than to control access to named objects. This lab focuses on the use of capabilities to limit privilege.

2 Lab Environment

This lab runs in the Labtainer framework, available at http://my.nps.edu/web/c3o/labtainers. That site includes links to a pre-built virtual machine that has Labtainers installed, however Labtainers can be run on any Linux host that supports Docker containers.

From your labtainer-student directory start the lab using:

```
labtainer capabilities
```

Links to this lab manual and to an empty lab report will be displayed. If you create your lab report on a separate system, be sure to copy it back to the specified location on your Linux system.

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.

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 checks the process' capabilities, and decides whether to grant the access or not.

3.1 Task 1: Experiencing Capabilities

In operating systems such a Linux, there are many privileged operations that can only be conducted by privileged users. Examples of privileged operations include configuring network interface cards, 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 more privileges than are needed for the intended tasks. Therefore, letting superusers perform these privileged operations is a violation of the *Least-Privilege Principle*.

Privileged operations are necessary in Linux and other Unix based operating systems. All `Set-UID` programs involve 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 the involved privileged operations do not need all privileges provided to superuser. This is dangerous: if the program is compromised, adversaries might get the root privilege.

Capabilities divide the root privilege into a set of distinct 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, the risks of privileged program can be reduced.

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 privileges assigned to certain privileged programs. First, as the unprivileged ubuntu user, run the following command:

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

The program should run successfully. If you look at the file attributes of the program `/bin/ping`, you will see that `ping` it is 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 therefore runs with root privileges. If there are vulnerabilities in `ping`, the entire system can be compromised. Using capabilities, we can remove unnecessary privileges from `ping`.

First, let us turn `/bin/ping` into a non-`Set-UID` program. This can be done via the following command:

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

Now, run `'ping www.google.com'`, and see what happens. The command should fail. This is because `ping` needs to open a 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. Let us only assign the `cap_net_raw` capability to `ping`, and see what happens:

```
sudo setcap cap_net_raw=ep /bin/ping
ping www.google.com
```

3.1.1 Task 1.1 Allow unprivileged users to run tcpdump

In addition to reducing privileges associated with setuid programs, capabilities can be used to allow unprivileged users to run selected programs without granting those users sudo privileges. A common example is the

/usr/bin/tcpdump program. In other Labtainer exercises, we use tcpdump to capture network traffic, however we do that but running

```
sudo tcpdump
```

Modify the tcpdump program so that an unprivileged user can run it.

3.1.2 Task 1.2 Convert passwd to use capabilities

Some setuid programs require several different capabilities to operate. The file: /usr/include/linux/capability.h describes the various capabilities available in Linux.

The /usr/bin/passwd program requires the following capabilities to operate:

```
cap_chown
cap_dac_override
cap_fowner
```

Modify the passwd program to use capabilities instead of setuid, then demonstrate that it still works by changing the ubuntu user password (which initially is ubuntu).

3.2 Task 2: Adjusting Privileges

With capabilities, it is possible to dynamically adjust the amount of privileges a process has, which is consistent with 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.

Review the `mypriv.c` program in the home directory. Answer the questions embedded in the source code of the program. Then use the `build.sh` script to compile and link the program. Then use `setcap` to set the `CAP_DAC_READ_SEARCH` capability, and run the program. Compare the results to what you expected.

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

- **Question 1:** After a program (running as normal user) disables a capability A, it is compromised by a buffer-overflow attack. The attacker successfully injects 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 2:** The same as the previous question, except replacing the buffer-overflow attack with the race condition attack. Namely, if the attacker exploits 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.

After finishing the lab, go to the terminal on your Linux system that was used to start the lab and type:

```
stoplab capabilities
```

When you stop the lab, the system will display a path to the zipped lab results on your Linux system. Provide that file to your instructor, e.g., via the Sakai site.