

# Capability Lab

Copyright © 2006 Wenliang Du, Syracuse University.

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. Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.2 or any later version published by the Free Software Foundation. A copy of the license can be found at <http://www.gnu.org/licenses/fdl.html>.

## 1 Lab Description

The learning objective of this lab is for students to apply the capability concept to enhance system security. In Unix, there are a number of privileged programs (e.g., Set-UID programs); when these programs are run, even by normal users, they run as `root` (i.e., system administrator); namely the running programs possess all the privileges that the `root` has, despite of the fact that not all of these privileges are actually needed for the intended tasks. This design clearly violates an essential security engineering principle, the *principle of least privilege*. As a consequence of the violation, if there are vulnerabilities in these programs, attackers might be able to exploit the vulnerabilities and abuse the `root`'s privileges.

Capability can be used to replace the Set-UID mechanism. In Trusted Solaris 8, `root`'s privileges are divided into 80 smaller capabilities. Each privileged program is only assigned the capabilities that are necessary, rather than given the `root` privilege. A similar capability system is also developed in Linux. In this lab, we will implement a simplified capability system for Minix.

## 2 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 should check the process' capability, and decides whether to grant the access or not.

### 2.1 Required Capabilities (60 points)

To make this lab accomplishable within a short period of time, we have only defined 5 capabilities. Due to our simplification, these five capabilities do not cover all of the `root`'s privileges, so they cannot totally replace Set-UID. They can only be used for privileged programs that just need a subset of our defined capabilities. For those programs, they do not need to be configured as a Set-UID program; instead, they can use our capability system. Here are the capabilities that you need to implement in this lab:

1. CAP\_READ: Allow read on files and directories. It overrides the ACL restrictions regarding read on files and directories.
2. CAP\_CHOWN: Overrides the restriction of changing file ownership and group ownership.
3. CAP\_SETUID: Allow to change the effective user to another user. Recall that when the effective user id is not `root`, callings of `setuid()` and `seteuid()` to change effective users are subject to certain restrictions. This capability overrides those restrictions.

4. CAP\_KILL: Allow killing of any process. It overrides the restriction that the real or effective user ID of a process sending a signal must match the real or effective user ID of the process receiving the signal.
5. CAP\_SYS\_BOOT: Allow rebooting the system.

A command should be implemented for the superuser to assign capabilities to (or remove capabilities from) a program. It should be noted that the above five capabilities are independent; if a capability is not assigned to a program, the program cannot gain this capabilities from other capabilities. For example, if a program has only the CAP\_SETUID capability, it should not be able to use this capability to gain any of the other capabilities. You should be warned that the above description of capabilities was *intentionally* made vague and incomplete, such that a design that exactly follows the description can have loopholes. It is your responsibility to clarify and complete the description. If you think that it is necessary to add restrictions to these capabilities to avoid loopholes, you should feel free to do that; in your report and demonstration, you need to justify your decisions.

## 2.2 Managing Capabilities (40 points)

We should also allow a process to manage its own capabilities. For example, when a capability is no longer needed in a process, we should allow the process to permanently remove this capability. Therefore, even if the process is compromised, attackers will not be able to gain this deleted capability. The following six operations are general capability management operations; you need to implement them in your capability system.

1. Deleting: A process can permanently delete a capability.
2. Disabling: A process can temporarily disable a capability. Note that unlike deleting, disabling is only temporary; the process can later enable it.
3. Enabling: A process can enable a capability that is temporarily disabled.
4. Copying: A process can give its own capabilities to its children processes.
5. Copy-control mechanism: The owner of a capability can control whether the receiver can make another copy or not.
6. (10 bonus points) Revocation: The owner of a capability can revoke the capability from all of its children processes.

## 3 Design and Implementation Issues

In this lab, you need to make a number of design choices. Your choices should be justified, and the justification should be included in your lab report.

### 3.1 Assigning Capability to Programs

Before a program becomes a privileged program, certain capabilities need to be assigned to this program. You need to consider the following issues related to capability assignment.

- Where should the capabilities of a program be stored? There are several ways to store capabilities. You need to justify your design decision. You can justify it from various aspects, such as security, usability, ease of use, etc. To help you, we list two possible methods in the following:
  - Save capabilities in a configuration file.
  - Save capabilities in the `I-nodes` of the program file.
- How can users set capabilities of a file?
- Who can assign capabilities to programs?

### 3.2 Capability in Process

When a program is executed, a process will be created to perform the execution. The process should carry the capability information. You need to consider the following issues related to processes:

- Where do you store capabilities? They can be stored in kernel space (e.g., capability list), in user space (e.g., cryptographic token), or in both spaces (like the implementation of file descriptor, where the actual capabilities are stored in the kernel and the indices to the capabilities are copied to the user space). Which design do you use? You should justify your decisions in your lab reports.
- You need to study the process-related data structures. They are defined in three places: file system (`/usr/src/fs`), memory management (`/usr/src/mm`), and kernel (`/usr/src/kernel`).
- How do you assign capability to a newly created process?
- When system boots up, a number of processes (e.g. file system process and memory management process) will be created; do they need to carry capabilities?

### 3.3 Use Capabilities for Access Control

When a process tries to access an object, the operating system checks the process' capability, and decides whether to grant the access or not. The following issues will give you some hints on how to design and implement such an access control system.

- To check capabilities, you need to modify a number of places in `Minix` kernel. Be very careful not to miss any place; otherwise you will have a loophole in your system. Please describe these places and your justification in your lab report.
- Where do you check capabilities? You should think about applying the *reference monitor* principle here.
- The capability implemented in this lab co-exists with the `Minix`'s existing ACL access control mechanism. How do you deal with their relationship? For example, if a process has the required capability, but ACL denies the access, should the access be allowed? On the other hand, if a process does not have the required capability, but ACL allows the access, should the access be allowed? In your lab report, you should draw a diagram to depict the relationship between your capability-checking module and the ACL-checking module.

- *Compatibility issue:* Keep in mind that there will be processes (especially those created during the bootup) that are not capability-enabled. The addition of capability mechanism will cause them not to work properly, because they do not carry any capability at all. You need to find a solution to make your capability system compatible with those processes.

### 3.4 Helpful Documents

We have linked several helpful documents to the lab web page. Make sure you read them, because they can save you a tremendous amount of time. These documents cover the following topics: (1) how to add new system calls? (2) how are system calls invoked? (3) process tables in the file system process and the memory management process.

**Very Important:** Please remember to backup a valid boot image before you make modifications; you might crash your systems quite often.

## 4 Submission and Demonstration

You should submit a detailed lab report to describe your design and implementation. You should also describe how you test the functionalities and security of your system. You also need to demonstrate your system to us. Please sign up a demonstration time slot with the TA. Please take the following into consideration when you prepare for demonstration:

- The total time of the demo will be 15 minutes, no more additional time would be given. So prepare your demonstration so you can cover the important features.
- You are entirely responsible for showing the demo. We will NOT even touch the keyboard during the demonstration; so you should not depend on us to test your system. If you fail to demo some important features of your system, we will assume that your system does not have those features.
- You need to practice before you come to the demonstration. If the system crashes or anything goes wrong, it is your own fault. We will not debug your problems, nor give you extra time for it.
- During the demo, you should consider yourself as salesmen, and you want to sell your system to us. You are given 15 minutes to show us how good your system is. So think about your sales strategies. If you have implemented a great system, but fail to show us how good it is, you are not likely to get a good grade.
- Do turn off the messages your system prints out for debugging purposes. Those messages should not appear in a demonstration.