Operating Systems Security II

Tom Kelliher, CS 325

Oct. 17, 2011

1 Administrivia

Announcements

Assignment

Read 4.5.

From Last Time

Outline

1. Introduction.
2. General object access control.
3. File protection mechanisms.
4. SELinux.

Coming Up

User authentication.

2 Introduction

1. An OS ought to provide a general protection mechanism for "objects."

2. Goals for such a mechanism:

   (a) Check every access — What if access has been revoked?

   (b) Enforce least privilege — Grant access to the minimal number of objects necessary for completing the task.

   Examples: restricted access to hs121ps; filesystem quotas; raw disk blocks.

   (c) Verify acceptable usage — Confirm that the operation applied to an object is appropriate according to the object's type.

   Examples: an attempt to delete a disk drive; executing a data file.

3. The difference between a "mechanism" and a "policy."

3 General Object Access Control

The basic questions:

1. Where do the permissions reside — with user or object?

2. The basic permissions: own, transfer, read, write, and execute.

3. Ease of adding a permission, removing a permission, removing all permissions.

4. Complexity of managing permissions.

5. Question: Can the user ever access the permissions directly?

3.1 Directories

1. Idea: Associate permissions with the user.

2. Think of the directory as a user-oriented list of what objects the user can access and how.

3.2 Access Control Lists

1. Now, the permissions reside with the object.

2. Associated with each object is a list of users permitted to access it and how.

3. Additionally, a default access may be specified. Implemented via a "wildcard" mechanism.

   Big improvement.

3.3 Capabilities

1. Capabilities are typically associated with the user.

   A set of "keys" carried by the user.

2. They might be derived from access control lists.

   Think of them as a cached form of the access control list, with a defined lifetime. For example, for this session.

3.4 Procedure-Oriented Access Control

This is the formalization of the notion of only allowing appropriate actions to be applied to an object.

4 File Protection Mechanisms

4.1 No Protection

The standard mechanism supplied by DOS and Windows.

4.2 Group Protection

1. The mechanism we love and hate; provided by Unix and Linux.

2. Three basic permissions: read, write, and execute.

3. Three classes of user: owner, group, and world.

4. The group class was added to allow sharing. For example, a project team. Issues with the group class:

   (a) How many groups can a user belong to at one time? HP-UX allowed only one at a time. A shell command allowed a user to change groups.

   If a user can only belong to one group **total**, a user would need multiple accounts if working on multiple projects.

   (b) If a file is created, what group is assigned to it? (The user's primary group.)

   (c) Root has to assign groups. Users can't create them on their own.

   (d) How well does the group class work with project groups.

   (Or, why does Tom assign group accounts?)

4.3 Password Assignment

1. Idea: assign a password in order to grant a set of permissions.

2. Problems:

(a) Loss of the password by the owner.

(b) Password theft. Assignment of new password and distribution to all legitimate users.

(c) Revocation. See password theft!!!

4.4 SUID and SGID

1. Allows a user to temporarily take on the permissions and power of another user or group.

2. Provides controlled access to objects.

3. Examples of SGID executables: write (tty), lockfile (mail; semaphore utility for sequencing access to mail spool).

4. Examples of SUID executables: su, passwd.

5 SELinux Object Protection

1. NSA development project to implement Mandatory Access Control (MAC). Originally implemented in FLASK OS. Later integrated into the Linux kernel.

   Owners are denied full control over objects they create. Instead, the system's security policy the access rights granted.

2. Standard Linux uses Discretionary Access Control. Access control is at the discretion of the owner. If root wants to leave a world-readable copy of `/etc/shadow` in `/tmp`, it can.

3. MAC can prevent a privileged process from writing to a file that a non-privileged process could read.

4. Some examples of the fine-grained control allowed:

   (a) A process can be allowed to append data to a log file, but neither truncate the file nor re-write entries.

(b) A process can be allowed to create and write files, but not delete them.

(c) Network programs can be granted access to bind to the ports they need, and no others.

These permissions can be on a per-process basis.

5. Consider `/etc/shadow`. Ordinarily, any root process can access it. Under SELinux, access can be greatly restricted, blocking a hacker who has obtained root.

6. Processes run within domains; certain resources are only available from specific domains; allowable transitions between domains can be specified.

   Example: `xscreensaver` has access to `/etc/shadow` via `chkpwd` program. Each is in its own domain, with the appropriate transition enabled. The privileges given to daemons are often carefully specified.