CI583 Data Structures and Operating Systems Security

Outline

Access control

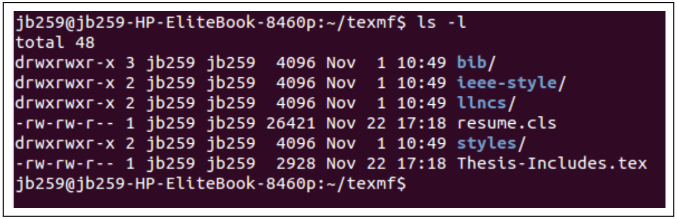
To recap access control at the file system level: in UNIX the Access Control List (ACL) is stored in 3 groups of 3 bits, representing the rights to read, write and execute a file.

The 3 groups define the rights of the owner of the file, the group that file belongs to, and everyone else.

This 9-bit value is often represented as an 3-digit octal number where 4=`read`, 2=`write` and 1=`execute`. Any combination of these numbers in an octal digit is unique, e.g. 750. What rights does the relevant group gave in this case?

Access control

The 9-bits are also represented in text, e.g. in the output of `ls -l`:



```
jb259@jb259-HP-EliteBook-8460p:~/texmf$ ls -l
total 48
drwxrwxr-x 3 jb259 jb259  4096 Nov  1 10:49 bib/
drwxrwxr-x 2 jb259 jb259  4096 Nov  1 10:49 ieee-style/
drwxrwxr-x 2 jb259 jb259  4096 Nov  1 10:49 llncs/
-rw-rw-r-- 1 jb259 jb259 26421 Nov 22 17:18 resume.cls
drwxrwxr-x 2 jb259 jb259  4096 Nov  1 10:49 styles/
-rw-rw-r-- 1 jb259 jb259  2928 Nov 22 17:18 Thesis-Includes.tex
jb259@jb259-HP-EliteBook-8460p:~/texmf$
```

Access control

Every process runs with the file and group permissions of the user that started it.

This is why processes such as the Apache web server normally run as special user such as `nobody` rather than as `root`).

However, we want users to be able to modify their own passwords, for instance, so they should be able to modify the files `/etc/passwd` and `/etc/shadow`. This is achieved with the `setuid` facility which enables processes to run as a particular user.

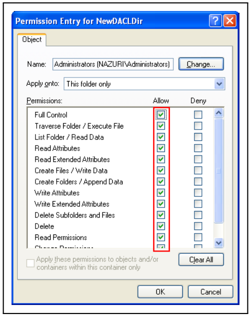
Access control

The `sudo` command allows the administrator to grant very specific rights to users to perform actions that would otherwise be beyond their permissions.

Two more features are `chroot` and `jail`, which allow the system to isolate a process by giving it a new, limited view of the file system.

Access control

The ACLs used in Windows offer more fine grained control than UNIX file permissions, primarily by detailing the type of things one might want to do to a file.



Permission Entry for NewDACLDir

Object

Name: Administrators (NAZURI\Administrators)

Apply onto: This folder only

Permissions:	Allow	Deny
Full Control	✓	☐
Traverse Folder / Execute File	✓	☐
List Folder / Read Data	✓	☐
Read Attributes	✓	☐
Read Extended Attributes	✓	☐
Create Files / Write Data	✓	☐
Create Folders / Append Data	✓	☐
Write Attributes	✓	☐
Write Extended Attributes	✓	☐
Delete Subfolders and Files	✓	☐
Delete	✓	☐
Read Permissions	✓	☐

☐ Apply these permissions to objects and/or containers within this container only

Also, there are two ACLs: the Discretionary ACL (specifying permissions similarly to UNIX) and the System ACL (determining which actions should be logged, whether they were permitted or not).

Is *NIX more secure than Windows?

Linux is often said to be “more secure” than Windows, normally without specifying exactly what that means.

In honeypot testing servers with some sort of default configuration are connected to the internet and the OS which resists being compromised for the longest period is the winner.

The winner is usually something like NetBSD, but is that just because people don’t write worms targeting NetBSD?

Is *NIX more secure than Windows?

- Privileges – common for ordinary user to be admin on Windows, at least until recently.
- Social engineering or password theft more effective because of above.
- Monoculture effect. Successful attack can be devastating. Compromising, say, OpenBSD will not have such a massive effect.

Is *NIX more secure than Windows?

- Userbase: size and type.
- Security-by-obscurity doesn't work. Linux is FOSS, meaning that malicious programmers can search it for weaknesses, but benevolent programmers do the same.

  Linus' law: *given enough eyeballs, all bugs are shallow*.

Access control getting more fine-grained

ACLs in both Linux and Windows are becoming more specific.

In Linux, `sudo` allows the ACL to be configured one permission at a time.

The use of `sudo` has superseded the need for `root` on several popular distros.

Access control getting more fine-grained

Windows' user account control, introduced in Vista, gives an administrator two tokens, one without special privileges and one with them.

The unprivileged token is used until the user tries to do something that requires enhanced rights, when the user is explicitly asked if they want to proceed.

For a mainstream OS, security needs to be *good enough*, and to keep out of the way most of the time.

Process isolation

We have talked about the ways in which an OS might isolate one process from another, by keeping separate contexts, address spaces, use of the base and bounds register, etc.

Google Chrome/Chromium OS takes this one step further.

Process isolation in Chrome OS

Any process belongs to one of the following namespaces:

- process-ID namespaces: sandboxes within which processes can detect each others' existence but not that of any other process on the system.
- interprocess communication namespaces: the IPC processes are threads that visit other processes, e.g. on behalf of the OS. These can't be hijacked by a thread from outside the namespace because they don't appear to exist.

Process isolation in Chrome OS

- virtual file system namespace: a modified view of the file system, so that a process can detect only those files it has some sort of permissions for. Another use for this is to give a process its own VFS namespace within a public directory, such as `/tmp`.
- network namespace: only the processes in this namespace can see the network devices and other network processes.

Process isolation in Chrome OS

Chrome OS also implements the Linux feature of control groups.

Processes are assigned to these groups when they are run, and the group they are in determines the resources they can use (which devices, for instance) and how much (e.g. a cap on processor time).

This can be used to circumvent a DoS attack (something which is normally seen as an application-level problem) at the OS level.

See `http://www.chromium.org/chromium-os`.

Other approaches

Security has been improving in Linux and Windows, but the only way to be sure that all software installed on a computer is harmless is for it to be formally verified. This is beyond the reach of any mainstream system.

In the 1980s the US government introduced a series of security-level specifications to be applied whenever a system stores classified information, in the so-called *Orange Book*.

Other approaches

This is an example of mandatory access control (MAC).

The comparison of rights and permissions of the subject (the initiator, such as a thread) and the object (e.g. a file or hardware device) is mandated at the OS level and meets a certain standard of integrity.

Role-based Access Control

See the Doeppner or Silberschatz books for details of the MAC approach.

This military model is not much use in “ordinary” systems. Only a small, specialised system could be made to fit in the higher categories.

Role-based Access Control

A more flexible approach is to focus on the roles of the users of a system and the uses to which data is put.

An example of this is the Role Based Access Control (RBAC) found in SELinux, a highly secure Linux distribution. SELinux has been verified as within the *mandatory protection* category from the Orange Book.

In SELinux, rules are specified that allow certain capabilities to *domains* (groups of users) and *types* (groups of objects, including programs and files).

Role-based Access Control

SELinux contains upwards of 20,000 rules that embody the following principles:

- A *subject* can have multiple *roles*.
- A *role* can have multiple *subjects*.
- A *role* can have many *permissions*.
- A *permission* can be assigned to many *roles*.
- An *operation* can be assigned many *permissions*.
- A *permission* can be assigned to many *operations*.

Role-based Access Control

RBAC sounds a lot like MAC but is more flexible and fine grained. However, adding new rules is complex and error prone, since it means checking that no existing rule has been invalidated.

At the scale of an entire OS then, RBAC is cumbersome. It is more often used at the level of a single application, such as in Oracle DBMS and MS SQL Server, and gives very precise control in this context.

After the break

- Virtualisation.
- Distributed and Network OS.