University of BRISTOL

Computer System- B Security

Introduction to OS Security

Entities in access control

Sanjay Rawat

bristol.ac.uk

Subjects/Principals

- We saw that access control mechanism has the notion of subjects and objects.
- Subjects are usually **users and process.**
- **Objects** are resources, like files.

Users and Process

- Modern OSes allow multiple users to login with different privileges.
- Process is created by *forking*, thereby ***inheriting the privileges of the parent process***.
- Thus we get a process tree.
- In linux, ***init*** is the is the root process of every process, including login sessions and OS tasks (systemd).
- Example (linux command):
  - `htop` *followed by 't' to get a tree view of the running processes.*
  - *`Pstree`*
- *`Process ID`* is a non-negative number to identify a running process.

Process Privilege

- Launching user’s privilege is used (remember subjects/principals)
- Under Unix-based systems, each process has associated user ID (uid) and group ID (gid).
- Theses identifiers are used to decide what accesses are granted to a process.
- Processes automatically inherit the permissions of their parents process.
- This is the basis of DAC!
- There is another identifier-- effective user ID (euid). In certain cases, it is set to the owner of the application (not the process!), e.g. set-UID prog.

Set User ID process

Set User ID process

- Problem: a low privilege process wants to do a high privilege task.

Set User ID process

- Problem: a low privilege process wants to do a high privilege task.
- Solution:
  - Elevate the privilege temporary!
  - Ask the high privilege process to do that!
  - **Set-user ID** mechanism
  - Differentiate between real user- and effective user-ID
  - Process runs with effective user's privilege, which can be root.
  - Run `ls -l /bin (or /usr/bin)` and observe permission bits of files. What do you notice, e.g. */bin/*mount

Set User ID process

- Problem: a low privilege process wants to do a high privilege task.
- Solution:
  - Elevate the privilege temporary!
  - Ask the high privilege process to do that!
  - **Set-user ID** mechanism
  - Differentiate between real user- and effective user-ID
  - Process runs with effective user's privilege, which can be root.
  - Run `ls -l /bin (or /usr/bin)` and observe permission bits of files. What do you notice, e.g. */bin/*mount
- Let's see one example of setting set-uid bit.

File Access Control

- Data file and programs have permission associated with them-- **read, write, execute**.
- Permission are decided for the three classes-- **owner, group, others**
- **Each class has 3 (bits of) permission**
  - **Decimal weights x:1, w:2, r:4**
- `chmod` command is used to change permission
  - `$chmod 755 file.txt`
- Fine-grained permission `setfacl`
- For directory, execute allows to enter (access) the directory.
-

File Descriptors

File Descriptors

- Rather than providing the whole path, process can use a shorthand (number) to access a file.

File Descriptors

- Rather than providing the whole path, process can use a shorthand (number) to access a file.
- OS maintains a table, called File Descriptor Table (surprise!) that maps files to an index.

File Descriptors

- Rather than providing the whole path, process can use a shorthand (number) to access a file.
- OS maintains a table, called File Descriptor Table (surprise!) that maps files to an index.
- `open()` syscall is used to get FD. At the time of `open()`, permissions are checked.

File Descriptors

- Rather than providing the whole path, process can use a shorthand (number) to access a file.
- OS maintains a table, called File Descriptor Table (surprise!) that maps files to an index.
- `open()` syscall is used to get FD. At the time of `open()`, permissions are checked.
- On further accesses (read/write), these permissions are checked w.r.t. the FD!

File Descriptors

- Rather than providing the whole path, process can use a shorthand (number) to access a file.
- OS maintains a table, called File Descriptor Table (surprise!) that maps files to an index.
- `open()` syscall is used to get FD. At the time of `open()`, permissions are checked.
- On further accesses (read/write), these permissions are checked w.r.t. the FD!
- File Descriptor leak vulnerability: parent process opens the file with high privilege, later a child process with low privilege accesses the file with the same FD!