System Security

Learn You a Capability For Great Good!

Mohamed Sabt

Univ Rennes, CNRS, IRISA

2023 / 2024

Part I

Introduction

Bank Analogy

- Context:
  - Alice wishes to keep all of her valuables in safe deposit boxes in the bank.
  - Occasionally, she would like one or more trustworthy friends to make deposits or withdrawals for her.

- There are two ways that the bank can control access to the box.
  - The bank maintains a list of people authorized to access each box.
  - The bank issues Alice one or more keys to each of the safe deposit boxes.

- Exercise:
  - Which way does represent the ACL?

The ACL Approach

- Authentication: The bank must authenticate.
- Bank's involvement: The bank must
  - store the list,
  - verify users.

- Forging access right: The bank must safeguard the list.
- Adding a new person: The owner must visit the bank.
- Delegation: A friend cannot extend her privilege to someone else.
- Revocation: If a friend becomes untrustworthy, the owner can remove her name.

The Capability Approach

- Authentication: The bank does not need to authenticate.
- Bank's involvement: The bank need not to be involved in any transactions

- Forging access right: The key cannot be forged.
- Adding a new person: The owner can give the key to other people
- Delegation: A friend can extend her privilege to someone else.
- Revocation: The owner can ask for the key back, but it may not be possible to know whether or not the friend has made a copy.

Alice in Hostile Environment 1/2

- Alice does have a social life.
  - She often goes to bars with her friends, some of which might be evil.
  - Therefore, Alice can get drunk; when people get drunk, they might do things or make mistakes that they regret to do.

- In the Capability approach,
  - Alice can choose not to carry the keys with her when she goes to drink.
  - This way, even if she get drunk, she cannot open the safe deposit box.

- In the ACL approach,
  - There is no such kind of protection by the least privilege protection.

Alice in Hostile Environment 2/2

- Alice often sends her employees to carry out tasks for her.
  - These tasks involve going to the bank several times, opening several deposit boxes. However, the employees might be kidnapped at any point of time.
  - Kidnaper can then force the employees to retrieve the valuables from the deposit boxes.

- In the Capability approach,
  - Employees can destroy the keys that will not be needed by the on-going tasks
  - Alice still has a copy of all the keys.
  - This way, the damage can be reduced to the minimum.

- In the ACL approach,
  - This kind of protection is hard to achieve.

Not a Myth!!

- It is widely known that most UNIX operating systems use Access Control List (ACL).

- However it is less well known that the capability concept has also been used in most UNIX operating systems for a long time.

- Exercise: give an example of a capability-based protection?

Part II

Concept & Properties

Definition

- The capability was introduced by Dennis and Van Horn in 1966.
- “A capability is a token, ticket, or key that gives the possessor permission to access an entity or object in a computer system”.

- Intuitive examples:
  - A movie ticket is a capability to watch a movie.
  - A key is a capability to enter a house.

Capability Structure

- A capability is implemented as a data structure that contains:
  - *Identifier:* addresses or names. e.g. a segment of memory, an array, a file, a printer, or a message port.
  - *Access right:* read, write, execute, access, etc.

Using Capabilities – Explicit Use

- You have to show your capabilities explicitly.

- This is what we do when we go to movie theaters: we show the doorkeeper our tickets.

- The following is an example that is quite common when a program tries to access a file:
  - PUT (file_capability, “Hello world");

Using Capabilities – Implicit Use

- There is no need to show the capabilities, but the system will automatically check whether one has the proper capabilities.

- An analogy to this would be having the theater doorkeeper to search your pockets for the right tickets.

- The capability-list method basically uses this approach.
  - Namely, each process carries a list of capabilities; when it tries to access an object, the access control system checks this list to see whether the process has the right capability.

Using Capabilities – Comparison

- The implicit approach is less efficient,
  - especially, when the capability-list is long.

- It might be easier to use, because, unlike the explicit approach, capabilities are transparent to users;
  - therefore, users do not need to be aware of the capabilities.

Capabilities Identifiers

- Capability on Users: Users are more persistent identifiers. Its capability can be stored in files.

- Capability on Processes: Processes are not persistent identifiers, they usually obtain capabilities dynamically.

- Capability on Programs.
  - Giving capabilities to programs can achieve privilege escalation or downgrading.
  - Set-UID programs are a special case: Set-UID programs have the root capability.

Capabilities Examples in LIDS

- LIDS for Linux Intrusion Detection System.

- *CAP CHOWN:* override the restriction of changing file ownership and group ownership.
- *CAP DAC READ SEARCH*: override all DAC restrictions regarding read and search on files and directories.
- *CAP KILL*: the capability to kill any process.
- *CAP NET RAW*: the capability to use RAW sockets
- *CAP SYS BOOT*: the capability to reboot.

Capabilities Storage 1/3

- Capabilities can be stored in a protected place.

- Users cannot touch the capability; they use them implicitly in:
  - ***Kernel***: the capability list is stored in the kernel (e.g. in the process data structure). Users cannot modify the contents of any capability. When users need their capabilities, the system will go to the kernel capability-list.

  - ***Tagged architecture***: the capability can be saved in memories that are tagged as read-only.

Capabilities Storage 2/3

- **In an unprotected place:** In some applications, users may have to carry their capabilities with themselves. When they request an access, they simply present their capability to the system.
- This is an explicit use of capabilities.

- How to prevent users from tampering with the capability? Because permissions are encoded in the capability, if users can tamper with the contents of a capability, they can gain unauthorized privileges.
- The protection can be achieved using cryptographic checksum: the capability issuer can put a cryptographic checksum on the capability (e.g. digital signature). Any tampering of the capability will be detected.

- This approach is widely used in distributed computing environments, where capabilities need to be carried from one computer to another; therefore, relying on kernel to protect capabilities is infeasible.

Capabilities Storage 3/3

- **Hybrid Approach:** users can use capabilities in an explicit manner, but the capabilities are stored in a safe place.

- The real capabilities are stored in a table, which resides in a protected place (e.g. kernel).
- Users are given the index to these capabilities. They can present the index to the system to explicitly use a capability.
- Forging an index by users does not grant the users with any extra capability.

Basic Operations 1/4

- **Create**: a capability is created for a user (or assign to a user).

- **Delegate**: a subject delegates its capability to other subjects. There are many interesting features related to delegation:
  - Expiration time: specify the lifetime of a delegated capability.
  - Propagation control: specify whether the users who get a capability via delegation can further delegate the capability.

Basic Operations 2/4

- **Revoke**: a subject revokes the capabilities it has delegated to other subjects.

- The implementation of revocation in general is a difficult problem. The followings are two common revocation schemes:
  - Approach 1: Have each capability point to an indirect object. When revoking a capability, we can simply delete the indirect object.
  - Approach 2: Use a random number. The owner can change the number. A user must also present the number in addition to the capability.
  - The above two approaches do not allow selective revocation.
  - Attach an expiration time to a delegated capability can achieve automatic revocation.

Basic Operations 3/4

- **Enable**: a subject enables a disabled capability.

- **Disable**: a subject *temporarily* disables a capability.

Basic Operations 4/4

- **Delete** capability: a subject *permanently* deletes a capability.

- It should be noted that *disabling* a capability is different from *deleting* a capability. They are both useful to achieve the least-privilege principle.
- When a capability is not needed anymore it should be permanently *removed* from the task. This way, even if the task is compromised to execute malicious code, the code cannot use the capability.
- When a capability will still be needed later, but will not be needed by a subtask (e.g. a procedure within a process), the capability should be *disabled*. When the capability is needed, it can be enabled.

Different Notions of Capabilities

- Subjects have capabilities, which
  - Give them accesses to resources.
  - Can be passed around.

- Capabilities used in Linux as a way to divide the root power into multiple pieces that can be given out separately.

- Capabilities solve the confused deputy problem (exercise: why?).

Part III

Case-Study

File Access

1. f = open(“/etc/passwd”, “r”);
2. read(f, buf, 10);
3. write(f, buf, 10);
/* Before the following statement is executed, the root modifies the permission on /etc/passwd to 600, normal users cannot read this file anymore. */
4. read(f, buf, 10) ;

ACL or Capability 1/2

- Because the /etc/passwd file has a permission 644, normal users can open the file for read.
  - Line 1 is based on ACL.

- We know that Line 2 will succeed, but Line 3 will fail.
  - There is an access control on both read and write.
  - Is the access control based on ACL?

ACL or Capability 2/2

- If the access control of read is based on ACL, line 4 should fail.

- However, we observe that line 4 succeeds.

- The access control decision for read is not based on ACL.
- Then what is it based on? It is actually based on capability.

File Descriptor 1/2

- File descriptor is an application of capability. When a file is open, a file descriptor is created and stored in the **filp** table.

```
/* (in src/fs/fproc.h) */
struct fproc  {                        /* Process Table */
   ......
   struct filp *fp_filp[OPEN_MAX];  /* the file descriptor table */
}

struct filp {           /* Filp Table */
   mode_t file_mode;    /*RW bits, telling how file is opened */
   int filp_flags;
   int filp_count;  /* how many file descriptors share this slot? */
   struct inode *filp_ino   /* pointer to the inode table */
   off_t filp_pos;
}
```

File Descriptor 2/2

- The user-space application is given the index of the file descriptor (we often call this index the file descriptor, but actually, it is just an index to the real descriptor).

- The filp table is actually a capability list.
  - It contains a list of file descriptors.
  - Each file descriptor contains a permission part that describes what the process can do to this file.

Capability Operations of File Descriptor

- ***Create**:* a capability is created via the open() system call.
  - Whether a process is allowed to create a capability depends on another access control mechanism, the Access Control List (ACL).
  - Namely, the ACL of the file will be checked to decide whether the process can open this file. If yes, a capability will be created.
  - This interesting example demonstrates one type of coordination between ACL and capability.

- ***Delete**:* a capability is deleted via the close().
  - This system call will remove the corresponding capability from the filp table.

- The other operations, such as delegation, revocation, enabling, and disabling, are not supported.

Part IV

Capabilities in Linux

Exercise

- Some tasks require special privileges.
- However, it is insecure to grant the users such privileges, because they might use them on some other tasks.

- How can we ensure that the users only get the privileges while carrying out the intended task?
  - Namely, the privileges will be taken back from the users once the task is finished.

Capabilities for Privileged Programs

- Similar to what the Set-UID programs are trying to achieve.

- The basic idea is to assign the privileges to programs, not to users.
  - Users gain the privileges when they run this program; they will lose the privileges when the program finishes.

- When a privileged program is executed, the capabilities are effective.
  - For example, if a program has a file-reading capability, it can read all files even if the user who runs the program is not superuser.

Capabilities in the Kernel

- Starting with kernel 2.2, Linux divides the root privileges into smaller privileges, known as capabilities.

- Capabilities are a per-thread attribute, and they can be independently enabled and disabled.

Capabilities Storage

- Store the capabilities in a configuration file, such as /etc/security/capability.conf.
  - When system bootup, configuration in this file will be read in kernel and saved in a capability array.

- Store the capabilities in the program's inode.
  - When a process is created to run the program, the process will be initialized with the program's capabilities.

Capabilities Examples

- CAP_CHOWN: Make arbitrary changes to file UIDs and GIDs.
- CAP_DAC_OVERRIDE: Bypass file read, write, and execute permission checks.
- CAP_DAC_READ_SEARCH: Bypass file read permission checks and directory read and execute permission checks.
- CAP_NET_ADMIN: Perform various network-related operations (e.g., setting privileged socket options, enabling multicasting, interface configuration, modifying routing tables).
- CAP_NET_RAW: Use RAW and PACKET sockets.
- CAP_SYS_PTRACE: Trace arbitrary processes using ptrace(2).

Capabilities Sets

- *Permitted Set:* This is the set of capability that a thread has.

- *Effective Set:* This is the set of capabilities that are currently effective in the process, i.e. the access control will use this set of capabilities.

- *Inheritable Set:* This is a set of capabilities preserved across execve.
  - It provides a mechanism for a process to assign capabilities to the permitted set of the new program during execve.

Executables Capabilities

- Since kernel 2.6.24, the kernel supports associating capability sets with an executable file using setcap.

- The file capability sets, in conjunction with the capability sets of the thread, determine the capabilities of a thread after an execve.
  - When a thread executes a program with file capabilities, the thread can get extra privileges.
  - We can replace Set-UID programs using file capabilities, i.e., instead of giving a privilege program the root privilege, we can assign a set of required capabilities to the program. This way, we can enforce the *principle of least privilege*.

Libcap Library

- The most convenient way for user-level programs to interact with the capability features in Linux, such as setting/getting thread capabilities, setting/getting file capabilities, etc.

- Capability Operations:
  - *Disabling capabilities:* only temporarily disable certain capabilities, i.e., remove the capabilities from the *effective set*; the capabilities are still in the *permitted set*, and can be enabled later.
  - *Enabling capabilities:* if a capability is in the *permitted set*, it can be enabled and thus becomes effective.
  - *Deleting capabilities:* if a capability is no longer needed, it can be permanently removed from the permitted set.

Part V

Capabilities Vs ACL

Representation

- ACL
  - Associate list with each object
  - Check user/group against list
  - Relies on authentication: need to know user

- Capabilities
  - Capability is unforgeable ticket
    - Random bit sequence, or managed by OS
    - Can be passed from one process to another
  - Reference monitor checks ticket
    - Does not need to know identify of user/process

Granularity

- In theory:
  - Dual representations of access control matrix

- ACL is based on users.

- Capabilities can be based on process, programs, and users.
  - Finer granularity: the principle of least privilege.

Forking

- ACL: depends on the child's subject.
  - Unix: child inherits parent's subject
  - Inherits all of the parent's permissions
  - Any program you run inherits all of your authority
  - Bad for least privilege.

- Capabilities: child has no caps by default
  - Parents can transfer some capabilities to the child.
  - Used to explicitly delegate the necessary privileges.
  - Defaults to Least Privileges.

Ambient Authority

- Ambient authority: means that a user's authority is automatically exercised, without the need of being selected.
  - Causes the confused deputy problem.

- Example:
  - You are carrying a lot of keys. When you walk to a door, the door automatically opens if you have the right key. You do not need to select a key.

- No ambient authority in capability systems.

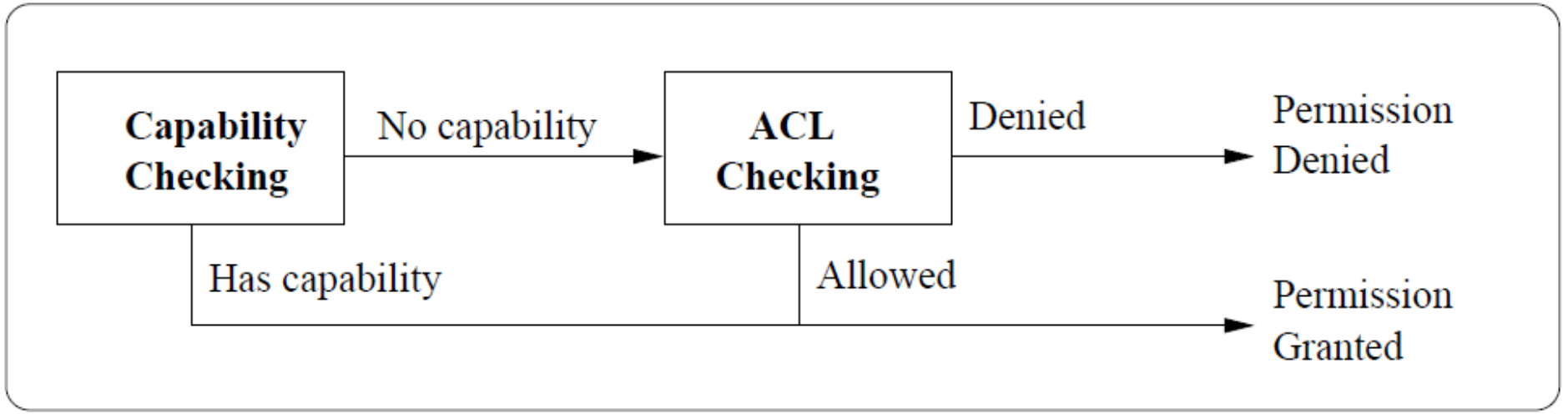
Meta-Conclusion

- Why do most operating systems use ACL, rather than capabilities

- Capability is more suitable for process level sharing, but not user-level sharing.
  - User-level sharing is what **was** really needed.

- Processes are more tightly coupled in capability-based systems
  - Because they need to pass capabilities around.
  - Programming may be more difficult.

Privilege Escalation



Privilege Restriction