Security Policies and Enforcement Mechanisms

Fall 2024

R. Sekar

Terminology and concepts

- Principals, Subjects, Objects
- Principle of least privilege
  - Throughout execution, each subject should be given the minimal access necessary to accomplish its task
    - Needs mechanisms for rights amplification and attenuation
- Reference monitors
  - Abstract machine that mediates all access
- Security kernel
  - Hardware, firmware and software elements that implement the reference monitor
- Trusted Computing Base
  - Totality of protection mechanisms in the system
    - Smaller TCB translates to greater assurance

Access control

- Typically, three kinds of entities
  - User (principal)
  - Subject: typically, a process acting on behalf of the user
  - Object: files, network sockets, devices, ...
- Goal: Control access to operations performed by subjects on objects
  - Basic: Read, Write, Execute
  - Additional: Append, Create, Delete
  - Advanced: Change permission, ownership

Discretionary Access Control

- Discretionary, i.e., permissions settings at owner's discretion
  - Permission on an object is set by its owner
  - The norm on OSes (UNIX derivatives, Windows, ...)
- Can be modeled as a matrix:

	O_1	O_2	O_3	$\cdots$
Alice	rw	w	—	$\cdots$
$\cdots$	$\cdots$ $\cdots$	$\cdots$	$\cdots$	$\cdots$
Zachary	r	wx	—	$\cdots$

- Implementations
  - Access-control lists (ACLs)
    - The AC-matrix column describing access rights on an object are stored with the object
    - Example: O_1: [Alice:rw, . . ., Zachary:r]
  - Capabilities
    - Row-wise representation of AC matrix, held by the user corresponding to the row
    - Example: Alice: [O_1:rw, O_2:w, O_3:$-$, . . .]

Managing Permissions

- Improve manageability using indirection
  - Groups
  - Roles (RBAC — Role-based access control)
- Inheritance
- Negative permissions

Implementation of DAC on UNIX: Objects

- All resources are "files"
- Each file has an owner and group owner
- For simplicity, original UNIX did not support ACL
  - Instead, permissions are divided into three groups
    - permissions for each of: file owner, owner group, and everyone else
    - Owner and group owner are attributes of the file
  - 3 bits of permission for each part: read/write/execute
- For directories, the interpretation is:
  - read means ability to list the directory
  - write means ability to create files in the directory
  - execute means the ability to access specific files if you know the name

Implementation of DAC on UNIX: Objects

- Permissions on newly created files determined by umask
  - Start with the mode specified in the third argument of open, and turn off the bits specified in umask
- chmod for changing permission
- chown for changing ownership (only root can use this syscall)
- Additional 3 bits of permission
  - setuid, setgid and sticky bits
  - Today, sticky bits is used to regulate access to shared directories:
    - Even if the directory is writable, you can only delete your own files
- More recent: Access control lists
  - But not in wide use, because most software is old and does not know about them.

Implementation of DAC on UNIX: Subjects

- Subjects inherit the userid, group and supplementary groups of the parent
  - Programs that perform user authentication (e.g., login, sshd) need to set this information
  - Exception: setuid/setgid bits
    - Privilege escalation mechanism
    - Can also be thought of as a delegation mechanism
- File permission checks are performed using this userid and groups
  - The primary as well as supplementary groups of a process are checked for permission
- No permission checks on superuser (userid 0)
  - Permission checks based on userid: usernames are used only for login
- Objects created by a subject inherit the subject's userid and group
  - File's default group owner is determined by the subject's primary group
  - But a file owner can set the file's group any of the supplementary groups of the owner

Effective, Real and Saved UID/GID

- Effective: the uid used for determining access privileges
- Real: the "real" user that is logged on, and on whose behalf a process is running
- Saved: allows processes to temporarily relinquish privileges but then restore original privileges
  - When executing a setuid executable, original euid is saved (or it could be explicitly saved)
  - Setting userid to saved userid is permitted

DAC on Windows vs UNIX

- OO-design: permissions can differ, depending on type of object
  - NTFS files offer additional rights: delete, modify ACL, take ownership
    - Files inherit permission from directory
  - Use of Registry for configuration data
    - Richer set of access permissions for registry entries (e.g., enumerate, create subkey, notify, ...)
- Mandatory file system locks
- No setuid mechanism

Capabilities

- "Tickets" to gain access to a resource
  - Combine objects and access rights into one package
  - Transferable
  - Must be unforgeable
- Examples
  - Passwords and cryptographic keys
  - Certificates
    - Anything cryptographically signed can be thought of as a capability
  - File descriptors
    - Handles to information maintained within OS kernel
  - Some cookies (e.g., session cookie) in web applications

Implementation drawbacks

- More difficult to implement than ACLs
  - Need forever unique object ids that don't change
  - Need to use crypto or rely on OS primitives that may be hard to realize
- Difficult to manage
  - How do we determine the permissions held by a user?
  - Do we want to allow them to pass around their capability? What about theft?
  - How long do we store them?
  - How can we revoke permissions?
- Result: Capabilities in their purest form are not widely used in OSes

Benefits

- Provide a better framework than ACLs when one or more of the following conditions hold:
  - Policy enforcement isn't centralized
  - Parties have limited trust on each other
  - Rights need to move with principals
- More examples
  - Web applications use cookies containing session ids to indicate when a user has successfully authenticated
    - Can use this capability even if their IP address changes, or after a browser crash/restart
  - Kerberos uses capabilities for access across hosts
    - Uses capabilities with different time scales
    - Accesses within a host are typically based on the ACL mechanism of the host OS

Mandatory Access Control (MAC)

- DAC Limitations
  1. "Trojan Horse" problem: assumes that users authorize all actions of their processes
     - What if a program changes permissions on a file without the user's knowledge?
  2. Provides no protection if a resource owner did not bother to set the ACL properly
- To overcome these problems, MAC moves the responsibility to a central point, typically the system administrator
  - Organizations want to control access to their resources
  - Don't want to rely on individual employees to ensure that organizational policies are enforced

MAC Example: Multi-Level Security (MLS)

- Motivation for MLS
  - Access control policies do not provide any way to control the manner in which information is used
    - Once an entity is given access to some information, it can use this information in any way
    - Can share it with anyone
- MLS policies control information flow, and hence control how information is used
  - Ensure certain global safety properties
- Developed originally in the context of protecting secrets in the military

MLS: Confidentiality Policies

- An object is labeled with a level *L*
  - Labels correspond to points in a lattice
  - Typical levels used in military include:
    - unclassified, classified, secret, top secret
- A subject is associated with a clearance level *C*
  - A subject can access an object if its clearance level is equal to or above the object's level
- In addition, information can also be compartmentalized
  - "Need-to-know" principle is used to decide who gets to access what information
    - e.g., top-secret information regarding nuclear fuel processing is made available to those working on nuclear-related projects

MLS: Bell-LaPadula Model [1973]

- To prevent leakage of sensitive information, we ensure:
  - No "read-up:"
    - A subject S can read object O only if $C(S) \geq L(O)$
  - No "write-down:"
    - A subject can write an object O only if $L(O) \geq C(S)$
- Ensures that information can flow only upwards in terms of confidentiality level
- Example: a subject with top-secret-clearance reads a top-secret file and then writes to a secret file
  - Without this "$*$" property, this behavior would be permitted and cause "top-secret" info to reach someone with just "secret" clearance.

MLS: Biba Model (Integrity)

- Designed to ensure integrity rather than confidentiality
  - In non-military settings, integrity is more important
- Conditions
  - No “read-down:”
    - A subject S can read object O only if $C(S) \leq L(O)$
    - A subject’s integrity can be compromised by reading lower integrity data, so this is disallowed
  - No “write-up:”
    - A subject S can write an object O only if $C(S) \geq L(O)$
    - The integrity of a subject’s output can’t be greater than that of the subject itself.
- Variation: Low Water-Mark Policy (LOMAC)
  - Allow read-downs, but downgrade subject to the level of the object
- Both policies ensure system integrity

Problems with Information Flow

- “Label creep:” More and more objects become sensitive, making it difficult for the system to be used by lower-clearance subjects
- No controlled mechanism for making exceptions
  - For instance, encryption programs need to read more sensitive info and write less sensitive (but encrypted) info
  - To accommodate this, “trusted” programs are exempted from the “$*$”-property
  - But the system provides no check on possible misuse of exceptions
- Motivate alternate approaches, or hybrid approaches

Alternative Approaches

- Key goal: Mitigate damage that may result from all-powerful root privileges
  - Break down root privilege into a number of sub-privileges
  - Decouple user privileges from program privileges
- Examples
  - Domain and type enforcement
    - SELinux
    - AppArmor (sort of)
  - Linux capabilities
    - Somewhat different from the classical notion of capabilities described earlier under DAC
    - These capabilities are associated with subjects, not users — Subjects are under the control of the OS, so many of the problems of classical capabilities can be avoided (unforgeability, unlimited lifetime, revocation, . . .)
    - Independent of objects

Domain and Type Enforcement (DTE)

- Subjects belong to domains
  - Users have default domains, but not all their processes belong to the same domain
    - Some processes transition to another domain, typically when executing another program
- Objects belong to types
- Policies specify which domains have what access rights on which types
  - Enable application of least-privilege principle
  - Example: a media player may need to write its configuration or data files, but not libraries or config files of other applications
- Domain transitions are an important feature
  - Can occur on exec, as specified by policy

DTE and SELinux

- Security-enhanced Linux combines standard UNIX DAC with DTE
  - Note: SELinux also supports other MAC mechanisms (e.g., MLS) but these are typically not enabled/configured
- Intuitively, the idea is to make access rights a function of (user, program, object)
  - Roughly speaking, MLS requires us to trust a program (and not enforce "$*$"-property), or fully trust it (i.e., it may do whatever it wants with information that it read)
  - In contrast, DTE allows us to express limited trust that is a function of the program, i.e., grant a program only those rights that it needs to carry out its function

DTE/SELinux Vs Information Flow

- In practice, DTE has turned out to be "one policy per application"
  - Scalability is clearly an issue
  - In addition, SELinux policies are quite complex
  - While DTE is able to gain additional power because it captures the fact that trust is not transitive, this very feature makes DTE policies difficult to manage
    - What are the overall system-wide assurances can be obtained, given a set of DTE policies developed independent of each other
- Information flow policies are simpler and closely relate to high-level objectives
  - Confidentiality or Integrity
  - But neither approach is easy enough for widespread use

Linux (POSIX) Capabilities

- Goal: Decompose root privilege into a number of “capabilities”
  - CAP_CHOWN
  - CAP_DAC_OVERRIDE
  - CAP_NET_BIND_SERVICE
  - CAP_SETUID
  - CAP_SYS_MODULE
  - CAP_SYS_PTRACE
- Differs from classical capabilities
  - Captures access rights, but not associated with any object
  - Unforgeable only because the capabilities are never present in the subject
    - They are maintained by the OS kernel for every process, similar to how subject ownership is maintained in the kernel

Linux (POSIX) Capabilities

- Effective, Permitted and Inheritable capabilities
  - Somewhat related to (and guided by) effective, real and saved userids
  - Effective: accesses will be checked against this set
  - Permitted: superset of effective, cannot be increased
    - Effective set can be set to include any subset of permitted
  - Inheritable: capabilities retained after `execve`
    - At `execve`, permitted and effective sets are masked with inheritable
- Attaching capabilities to executables
  - Allowed: capabilities not in this set are taken away on `execve`
  - Forced: “setuid”-like feature — given to executable even if parent does not have the capability
  - Effective: Indicates which of the permitted bits are to be transferred to effective

Commercial Policies

- High-level policies in commercial environments differ from those in military environments.
- Examples:
  - Chinese Wall (conflict of interest)
  - Clark-Wilson
- Common principles:
  - Separation of duty: critical functions need to be performed by multiple users.
  - Auditing: ensure actions can be traced and attributed, and if necessary, reverted (recoverability).

Clark-Wilson Policy

- Focuses on data integrity rather than confidentiality.
  - Based on the observation that in the "real-world," errors and fraud are associated with loss of data integrity.
- Based on the concept of well-formed transactions (WFTs):
  - Data is processed by a series of WFTs.
  - Each WFT takes the system from one consistent state to another:
    - Operations within a WFT may temporarily make system state inconsistent.

Clark-Wilson Policy

- What about the integrity of WFTs?
  - WFTs ensure that the system state is consistent, but don't guarantee that the transactions themselves were correct.
    - Was that a fraudulent money transfer?
    - Was that travel voucher properly inspected?
  - Relies primarily on *separation of duty.*
- *Auditing* to verify integrity of transactions
- *Recovery:* Maintain adequate logs so that WFTs in error can be undone.

Chinese Wall Policy

- Addresses "conflict of interest."
  - Common in the context of the financial industry.
  - Regulatory compliance, auditing, advising, consulting, etc.
- Defined in terms of:
  - CD: objects related to a single company.
  - COI classes: sets of companies that are competitors.
  - Policy: no person can have access to two CDs in the same COI class.
    - Implies past, present, or future access.

Policy Management

- Goal: simplify the setup and administration of security policies.
- Topics:
  - Role-based access control (RBAC).
  - Administrative policies:
    - Who can change what policies.
  - Delegation and trust management.

RBAC

- Roles vs groups: Essentially the same mechanism but different interpretations.
  - Role: a set of permissions.
  - Group: a set of users.
- An extra level of indirection to simplify policy management.
  - Based on the functions performed by a user, he/she is given one or more roles.
    - When the user's responsibilities change, the user's roles are updated.
    - When the permissions needed to perform a function are changed, the corresponding role's permissions are updated. — Does not require any updating of user information.

Delegation

- Ability to transfer certain rights to another entity so that it may act on behalf of the first entity.
  - Essential for scalability and in the context of distributed systems.
- Implementation:
  - The issue is one of trust and granularity.
  - Multiple levels of delegation rely on a chain of trust.
    - Can be implemented using certificates.
- Trust management:
  - Systems designed to manage delegation and enforce security policies in the presence of delegation rules and certificates.