NYIT
NEW YORK INSTITUTE OF TECHNOLOGY

CSCI 620 – Operating Systems Security

Chapter 2

Chapter 2

Access Control Fundamentals

Chapter Overview

- Protection Systems
- Mandatory Protection Systems
- Reference Monitors
- Definition of a Secure Operating System
- Assessment criteria to be used against each operating system.

Introduction

- An access enforcement mechanism authorizes requests (e.g., system calls) from multiple subjects (e.g., users, processes, etc.) to perform operations (e.g., read, write, etc.) on objects (e.g., files, sockets, etc.).
- An operating system provides an access enforcement mechanism.

Protection System

- The security requirements of a operating system are defined in its *protection system*.

Definition 2.1. *A protection system consists of a protection state, which describes the operations that system subjects can perform on system objects, and a set of protection state operations, which enable modification of that state.*

- A protection system enables the definition and management of a ***protection state***. A *protection state* consists of the specific system subjects, the specific system objects, and the operations that those subjects can perform on those objects.
- A protection system also defines *protection state operations* that enable a protection state to be modified. For example, protection state operations are necessary to add new system subjects or new system objects to the protection state.

Lampson's Access Matrix

- Lampson defined the idea that a protection state is represented by an *access matrix*, in general.

Definition 2.2. An *access matrix* consists of a set of subjects $s \in S$, a set of objects $o \in O$, a set of operations $op \in OP$, and a function $ops(s, o) \subseteq OP$, which determines the operations that subject *s* can perform on object *o*. The function *ops(s, o)* is said to return a set of operations corresponding to cell *(s, o)*.

- Figure 2.1 shows an access matrix. The matrix is a two-dimensional representation where the set of subjects form one axis and the set of objects for the other axis. The cells of the access matrix store the operations that the corresponding subject can perform on the corresponding object. For example, subject Process 1 can perform read and write operations on object File 2.

Lampson's Access Matrix

	File 1	File 2	File 3	Process 1	Process 2
Process 1	Read	Read, Write	Read, Write	Read	-
Process 2	-	Read	Read, Write	-	Read

Figure 2.1: Lampson's Access Matrix

An Access Matrix

	BIBLIOG	TEMP	F	HELP.TXT	C_COMP	LINKER	SYS_CLOCK	PRINTER
USER A	ORW	ORW	ORW	R	X	X	R	W
USER B	R	-	-	R	X	X	R	W
USER S	RW	-	R	R	X	X	R	W
USER T	-	-	-	R	X	X	R	W
SYS_MGR	-	-	-	RW	OX	OX	ORW	O
USER_SVCS	-	-	-	O	X	X	R	W

R: Read

W: Write

O: Own

X: Execute

Lampson's Access Matrix

- If the subjects correspond to processes and the objects correspond to files, then we need protection state operations to update the protection state as new files and processes are created.
- For example, when a new file is created, at least the creating process should gain access to the file. In this case, a protection state operation *create_file(process, file)* would add a new column for the new file and add read and write operations to the cell *(process, file)*.
- Lampson's access matrix model also defines operations that determine which subjects can modify cells. For example, Lampson defined an ***own*** operation that defines ownership operations for the associated object.

Lampson's Access Matrix

- When a subject is permitted for the own operation for an object *o*, that subject can modify the other cells associated with that object *o*.
- Lampson also explored delegation of ownership operations to other subjects, so others may manage the distribution of permissions.
- The access matrix is used to define the ***protection domain*** of a process.

- **Definition 2.3.** A ***protection domain*** specifies the set of resources (objects) that a process can access and the operations that the process may use to access such resources.

- By examining the rows in the access matrix, one can see all the operations that a subject is authorized to perform on system resources.

Access Control List

- Because the access matrix would be a sparse data structure in practice (i.e., most of the cellswould not have any operations), other representations of protection states are used in practice.
- One representation stores the protection state using individual object columns, describing which subjects have access to a particular object.
- This representation is called an *access control list* or ACL. The other representation stores the other dimension of the access matrix, the subject rows.

Access Control List

- In this case, the objects that a particular subject can access are stored. This representation is called a capability list or C-List.
- There are advantages and disadvantages to both the C-List and ACL representations of protection states.
- For the ACL approach, the set of subjects and the operations that they can perform are stored with the objects, making it easy to tell which subjects can access an object at any time. Administration of permissions seems to be more intuitive.
- C-Lists store the set of objects and operations that can be performed on them are stored with the subject, making it easy to identify a process's protection domain.

So, what is a protection system?

- Informally, a protection system is a description of what is allowed in a computer system, together with a set of rules that allow us to modify a description.
- Formally:
  - A *protection system* consists of a *protection state*, and a set of *protection state operations* which enable modifications to that state.

Discretionary Access Control

- The access matrix model shown before presents a problem for secure systems:
  - untrusted processes can tamper with the protection system. Using protection state operations, untrusted user processes can modify the access matrix by adding new subjects, objects, or operations assigned to cells. Consider Figure 2.1. Suppose Process 1 has ownership over File 1. It can then grant any other process read or write (or potentially even ownership) access over File 1.
- A protection system that permits untrusted processes to modify the protection state is called a *Discretionary Access Control* (DAC) system. This is because the protection state is at the discretion of the users and any untrusted processes that they may execute.

Problems with DAC

- We say that the protection system defined in Definition 2.1 aims to enforce the requirement of ***protection***: one process is protected from the operations of another only if both processes behave benignly.
- If no user process is malicious, then with some degree of certainly, the protection state will still describe the true security goals of the system, even after several operations have modified the protection state.
- Suppose that a File 1 in Figure 2.1 stores a secret value, such as a private key in a public key pair, and File 2 stores a high integrity value like the corresponding public key. If Process 1 is non-malicious, then it is unlikely that it will leak the private key to Process 2 through either File 1 or File 2 or by changing the Process 2's permissions to File 1.
- However, if Process 1 is malicious, it is quite likely that the private key will be leaked.

Problems with DAC

- The access matrix protection system does not ensure the ***integrity*** of the public key file File 2, either.
- In general, an attacker must not be able to modify any user's public key because this could enable the attacker to replace this public key with one whose private key is known to the attacker. Then, the attacker could masquerade as the user to others. Thus, the integrity compromise of File 2 also could have security ramifications.
- Clearly, the access matrix protection system cannot protect File 2 from a malicious Process 1, as it has write access to File 2.
- Further, a malicious Process 2 could enhance this attack by enabling the attacker to provide a particular value for the public key.
- ***Buffer overflow vulnerabilities*** are used in this manner for a malicious process (e.g., Process 2) to take over a vulnerable process (e.g., Process 1) and use its permissions in an unauthorized manner.

Mandatory Protection Systems

- **Definition 2.4.** A *mandatory protection system* is a protection system that can only be modified by trusted administrators via trusted software, consisting of the following state representations:
  - A *mandatory protection state* is a protection state where subjects and objects are represented by
  - *labels* where the state describes the operations that subject labels may take upon object labels;
  - A *labeling state* for mapping processes and system resource objects to labels;
  - A *transition state* that describes the legal ways that processes and system resource objects may be relabeled.

Mandatory Protection Systems

- For secure operating systems, the subjects and objects in an access matrix are represented by system-defined *labels*.
- A label is simply an abstract identifier—the assignment of permissions to a label defines its security semantics.
- Labels are tamperproof because:
  1. the set of labels is defined by trusted administrators using trusted software and
  2. the set of labels is immutable. administrators define the access matrix's labels and set the operations that subjects of particular labels can perform on objects of particular labels.

- Such protection systems are ***Mandatory Access Control (MAC)*** systems because the protection system is immutable to untrusted processes.
- Since the set of labels cannot be changed by the execution of user processes, we can prove the security goals enforced by the access matrix and rely on these goals being enforced throughout the system's execution.

Labeling state

- *A **Labeling state*** assigns labels to new subjects and objects. Figure 2.2 shows that processes and files are associated with labels in a fixed protection state.
- When newfile is created, it must be assigned one of the object labels in the protection state. In Figure 2.2, it is assigned the secret label.
- Likewise, the process newproc is also labeled as unclassified. Since the access matrix does not permit unclassified subjects with access to secret objects, newproc cannot access newfile.
- As for the protection state, in a secure operating system, the labeling state must be defined by trusted administrators and immutable during system execution.

In Pictures....

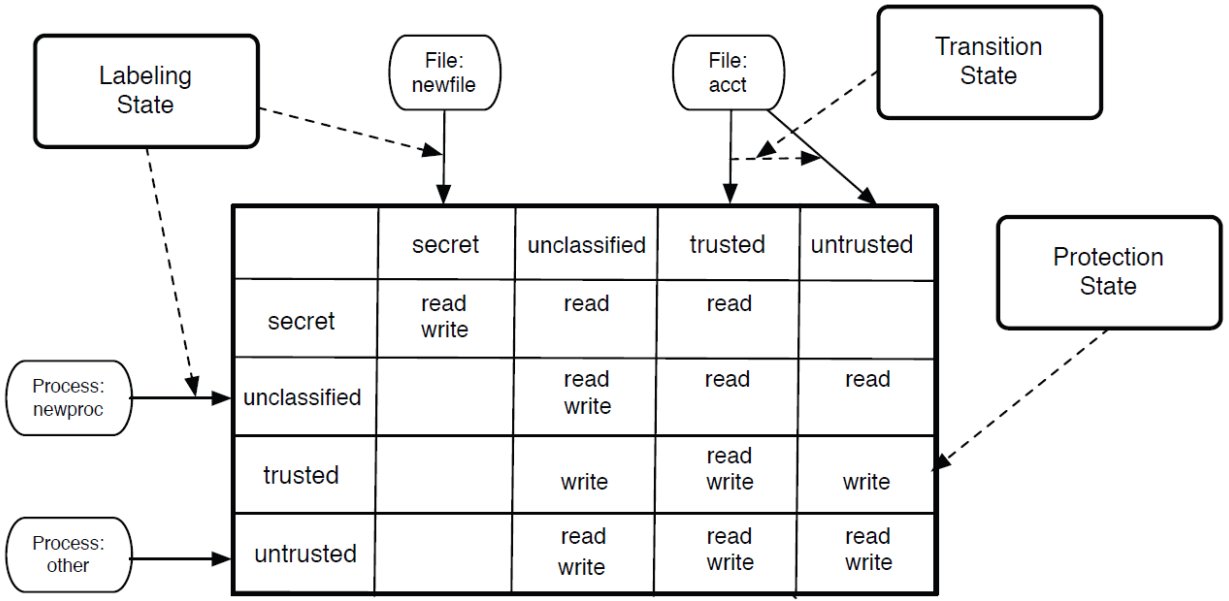

Figure 2.2: A Mandatory Protection System: The *protection state* is defined in terms of labels and is immutable. The immutable *labeling state* and *transition state* enable the definition and management of labels for system subjects and objects.

Protection Domain Transitions

- A ***transition state*** enables a secure operating system to change the label of a process or a system resource.
- For a process, a label transition changes the permissions available to the process (i.e., its protection domain), so such transitions are called *protection domain transitions* for processes.
- As an example where a protection domain transition may be necessary, consider when a process executes a different program.

Protection Domain Transitions

- When a process performs an execve system call the process image (i.e., code and data) of the program is replaced with that of the file being executed.
- Since a different program is run as a result of the execve system call, the label associated with that process may need to be changed as well to indicate the requisite permissions or trust in the new image.

How it works

- The labels are fixed and define the access relations.
- Subjects and objects are assigned different labels according to need.
- When circumstances changes, labels may change also.
- However, permissions are constrained to those defined by the labels
- In addition, label assignment/reassignment is a “privileged operation”.

Reference Monitors

- A ***reference monitor*** is the classical access enforcement mechanism.Figure 2.3 presents a generalized view of a reference monitor. It takes a request as input, and returns a binary response indicating whether the request is authorized by the reference monitor's access control policy.
- We identify three distinct components of a reference monitor:
  (1) its interface;
  (2) its authorization module; and
  (3) its policy store.
- The interface defines where the authorization module needs to be invoked to perform an authorization query to the protection state, a labeling query to the labeling state, or a transition query to the transition state.
- The authorization module determines the exact queries that are to be made to the policy store. The policy store responds to authorization, labeling, and transition queries based on the protection system that it maintains.

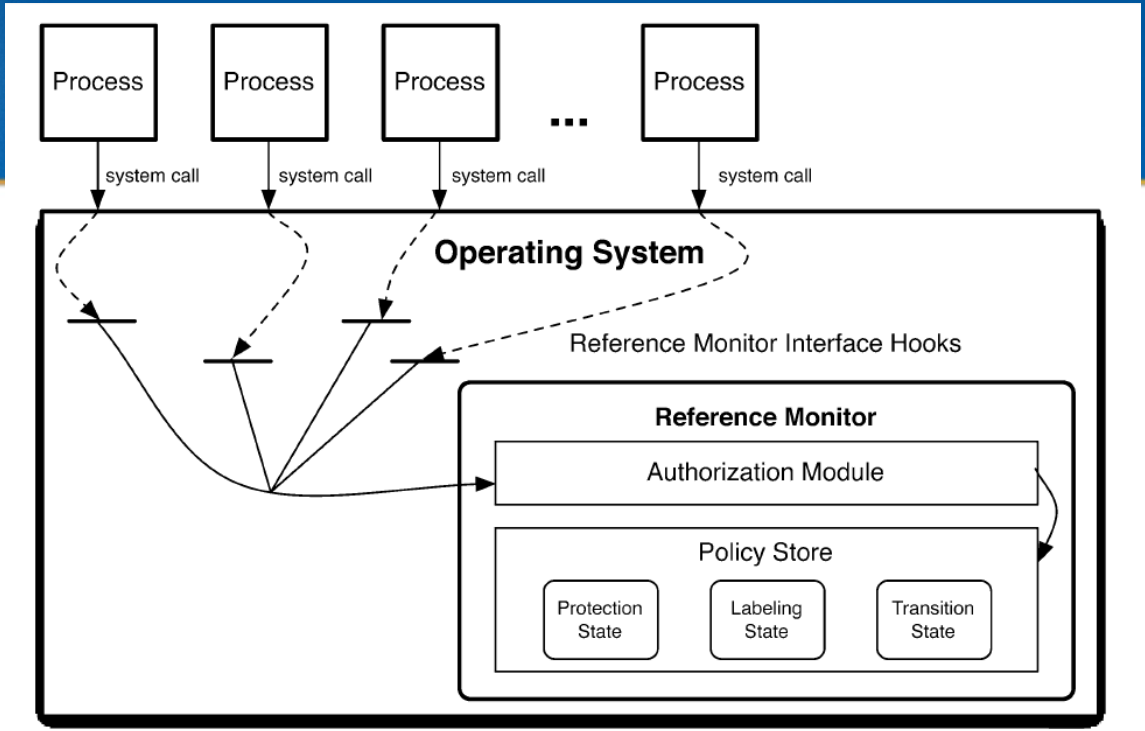

Figure 2.3: A *reference monitor* is a component that authorizes access requests at the *reference monitor interface* defined by individual *hooks* that invoke the reference monitor's *authorization module* to submit an authorization query to the *policy store*. The policy store answers authorization queries, labeling queries, and label transition queries using the corresponding states.

Reference Monitor Interface

- ***Reference Monitor Interface*** defines where protection system queries are made to the reference monitor.
- In particular, it ensures that all security-sensitive operations are authorized by the access enforcement mechanism.
- By a *security-sensitive operation*, we mean an *operation* on a particular *object* (e.g., file, socket, etc.) whose execution may violate the system's security requirements.
- For example, an operating system implements file access operations that would allow one user to read another's secret data (e.g., private key) if not controlled by the operating system.
- Labeling and transitions may be executed for authorized operations.

Authorization Module

- The core of the reference monitor is its ***authorization module***. The authorization module takes interface's inputs (e.g., process identity, object references, and system call name), and converts these to a query for the reference monitor's policy store.
- The challenge for the authorization module is to map the process identity to a subject label, the object references to an object label, and determine the actual operations to authorize (e.g., there may be multiple operations per interface).
- The protection system determines the choices of labels and operations, but the authorization module must develop a means for performing the mapping to execute the "right" query.
- For the open request above, the module responds to the individual authorization requests from the interface separately. For example, when a directory in the file path is requested, the authorization module builds an authorization query. The module must obtain the label of the subject responsible for the request (i.e., requesting process), the label of the specified directory object (i.e., the directory inode), and the protection state operations implied the request (e.g., read or search the directory).

Policy Store

- The ***policy store*** is a database for the protection state, labeling state, and transition state.
- An authorization query from the authorization module is answered by the policy store. These queries are of the form *{subject_label, object_label, operation_set}* and return a binary authorization reply.
- Labeling queries are of the form *{subject_label, resource}* where the combination of the subject and, optionally, some system resource attributes determine the resultant resource label returned by the query.
- For transitions, queries include the *{subject_label, object_label, operation, resource},* where the policy store determines the resultant label of the resource. The resource may be either be an active entity (e.g., a process) or a passive object (e.g., a file). Some systems also execute queries to authorize transitions as well.

What is a Secure Operating System?

- **Definition 2.5.** A *secure operating system* is an operating system where its access enforcement mechanism satisfies the *reference monitor concept*:
  - Complete mediation: all security security-sensitive operations are checked.
  - Tamper-proof: Cannot be modified by untrusted processes
  - Verifiable.

Reference Monitor Concept

- **Definition 2.6.** The ***reference monitor concept*** defines the necessary and sufficient properties of any system that securely enforces a mandatory protection system, consisting of three guarantees:

- **Complete Mediation**: The system ensures that its access enforcement mechanism mediates all security-sensitive operations.
- **Tamperproof**: The system ensures that its access enforcement mechanism, including its protection system, cannot be modified by untrusted processes.
- **Verifiable**: The access enforcement mechanism, including its protection system,“must be small enough to be subject to analysis and tests, the completeness of which can be assured”.
- That is, we must be able to prove that the system enforces its security goals correctly.

Complete Mediation

- ***Complete Mediation*** of security-sensitive operations requires that all program paths that lead to a security-sensitive operation be mediated by the reference monitor interface.
- The trivial approach is to mediate all system calls, as these are the entry points from user-level
- processes. While this would indeed mediate all operations, it is often insufficient.
- For example, some system calls implement multiple distinct operations. The open system call involves opening a set of directory objects, and perhaps file links, before reaching the target file.
- The subject may have different permission for each of these objects, so several, different authorization queries would be necessary.

Complete Mediation

- Also, the directory, link, and file objects are not available at the system call interface, so the interface would have to compute them, which would result in redundant processing (i.e., since the operating system already maps file names to such objects).
- But worst of all, the mapping between the file name passed into an open system call and the directory, link, and file objects may be changed between the start of the system call and the actual open operation (i.e., by a well-timed rename operation).
- This is called a time-of-check-to-time-of-use (TOCTTOU) attack [30], and is inherent to the open system call.

Tamperproof

- Verifying that a reference monitor is ***tamperproof*** requires verifying that all the reference monitor components, the reference monitor interface, authorization module, and policy store, cannot be modified by processes outside the system's trusted computing base (TCB).
- This also implies that the TCB itself is high integrity, so we ultimately must verify that the entire TCB cannot be modified by processes outside the TCB.
- Thus, we must identify all the ways that the TCB can be modified, and verify that no untrusted processes (i.e., those outside the TCB) can perform such modifications.

Tamperproof

- First, this involves verifying that the TCB binaries and data files are unmodified. This can be accomplished by a multiple means, such as file system protections and binary verification programs.
  - Note that the verification programs themselves must also be protected.
  - Second, the running TCB processes must be protected from modification by untrusted processes.
- Again, system access control policy may ensure that untrusted processes cannot communicate with TCB processes, but for TCB processes that may accept inputs from untrusted processes, they must protect themselves from malicious inputs, such as buffer overflows.

Verifiable

- Finally, we must be able to verify that a reference monitor and its policy really enforce the system security goals.
- This requires verifying the correctness of the interface, module, and policy store software, and evaluating whether the mandatory protection system truly enforces the intended goals.
- First, verifying the correctness of software automatically is an unsolved problem. Tools have been developed that enable proofs of correctness for small amounts of code and limited properties, but the problem of verifying a large set of correctness properties for large codebases appears intractable.

Verifiable

- In practice, correctness is evaluated with a combination of formal and manual techniques which adds significant cost and time to development.
- As a result, few systems have been developed with the aim of proving correctness, and any comprehensive correctness claims are based on some informal analysis (i.e., they have some risk of being wrong).
- Second, testing that the mandatory protection system truly enforces the intended security goals appears tractable, but in practice, the complexity of systems makes the task difficult. Because the protection, labeling, and transition states are immutable, the security of these states can be assessed.

Assessment Criteria

- For each system that we examine, we must specify precisely how each system enforces the reference monitor guarantees in order to determine how an operating system aims to satisfy these guarantees.
- In doing this, it turns out to be easy to expose an insecure operating system, but it is difficult to define how close to “secure” an operating system is. Based on the analysis of reference monitor guarantees above, we list a set of dimensions that we use to evaluate the extent to which an operating system satisfies these reference monitor guarantees.

Complete Mediation

- **Complete Mediation**: How does the reference monitor interface ensure that all security-sensitive operations are mediated correctly?

- In this answer, we describe how the system ensures that the subjects, objects, and operations being mediated are the ones that will be used in the security-sensitive operation. This can be a problem for some approaches, in which the reference monitor does not have access to the objects used by the operating system.

Tamperproof

- **Tamperproof**: How does the system protect the reference monitor, including its protection system, from modification?

- In modern systems, the reference monitor and its protection system are protected by the operating system in which they run.
- The operating system must ensure that the reference monitor cannot be modified and the protection state can only be modified by trusted computing base processes.

Verifiable

- 6. **Verifiable**: What is basis for the correctness of the system's trusted computing base?

- We outline the approach that is used to justify the correctness of the implementation of all trusted computing base code.

QUESTIONS?