>

DI-FCT-UNL
Segurança de Redes e Sistemas de Computadores

Mestrado Integrado em Engenharia Informática
2º Semestre, 2018/2019

- *Access Control*
- *Operating System Security*

# Outline

- **Access control**
  - Principles of Access Control Models
  - Access Control Policy Models: MAC, DAC, RBAC, ABAC
- **Operating System Security**
  - OS security background
  - Case study: Linux, Windows
  - Virtual machines

# Access Control

- **Topics**
  - Principles of Access Control Models
  - Access Control Policy Models: MAC, DAC, RBAC, ABAC
  - Mandatory Access Control (MAC)
  - Discretionary Access Control (DAC)
    - Case Study Example:  Unix File System
    - setUID programs
  - Role Based Access Model (RBAC)
    - Case Study Example: RBAC in a Banking System
  - Attribute-Based Access Control (ABAC)

# Outline

- **Access control topics**
  - Principles of Access Control Models
  - Access Control Policy Models: MAC, DAC, RBAC, ABAC
  - Mandatory Access Control (MAC)
  - Discretionary Access Control (DAC)
    - Case: Unix File System
  - Role Based Access Model (RBAC)
    - Example: RBAC in a Banking System
  - Attribute-Based Access Control (ABAC)

# What is Access Control ?

- “The prevention of unauthorized use of a resource, including the prevention of use of a resource in an unauthorized manner“

- Or (as defined in RFC 4949): “The measures that implement and assure security services in a computer system, particularly those that assure access control service.”

- A central mechanism of computer security
  - Related to the materialization of the Access-Control Security Property
  - (Remember the OSI X.800 Framework and Security Services and Mechanisms Typology)

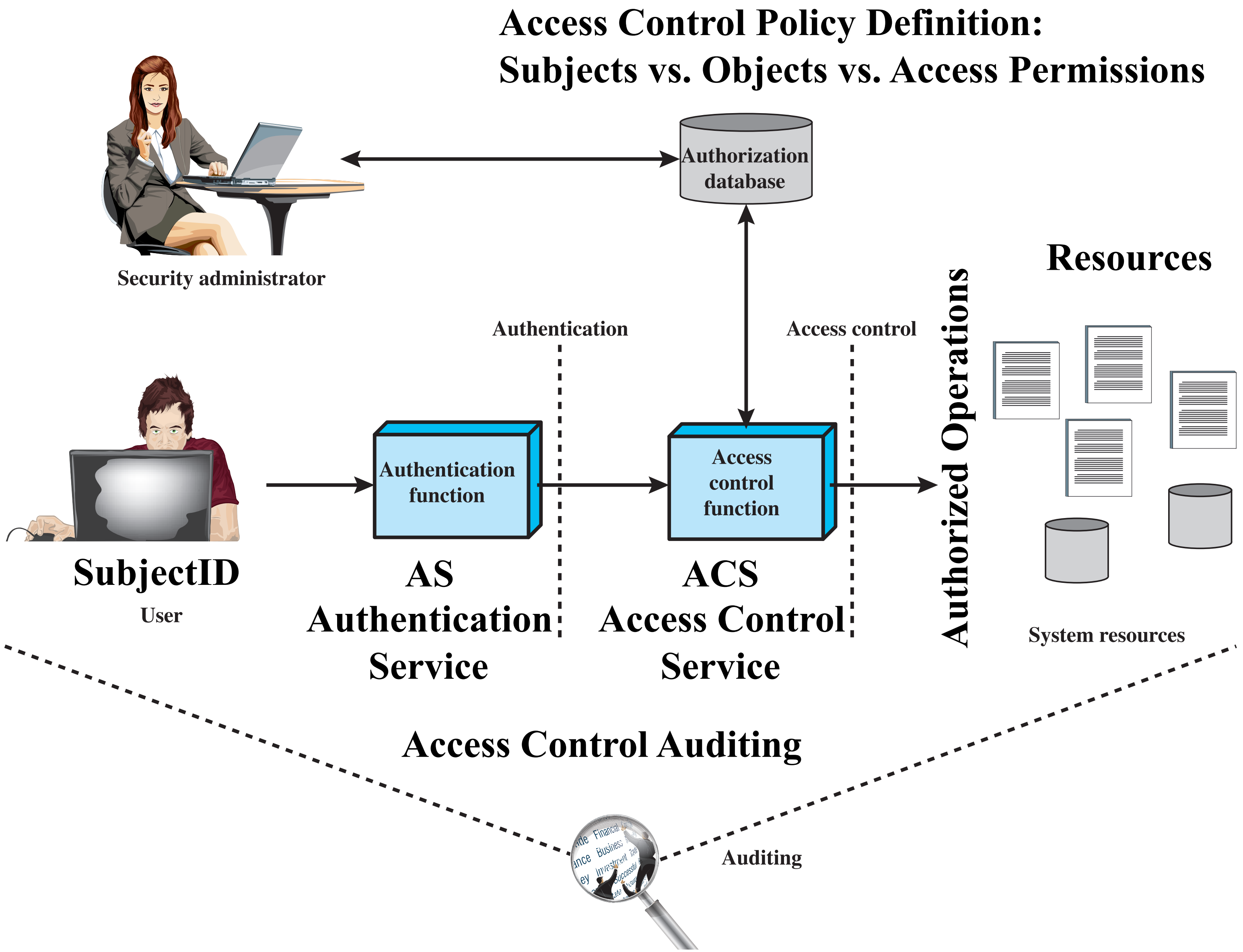
# Access Control: Assumptions

- Assume principals, subjects or users (Principals – PrincipalIDs, SubjectIDs, UserIDs, … ) or groups (aggregated Principals or Subjects as GroupIDs)
  - Authenticate to system
    - Access control is applied over (supposed) authenticated subjects or principals
      - Relates to the need of Authentication Services
    - **But … authentication and access control are two different security services (separation of concerns) use very different mechanisms !!!**
    - **Implemented by different processes (ex., different machines in a Data Center)**

# Access Control: Assumptions

- Access control services: assignment of **access rights (or permissions)** to access certain resources on system
  - **Permission to access a resource** is also called **authorization**
  - An Access Control Service requires the definition of Access Control Policies
  - Verification and enforcement made via an **Access-Control Service Reference Monitor**
    - Set and verification of access control enforcements (as access-control policy definitions) providing the related control guarantees
    - **Access-Control Reference Monitor: a trusted process that verify/monitors/apply the access control enforcements**
      - » **allowing or denying the access**
      - » **for the execution of specific operations (OPi) on resources (Rj) intended by well-defined (and previously authenticated) principals (SujectIDs)**

# Access Control Principles



Access Control Policy Definition:
Subjects vs. Objects vs. Access Permissions

Authorization database

Security administrator

Resources

Authentication

Access control

Authentication function

Access control function

Authorized Operations

SubjectID

User

AS
Authentication
Service

ACS
Access Control
Service

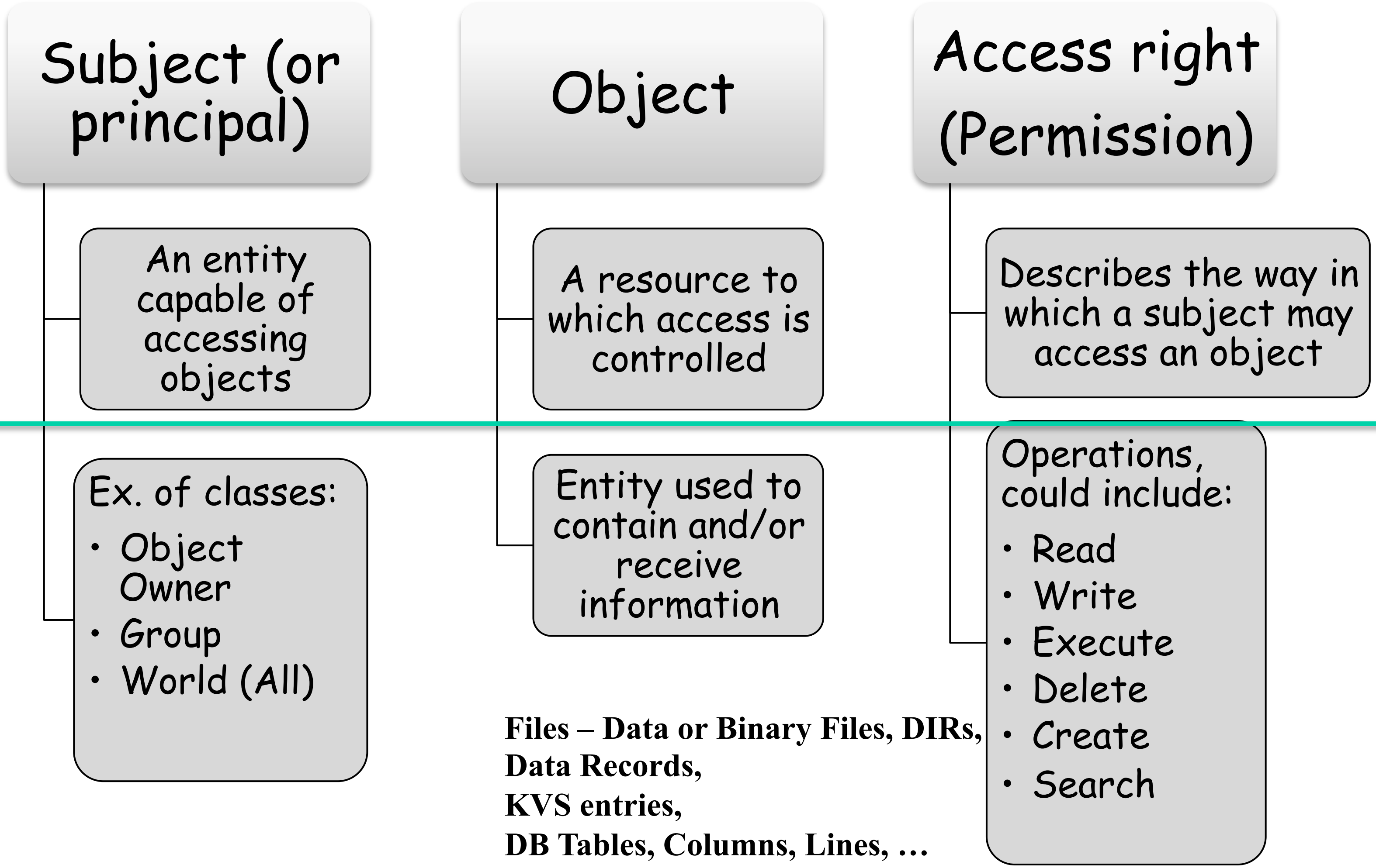
System resources

Access Control Auditing

Auditing

# Access Control Elements

- **Subjects** (or Principals), **Objects** and Access Rights or **Permissions**



Subject (or principal)

An entity capable of accessing objects

Ex. of classes:
- Object Owner
- Group
- World (All)

Object

A resource to which access is controlled

Entity used to contain and/or receive information

Access right (Permission)

Describes the way in which a subject may access an object

Operations, could include:
- Read
- Write
- Execute
- Delete
- Create
- Search

Ex: Concretization

**Files – Data or Binary Files, DIRs,**
**Data Records,**
**KVS entries,**
**DB Tables, Columns, Lines, …**
**Devices ,…**

Design criteria in Access Control Requirements (1)

- Base Access control design principles:

# Possible Access Control Drawbacks

- Important issue: possible limitations of coarse-grain access control enforcements
  - Devices / Sensors / Data in smartphones, tablets (ex., Android Access Control Ecosystem)
    - Two only permissions: ALL or NOTHING
- What about involved SubjectIDs vs. Users
    - Only one user: USER is also the SYS ADMIN
    - Can do everything ! She/he installs and executes everything and access to a large number (or all) resources
- Access Control Monitoring Level (in a TCB at Application-level or Middleware Level, out of the Base OS Foundations or HW)

# Issues in smartphones/tablets …

- Problem: Are current smartphones/tablets ready to be used in BYOD paradigms, running sensitive and no-sensitive apps in the same execution eco-system ?
- Different issues involved, but access control is one of the most prevalent problems
  - Lack of proper TRUSTED EXECUTION ENVIRONMENT and complete approach of Access Control System Design Principles
  - Lack of Fine-Grain Access Control
  - No separation of roles: SYSADMIN and USER
  - Too High Level Trust Computing Base Assumptions:
    - Ex. in ANDROID Devices: OS, Device Drivers, Delvik VM, Application Level Support Libraries
  - A situation where "the user" … can be easily "the adversary" !

# Relevant Design Principles in Designing Access Control Services

# PoLP – Principle of the least privilege

- Also known as the principle of the minimum privilege or least minimum authority
  - Requires that in a particular abstraction layer of a computing environment, every module (interface, endpoint, process, program, named user) depending on the subject or principal level) must be able to access only the information, processing or resources hat are strictly necessary for its legitimate purpose and nor much than this
  - Default: (the minimum privileges) for a certain level of principal authentcation

# Non Escalading Privileges

- The principle of non-escalading privileges with the same authentication proofs and subject security level
  - Escalation of privileges must require the control and scrutiny of improved authentication level guarantees associated to the improvement of the security level of the involved subject

# Deputy Attacks Avoidance

- The principle of avoidance of deputy attacks
  - Privileges are applied to specific subjects and cannot be associated to other subjects without the proper delegation control

# Proper Course Grain Control

- The principle of the proper course grain granularity
  - Privileges must be scrutinized and controlled at the proper level of well-defined granularity
    - Fine grain access control … is better

# Violation of base principles

- Important issue: limitations of coarse-grain access control enforcements combined with privilege escalading problems due to the violation of the discussed principles
- Consequences:
    - Lots of access control problems
    - Confused Deputy Problem: a computer program that is innocently fooled by some other party into misusing its authority.
        - Ex., Use of the Video Camera. Microphone, GPS location, SD card, etc. … by a App with given authorization as resources that will be used illicitly by another installed App without authorization for that
            - » One of the more prevalent attacks on current ANDROID OS devices
        - All are examples of the violation/limitation of the PRINCIPLE OF THE LEAST PRIVILEGE in Access Control System Design Assumptions !

# Access control must be established by design

> *See also bibliography for more details*

- Fine and coarse specifications
  - Grain of Access Control Enforcements
  - Fine-grained specifications allow access regulated at the level of individual fields / records in files, etc;
  - Each individual access by a user rather than a sequence of accesses.
  - System administrators should also be able to choose coarse-grain specification for classes of resource access
    - Segmentation and delimitation of the operations scope
- Principle of Least Privilege specifications
  - Relevant to define the default settings for each authorization level
  - Tends to limit damage that can be caused by an accident, error, or unauthorized act, as a default-behavior

## Design criteria in Access Control Requirements

- Other relevant design criteria for access control specifications by design

# Reliable Inputs and Separation of Duties

- Reliable input
  - Assumes that a user is authentic (previously authenticated); thus, an authentication mechanism is needed as a front end to an access control system.
  - Any user inputs to the access control system must also be reliable (and supposed that are inputs originated by authenticated correct users)
- Separation of duty
  - Should divide steps in a system function among different individuals, so as to keep a single individual from subverting the process.

# Default settings: Open vs Closed Access Control Policies

- Open vs. closed policies (and security vs. usability tradeoffs)
  - A closed policy only allows accesses that are specifically (strictly) authorized: better for security
  - An open policy allows all accesses (everything is authorized) except those expressly prohibited: better for usability

# Other design issues

- Policy combinations, consistency and conflict resolution
  - May apply multiple policies to a given class of resources
  - Need a consistent well-defined procedure for solving conflicts between non-consistent policies

- Administrative enforcement policies and cardinality of targeted subjects
  - To specify who can add, delete, or modify authorization rules, and also need access control and other control mechanisms to enforce these administrative policies.

# Access Control vs. Auditing

- Auditing is a relevant function related to Access Control Policy Enforcement and Management
- Allows for performing posteriori analysis of all authorized or no-authorized requests and access activities of subjects in a system
- All operation requests and activities on resources must be logged for:
  - Acting as a deterrent
  - Identifying/Detection attempted or actual violations
  - Identifying/Detection of flaws
  - Preventing authorized users from attempted or actual violations and misusing privileges
  - For other management functions: alarmist-functions, statistics, security forensics, …

# Outline

- **Access control topics**
  - Principles of Access Control Models
  - Access Control Policy Models: MAC, DAC, RBAC, ABAC
  - Mandatory Access Control (MAC)
  - Discretionary Access Control (DAC)
    - Case: Unix File System
  - Role Based Access Model (RBAC)
    - Example: RBAC in a Banking System
  - Attribute-Based Access Control (ABAC)

# Base Access Control Policies

- Discretionary access control **(DAC)**
  - Controls access based on the identity of the requestor and on access rules (authorizations) stating what requestors are (or are not) allowed to do
  - the data owner determines who can access specific resources.
- Mandatory access control **(MAC)**
  - Controls access based on comparing security labels with security clearances
- Role-based access control **(RBAC)**
  - Controls access based on the roles that users have within the system and on rules stating what accesses are allowed to users in given roles
- Attribute-based access control **(ABAC)**
  - Controls access based on attributes of the user, the resource to be accessed, and current environmental conditions
  - Access rights are granted to users through the use of policies which evaluate possible combined attributes (user attributes, resource attributes and environment conditions)

# Outline

- **Access control topics**
  - Principles of Access Control Models
  - Access Control Policy Models: MAC, DAC, RBAC, ABAC
  - Mandatory Access Control (MAC)
  - Discretionary Access Control (DAC)
    - Case:  Unix File System
  - Role Based Access Model (RBAC)
    - Example: RBAC in a Banking System
  - Attribute-Based Access Control (ABAC)
  - Complementary issues

# MAC Policy

- Access control enforcement under the rigid control of the system

- MLS (Multi Level Security) Model: perhaps the most popular mandatory access control approach:
  - Access fully controlled based on security levels assigned to objets and subjets
    - Objects have fixed security levels
    - Subjects have fixed security levels
    - MLS provide "oneway" flow relations in the lattice of security levels

# MAC Policy Enforcement

- MAC Level Enforcement:
  - Example of OS access control policies and separation of duties:
    - Kernel-Based Mandatory Access Control
      - Only code running in supervised mode can access/manage OS level system resources
      - Access only possible by code executed beyond the System Calls (Calls from running Processes, executing with system-level privilege)
  - Could we consider MAC policies as appropriate at user space level (ex., Permissions in a File-System Service) ?
      - In a MAC policy, users couldn't have much freedom to determine who can access to their own files.

# MAC policy concretizations: OS (1)

- Refers to a type of access control by which an OS constrains the ability of a *subject* or *initiator* to access or generally perform some sort of operation on an *object* or *target*, directly controlled by the OS kernel in supervised running model
- In practice, a subject is usually a process or thread; objects are constructs such as files, directories, TCP/UDP ports, shared memory segments, IO devices, etc.
- Subjects and objects each have a set of security attributes.
- Operation
  - Whenever a subject attempts to access an object, an authorization rule directly defined and enforced by the operating system kernel examines these security attributes and decides whether the access can take place.
  - Any operation by any subject on any object is tested against the set of authorization rules (aka OS *policy*) to determine if the operation is allowed

# MAC policy concretizations: DBMS (2)

- A DBMS (Data Base Management System) in its access control service, can apply mandatory access control;
- Implemented by the DBMS runtime support environment
- In this case, the objects are tables, views, procedures, etc. with certain operations only accessible by pre-defined DB administrator subjects

- But, what if we want to support users in defining access-control delegation policies on "owned" tables, attributes, etc …
  - Lack of flexibility for other users accessing the DB

# Another MAC based model approach

- Use of security labels (corresponding to classifications of security levels) reflecting the sensitivity of information contained in an object
  - Examples:

  TOP SECRET (TS), SECRET (S), CONFIDENTIAL (C), UNCLASSIFIES (U) forming an ordered SET: TS > S > C > U

  Each level is set to dominate itself and all the other levels below in the SET HIERARCHY

- Use of clearances: security levels associated to subjects, reflecting:
- The user's trustworthiness not to disclose sensitive information to other users that are not cleared to access it

# Outline

- **Access control topics**
  - Principles of Access Control Models
  - Access Control Policy Models: MAC, DAC, RBAC, ABAC
  - Mandatory Access Control (MAC)
  - ▶ Discretionary Access Control (DAC)
    - Case: Unix File System
    - setUID programs
  - Role Based Access Model (RBAC)
    - Example: RBAC in a Banking System
  - Attribute-Based Access Control (ABAC)

# DAC policy

- DAC level Enforcement:
  - The data owner determines who can access specific resources.
  - For example, a system administrator may create a hierarchy of files to be accessed based on certain default permissions for certain users, groups of users and allowed operations. Then owners can rewrite these permissions

  - We say that this access control model is under the discretion of the user
    - More flexible and popular than the MAC policy enforcement model
    - But DAC doesn't't address real assurance  on the flow if information in the system because DAC models don't impose any restriction on the usage of information/resources if the user has the right privilege
      - Opens the door for possible easy bypass practices under the user freedom, for example if a user can read an object she/he can pass (copy) it to another user not having the read privilege overt the same object

# DAC policy and Unix File System

- Example of UNIX File system permissions:
  - Read, Write, Execute Permissions
  - Principals: Owner Principals (UserIDs), GroupIDs and All (Others)
  - DAC definitions: defined and managed by the resource owners and owners can pass the owning to other principals
  - Permissions scrutiny by a Kernel-Based Access Control Monitor (running as Module in supervised mode), deciding on each operation that a process (running with a correspondent efective UID - eUID) intends to apply on a resource (files, directories, device-drivers, sockets, message-queues, etc)
    - Remember: in UNIX everything (all the resources) are accessed as "file-system descriptors containing the access control attributes)

# DAC concretization

- By different schemes in which an owner entity may enable another entity to access some resource to perform some operation

- For example, can be provided **using an access control matrix**
  - One dimension consists of identified subjects that may attempt data access to the resources
  - The other dimension lists the objects that may be accessed
- Each entry in the matrix indicates the access rights of a particular subject for a particular object

# DAC and Access Control Matrix

Access Control Matrix represents a conceptual model specifying rights (authorizations or permissions) that each subject possesses to access (to operate) each object

- Clearly separates Authentication from Access Control
- The defined access control enforcements are then ensured by special processes: reference monitors
  - See the Access Control Matrix as a “private” data-structure only managed by the Reference Monitor

**OBJECTS**

| | | **File 1** | **File 2** | **File 3** | **File 4** |
|---|---|---|---|---|---|
| | **User A** | **Own**<br>**Read**<br>**Write** | | **Own**<br>**Read**<br>**Write** | |
| **SUBJECTS** | **User B** | **Read** | **Own**<br>**Read**<br>**Write** | **Write** | **Read** |
| | **User C** | **Read**<br>**Write** | **Read** | | **Own**<br>**Read**<br>**Write** |

# Access control matrix

- Example access rights/modes:
  - For files: read, write, execute, own
    - Typically implemented by the OS File System functions
  - For Bank accounts: inquiry, debit, credit, profile, …
    - Typically implemented at application-level
- Typically the access control matrix is sparsed and hence not implemented as a real matrix, but as:
  - **Access control lists**
  - **Capabilities**

# Access control lists

- Each object associated to an ACL, corresponding to store the access matrix by column
- Looking at an object's ACL, it easy to determine the modes of access (corresponding to allowed operations and permissions for the defined subjects)
- Easy to revoke permissions

- Example: in UNIX we can create ACLs on files or folders (directories) in the file system
  - System calls: getfacl(), setfacl()

- Problem: for scalability purposes will be difficult to find all accesses (permissions) established for a certain subject

- To reduce list lengths, we can use groups (as a subject entity) instead of using individual subjects and then we can apply permissions to subject-groups

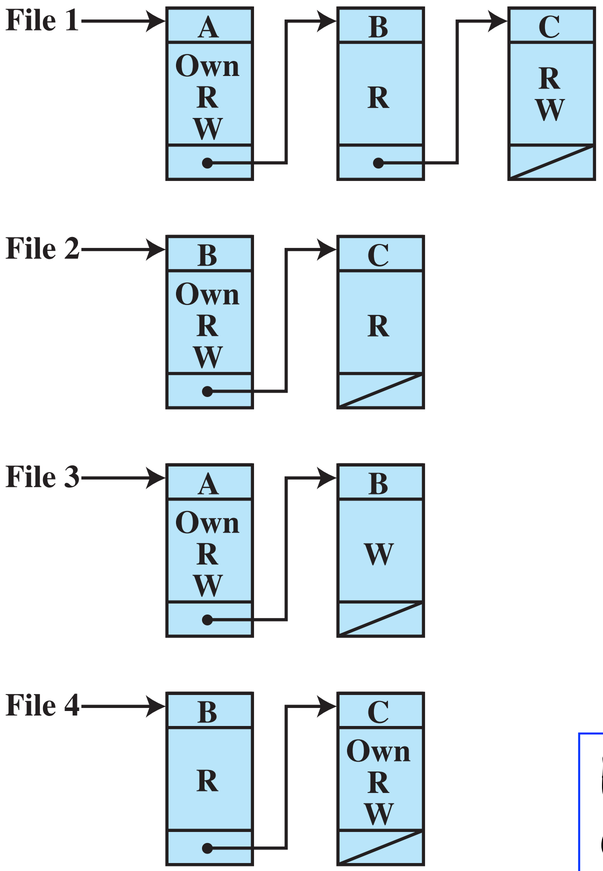
# Capabilities

- The capability model is a dual approach to ACLs
  - Each subject is associated with a list (the capability list)
  - The capability list of a subject has the list of objects for which the subject has some kind of permission access
    - Advantage: easy to find permissions for the given subjects or to revoke permissions for subjects
    - Drawback: difficult to find all subjects that have permissions for a given object

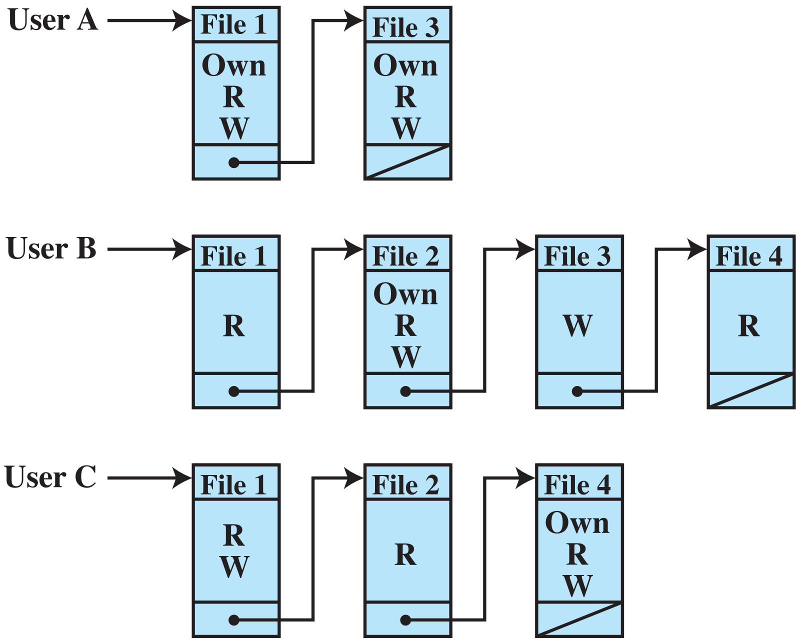
      => Typical Operating System Strategies are based in ACLs

# ACLs vs Capability Lists

## Access Control Lists



## Capability Lists



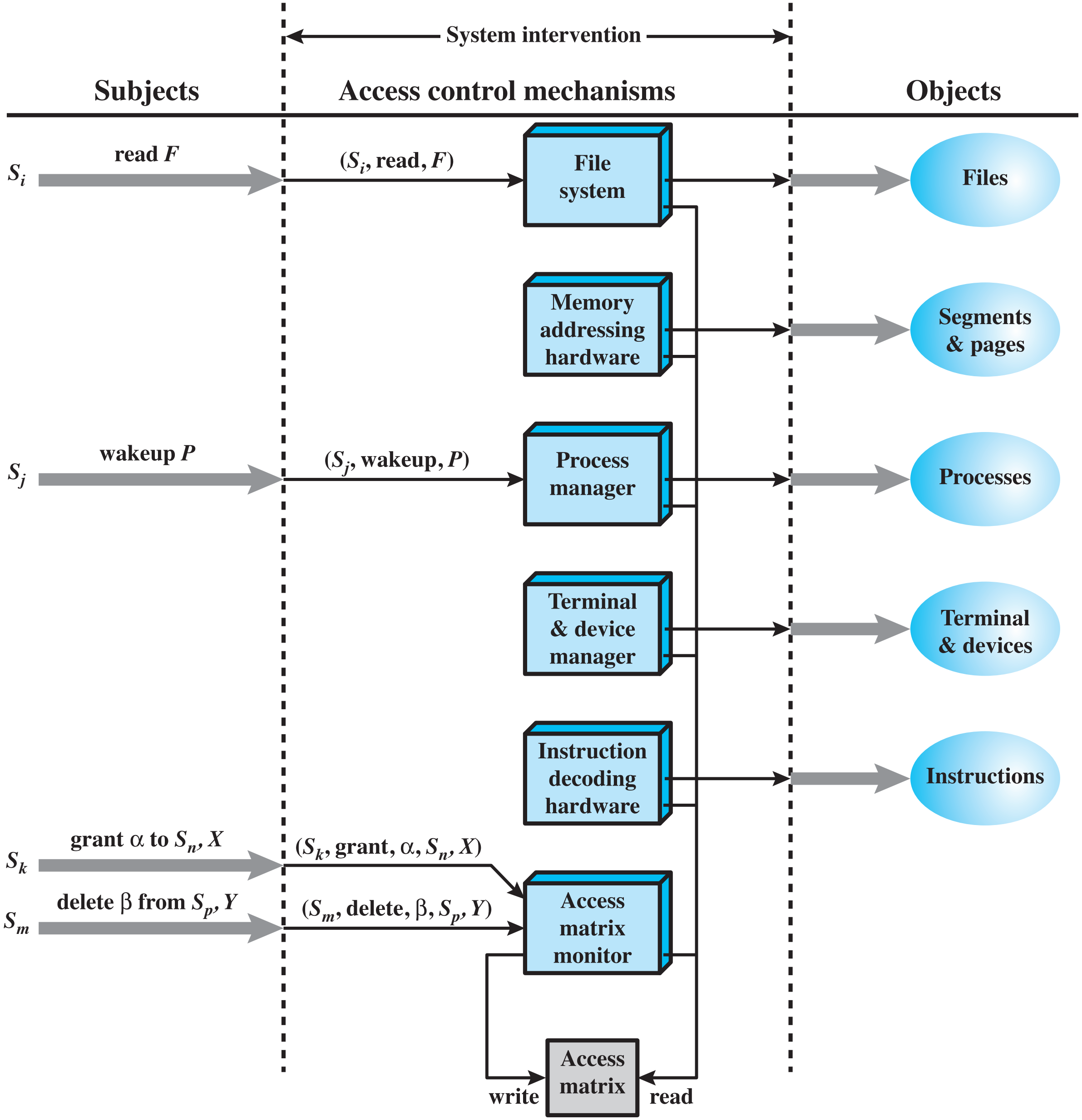
Example:
Object: Files
Subjects: Users

# Protection Domains

- A protection domain is a set of objects with associated access rights
- In access matrix view, each row defines a protection domain
  - Not necessarily just a user
  - May be a limited subset of user's rights
  - Applied to a more restricted process

- The association between a process and a domain may be static or dynamic
  - Ex., during a process execution it may require different access rights for each procedure
  - In general: minimization of access rights overtime (controlled in each protection domain)

# Access Control Function



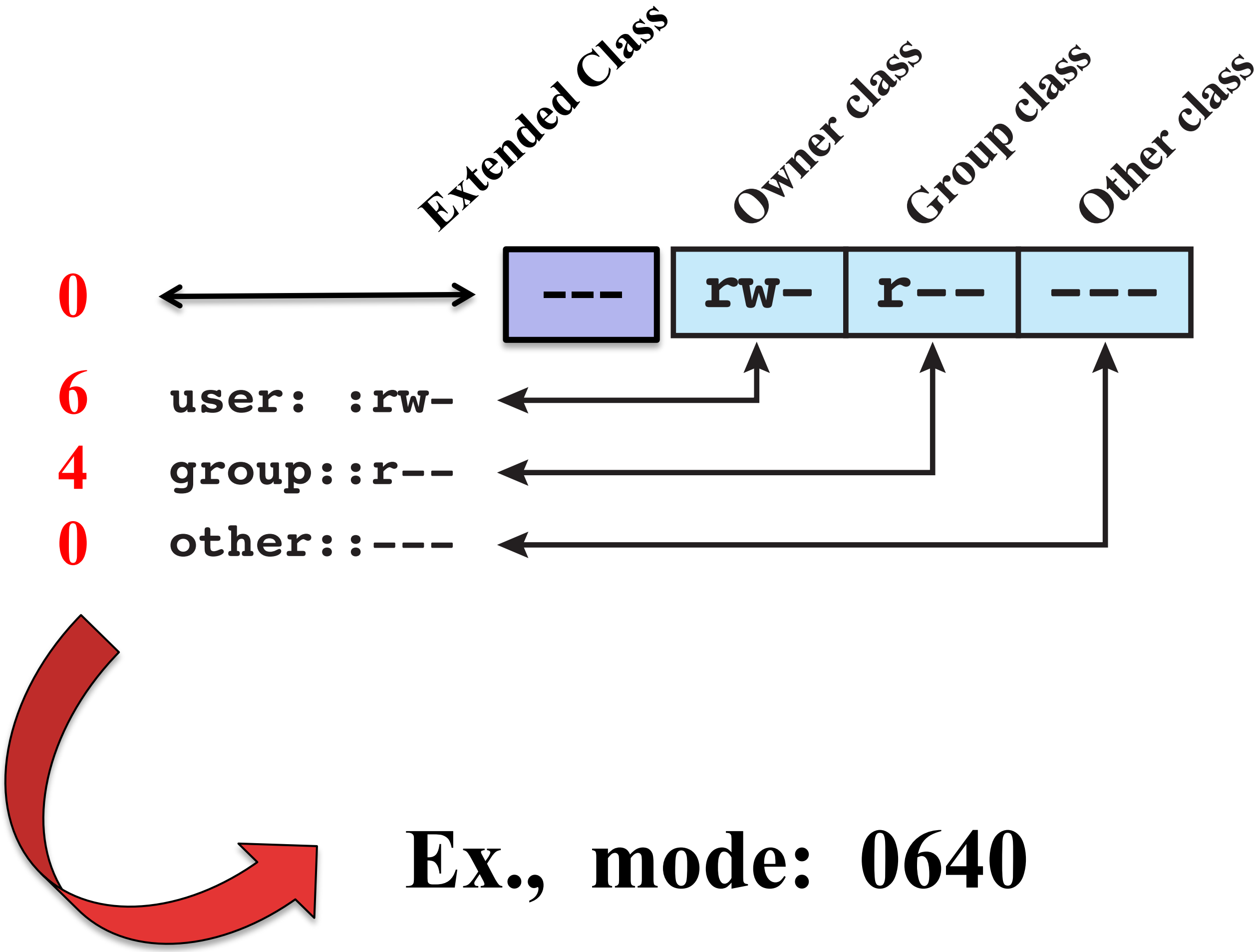
# DAC and UNIX File System Concepts

- UNIX files administered using inodes
  - Control structure with key info on file
    - Attributes, permissions of a single file
  - May have several names for same inode
  - Have inode table / list for all files on a disk
    - Copied to memory when disk mounted

- Directories form a hierarchical tree
  - May contain files or other directories
  - Are a file of names and inode numbers

# UNIX File Access Control

- Expression of DAC in the UNIX File System



Extended Class

Owner class

Group class

Other class

---

rw-

r--

---

0

6 user: :rw-

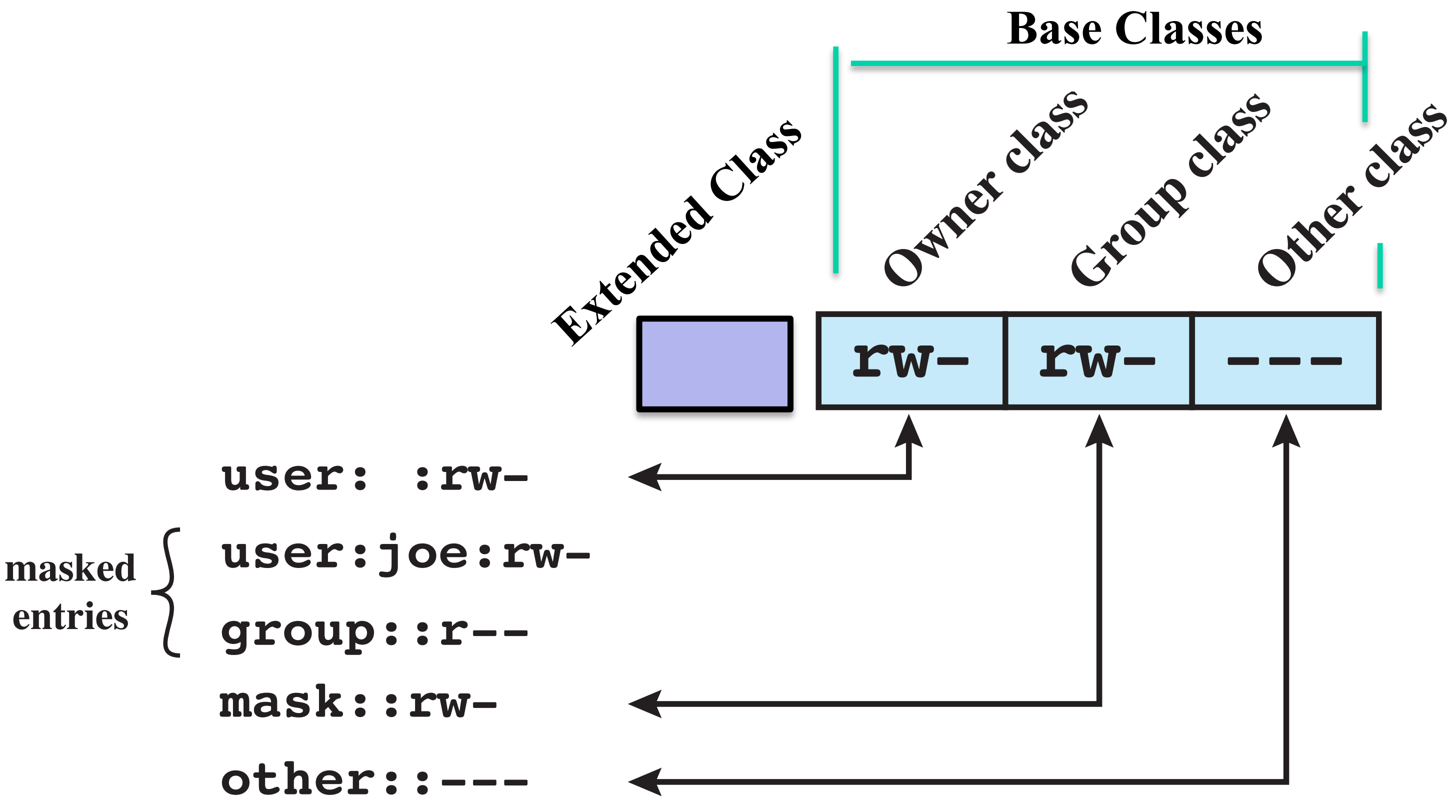
4 group::r--

0 other::---

**Ex., mode: 0640**

# Extended File Access Control

- Expression of DAC in the UNIX File System



# UNIX File Access Control

- "set user ID"(SetUID) or "set group ID"(SetGID)
  - The system temporarily uses rights of the file owner / group in addition to the real user's rights when making access control decisions
  - Enables privileged programs to access files / resources not generally accessible
- Sticky bit
  - on directory limits rename/move/delete to owner
- Superuser
  - is exempt from usual DAC restrictions

# Examples

- See *chown* and *chgrp* in UNIX file system
- **chown** -- change file owner and group
  - **chown [-fhv] [-R [-H | -L | -P]]** owner[:group] file ...
  - **chown [-fhv] [-R [-H | -L | -P]]** :group file ...
- **chgrp** -- change group
  - **chgrp [-fhv] [-R [-H | -L | -P]]** group file ...

# Examples

- See *chmod* in UNIX file system
  - **chmod** [**-fv**] [**-R** [**-H** | **-L** | **-P**]] mode file ...
  - **chmod** [**-fv**] [**-R** [**-H** | **-L** | **-P**]] [-a | +a | =a] ACE file ...
  - **chmod** [**-fhv**] [**-R** [**-H** | **-L** | **-P**]] [**-E**] file ...
  - **chmod** [**-fhv**] [**-R** [**-H** | **-L** | **-P**]] [**-C**] file ...
  - **chmod** [**-fhv**] [**-R** [**-H** | **-L** | **-P**]] [**-N**] file ...
- Access control modes (can combine them):
  - Modes: 4000, 2000, 1000
    - for setting eUID on owner, group and sticky-bit respectively
  - Modes: 0400, 0200, 0100 for w r x to the owner
  - Modes: 0040, 0020, 0010 for w r x to the group
  - Modes: 0004, 0002, 0001 for w r x for others

# Extensions: UNIX Access Control Lists

- Many UNIX-based distributions support ACLs as extended mechanism
  - Can specify any number of additional users / groups and associated rwx permissions
  - ACLs are optional extensions to the standard permissions
  - Group permissions also set max ACL permissions
- When access is required
  - Select most appropriate ACL
    - owner, named users, owning / named groups, others
  - Check if have sufficient permissions for access

# MAC and DAC Enhanced Linux Distributions

- See more on different MAC evolved mechanisms for security enhanced implementations on UNIX/LINUX distributions, SUSE Linux-App Armor, Tomoyo Linux, Trusted Solaris, Windows (since 2008), Mac OS-X and others
- Ex., summary on:
- https://en.wikipedia.org/wiki/Mandatory_access_control

# Outline

- **Access control topics**
  - Principles of Access Control Models
  - Access Control Policy Models: MAC, DAC, RBAC, ABAC
  - Mandatory Access Control (MAC)
  - Discretionary Access Control (DAC)
    - Case:  Unix File System
  - ▶ Role Based Access Model (RBAC)
    - Example: RBAC in a Banking System
  - Attribute-Based Access Control (ABAC)

# RBAC based policies

- Neither DAC nor MAC approaches satisfy the need of "organizational or business" models, that require more flexibility for access control management:
  - MAC: rise form rigid environments, like those of strict hierarchies, like those of military hierarchies
  - DAC: rise from cooperative yet environments with user autonomy (ex., academic users/ researchers)

# RBAC (Role Based Access Control) policy

- ROLE: a set of actions and responsibilities associated to a particular organizational activity

  Instead of specifying all the permissions for each subject (ex., a single user), the permissions are specified for ROLES

- Then, RBAC allows access based on the defined roles (ex., job titles or organizational job functions)
  - Require an implicit controlled mapping (association) between subjects and such roles, as organizational responsibilities
  - Users are given authorization to adopt roles (may or may not be allowed to play with more than one role) but in each access she/he is playing with one specific role
  - A user playing in a certain role is allowed to execute operations with the permissions for which the role is authorized
  - Usually, the mappings must be defined by previously defined subjects or roles, following the principle of non-escalading privileges

# RBAC Advantages

- RBAC largely eliminates discretion when providing access to objects. For example, a human resources specialist should not have permissions to create network accounts; this should be a role reserved for network administrators.

- In some sense we have the best of the MAC and DAC models:
  - Allow the specification of permissions to be granted to users (or groups) on objects, like in the DAC model
  - But we also have the possibility to specify restrictions (like in the MAC model) on the assignment or on the use of such restrictions

# RBAC in summary

- Simplification of authorization management
- Hierarchical roles further simplify by allowing generalization and specialization
- Allows for a proper adaptation of different roles to operate at the least privilege
- Promotes the principle of separation of duties, preventing the misuse of the system
- Can be used with more flexibility, by allowing eh specification of permissions over object classes and not only to individual objects

# Outline

- **Access control topics**
  - Principles of Access Control Models
  - Access Control Policy Models: MAC, DAC, RBAC, ABAC
  - Mandatory Access Control (MAC)
  - Discretionary Access Control (DAC)
    - Case:  Unix File System
  - Role Based Access Model (RBAC)
    - Example: RBAC in a Banking System
  - ▶ Attribute-Based Access Control (ABAC)

# ABAC (Attribute Based Access Control)

- An access control model whereby access rights are granted to users through the use of policies which evaluate attributes (user attributes, resource attributes and environment conditions)
  - We can imagine context-aware attributed for specific ABAC models: Time-leasing conditions, Location Validity, Operation-Flow Controls, Behavioral Biometric Usage Conditions, …

# Other AC policies (1)

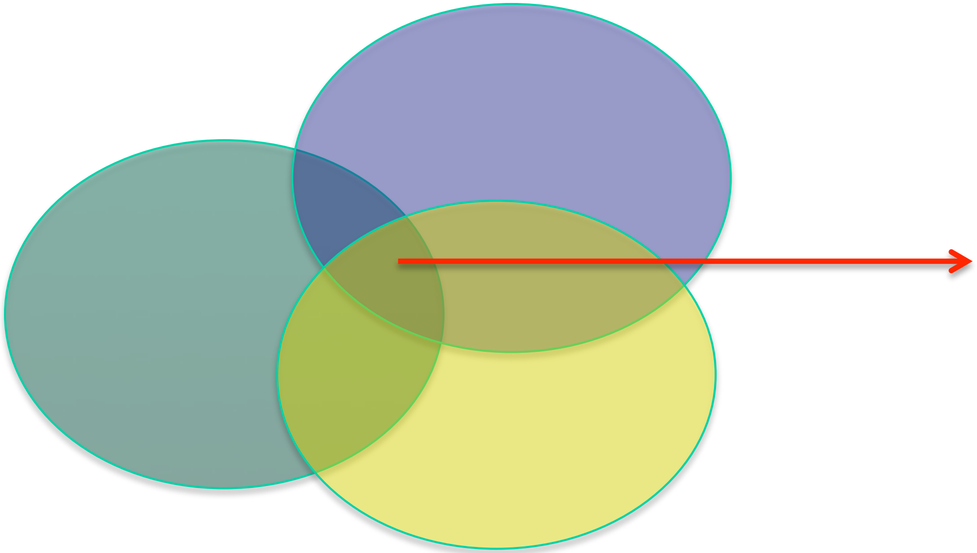
- Possible variants are sometimes defined with other designations (classified by different authors as access control policy models). Examples include:
  - **HBAC – History Based Access Control**
    - Access is granted or declined based on the real-time evaluation of a history of activities of the inquiring party, e.g. behavior, time between requests, content of requests or state-machine of operation-flows.
  - **IBAC – Identity Based Access Control**
    - In such policies network administrators can more effectively manage activity and access based on specific individual needs.
  - **OrBAC – Organization-Based Access Control**
    - OrBAC model allows the policy designer to define a security policyfor organizational or business functions independently of the implementation. Usually, we can map on designed RBAC and ABAC restrictions
  - **Context-Based Access Control (particular instances of ABAC)**
    - When permissions are identified by usage context (ex., determined by temporal, spatial, device-access type or contextual profiling context factors, or a combination of such type of factors)

# Other AC policies (2)

- Possible variants are sometimes defined with other designations (classified by different authors as access control policy models). Examples include:
  - **RAC - Rule Based Access Control**
    - RAC methods are defined largely as context-based access control. Example of this would be only allowing students to use the labs during a certain time of day.
    - Some overlaps with ABAC and/or RBAC
  - **ResBAC – Responsibility Based Access Control**
    - Information is accessed based on the responsibilities assigned to an actor or a business role
    - Some overlaps ABAC and/or RBAC and/or OrBAC

# Multiple policy models

- AC policies are not necessarily exclusive in a system design
  - Can be combined for a more suitable (and complex) protection system
  - In such combinations, permissions are usually applied by determining the defined interceptions



Interception of multiple combined access control models

# AC administration

- Ruled by Administrative policies that determine who is authorized to create, modify or remove the allowed access permissions
  - In a MAC model: by a security administrator
  - In DAC or RBAC models: there are possibly many types of administrative policies under different administrative entities
    - Ex., by the segregation of administrative domains and segregated system administrators

# Conclusions

- **Discussed access control topics**
  - Principles of Access Control Models
  - Access Control Policy Models: MAC, DAC, RBAC, ABAC
  - Mandatory Access Control (MAC)
  - Discretionary Access Control (DAC)
    - Case: Unix File System
  - Role Based Access Model (RBAC)
    - Example: RBAC in a Banking System
  - Attribute-Based Access Control (ABAC)

Suggested reading: W, Stallings and L. Brown,
Computer Security – Principles and Practice, 3rd Ed., Prentice Hall, 2015
Chap 4

- OS Level Security

# SO Security Background

- Computer client and server systems are central components of the IT infrastructure for most organizations.
  - The client systems provide access to organizational data and applications, supported by the servers housing those data and applications.

- Given that most large software systems will almost certainly have a number of potential security weaknesses (SW vulnerabilities, installation of malicious software, …
  - it is currently necessary to manage the installation and continuing operation of these systems to provide appropriate levels of security despite the expected presence of these potential vulnerabilities.
  - SW Security requires a "by design approach": more on the MIEI Software Security Course

- **But we may be able to use SW systems with SO Security baseline assumptions**

# SO Security Background

- **But we may be able to use SW systems with SO Security baseline assumptions, …**
- ***Answering to these questions:***
  - How to provide systems security as a hardening process that includes:
    - Planning
    - installation,
    - configuration,
    - update,
    - And maintenance

    **of the operating system and the OS packaged key applications in use**

# SO Security Background

- Approach detailed in [NIST08] and summarized in the provided course bibliography

Covered issues:

- A set of OS relevant topics
  - Illustrative aspects on Linux and Windows systems in particular
  - We conclude with a discussion on securing virtualized systems, where multiple virtual machines may execute on the one physical system.

# Topics

- Base OS Security
  - OS Security Layers
    - Typical measures in OS Security Management
  - Security Maintenance
  - Linux/Unix Security Issues
  - Windows Security Issues
  - Role of Virtualization

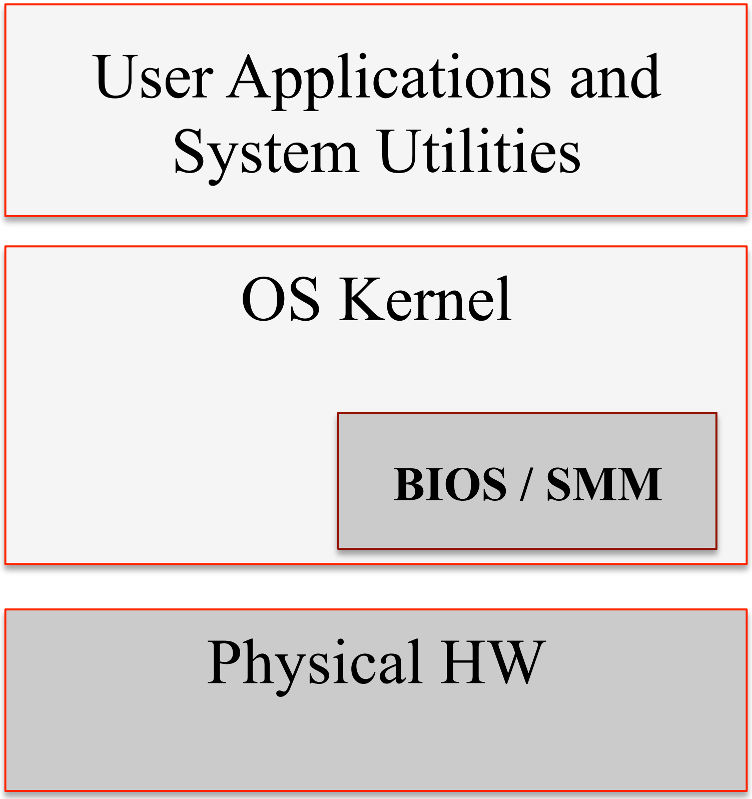
# Topics

- Base OS Security
  - OS Security Layers
    - Typical measures in OS Security Management
  - Security Maintenance
  - Linux/Unix Security Issues
  - Windows Security Issues
  - Role of Virtualization

©rev, DI-FCT-UNL, 2018/2019

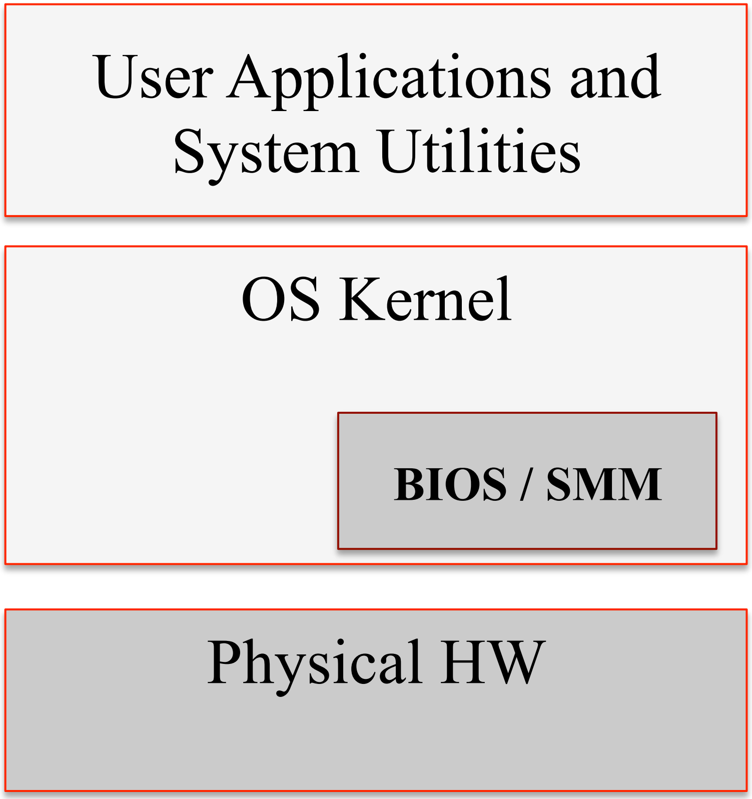
# Process model

- Access to hardware and SO resources only through system calls
- OS code runs in CPU kernel mode; applications and system utilities in User Mode
- Only privileged instructions allow
  - Programming the MMU
  - Programming the I/O controllers
  - Programming the clock



# Operating System and Security Layers

- Each layer of code needs measures in place to provide appropriate security services
- Each layer is vulnerable to attacks:
  - from below
  - If the lower layers are not secured appropriately



# Security Measures

- The 2010 Australian Defense Signals Directorate (DSD) list the "Top 35 Mitigation Strategies"
  - See the bibliography
- Over 70% of the targeted cyber intrusions investigated by DSD in 2009 could have been prevented the top four measures for prevention are:
  - patch operating systems and applications using auto-update
  - patch third-party applications
  - restrict admin privileges to users who need them
  - white-list approved applications

- Among other security approches:
  - SW Security and Security by Design !

# Operating System Security

- Possible for a system to be compromised during the installation process before it can install the latest patches
- Building and deploying a system should be a planned process designed to counter this threat
- Process must:
  - assess risks and plan the system deployment
  - secure the underlying operating system and then the key applications
  - ensure any critical content is secured
  - ensure appropriate network protection mechanisms are used
  - ensure appropriate processes are used to maintain security

**Security in the installation process …**
**And (important !) Security Maintenance and Auditing**

©rev, DI-FCT-UNL, 2018/2019

# Topics

- Base OS Security
  - OS Security Layers
    - Typical measures in OS Security Management
  - ▶ Security Maintenance
  - Linux/Unix Security Issues
  - Windows Security Issues
  - Role of Virtualization

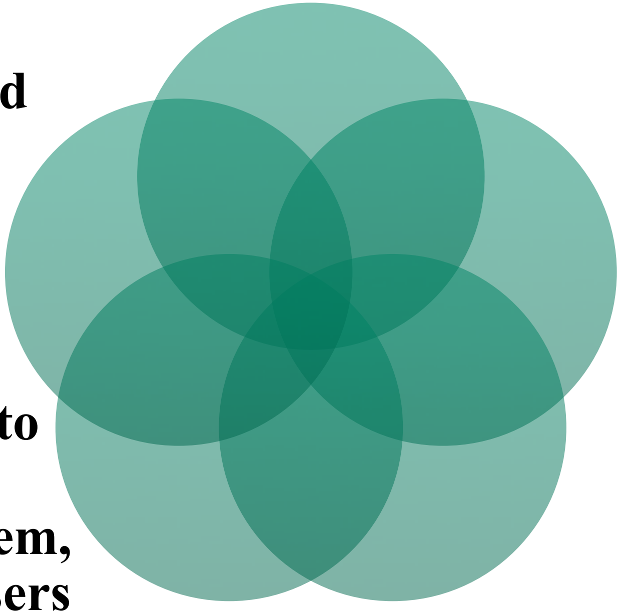
# Security Maintenance Issues

- Summary of relevant issues:
  - System Security Planning / Planning Processes
  - OS Security Hardening
  - Initial Setup and Patching Management
  - Secure Configuration of Applications and Utilities
  - Encryption Technology
  - Security Maintenance
    - Logging, Auditing, Backup/Archiving

# System Security Planning

The first step in deploying a new system is planning

Plan needs to identify appropriate personnel and training to install and manage the system



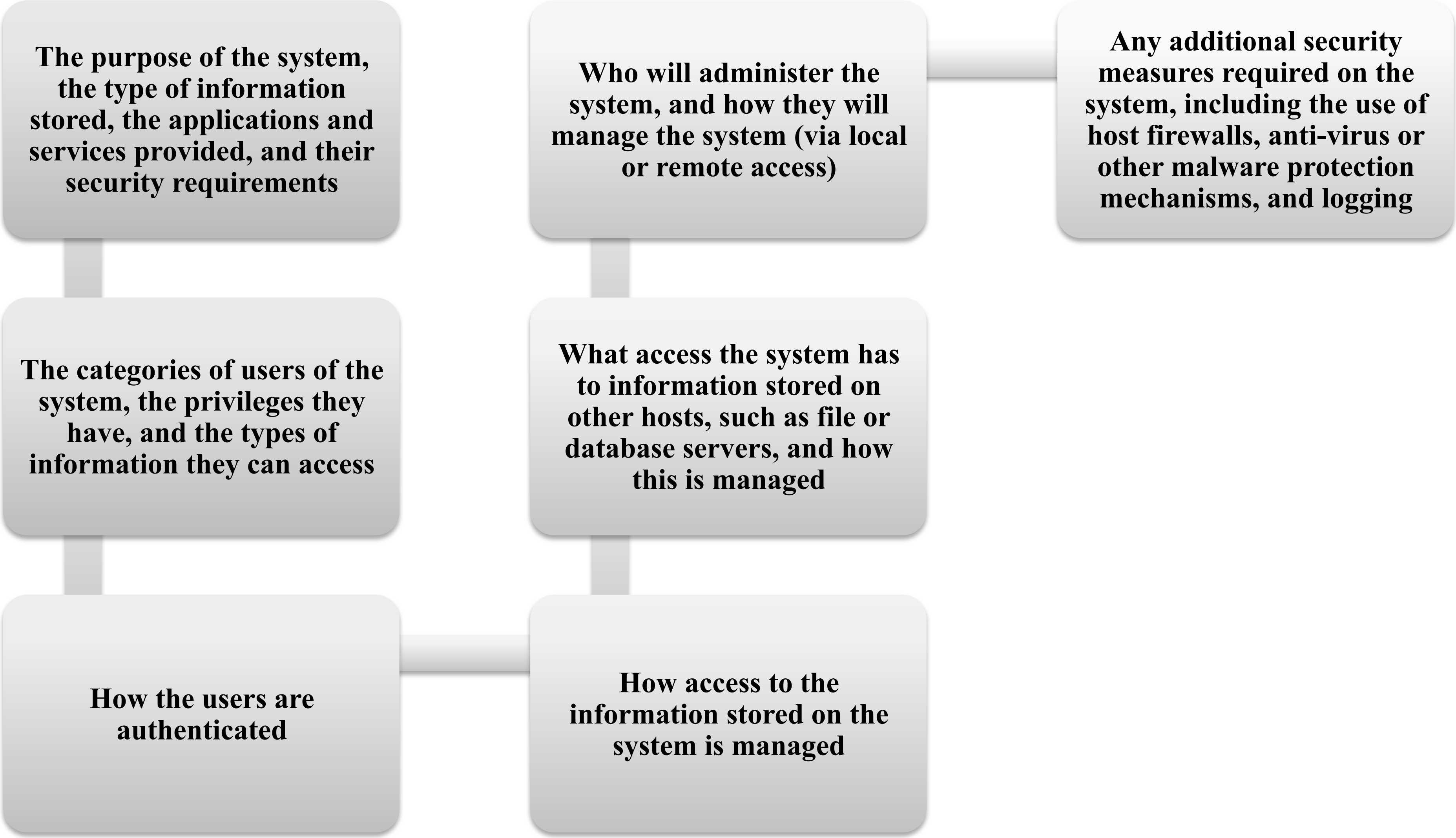
Planning should include a wide security assessment of the organization

Planning process needs to determine security requirements for the system, applications, data, and users

Aim is to maximize security while minimizing costs

# System Security Planning Process

## Ex., NIST Defined Issues for System Security Planning



The purpose of the system, the type of information stored, the applications and services provided, and their security requirements

The categories of users of the system, the privileges they have, and the types of information they can access

How the users are authenticated

How access to the information stored on the system is managed

What access the system has to information stored on other hosts, such as file or database servers, and how this is managed

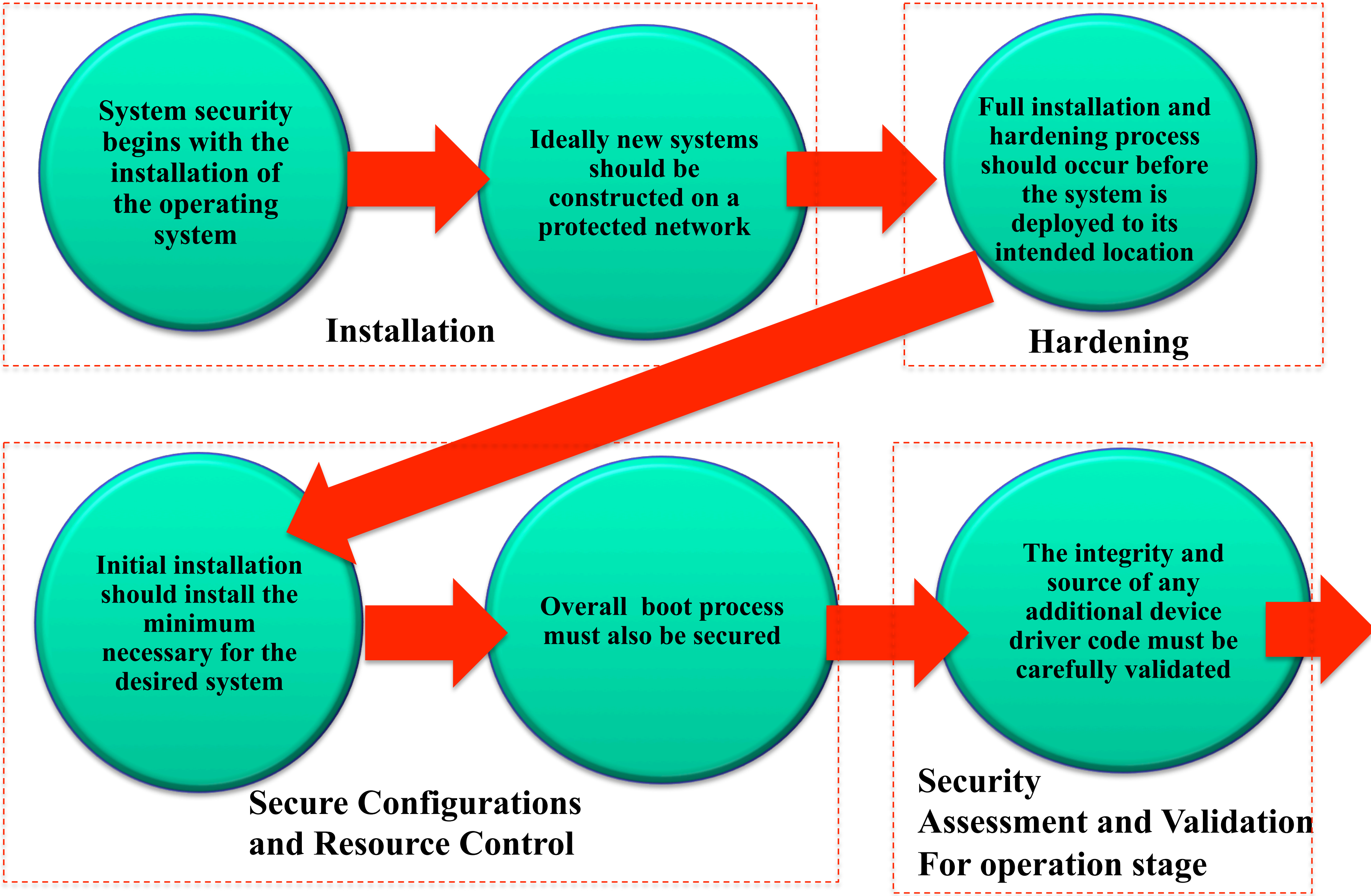
Who will administer the system, and how they will manage the system (via local or remote access)

Any additional security measures required on the system, including the use of host firewalls, anti-virus or other malware protection mechanisms, and logging

# Operating Systems Hardening

- First critical step in securing a system is … **to secure the base operating system**
- Basic relevant steps: **IHCT Sequence**
  - **Install** and patch the operating system
  - **Harden** and configure the operating system to adequately address the identified security needs of the system
  - **Configure** trustable installed additional security controls, such as anti-virus, host-based firewalls, and intrusion detection system (IDS)
  - **Test** the security of the basic operating system to ensure that the steps taken adequately address its security needs

# Initial Setup and Patching (1)



System security begins with the installation of the operating system

Ideally new systems should be constructed on a protected network

Installation

Full installation and hardening process should occur before the system is deployed to its intended location

Hardening

Initial installation should install the minimum necessary for the desired system

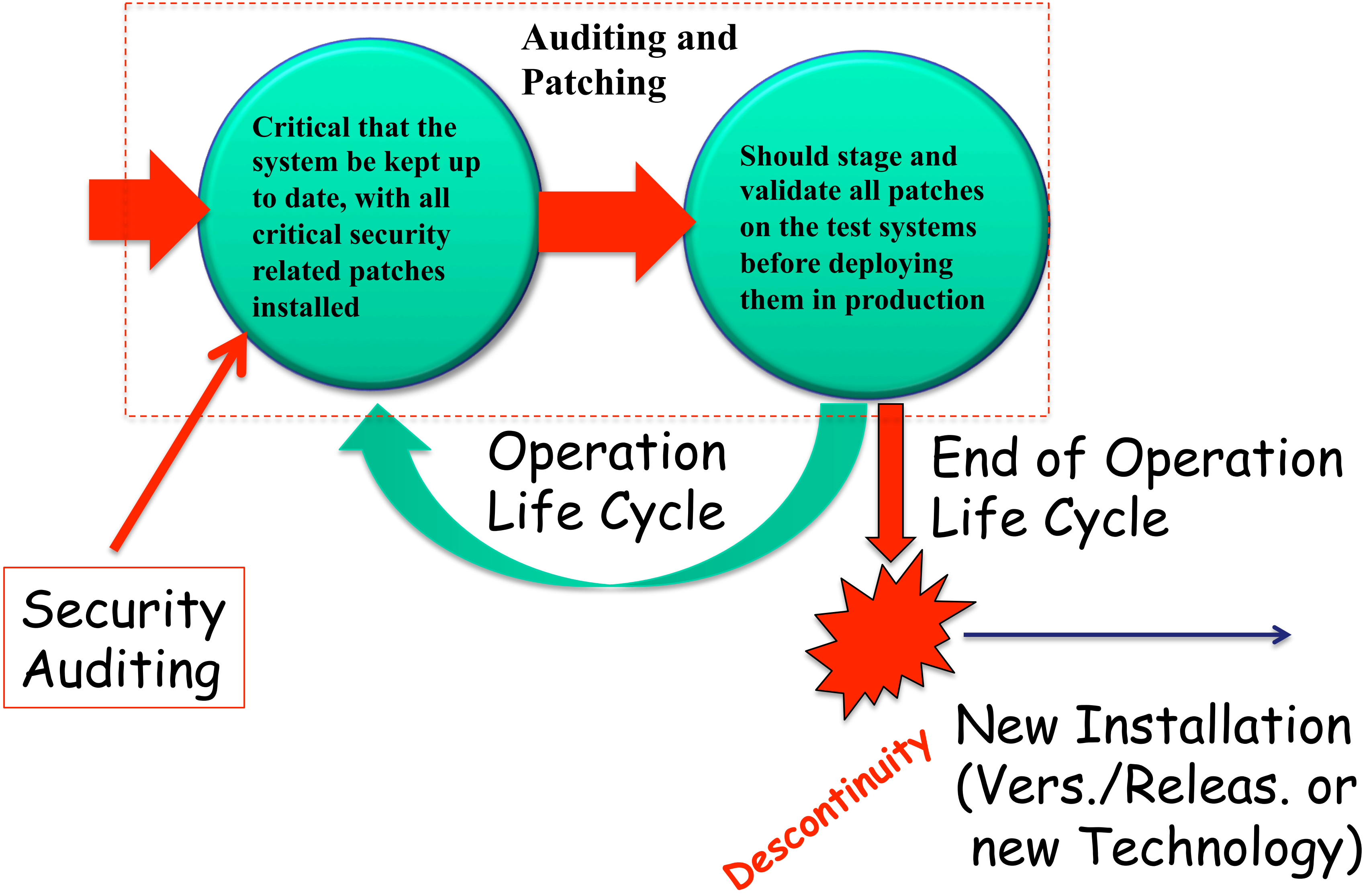
Overall boot process must also be secured

Secure Configurations and Resource Control

The integrity and source of any additional device driver code must be carefully validated

Security Assessment and Validation For operation stage

# Initial Setup and Patching (1)



Auditing and Patching

Critical that the system be kept up to date, with all critical security related patches installed

Should stage and validate all patches on the test systems before deploying them in production

Operation Life Cycle

End of Operation Life Cycle

Security Auditing

Descontinuity

New Installation (Vers./Releas. or new Technology)

# Hardening Principle

- Remove Unnecessary Services, Applications, and Protocols

- If fewer software packages are available to run the risk is reduced
- System planning process should identify what is actually required for a given system

- When performing the initial installation the supplied defaults should not be used
  - Default configuration is set to maximize ease of use and functionality rather than security
  - If additional packages are needed later they can be installed when they are required

# Security Configurations: User-Setup

- Configure Users, Groups, and Authentication

- Not all users with access to a system will have the same access to all data and resources on that system
- Elevated privileges should be restricted to only those users that require them, and then only when they are needed to perform a task
- Minimal Privileges as Necessary

- System planning process should consider:
  - Categories of users on the system
  - Privileges they have
  - Types of information they can access
  - How and where they are defined and authenticated
- Default accounts included as part of the system installation should be secured
  - Those that are not required should be either removed or disabled
  - Policies that apply to authentication credentials configured

# Security Configurations: Resource Control

Configuration of Resource-Controls

Installation of Additional Reosurce Controls

- Once the users and groups are defined, appropriate permissions can be set on data and resources
- Many of the security hardening guides provide lists of recommended changes to the default access configuration

- Further security possible by installing and configuring additional security tools:
  - Anti-virus software
  - Host-based firewalls
  - IDS or IPS software
  - Application white-listing

# Security Assessment

- Test the System Security

- Final step in the process of initially securing the base operating system is security testing
- goal:
  - Ensure the previous security configuration steps are correctly implemented
  - Identify any possible vulnerabilities

- Checklists are included in security hardening guides
- There are programs specifically designed to:
  - Review a system to ensure that a system meets the basic security requirements
  - **Scan for known vulnerabilities and poor configuration practices**
- Should be done following the initial hardening of the system
- Repeated periodically as part of the security maintenance process
  - **=> Security Auditing Process** →

# Application Configuration

- May include:
  - Creating and specifying appropriate data storage areas for application
  - Making appropriate changes to the application or service default configuration details
- Important !!! Some applications or services may include:
  - Default data
  - Scripts / Default Scripts, Examples, etc. …
  - Default User accounts
  - Permissions
- Review all this: Hardening + Application Resources + Reduction of Permissions (Least Privilege Principle)
- Of particular concern with remotely accessed services such as Web and File Transfer Services
  - Risk from this form of attack is reduced by ensuring that most of the files can only be read, but not written, by the server

# Encryption

A key enabling technology that may be used to secure data both in transit and when stored

But must be configured and appropriate cryptographic keys created, signed, and secured

**Maintenance of Cryptographic Keys: Better in dedicated Secure Crypto Devices**
**Carefully Management of Permissions when Secrecy Material is in the File Systems**

**Some critical examples:**

if secure network services are provided using TLS or IPsec suitable public and private keys must be generated for each of them

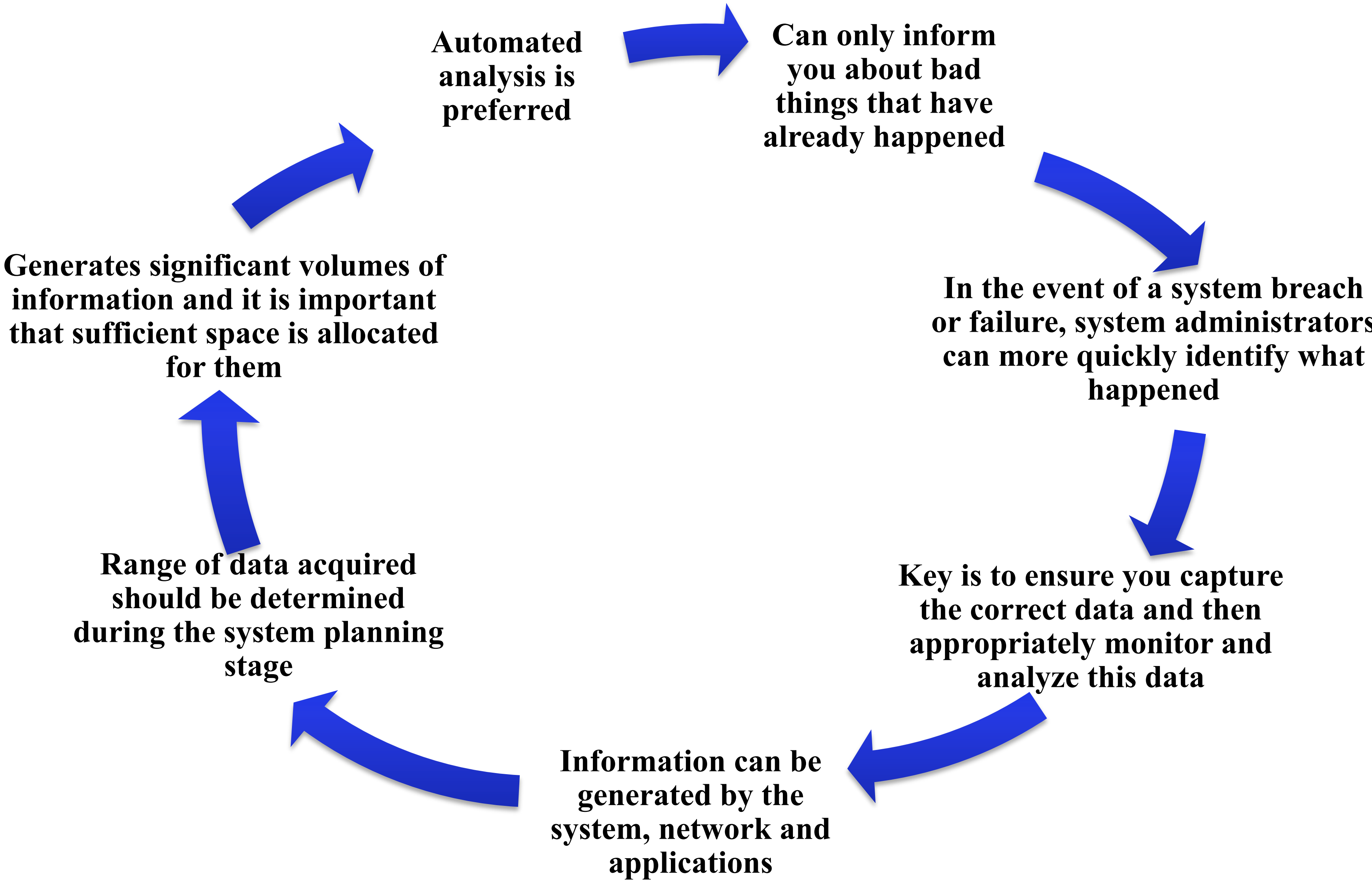
If secure network services are provided using SSH, appropriate server and client keys must be created

Cryptographic File Systems are another use of encryption

# Security Maintenance

- Process of maintaining security is continuous
  - Security as a Process (not an End)
- Security maintenance includes:
  - Monitoring and analyzing logging information
  - Performing regular backups
  - Recovering from security compromises
  - Regularly testing system security
    - AUDITING
  - Using appropriate software maintenance processes to patch and update all critical software, and to monitor and revise configuration as needed

# Logging



Automated analysis is preferred

Can only inform you about bad things that have already happened

In the event of a system breach or failure, system administrators can more quickly identify what happened

Key is to ensure you capture the correct data and then appropriately monitor and analyze this data

Information can be generated by the system, network and applications

Range of data acquired should be determined during the system planning stage

Generates significant volumes of information and it is important that sufficient space is allocated for them

# Data Backup and Archive

- Performing regular backups of data is a critical control that assists with maintaining the integrity of the system and user data
  - May be … legal, ethical, or operational compliance requirements for the retention of data will be necessary to analyze and to address

# Data Backup and Archive

**Backup:**

- Process of making copies of data at regular intervals
- Incremental vs. Total Backups

**Archive:**

The process of retaining copies of data over extended periods of time in order to meet legal and operational requirements to access past data

**Concerns:**

The needs and the policy requirements relating to backup and archive should be determined during the system planning stage

- kept online or offline
- Options: stored locally or transported to one or more remote sites (physically different)
  - Trade-offs include ease of implementation and cost versus greater security and robustness against different threats

# Topics

- Base OS Security
  - OS Security Layers
    - Typical measures in OS Security Management
  - Security Maintenance
  - ▶ Linux/Unix Security Issues
  - Windows Security Issues
  - Role of Virtualization

# Linux Security

- Linux has evolved into one of the most popular and versatile operating systems
- many features mean broad attack surface
- can create highly secure Linux systems
- will review:
  - Discretionary Access Controls
  - typical vulnerabilities and exploits in Linux
  - best practices for mitigating those threats
  - new improvements to Linux security model

# Linux/Unix Security

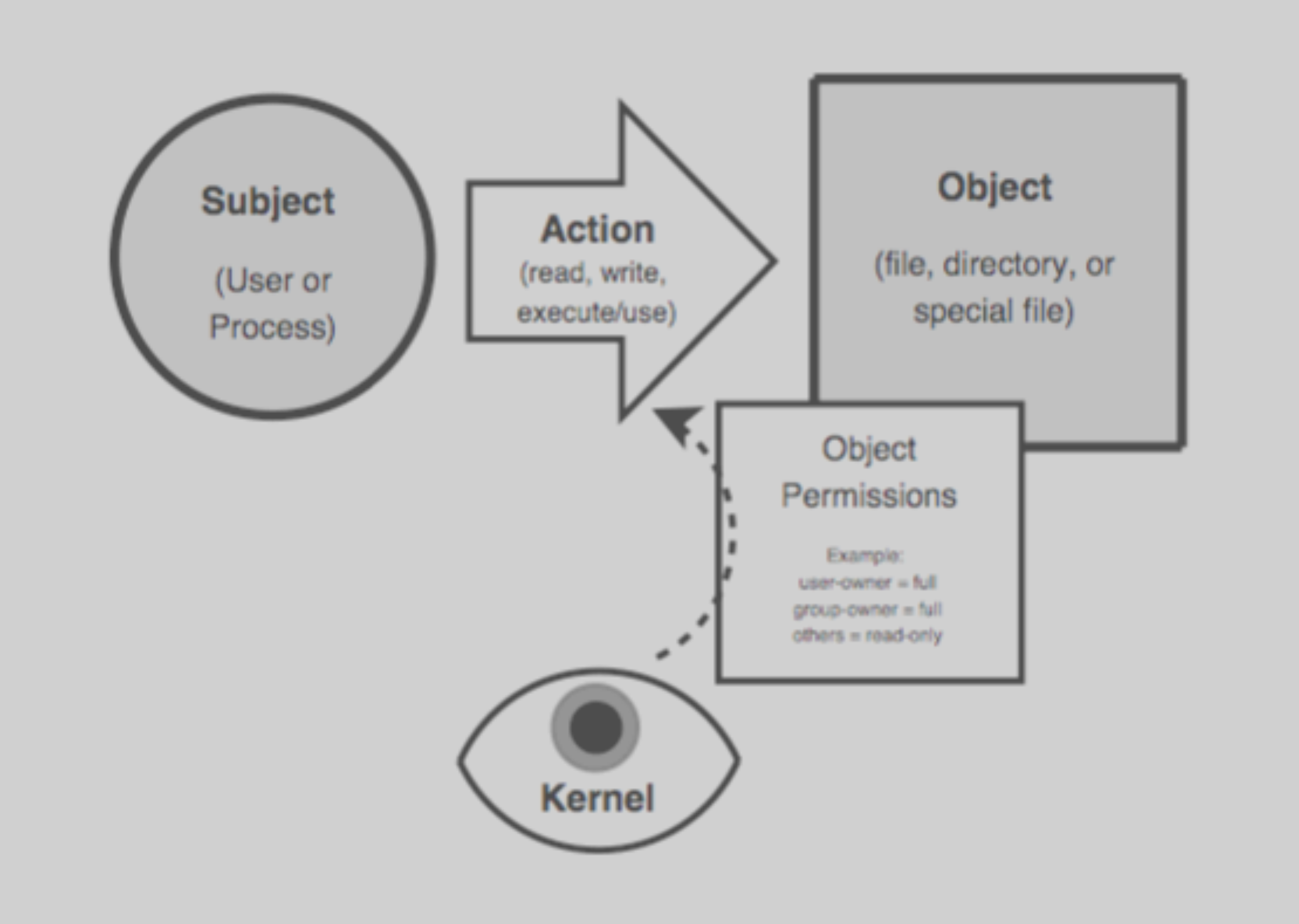
- Patch management
  - keeping security patches up to date is a widely recognized and critical control for maintaining security

- Application and service configuration
  - Most commonly implemented using separate text files for each application and service
  - Generally located either in the /etc directory or in the installation tree for a specific application
  - Individual user configurations that can override the system defaults are located in hidden “dot” files in each user’s home directory
  - Most important changes needed to improve system security are to disable services and applications that are not required

# Linux Security Model

- Linux’s traditional security model is:
  - people or proceses with “root” privileges can do anything
  - other accounts can do much less
- hence attacker’s want to get root privileges
- can run robust, secure Linux systems
- crux of problem is use of **Discretionary Access Controls** (DAC)

# Linux Security Transactions



# File System Security

- in Linux *everything* as a file
  - e.g. memory, device-drivers, named pipes, and other system resources
  - hence why filesystem security is so important
- I/O to devices is via a "special" file
  - e.g. /dev/cdrom
- have other special files like named pipes
  - a conduit between processes / programs

# Users and Groups

- a user-account (user)
  - represents someone capable of using files
  - associated both with humans and processes
- a group-account (group)
  - is a list of user-accounts
  - users have a main group
  - may also belong to other groups
- users & groups are **not** files

# Users and Groups

- user's details are kept in /etc/password

```
maestro:x:200:100:Maestro Edward
  Hizzersands:/home/maestro:/bin/bash
```

- additional group details in /etc/group

```
conductors:x:100:

pianists:x:102:maestro,volodya
```

- use **useradd**, **usermod**, **userdel** to alter

# File Permissions

- files have two owners: a user & a group
- each with its own set of permissions
- with a third set of permissions for other
- permissions are to read/write/execute in order user/group/other, cf.

```
-rw-rw-r--  1  maestro user 35414
  Mar 25 01:38 baton.txt
```

- set using **chmod** command

# Directory Permissions

- read = list contents
- write = create or delete files in directory
- execute = use anything in or change working directory to this directory
- e.g.

```
$ chmod g+rx extreme_casseroles
$ ls -l extreme_casseroles
 drwxr-x--- 8 biff  drummers 288  Mar
  25 01:38 extreme_casseroles
```

# Sticky Bit

- originally used to lock file in memory (obsolete now)
- now used on directories to limit delete
  - if set must own file or dir to delete
  - other users cannot delete even if have write
- set using chmod command with +t flag, e.g.

```
chmod +t extreme_casseroles
```

- directory listing includes t or T flag

```
drwxrwx--T  8  biff  drummers  288  Mar
  25 01:38 extreme_casseroles
```

- only apply to specific directory not child dirs

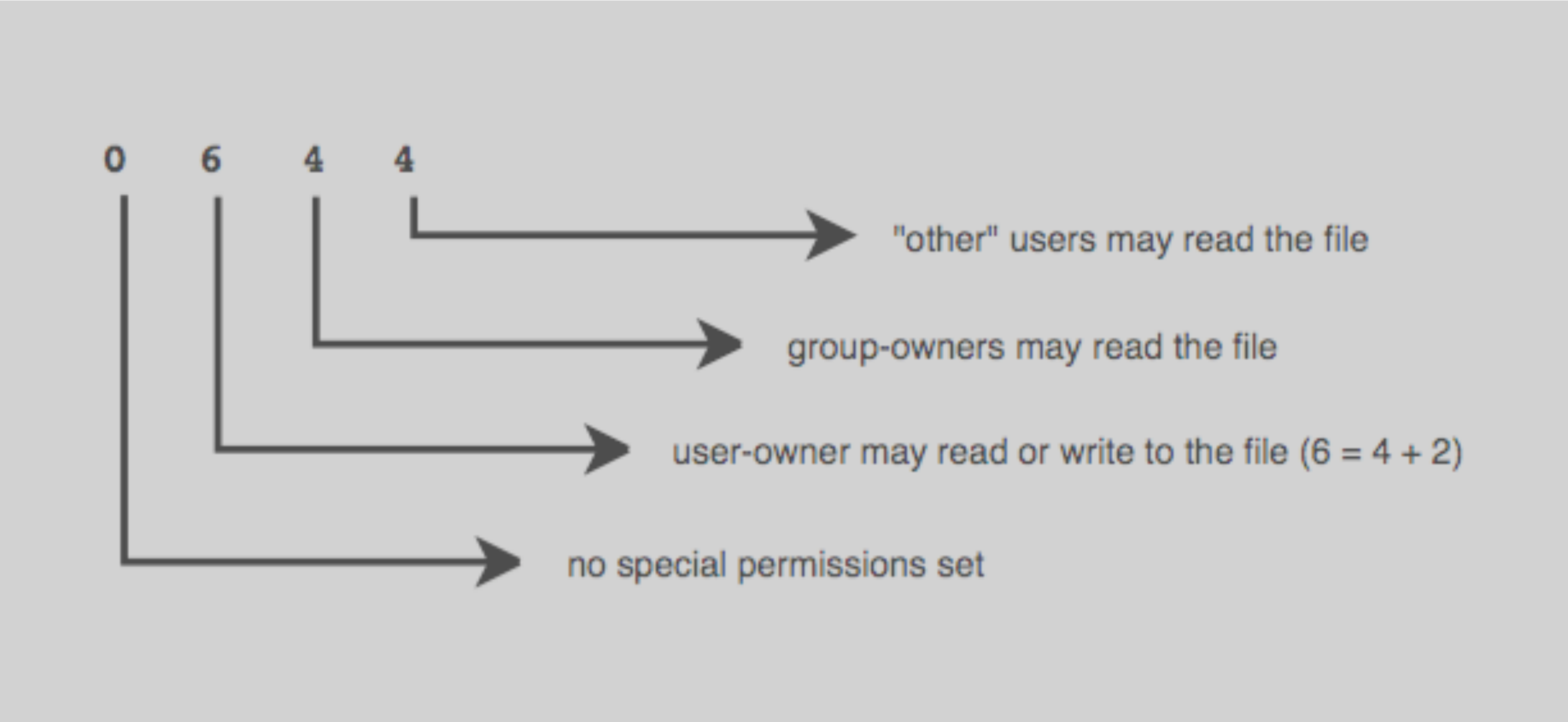
# SetUID and SetGID

- setuid bit means program "runs as" owner
  - no matter who executes it
- setgid bit means run as a member of the group which owns it
  - again regardless of who executes it
- "run as" = "run with same privileges as“
- *are very dangerous* if set on file owned by root or other privileged account or group
  - only used on executable files, not shell scripts

# SetGID and Directories

- setuid has no effect on directories
- setgid does and causes any file created in a directory to inherit the directory's group
- useful if users belong to other groups and routinely create files to be shared with other members of those groups
  - instead of manually changing its group

# Numeric File Permissions



0 6 4 4

"other" users may read the file

group-owners may read the file

user-owner may read or write to the file (6 = 4 + 2)

no special permissions set

# Kernel vs User Space

- Kernel space
  - refers to memory used by the Linux kernel and its loadable modules (e.g., device drivers)
- User space
  - refers to memory used by all other processes
- since kernel enforces Linux DAC and security critical to isolate kernel from user
  - so kernel space never swapped to disk
  - only root may load and unload kernel modules

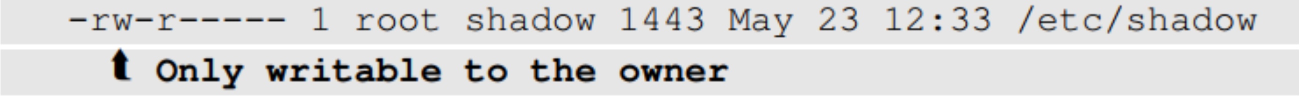
# setuid root Vulnerabilities

- a **setuid root** program runs as root
  - *no matter who executes it*
- used to provide unprivileged users with access to privileged resources
- must be very carefully programmed
- if can be exploited due to a software bug
  - may allow otherwise-unprivileged users to use it to wield unauthorized root privileges
- distributions now minimise setuid-root programs
- system attackers still scan for them!

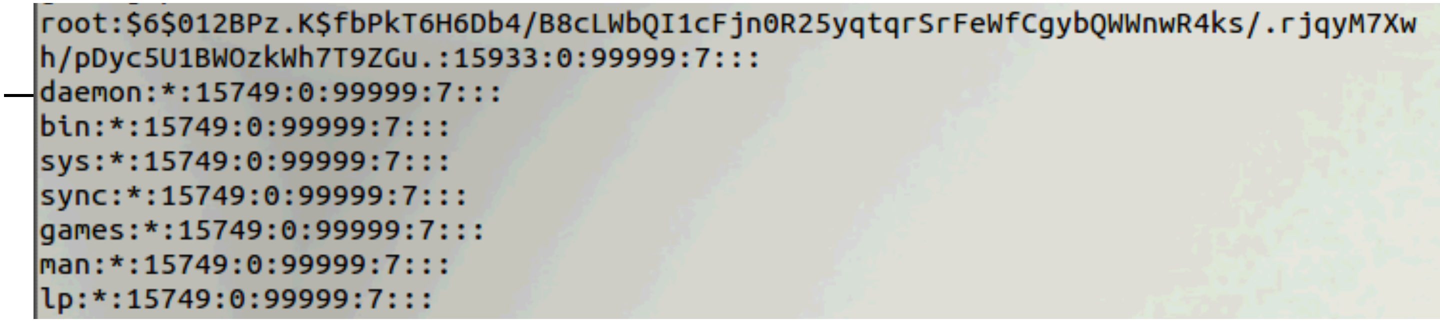
# Set-UID Privileged Programs

## Need for Privileged Programs : Password Dilemma

- Permissions of /etc/shadow File:



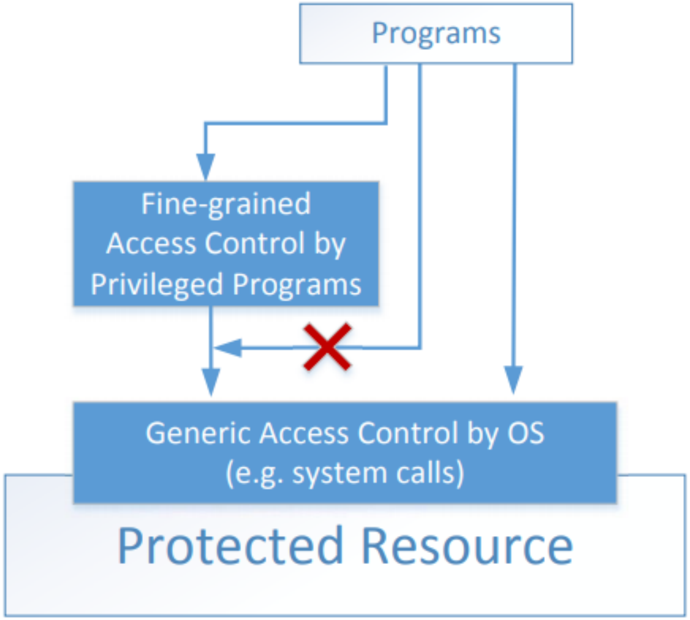
```
  -rw-r----- 1 root shadow 1443 May 23 12:33 /etc/shadow
     ↑ Only writable to the owner
```



```
root:$6$012BPz.K$fbPkT6H6Db4/B8cLWbQI1cFjn0R25yqtqrSrFeWfCgybQWWnwR4ks/.rjqyM7Xw
h/pDyc5U1BWOzkWh7T9ZGu.:15933:0:99999:7:::
daemon:*:15749:0:99999:7:::
bin:*:15749:0:99999:7:::
sys:*:15749:0:99999:7:::
sync:*:15749:0:99999:7:::
games:*:15749:0:99999:7:::
man:*:15749:0:99999:7:::
lp:*:15749:0:99999:7:::
```

# Two-Tier Approach

- Implementing fine-grained access control in operating systems make OS over complicated.
- OS relies on extension to enforce fine-grained access control
- Privileged programs are such extensions


Programs
Fine-grained Access Control by Privileged Programs
Generic Access Control by OS (e.g. system calls)
Protected Resource

# Types of Privileged Programs

- Daemons
  - Computer program that runs in the background
  - Needs to run as root or other privileged users
- Set-UID Programs
  - Widely used in UNIX systems
  - Program marked with a special bit

# Set-UID Concept

- **Allow user to run a program with the program owner's privilege.**
- Allow users to run programs with temporary elevated privileges
- Example: the passwd program

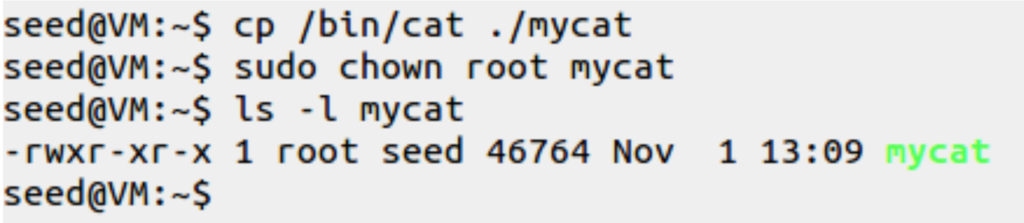
```
$ ls -l /usr/bin/passwd
-rwsr-xr-x 1 root root 41284 Sep 12  2012 /usr/bin/passwd
```

# Set-UID Concept

- Every process has two User IDs.
- **Real UID (RUID)**: Identifies real owner of process
- **Effective UID (EUID)**: Identifies privilege of a process
  - Access control is based on EUID
- When a normal program is executed, RUID = EUID, they both equal to the ID of the user who runs the program
- When a Set-UID is executed, RUID ≠ EUID. RUID still equal to the user's ID, but EUID equals to the program **owner**'s ID.
  - If the program is owned by root, the program runs with the root privilege.

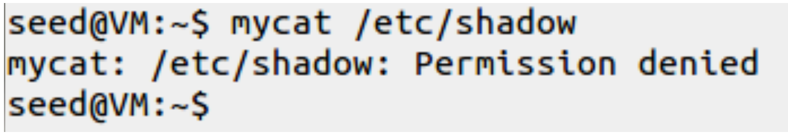
# Turn a Program into Set-UID

- Change the owner of a file to root :



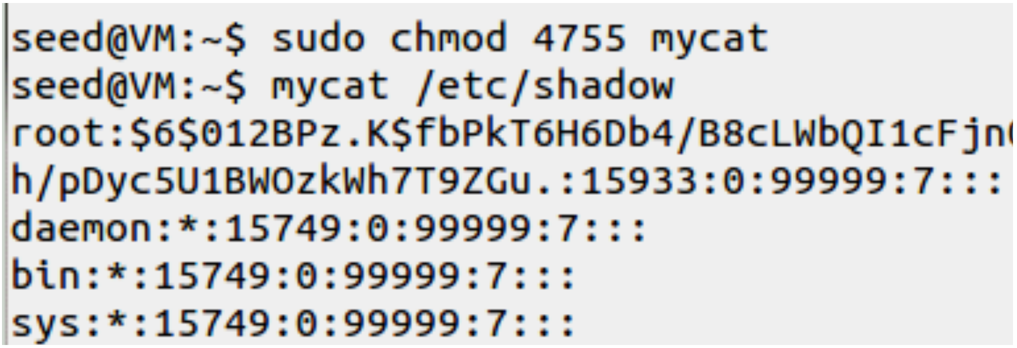
```
seed@VM:~$ cp /bin/cat ./mycat
seed@VM:~$ sudo chown root mycat
seed@VM:~$ ls -l mycat
-rwxr-xr-x 1 root seed 46764 Nov  1 13:09 mycat
seed@VM:~$
```

- Before Enabling Set-UID bit:



```
seed@VM:~$ mycat /etc/shadow
mycat: /etc/shadow: Permission denied
seed@VM:~$
```

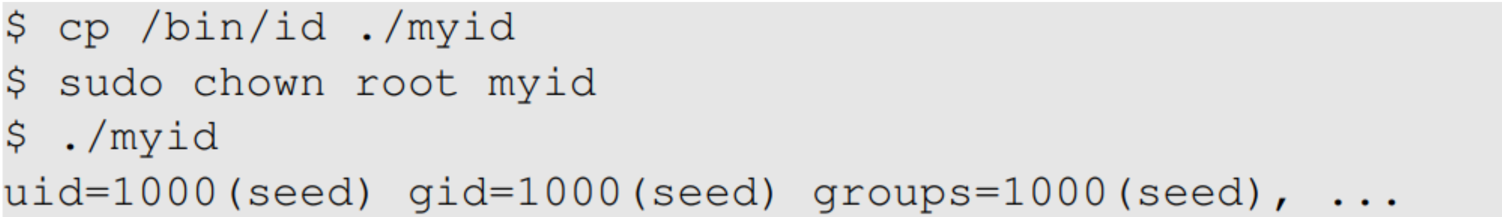
- After Enabling the Set-UID bit :



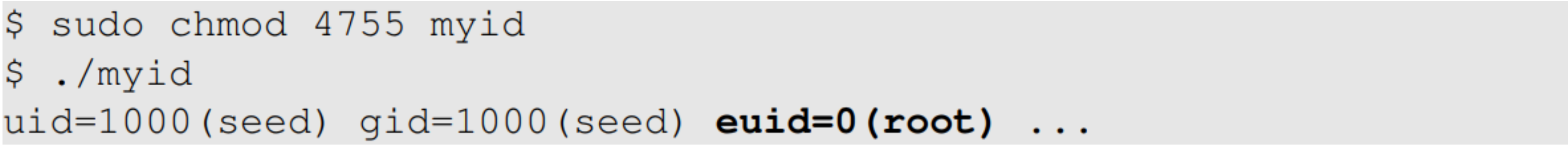
```
seed@VM:~$ sudo chmod 4755 mycat
seed@VM:~$ mycat /etc/shadow
root:$6$012BPz.K$fbPkT6H6Db4/B8cLWbQI1cFjn
h/pDyc5U1BWOzkWh7T9ZGu.:15933:0:99999:7:::
daemon:*:15749:0:99999:7:::
bin:*:15749:0:99999:7:::
sys:*:15749:0:99999:7:::
```

# How it Works

A Set-UID program is just like any other program, except that it has a special marking, which a single bit called Set-UID bit

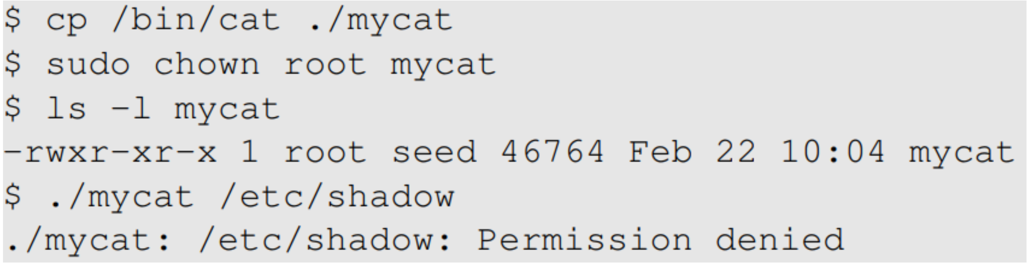


```
$ cp /bin/id ./myid
$ sudo chown root myid
$ ./myid
uid=1000(seed) gid=1000(seed) groups=1000(seed), ...
```



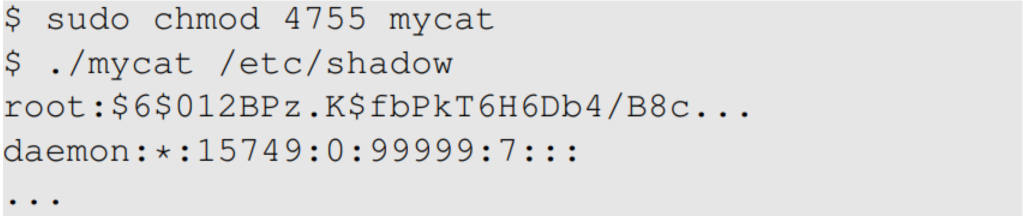
```
$ sudo chmod 4755 myid
$ ./myid
uid=1000(seed) gid=1000(seed) euid=0(root) ...
```

# Example of Set UID



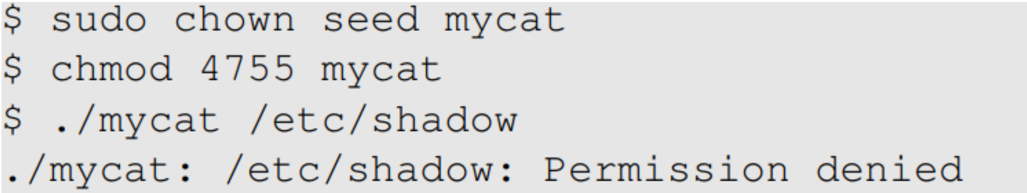
```
$ cp /bin/cat ./mycat
$ sudo chown root mycat
$ ls -l mycat
-rwxr-xr-x 1 root seed 46764 Feb 22 10:04 mycat
$ ./mycat /etc/shadow
./mycat: /etc/shadow: Permission denied
```

Not a privileged program



```
$ sudo chmod 4755 mycat
$ ./mycat /etc/shadow
root:$6$012BPz.K$fbPkT6H6Db4/B8c...
daemon:*:15749:0:99999:7:::
...
```

Become a privileged program



```
$ sudo chown seed mycat
$ chmod 4755 mycat
$ ./mycat /etc/shadow
./mycat: /etc/shadow: Permission denied
```

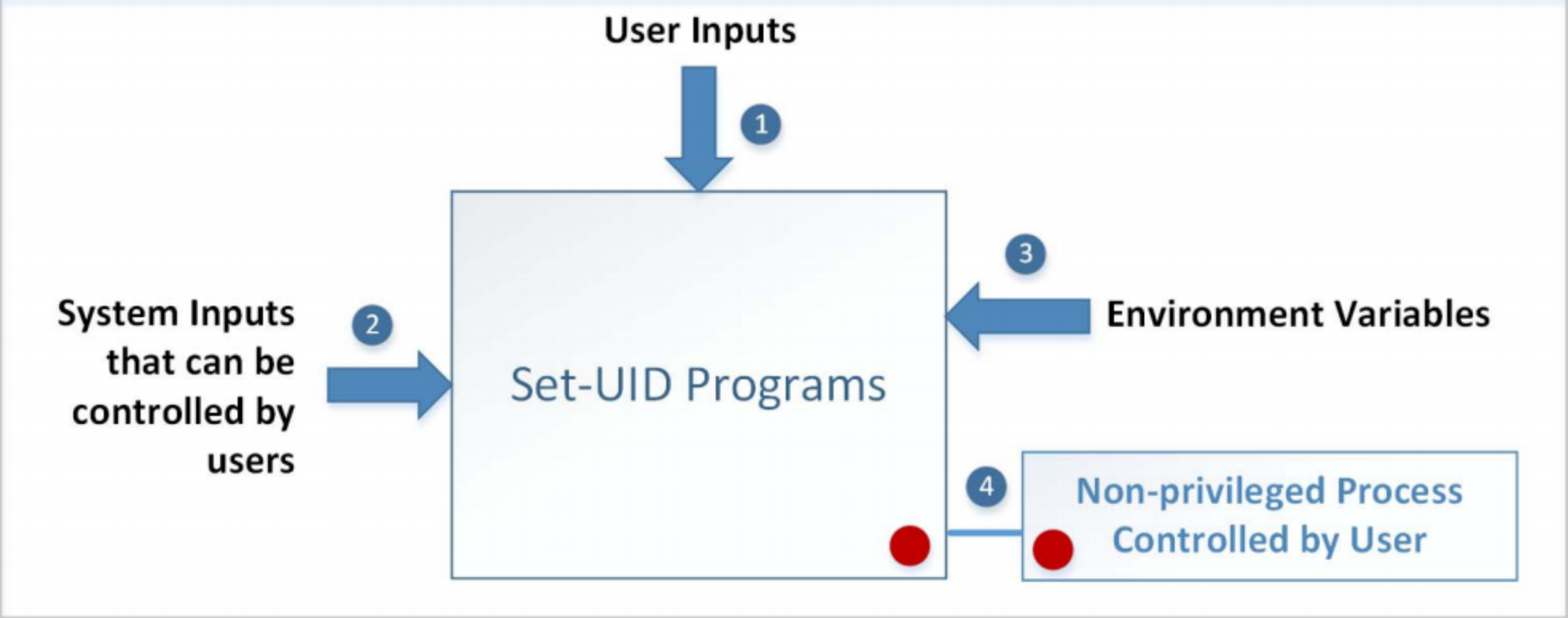
It is still a privileged program, but not the root privilege

# How is Set-UID Secure?

- Allows normal users to escalate privileges
  - This is different from directly giving the privilege (sudo command)
  - Restricted behavior

- Unsafe to turn all programs into Set-UID
  - Example: /bin/sh
  - Example: vi

# Attack Surfaces of Set-UID Programs


User Inputs
1
System Inputs that can be controlled by users
2
Set-UID Programs
3
Environment Variables
4
Non-privileged Process Controlled by User

# Attacks via User Inputs

## User Inputs: Explicit Inputs

- Buffer Overflow : Overflowing a buffer to run malicious code
- Format String Vulnerability
  - Changing program behavior using user inputs as format strings

# Attacks via System Inputs

## System Inputs

- Race Condition
  - Symbolic link to privileged file from an unprivileged file
  - Influence programs
  - Writing inside world writable folder

# Attacks via Environment Variables

- Behavior can be influenced by inputs that are not visible inside a program.
- Environment Variables : These can be set by a user before running a program.

# Attacks via Environment Variables

- PATH Environment Variable
  - Used by shell programs to locate a command if the user does not provide the full path for the command
  - system(): call /bin/sh first
  - system("ls")
    - /bin/sh uses the PATH environment variable to locate "ls"
    - Attacker can manipulate the PATH variable and control how the "ls" command is found

# Capability Leaking

- In some cases, Privileged programs downgrade themselves during execution
- Example: The su program
  - This is a privileged Set-UID program
  - Allows one user to switch to another user ( say user1 to user2 )
  - Program starts with EUID as root and RUID as user1
  - After password verification, both EUID and RUID become user2's (via privilege downgrading)
- Such programs may lead to capability leaking
  - Programs may not clean up privileged capabilities before downgrading

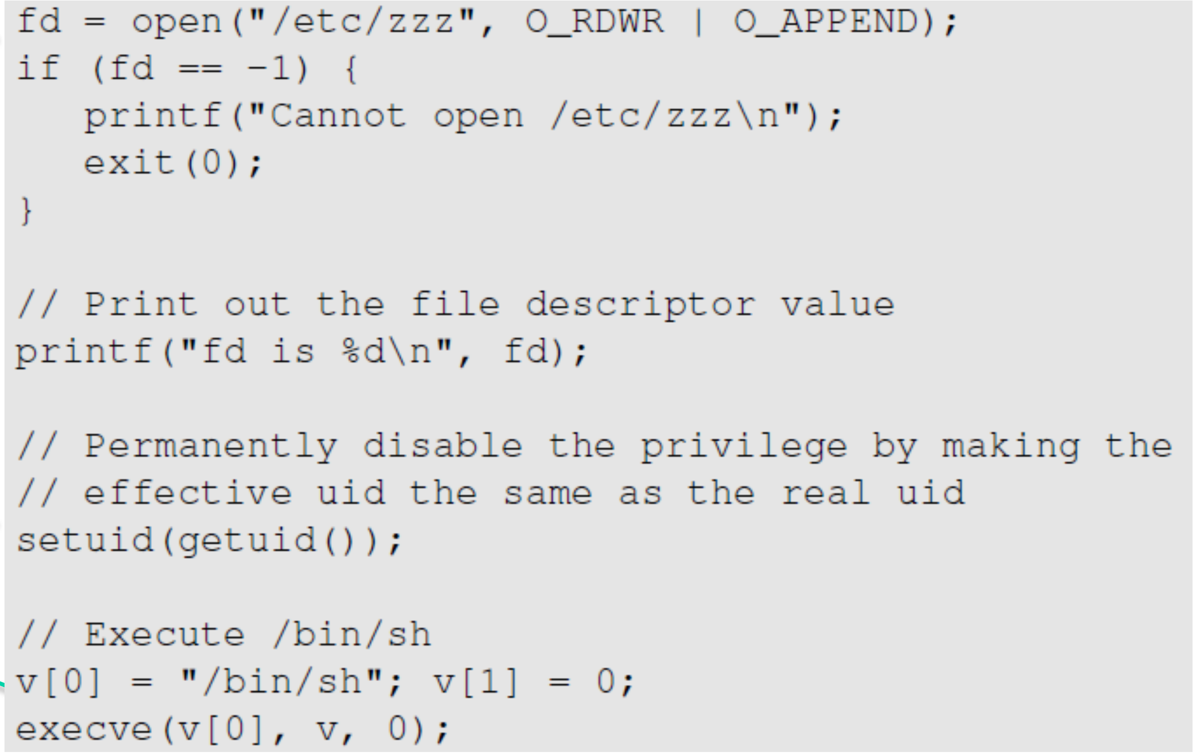
# Attacks via Capability Leaking: An Example

The /etc/zzz file is only writable by root

File descriptor is created
(the program is a root-owned Set-UID program)

The privilege is downgraded

Invoke a shell program, so the behavior restriction on the program is lifted

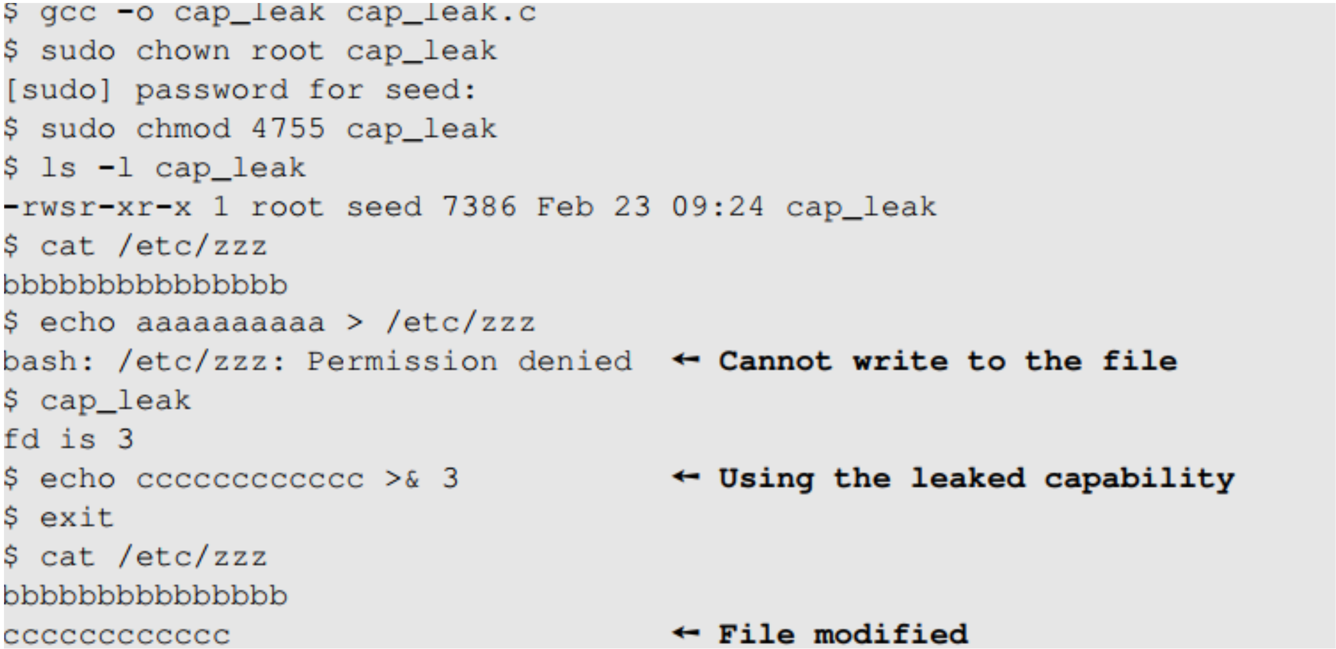


```
fd = open("/etc/zzz", O_RDWR | O_APPEND);
if (fd == -1) {
   printf("Cannot open /etc/zzz\n");
   exit(0);
}

// Print out the file descriptor value
printf("fd is %d\n", fd);

// Permanently disable the privilege by making the
// effective uid the same as the real uid
setuid(getuid());

// Execute /bin/sh
v[0] = "/bin/sh"; v[1] = 0;
execve(v[0], v, 0);
```

# Attacks via Capability Leaking (Continued)

The program forgets to close the file, so the file descriptor is still valid.

Capability Leak



```
$ gcc -o cap_leak cap_leak.c
$ sudo chown root cap_leak
[sudo] password for seed:
$ sudo chmod 4755 cap_leak
$ ls -l cap_leak
-rwsr-xr-x 1 root seed 7386 Feb 23 09:24 cap_leak
$ cat /etc/zzz
bbbbbbbbbbbbbbb
$ echo aaaaaaaaaa > /etc/zzz
bash: /etc/zzz: Permission denied   ← Cannot write to the file
$ cap_leak
fd is 3
$ echo cccccccccccc >& 3             ← Using the leaked capability
$ exit
$ cat /etc/zzz
bbbbbbbbbbbbbbb
cccccccccccc                         ← File modified
```

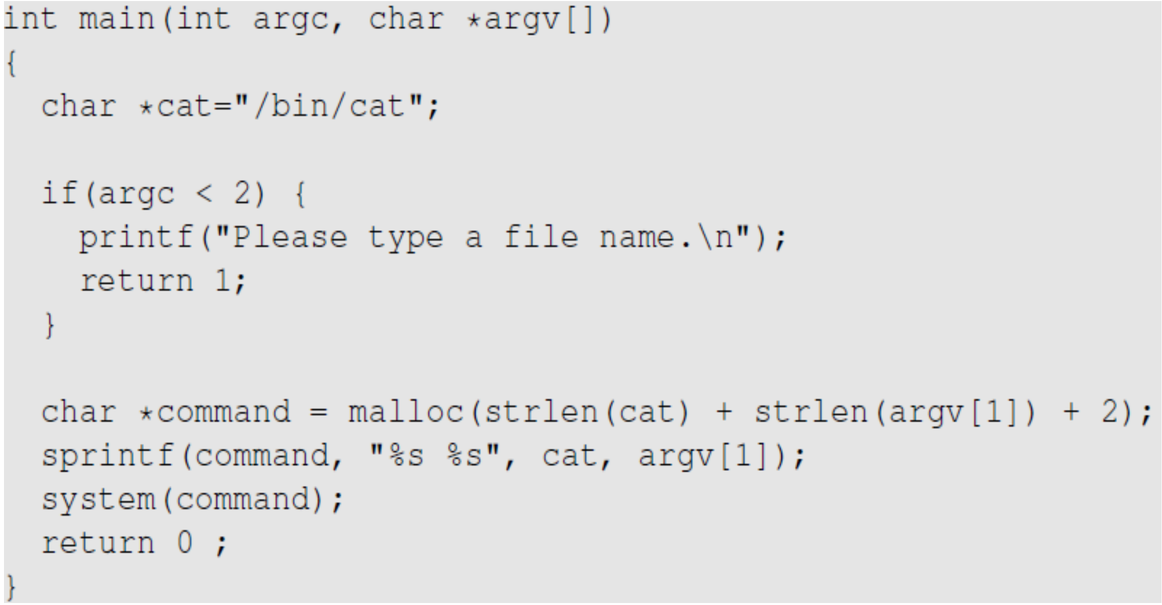
How to fix the program?

Destroy the file descriptor before downgrading the privilege (close the file)

# Invoking Programs

- Invoking external commands from inside a program
- External command is chosen by the Set-UID program
  - Users are not supposed to provide the command (or it is not secure)
- Attack:
  - Users are often asked to provide input data to the command.
  - If the command is not invoked properly, user's input data may be turned into command name. This is dangerous.

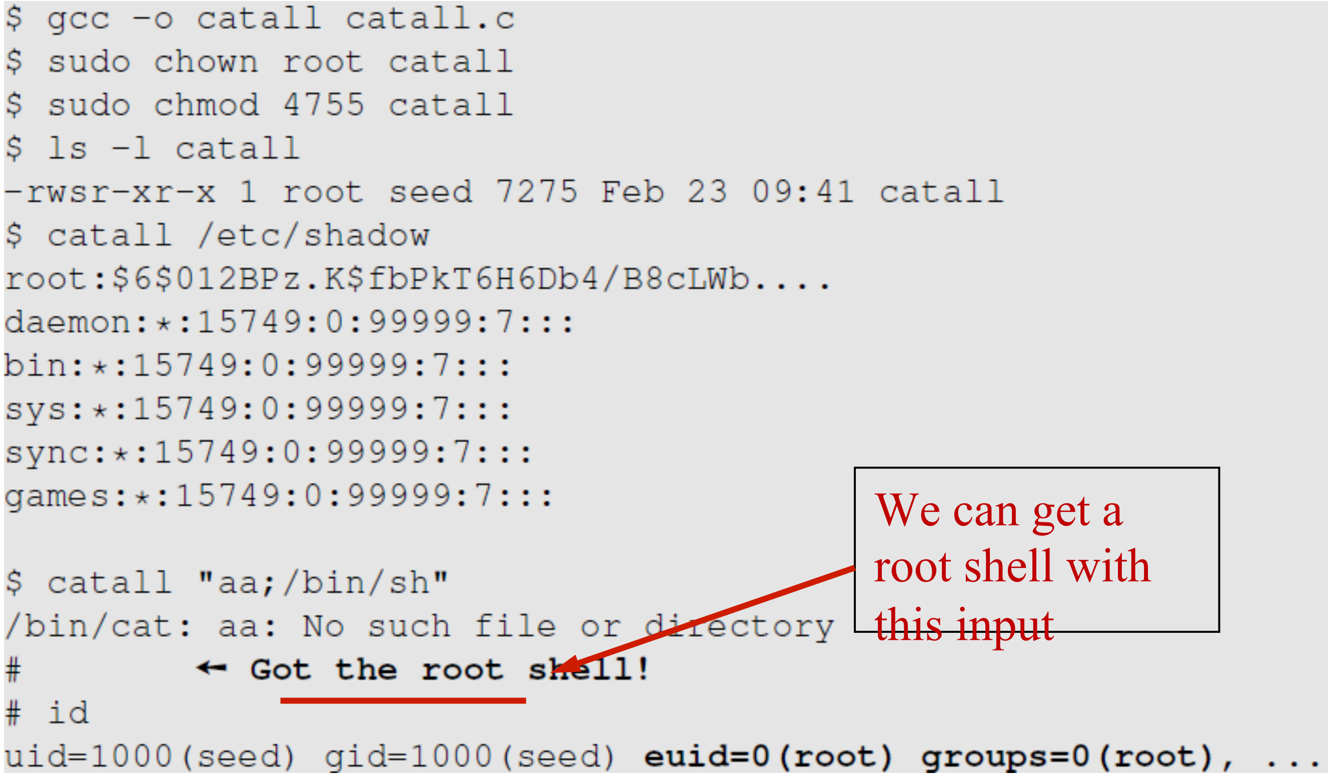
# Invoking Programs : Unsafe Approach



```
int main(int argc, char *argv[])
{
  char *cat="/bin/cat";

  if(argc < 2) {
    printf("Please type a file name.\n");
    return 1;
  }

  char *command = malloc(strlen(cat) + strlen(argv[1]) + 2);
  sprintf(command, "%s %s", cat, argv[1]);
  system(command);
  return 0 ;
}
```

- The easiest way to invoke an external command is the system() function.
- This program is supposed to run the /bin/cat program.
- It is a root-owned Set-UID program, so the program can view all files, but it can't write to any file.

# Invoking Programs : Unsafe Approach ( Cont.)

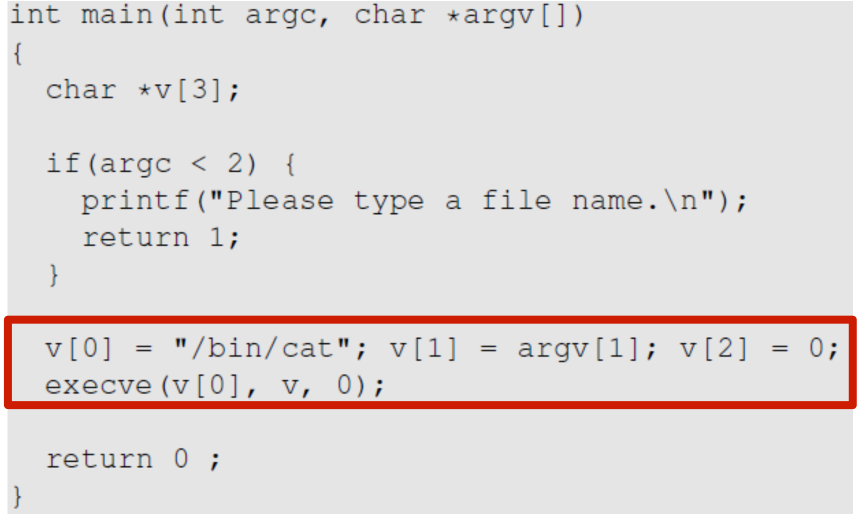


```
$ gcc -o catall catall.c
$ sudo chown root catall
$ sudo chmod 4755 catall
$ ls -l catall
-rwsr-xr-x 1 root seed 7275 Feb 23 09:41 catall
$ catall /etc/shadow
root:$6$012BPz.K$fbPkT6H6Db4/B8cLWb....
daemon:*:15749:0:99999:7:::
bin:*:15749:0:99999:7:::
sys:*:15749:0:99999:7:::
sync:*:15749:0:99999:7:::
games:*:15749:0:99999:7:::

$ catall "aa;/bin/sh"
/bin/cat: aa: No such file or directory
#        ← Got the root shell!
# id
uid=1000(seed) gid=1000(seed) euid=0(root) groups=0(root), ...
```

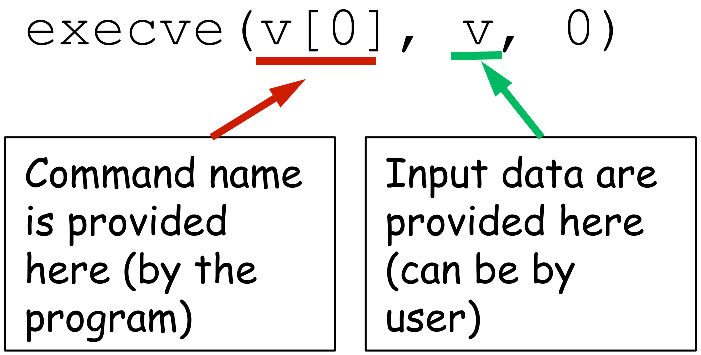
We can get a root shell with this input

**Problem**: Some part of the data becomes code (command name)

# Invoking Programs Safely: using **execve()**



```
int main(int argc, char *argv[])
{
  char *v[3];

  if(argc < 2) {
    printf("Please type a file name.\n");
    return 1;
  }

  v[0] = "/bin/cat"; v[1] = argv[1]; v[2] = 0;
  execve(v[0], v, 0);

  return 0 ;
}
```



```
execve(v[0], v, 0)
```

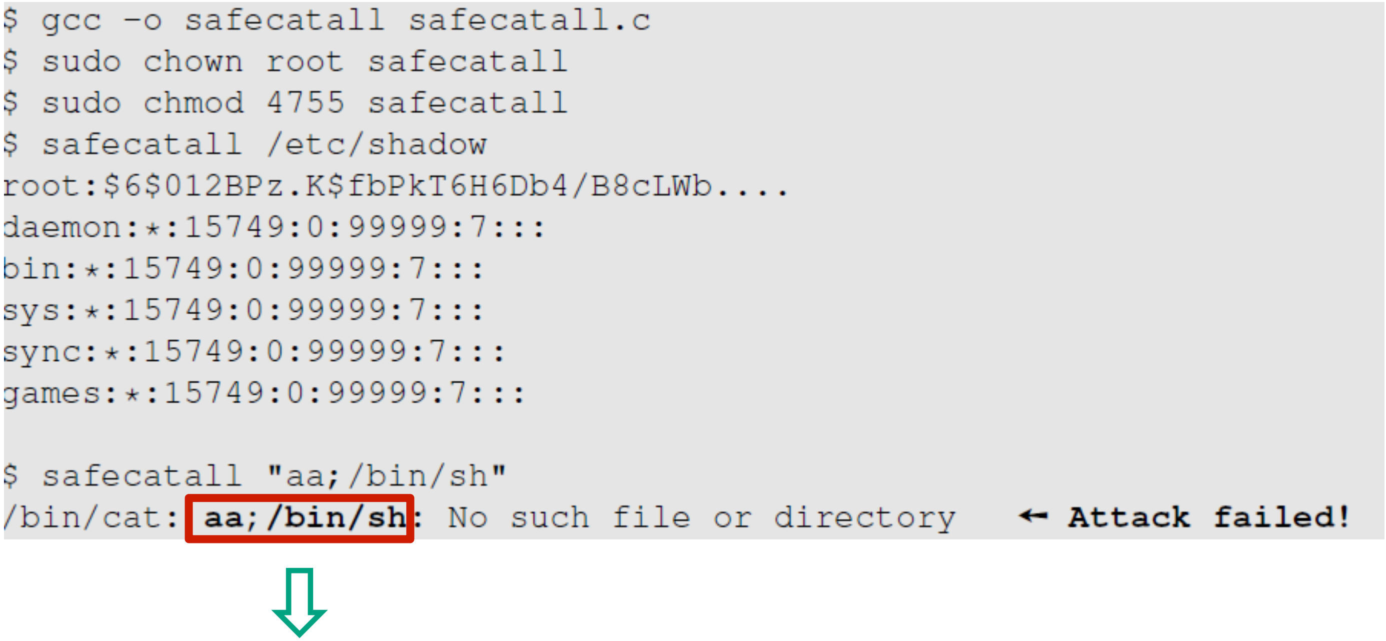
Command name is provided here (by the program)

Input data are provided here (can be by user)

**Why is it safe?**

Code (command name) and data are clearly separated; there is no way for the user data to become code

# Invoking Programs Safely ( Continued)



```
$ gcc -o safecatall safecatall.c
$ sudo chown root safecatall
$ sudo chmod 4755 safecatall
$ safecatall /etc/shadow
root:$6$012BPz.K$fbPkT6H6Db4/B8cLWb....
daemon:*:15749:0:99999:7:::
bin:*:15749:0:99999:7:::
sys:*:15749:0:99999:7:::
sync:*:15749:0:99999:7:::
games:*:15749:0:99999:7:::

$ safecatall "aa;/bin/sh"
/bin/cat: aa;/bin/sh: No such file or directory   ← Attack failed!
```

The data are still treated as data, not code

# Additional Consideration

- Some functions in the exec() family behave similarly to execve(), but may not be safe
  - execlp(), execvp() and execvpe() duplicate the actions of the shell. These functions can be attacked using the PATH Environment Variable

# Principle of Isolation

Principle: Don’t mix code and data.

Attacks due to violation of this principle :

- system()  code execution
- Buffer  Overflow attacks
- …

# Principle of Least Privilege

- A privileged program should be given the power which is required to perform it's tasks.
- Disable the privileges (temporarily or permanently) when a privileged program doesn't need those.
- In Linux, seteuid() and setuid() can be used to disable/discard privileges.
- Different OSes have different ways to do that.

# Linux System Hardening

- consider how to mitigate Linux security risks at system and application levels
- first look at OS-level security tools and techniques that protect the entire system

# OS Installation

- security begins with O/S installation
- especially what software is run
  - since unused applications liable to be left in default, un-hardened and un-patched state
- generally should not run:
  - X Window system, RPC services, R-services, inetd, SMTP daemons, telnet etc
- also have some initial system s/w configuration:
  - setting root password
  - creating a non-root user account
  - setting an overall system security level
  - enabling a simple host-based firewall policy
  - enabling SELinux

# Patch Management

- installed server applications must be:
  - configured securely
  - kept up to date with security patches
- patching can never win "patch rat-race"
- have tools to automatically download and install security updates
  - e.g. up2date, YaST, apt-get
  - note should not run automatic updates on change-controlled systems without testing

# Network Access Controls

- network a key attack vector to secure
- TCP wrappers a key tool to check access
  - originally tcpd inetd wrapper daemon
  - before allowing connection to service checks
    - if requesting host explicitly in hosts.allow is ok
    - if requesting host explicitly in hosts.deny is blocked
    - if not in either is ok
  - checks on service, source IP, username
  - now often part of app using libwrappers

# Network Access Controls

- also have the very powerful **netfilter** Linux kernel native firewall mechanism
  - and **iptables** user-space front end
- as useful on firewalls, servers, desktops
- direct config tricky, steep learning curve
- do have automated rule generators
- typically for "personnal" firewall use will:
  - allow incoming requests to specified services
  - block all other inbound service requests
  - allow all outbound (locally-originating) requests
- if need greater security, manually config

# User Management

- guiding principles in user-account security:
  - need care setting file / directory permissions
  - use groups to differentiate between roles
  - use extreme care in granting / using root privs
- commands: chmod, useradd/mod/del, groupadd/ mod/del, passwd, chage
- info in files /etc/passwd & /etc/group
- manage user’s group memberships
- set appropriate password ages

# Root Delegation

- have "root can to anything, users do little" issue
- "su" command allows users to run as root
  - either root shell or single command
  - must supply root password
  - means likely too many people know this
- SELinux RBAC can limit root authority, complex
- "sudo" allows users to run as root
  - but only need their password, not root password
  - /etc/sudoers file specifies what commands allowed
- or configure user/group perms to allow, tricky

# Logging

- effective logging a key resource
- Linux logs using syslogd or Syslog-NG
  - receive log data from a variety of sources
  - sorts by **facility** (category) and **severity**
  - writes log messages to local/remote log files
- Syslog-NG preferable because it has:
  - variety of log-data sources / destinations
  - much more flexible "rules engine" to configure
  - can log via TCP which can be encrypted
- should check and customized defaults

# Log Management

- balance number of log files used
  - size of few to finding info in many
- manage size of log files
  - must rotate log files and delete old copies
  - typically use logrotate utility run by cron
  - to manage both system and application logs
- must also configure application logging

# Application Security

- this is a large topic (other course - Software Security)
- many security features are implemented in similar ways across different applications
- will review issues such as:
  - running as unprivileged user/group
  - running in chroot jail
  - modularity
  - encryption
  - logging

# Running in chroot Jail

- chroot confines a process to a subset of /
  - maps a virtual "/" to some other directory
  - useful if have a daemon that should only access a portion of the file system, e.g. FTP
  - directories outside the chroot jail aren't visible or reachable at all
- contains effects of compromised daemon
- complex to configure and troubleshoot
  - must mirror portions of system in chroot jail

# Modularity

- applications running as a single, large, multipurpose process can be:
  - more difficult to run as an unprivileged user
  - harder to locate / fix security bugs in source
  - harder to disable unnecessary functionality
- hence modularity a highly prized feature
  - providing a much smaller attack surface
- cf. postfix vs sendmail, Apache modules

# Encryption

- sending logins & passwords or application data over networks in clear text exposes them to network eavesdropping attacks
- hence many network applications now support encryption to protect such data
  - often using OpenSSL library
- may need own X.509 certificates to use
  - can generate/sign using openssl command
  - may use commercial/own/free CA

# Logging

- applications can usually be configured to log to any level of detail (debug to none)
- need appropriate setting
- must decide if use dedicated file or system logging facility (e.g. syslog)
  - central facility useful for consistent use
- must ensure any log files are rotated

# Mandatory Access Controls

- Linux uses a DAC security model
- but Mandatory Access Controls (MAC) impose a global security policy on all users
  - users may not set controls weaker than policy
  - normal admin done with accounts without authority to change the global security policy
  - but MAC systems have been hard to manage
- Novell's SuSE Linux has AppArmor
- RedHat Enterprise Linux has SELinux
- pure SELinux for high-sensitivity, high-security

# SELinux

- is NSA's powerful implementation of mandatory access controls for Linux
- Linux DACs still applies, but if it allows the action SELinux then evaluates it against its own security policies
- "subjects" are processes (run user cmds)
- actions are "permissions"
- objects not just files & dirs
- to manage complexity SELinux has:
  - "that which is not expressly permitted, is denied"
  - groups of subjects, permissions, and objects

# Topics

- Base OS Security
  - OS Security Layers
    - Typical measures in OS Security Management
  - Security Maintenance
  - Linux/Unix Security Issues
  - ▶ Windows Security Issues
  - Role of Virtualization

# Windows 8/10 Security Design Principles

- **Access control lists** (**ACLs**) – both attribute-based and claim-based
- Rudimentary capabilities functionally called **integrity levels**
- File system and communication encryption
- Exploit mitigations – **address-space layout randomization** (**ASLR**), **Data Execution Prevention** (**DEP**) ...
- Several digital signature facilities

# Windows 8/10 Security Design Principles

- Security is based on user accounts
  - Each user has unique security ID
  - Login to ID creates security access token
    - Includes security ID for user, for user's groups, and special privileges
    - Every process gets copy of token
    - System checks token to determine if access allowed or denied
- Uses a subject model to ensure access security
  - A subject tracks and manages permissions for each program that a user runs
- Each object in Windows has a security attribute defined by a security descriptor
  - For example, a file has a security descriptor that indicates the access permissions for all users

# Windows 8/10 Security Design Principles

- Win added mandatory integrity controls – assigns **integrity label** to each securable object and subject
  - Subject must have access requested in discretionary access-control list to gain access to object
- Security attributes described by security descriptor
  - Owner ID, group security ID, discretionary access-control list, system access-control list
- Objects are either **container objects** (containing other objects, for example a file system directory) or **noncontainer objects**
  - By default an object created in a container inherits permissions from the parent object

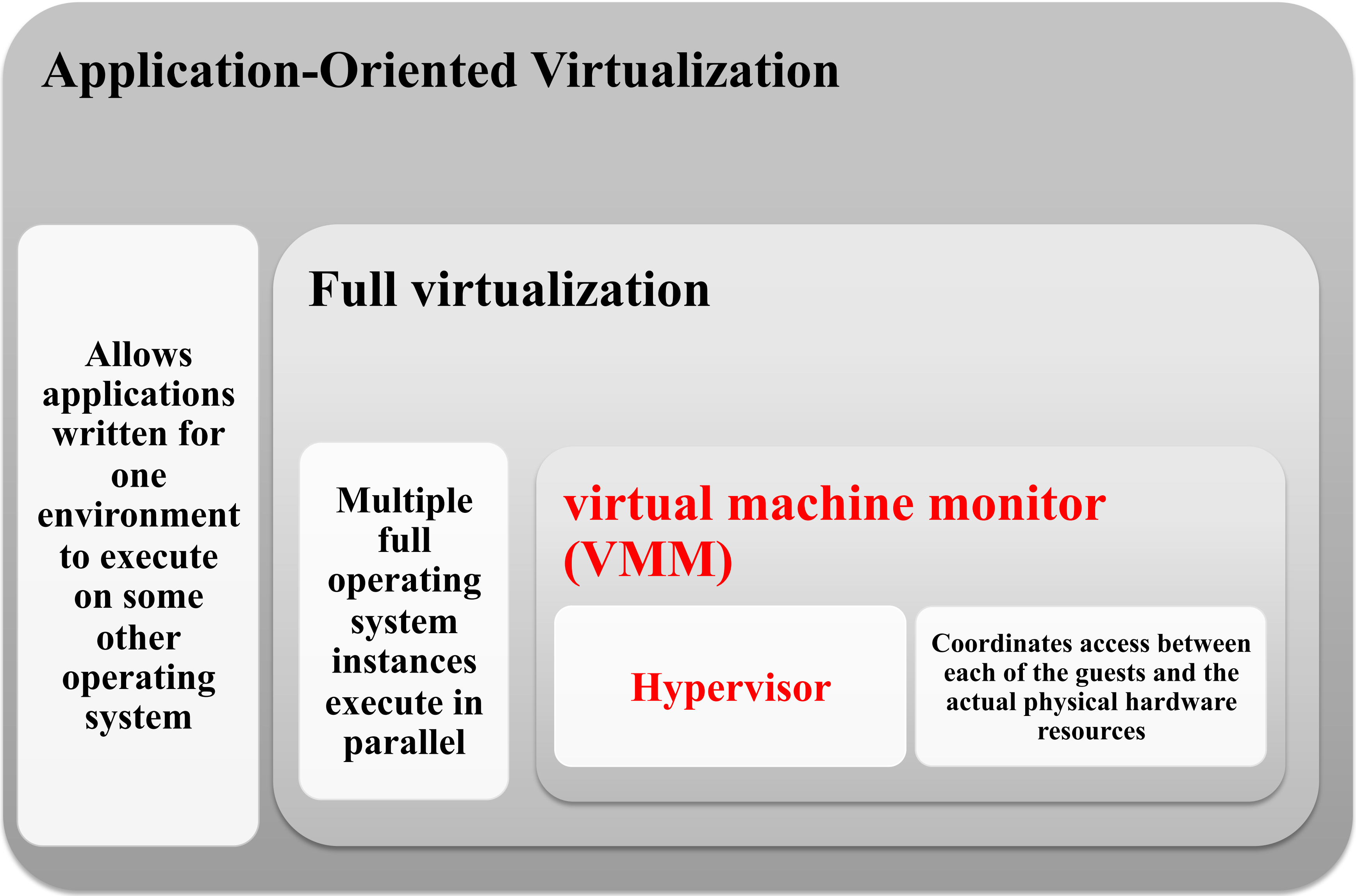
# Topics

- Base OS Security
  - OS Security Layers
    - Typical measures in OS Security Management
  - Security Maintenance
  - Linux/Unix Security Issues
  - Windows Security Issues
  - ▶ Role of Virtualization

# Virtualization

- Technology that provides an abstraction of the resources used by some software which runs in a simulated environment called a virtual machine (VM)
- Benefits include:
  - better efficiency in the use of the physical system resources
  - provides support for multiple distinct operating systems and associated applications on one physical system
  - And (last but not the least !!!!): **raises additional security concerns**

# Virtualization Approaches



Application-Oriented Virtualization

Allows applications written for one environment to execute on some other operating system

Full virtualization

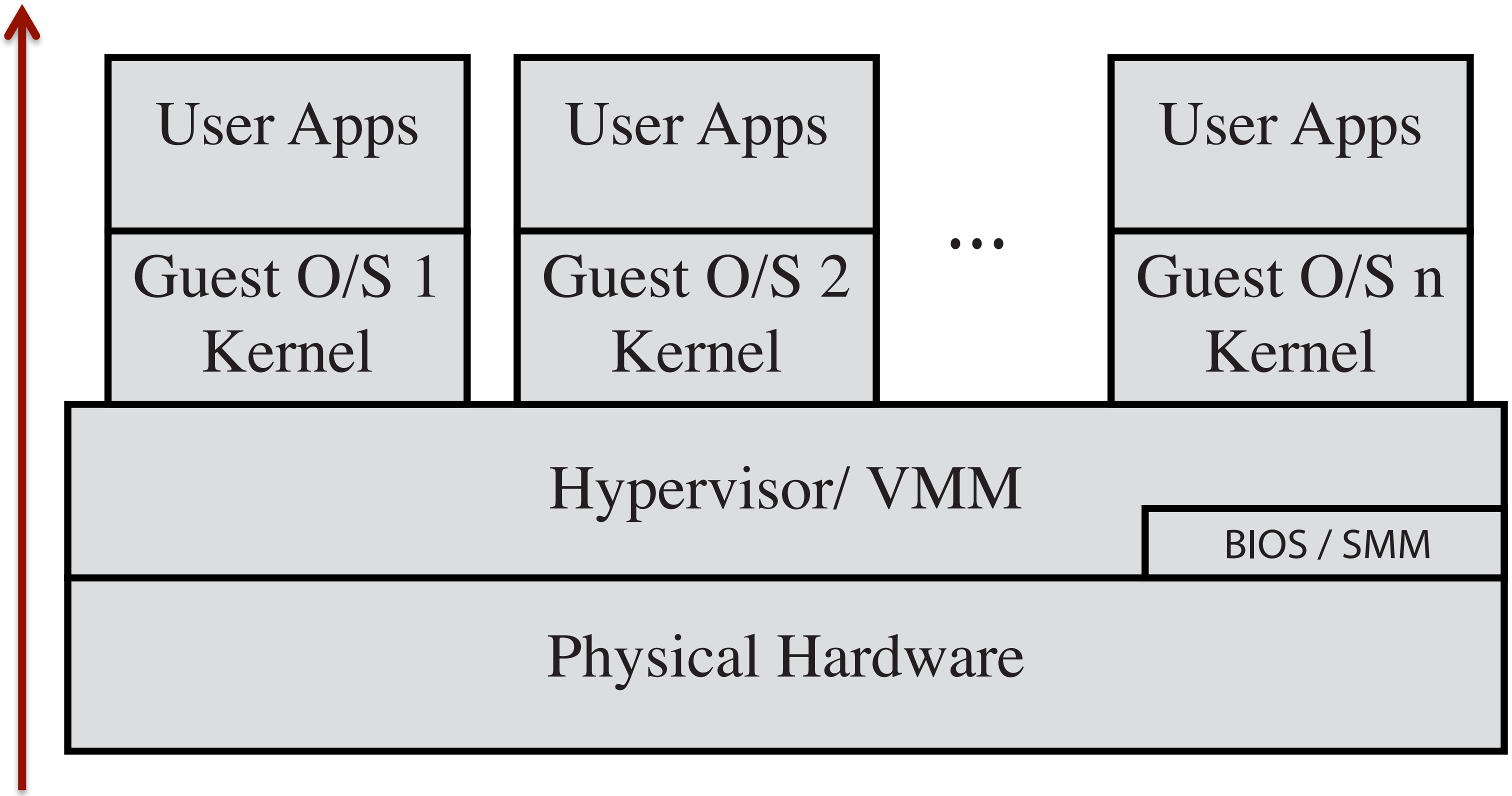
Multiple full operating system instances execute in parallel

virtual machine monitor (VMM)

Hypervisor

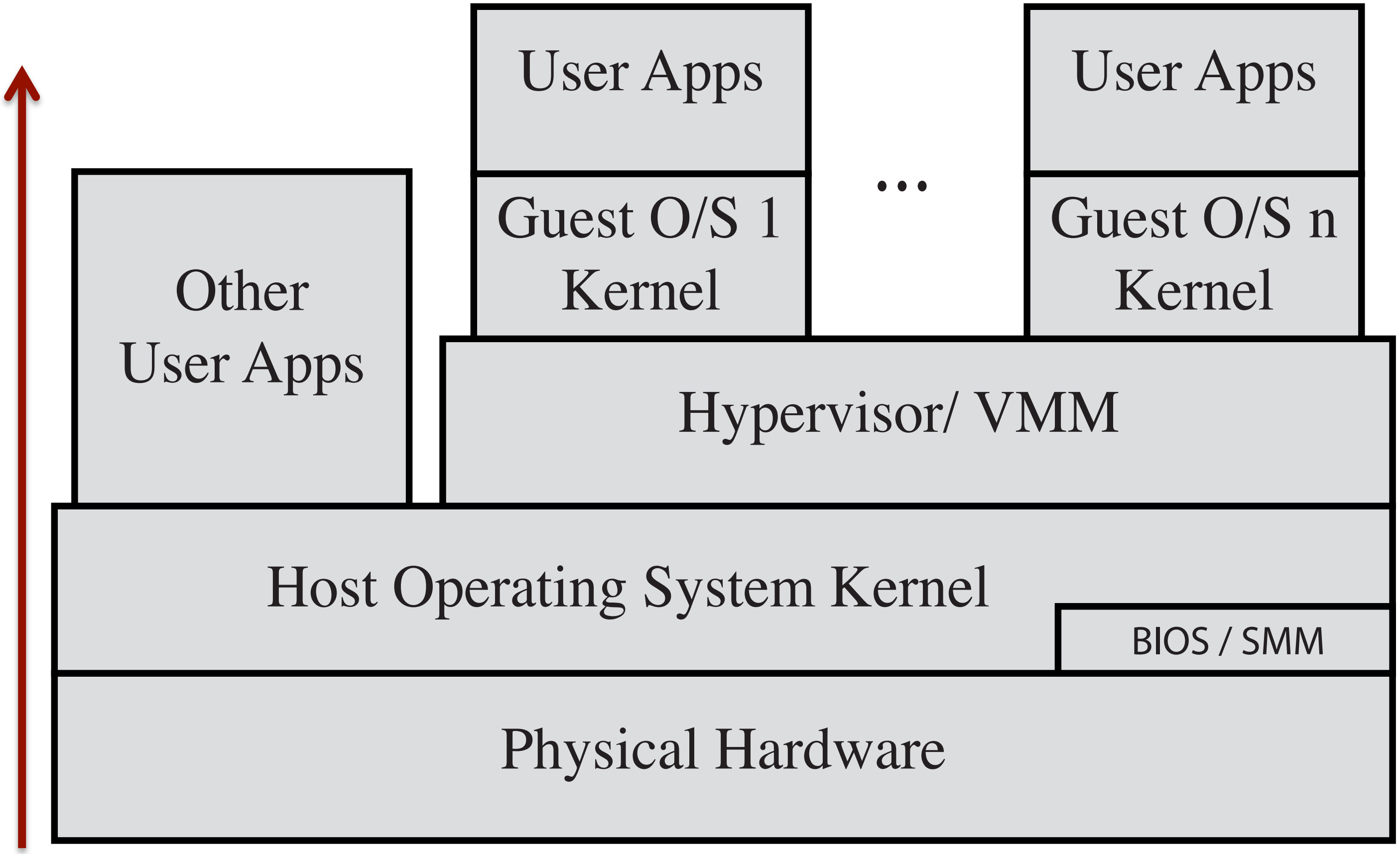
Coordinates access between each of the guests and the actual physical hardware resources

# Native Virtualization Security Layers



Native Virtualization Security Layers

# Hosted Virtualization Security Layers



Native Virtualization Security Layers

# Security issues in Virtualization

- Security concerns include different levels of approach:
  - Guest OS isolation
    - To ensuring that programs executing within a guest OS may only access and use the resources allocated to it and nothing more
  - Guest OS monitoring by the hypervisor
    - Which has privileged access to the programs and data in each guested OS
  - Virtualized environment security
    - Particularly image and snapshot management which attackers may attempt to view or modify

# Securing Virtualization Systems

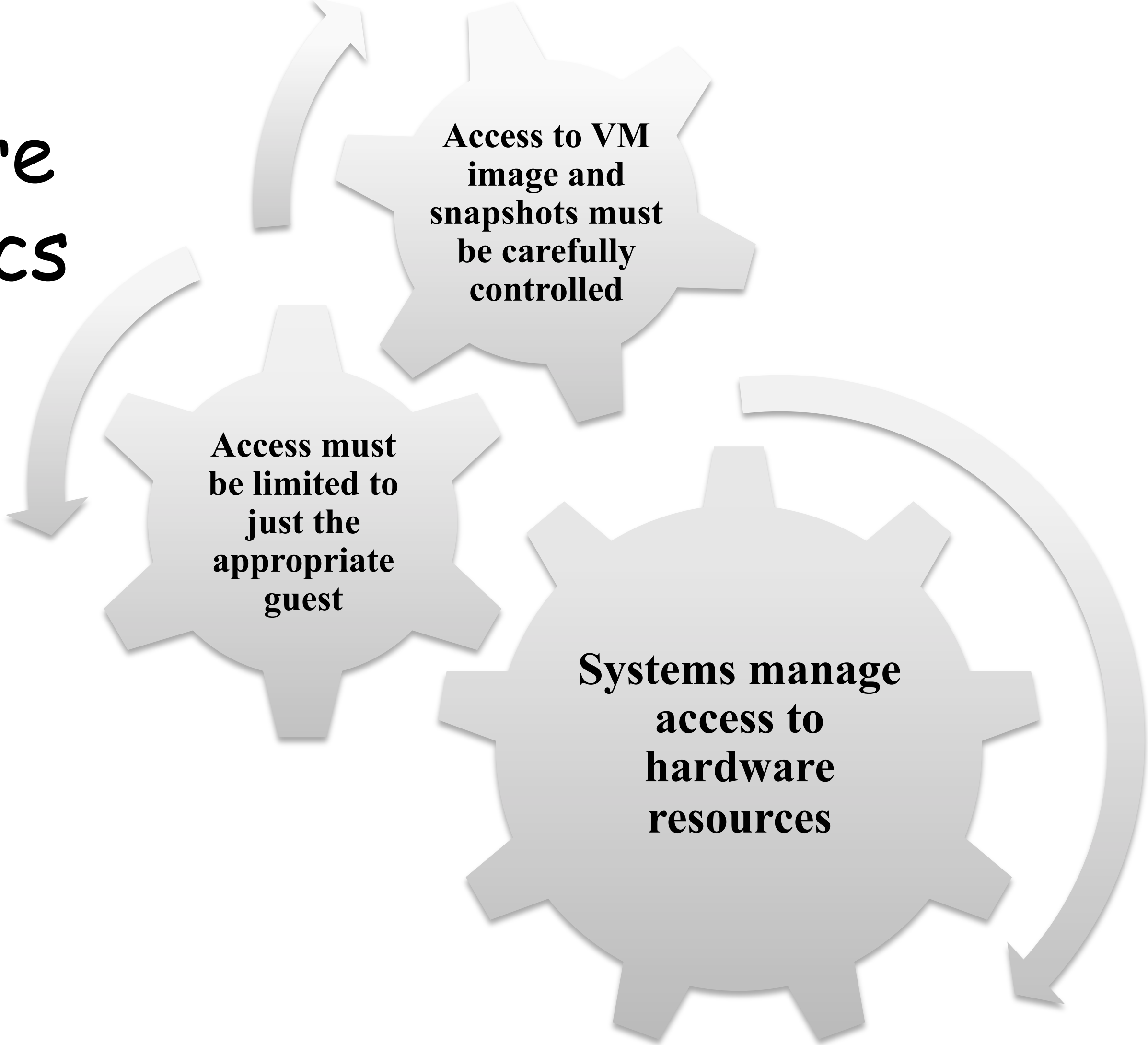
**Security issues that must be addressed when virtualization is used.**

- Carefully plan the security of the virtualized system
- Secure all elements of a full virtualization solution and maintain their security
- Ensure that the hypervisor is properly secured
- Restrict and protect administrator access to the virtualization solution

# Hypervisor Security

- The Hypervisor should be
  - Secured using a process similar to securing an operating system
  - Initially installed in an isolated environment
  - Configured so that it is updated automatically
  - Monitored for any signs of compromise
  - Accessed only by authorized administration
- May support both local and remote administration so must be configured appropriately
- Remote administration access should be considered and secured in the design of any network firewall and IDS capability in use
- Ideally administration traffic should use a separate network with very limited access provided from outside the organization

# Virtualization Infrastructure Security Basics



Access to VM image and snapshots must be carefully controlled

Access must be limited to just the appropriate guest

Systems manage access to hardware resources

# Security of virtualized resources (1)

- Virtualized systems manage access to hardware resources such as disk storage and network interfaces.
- This access must be limited to just the appropriate guest OSs that use any resource.
  - As we noted, the configuration of network interfaces and use of an internal virtual network may present issues for organizations that wish to monitor all network traffic between systems. This should be designed and handled as needed.
- Access to VM images and snapshots must be carefully controlled, since these are another potential point of attack.

# Security of virtualized resources (2)

- Hosted virtualized systems, as typically used on client systems, pose some additional security concerns.
  - These result from the presence of the host OS under, and other host applications beside, the hypervisor and its guest OSes.
  - Hence there are yet more layers to secure !
  - Further, the users of such systems often have full access to configure the hypervisor, and to any VM images and snapshots.
    - In this case, the use of virtualization is more to provide additional features, and to support multiple operating systems and applications, than to isolate these systems and data from each other, and from the users of these systems.

# Security of virtualized resources (3)

- It is possible to design a host system and virtualization solution that is more protected from access and modification by the users.
  - This approach may be used to support well-secured guest OS images used to provide access to enterprise networks and data, and to support central administration and update of these images.

- However, there will remain security concerns from possible compromise of the underlying host OS, unless it is adequately secured and managed.

# Suggested readings (bibliography)

- W. Stallings, L. Brown, Computer Systems Security

  Chap. 12

  See also for Linux Security and Windows Security

  Chap. 25 – Linux Security

  Chap 26 – Windows and Windows Vista Secruity

  * Ver material disponibilizado (CLIP)