chp 4

Access Control Definition

↳ process of regulating access to system resources based on security policy and only permitted by authorised users, programs to use them

Basic Security Requirements	
1	Limit information system access to authorized users, processes acting on behalf of authorized users, or devices (including other information systems).
2	Limit information system access to the types of transactions and functions that authorized users are permitted to execute.
Derived Security Requirements	
3	Control the flow of CUI in accordance with approved authorizations.
4	Separate the duties of individuals to reduce the risk of malevolent activity without collusion.
5	Employ the principle of least privilege, including for specific security functions and privileged accounts.
6	Use non-privileged accounts or roles when accessing nonsecurity functions.
7	Prevent non-privileged users from executing privileged functions and audit the execution of such functions.
8	Limit unsuccessful logon attempts.
9	Provide privacy and security notices consistent with applicable CUI rules.
10	Use session lock with pattern-hiding displays to prevent access and viewing of data after period of inactivity.
11	Terminate (automatically) a user session after a defined condition.
12	Monitor and control remote access sessions.
13	Employ cryptographic mechanisms to protect the confidentiality of remote access sessions.
14	Route remote access via managed access control points.
15	Authorize remote execution of privileged commands and remote access to security-relevant information.
16	Authorize wireless access prior to allowing such connections.
17	Protect wireless access using authentication and encryption.
18	Control connection of mobile devices.
19	Encrypt CUI on mobile devices.
20	Verify and control/limit connections to and use of external information systems.
21	Limit use of organizational portable storage devices on external information systems.
22	Control CUI posted or processed on publicly accessible information systems.

CUI = controlled unclassified information

Access Control Principles

↳ computer security largely revolves around access control

↳ ensure only authorised users/processes can interact with specific system resources

Computer security

↳ measurses that ensure security services

↳ especially those that enforce access control

↳ shows AC as a single module (Access control) but in reality involves multiple components working together

↳ The AC mechanism acts as a mediator b/w a user and system resources

↳ Access is granted through these 3 functions

1. Authentification

↳ verifies if the entity can access the system at all

2. Authorization

↳ checks the AC database (managed by security administrator) to see if the specific request is permitted

3. Audit

↳ tracks and records user access to the system resources for monitoring purposes

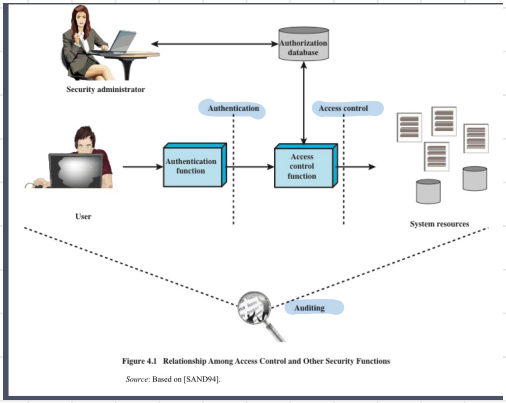


Figure 4.1 Relationship Among Access Control and Other Security Functions

Source: Based on [SAND94].

Access Control Principles

1. Discretionary Access Control (DAC)
↳ traditional method of AC

↳ Grants access based on
  ↳ users identity and
  ↳ Predefined rules → states what requestors are allowed to do

2. Mandatory Access Control (MAC)
↳ best for military info security

↳ Grants access by
  matching security labels of resources
  with users security clearances

3. Role Based Access Control (RBAC)
↳ Grants access based on the
  ↳ users role within the system and
  ↳ rules specifying permissions for each role

have become increasingly popular

4. Attribute Based Access Control (ABAC)
↳ Grants access based on
  ↳ user attributes
  ↳ resource characteristics
  ↳ enviormental conditions

BASIC ELEMENTS OF AC

1. Subject
→ e.g users

↳ an entity that can access objects

↳ are held accountable for actions they iniate
  which may be recorded in audit trials

↳ 3 main types of subjects
  1. Owner : creator/administrator with full control over a resource
  2. Group : group of user (can belong to multiple groups) who are granted shared access rights
  3. World : users with least access, not in owners/group category

2. Object
e.g databases

↳ a resource controlled by access permissions

↳ It's an entity used to contain/receive information
  e.g files, directories, mailboxes, messages, pages
  bits, bytes, words, network ports

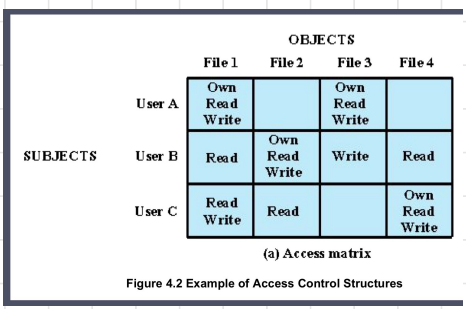
3. Access rights
↳ defines how a subject interacts with an object
  ↳ read
  ↳ write
  ↳ execute
  ↳ delete
  ↳ create
  ↳ search

1. Discretionary Access Control (DAC)

↳ Grants access to others for certain resources
↳ Represented by an access matrix

Access Matrix

↳ Rows: subjects (entities that request access)
↳ Columns: Objects (resources to be accessed)
↳ each matrix entry defines the specific access rights a subject has for an object



		OBJECTS			
		File 1	File 2	File 3	File 4
SUBJECTS	User A	Own Read Write		Own Read Write	
	User B	Read	Own Read Write	Write	Read
	User C	Read Write	Read		Own Read Write

(a) Access matrix

Figure 4.2 Example of Access Control Structures

↳ user A owns file 1,3 and has read, write access to those
↳ User B has read access to file 1,4

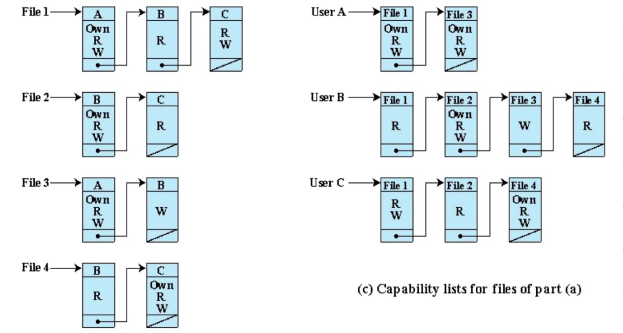
Access Matrix
↳ is often too large and sparce, to implement directly
↳ So its broken down in 2 ways

1. Access control lists (ACLs)

↳ breaks down matrix by columns
↳ with each obj having
  ↳ list of users
  ↳ their permitted access rights
PRO ↳ easy to see which user has access to a specif resource
CON ↳ hard to determine permissions of a particular user across resource

2 Capability Tickets

↳ breaks down matrix by rows
↳ with each user holding tickets that specify
  ↳ their authorised objects
  ↳ their operations
PRO ideal for distributed enviorments where securing each ticket may be difficult
CON ↳ users may share these tickets with others which raises security disadvantages of protection
SOL the system can store the tickets in secure memory or include a unforgetable token that is verified when access requested



(b) Access control lists for files of part (a)

(c) Capability lists for files of part (a)

Figure 4.2 Example of Access Control Structures

Authorization Table

- an alternate to Access Matrix for AC
  - row → 1 access right for a single subject to a specific resource
- can function like both ACLs and capability lists
  - sorting by subject (similar view to) → capability list
  - sorting by object (similar view to) → ACL

PRO
- can be efficiently implemented using a relational database making it easy to manage and access
- fully populated (unlike the sparse access matrix)

Subject	Access Mode	Object
A	Own	File 1
A	Read	File 1
A	Write	File 1
A	Own	File 3
A	Read	File 3
A	Write	File 3
B	Read	File 1
B	Own	File 2
B	Read	File 2
B	Write	File 2
B	Write	File 3
B	Read	File 4
C	Read	File 1
C	Write	File 1
C	Read	File 2
C	Own	File 4
C	Read	File 4
C	Write	File 4

Table 4.2

Authorization Table for Files in Figure 4.2

(Table is on page 113 in the textbook)

		OBJECTS								
		subjects			files		processes		disk drives	
		S_1	S_2	S_3	F_1	F_2	P_1	P_2	D_1	D_2
SUBJECTS	S_1	control	owner	owner control	read *	read owner	wakeup	wakeup	seek	owner
	S_2		control		write *	execute			owner	seek *
	S_3			control		write	stop			

* - copy flag set

Figure 4.3 Extended Access Control Matrix

read in A[S1, F1] means
sub S1 has read access
to file F1

Extended ACM

Access Control Matrix

↳ to represent protection state, ACM is extended to include not just files but

- ↳ Processes (rights may include) stoping/waking processes
- ↳ Devices (rights may include) reading/writing, controlling, blocking/unblocking the device
- ↳ Memory locations (rights may include) reading/writing protected memory regions
- ↳ Subjects (rights may include) granting/deleting access rights for other subjects

Access Control Function

↳ everytime a subject attempts to access an obj, the controller of that obj, check the current Access Matrix to decide wether an access should be allowed

↳ an access attempt triggers the following steps

1. Request
   a sub S_0 request access (read/right) to obj X
2. Message Generation:
   the system generates a message $(S0, \alpha, X)$ (α: type of access requested)
3. Access check:
   the controller for X, checks if α is in A[S0, X]
   - ↳ if exists → access granted
   - ↳ if not exist → access denied

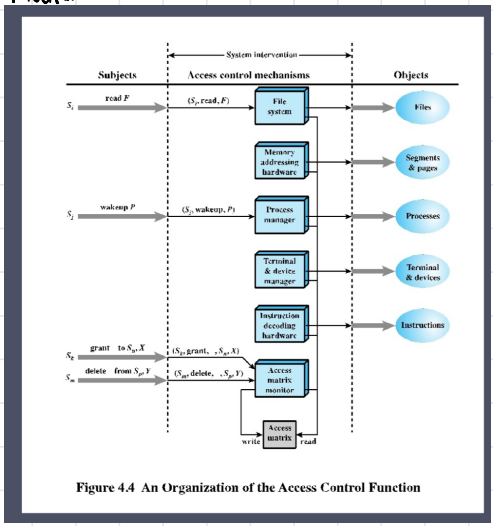


Figure 4.4 An Organization of the Access Control Function

Rule	Command (by S_o)	Authorization	Operation
R1	**transfer** $\left\{ \begin{matrix} \alpha * \\ \alpha \end{matrix} \right\}$ **to** S, X	'α*' in $A[S_o, X]$	store $\left\{ \begin{matrix} \alpha * \\ \alpha \end{matrix} \right\}$ in $A[S, X]$
R2	**grant** $\left\{ \begin{matrix} \alpha * \\ \alpha \end{matrix} \right\}$ **to** S, X	'owner' in $A[S_o, X]$	store $\left\{ \begin{matrix} \alpha * \\ \alpha \end{matrix} \right\}$ in $A[S, X]$
R3	**delete** α **from** S, X	'control' in $A[S_o, S]$ or 'owner' in $A[S_o, X]$	delete α from $A[S, X]$
R4	$w \leftarrow$ **read** S, X	'control' in $A[S_o, S]$ or 'owner' in $A[S_o, X]$	copy $A[S, X]$ into w
R5	**create object** X	None	add column for X to A; store 'owner' in $A[S_o, X]$
R6	**destroy object** X	'owner' in $A[S_o, X]$	delete column for X from A
R7	**create subject** S	none	add row for S to A; execute **create object** S; store 'control' in $A[S, S]$
R8	**destroy subject** S	'owner' in $A[S_o, S]$	delete row for S from A; execute **destroy object** S

Table 4.3

Access Control System Commands

(Table is on page 116 in the textbook)

Protection Domains:
A collection of objects along with the access rights to those objects.

- **Flexible capabilities**: Access rights can be associated with different domains.
- **Access matrix relation**: Each row in the matrix represents a protection domain.
- **Process control**: Users can create processes with limited access rights.
- **Static or dynamic association**: A process can be tied to a domain either permanently or temporarily.
- **User mode:**
  - Restricted access to certain memory areas and instructions.
- **Kernel mode:**
  - Allows execution of privileged instructions and access to protected memory.

UNIX Files Access Control

↳ UNIX manages files using inodes (index nodes)
which store essential info about each file

Inodes

↳ Controls structure that contains attributes, permissions and control data
↳ Multiple file names can point to the same inode
↳ each active node corresponds to exactly 1 file

Inode Table

↳ located on disk
↳ holds the inodes for all files in the system
↳ when a file is opened, its inode is loaded into a memory resident inode table

Directories

↳ organised as a hierarchical tree
↳ can contain both files and subdirectories
↳ store file names and pointers to their corresponding inodes

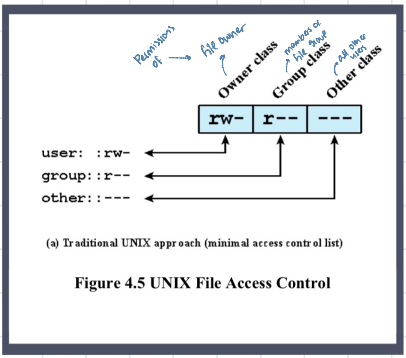
UNIX File Access Control

User Identification:
- Each UNIX user has a unique user ID (UID).
- Users belong to a primary group and potentially other groups, each identified by a group ID (GID).

File Ownership:
- Files are owned by a specific user (identified by UID) and belong to a group (either the creator's primary group or the group of the parent directory with SetGID permission).

Protection Bits:
- Each file has 12 protection bits stored in its inode.
- Permissions:
  - Nine bits specify read, write, and execute permissions:
    - Owner: Permissions for the file's owner.
    - Group: Permissions for members of the file's group.
    - Others: Permissions for all other users.
- Permissions follow a hierarchy: owner > group > others, with the most relevant permissions applying.

Directory Permissions:
- For directories:
  - Read and write bits allow listing and creating/deleting files.
  - Execute bit allows users to access or search within the directory.



(a) Traditional UNIX approach (minimal access control list)

Figure 4.5 UNIX File Access Control

Traditional UNIX File Access Control

Special Permissions and User Access in UNIX

Special Permission Bits:

SetUID (Set User ID):
- When set on an executable file, it allows the user executing the file to temporarily gain the privileges of the file's owner.
- This is useful for running privileged programs that need access to restricted files.

SetGID (Set Group ID):
- When set on an executable, it grants the executing user the privileges of the file's group.
- When set on a directory, new files created within inherit the directory's group.

Sticky Bit:
- When applied to a file, it is no longer used.
- When set on a directory, it allows only the owner of a file to rename, move, or delete that file, aiding in management of shared temporary directories.

Superuser:
- The superuser has unrestricted access across the system and is exempt from normal file access controls.
- Programs owned by the superuser with SetUID can grant any user full access to the system, requiring careful handling.

Access Management:
- The traditional UNIX access control works well for simple file sharing among a few groups.
- In cases with many user groups needing varied access, managing permissions becomes complex.
- Access Control Lists (ACLs) can be used in modern UNIX systems to address this complexity.

Protection Domain Structure:
- The UNIX access control scheme features a simple protection domain structure, where changing the user ID corresponds to switching domains.

Access Control Lists (ACLs) in UNIX

Modern UNIX systems support ACLs

- FreeBSD, OpenBSD, Linux, Solaris

FreeBSD allows

- Setfacl command assigns a list of UNIX user IDs and groups
- Any number of users and groups can be associated with a file
- Read, write, execute protection bits
- A file does not need to have an ACL
- Includes an additional protection bit that indicates whether the file has an extended ACL

When a process requests access to a file system object two steps are performed:

- Step 1 selects the most appropriate ACL
- Step 2 checks if the matching entry contains sufficient permissions

FreeBSD and many UNIX systems with extended ACLs (Access Control Lists) use this approach:

1. The owner and other classes in the 9-bit permission field work like in basic ACLs.
2. The group class shows the permissions for the file's owner group, which sets the maximum permissions that can be assigned to named users or groups other than the owner. This group entry acts as a mask.
3. Additional named users and groups can have their own 3-bit permission fields. Their permissions are checked against the mask; if a permission isn't in the mask, it is denied.

When a process wants to access a file, two steps happen:

1. The system finds the ACL entry that best matches the process. It checks in this order: owner, named users, groups (owning or named), and others. Only one entry controls access.
2. The system checks if the matching entry has enough permissions. A process can belong to multiple groups, so several group entries might match. If any matching group entry has the needed permissions, access is granted. If none do, access is denied, regardless of which entry is checked.

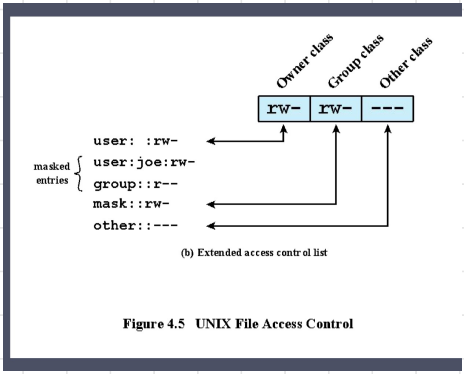


Figure 4.5 UNIX File Access Control

Traditional Discretionary Access Control (DAC) systems
set access rights for individual users and user groups.

In contrast,

Role-Based Access Control (RBAC)

- focuses on the roles users have in a system rather than their identities.
- RBAC defines roles based on job functions within an organization and assigns access rights to these roles instead of to individual users.
- Users can be assigned to different roles based on their responsibilities, either permanently or as needed.
- the relationships between users and roles, are many-to-many.
- The number of users can change frequently, and users may be dynamically assigned to multiple roles.

However, the set of roles in a system usually remains stable, with only occasional changes.
Each role has specific access rights to various resources, and these rights are not likely to change often.

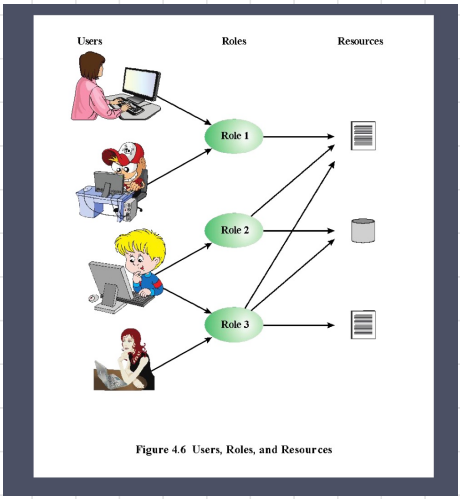


Figure 4.6 Users, Roles, and Resources

Access Control Matrix Representation of RBAC

↳ used to depict key elements of RBAC system

↳ The top half

↳ each matrix entry is either

↳ marked → indicates user is assigned to this role

a single user can be assigned to multiple roles → more than 1 mark in a column

↳ blank

↳ The bottom half

↳ similar to DAC matrix

↳ with roles → subjects

↳ can also be treated as objects

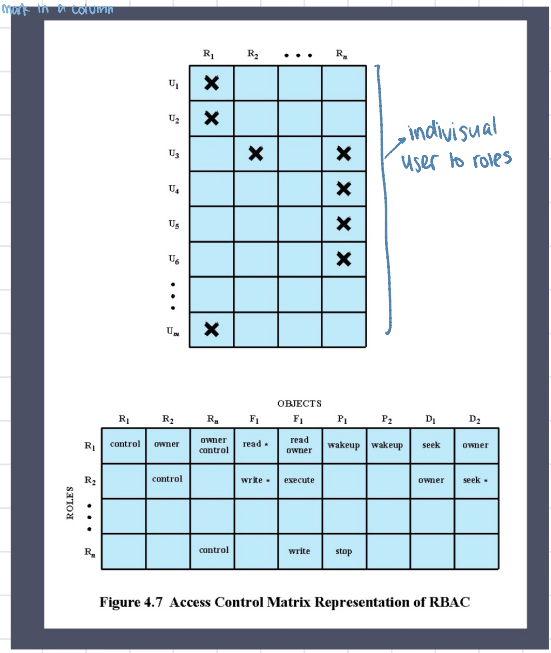


Figure 4.7 Access Control Matrix Representation of RBAC

RBAC (Role-Based Access Control)
can include various functions and services,

- outlines four related RBAC models, as shown in Figure 4.8a and Table 4.4:
  RBAC0: The basic model with minimum functionality.
  RBAC1: Builds on RBAC0 by adding role hierarchies, allowing roles to inherit permissions from others.
  RBAC2: Includes RBAC0 and adds constraints to limit how components can be configured.
  RBAC3: Combines all features of RBAC0, RBAC1, and RBAC2.

- In the RBAC0 model (Figure 4.8b), there are four key entities:
  User: An individual with access to the system, identified by a user ID.
  Role: A job function that defines authority and responsibilities within the system.
  Permission: Approval for a specific type of access to resources, also called access rights or privileges.
  Session: A link between a user and a selected set of roles they can use at a given time.

  The diagram shows relationships:
- There is a many-to-many relationship between users and roles,
  - meaning one user can have multiple roles
  - many users can share a role.
- There is also a many-to-many relationship between roles and permissions.
- A session allows a user to connect with one or more roles temporarily,
  using only the roles necessary for a specific task,
  reflecting the principle of least privilege.

- These many-to-many relationships give RBAC
  more flexibility and detail than
  traditional DAC (Discretionary Access Control) systems.

  Without this flexibility, users might get more access than needed.
  For example, users may need to list directories and
  modify files but not create new ones,
  or they may need to add records to a file without changing existing ones.

one → many

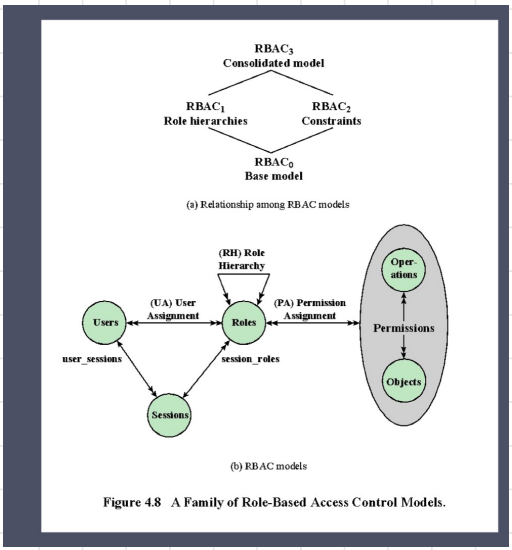


Figure 4.8 A Family of Role-Based Access Control Models.

Table 4.4
Scope RBAC Models

Models	Hierarchies	Constraints
$RBAC_0$	No	No
$RBAC_1$	Yes	No
$RBAC_2$	No	Yes
$RBAC_3$	Yes	Yes

Role hierarchy

- illustrate the ranking of job roles in an organization,
- where roles with more responsibility have greater authority to access resources.

- A lower-level role typically inherits some access rights from its higher-level counterpart.
- This inheritance allows one role to include the access rights of a subordinate role.

In the accompanying diagram (Figure 4.9),
- subordinate roles are placed lower.
- A line connecting two roles shows that the upper role encompasses all access rights of the lower role, along with additional rights not available to the lower role.
- A single role can inherit rights from multiple subordinate roles.

- For instance,
  - the Project Lead role inherits rights from both the Production Engineer and Quality Engineer roles.
  - Meanwhile, both the Production Engineer and Quality Engineer roles inherit rights from the Engineer role,
  - but each has its own additional access rights as well.
- Therefore, these two roles share some access rights from the Engineer role while also having unique ones.

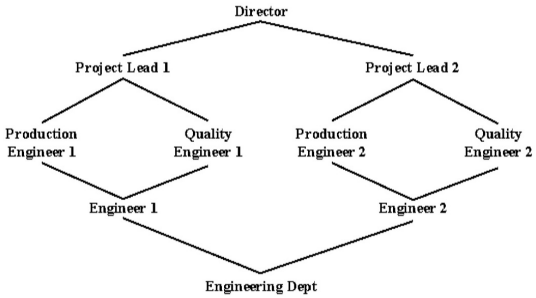


Figure 4.9 Example of Role Hierarchy

Constraints in RBAC

Role Based Access Control Models

↳ allows RBAC to lift an organizations specific administrative and security needs
↳ role relationships defines how roles relate to one another
or sets conditions on roles

TYPES OF CONSTRAINTS

Mutually Exclusive Roles

↳ a user can only hold 1 role in a defined set
either → during a session
→ as permanent assignment

↳ Permissions with this set can be assigned to only one role

Cardinality

↳ limits the max no of users assigned to certain roles

Prerequisite Roles

↳ requires users to specifically hold a specific role before being assigned another role

3. ABAC

Attribute Based Access Control

↳ defines access based on conditions related to both resource and subject attributes

PROS

↳ highly flexible and expressive

CONS

↳ Performance concerns arise from evaluating conditions on both user and resource attributes for each access request

applications

↳ web services

↳ explored for cloud services

ABAC Model: Attributes

Subject Attributes

↳ defines the identity and characteristic of the active entity

↳ is an active entity → user → process that initiates actions causing info to flow b/w objects or altering the systems state

Object Attributes

↳ describes the passive entity that contains or receives info

↳ these attributes are used to guide access control decisions

Enviorment Attributes

↳ represents the context in which access occurs.

↳ often overlooked in most access control policies

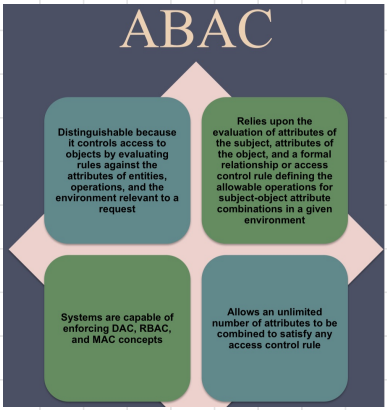


Figure 4.10 shows
the core components of an Attribute-Based Access Control (ABAC) system in a logical structure.
Here's how access works in this system:

1. A subject (user) requests access to an object (resource).
   This request goes to the access control mechanism.
   The access control mechanism checks a set of predefined rules (2a) within an access control policy.

2. It evaluates:
   Attributes of the subject (2b),
   Attributes of the object (2c),
   Environmental conditions (2d) at the time of the request.
3. If these rules authorize access,
   the mechanism grants it;
   otherwise, access is denied.

This setup uses four independent sources of information to decide on access.
The system designer chooses relevant attributes for subjects, objects, and environmental factors
and can define rules for any combination of these attributes,
making this approach highly flexible and adaptable.

However, this flexibility comes at a cost,
as it adds complexity to both the design and implementation,
potentially affecting system performance.
Balancing this tradeoff is up to the system authority.

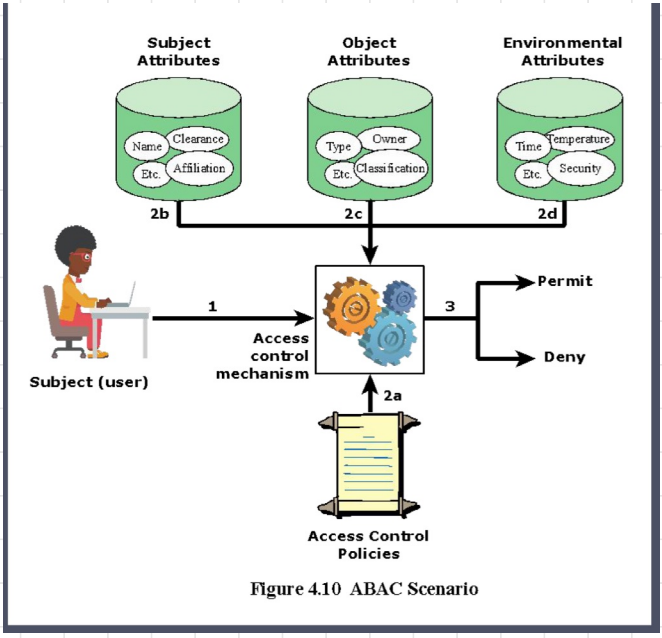


Figure 4.10 ABAC Scenario

Q3.

Consider the access control policy of an online entertainment store in Figure 1, which we studied in the class.

```
R1:can_access(u, m, e) ←
   (Age(u) ≥ 17 ∧ Rating(m) ∈ {R, PG-13, G}) ∨
   (Age(u) ≥ 13 ∧ Age(u) < 17 ∧ Rating(m) ∈ {PG-13, G}) ∨
   (Age(u) < 13 ∧ Rating(m) ∈ {G})
R2:can_access(u,  m,  e) ←
   (MembershipType(u) = Premium) ∨
   (MembershipType(u) = Regular ∧ MovieType(m) = OldRelease)
R3:can_access(u,  m,  e) ←R1 ∧ R2
```

Figure 1

Now perform the following two tasks:

i. Suppose the store takes user attributes from google.com. **Draw a compact free-hand labeled diagram** of an architecture that can decides to allow or deny access to any user. Explain the parameters u, m and e. **[3]**

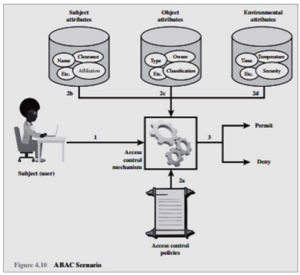


Figure 4.10 **ABAC Scenario**

ii. **Change the policy in Figure 1** to allow access to under thirteen users only when social security number of their guardian is in the database. **[2]**

The social security number of minor might not be available from google.com therefore the store's web portal need to ask and store it while registering the minor user.

Modify under 13 part of R1 → R1: Age(u) < 13 AND Rating(m) ∈ {G} AND SSN_Gardian(u) £ QueryDB (u))

Figure 4.11 contrasts the scope and complexity of ABAC and DAC using access control lists (ACLs). The figure highlights both the structural differences and the trust needs of each model.

In **DAC**,
access control relies primarily on the object owner,
who controls access by adding users to an ACL,
making the owner the main point of trust.

In contrast, **ABAC**
relies on multiple trust sources beyond the object owner,
including Subject Attribute Authorities, Policy Developers, and Credential Issuers.
This broad trust network adds complexity
and requires more coordinated trust relationships
for ABAC to function correctly.

The NIST guide suggests that organizations using ABAC establish a governance body to oversee identity, credential, and access management across the enterprise. Subordinate groups within the organization should have similar bodies to maintain consistency in ABAC deployment. NIST also recommends developing a trust model to define relationships, ownership, and accountability for information. This model aids organizations in confidently sharing information by clarifying its use, protection, and trustworthiness.

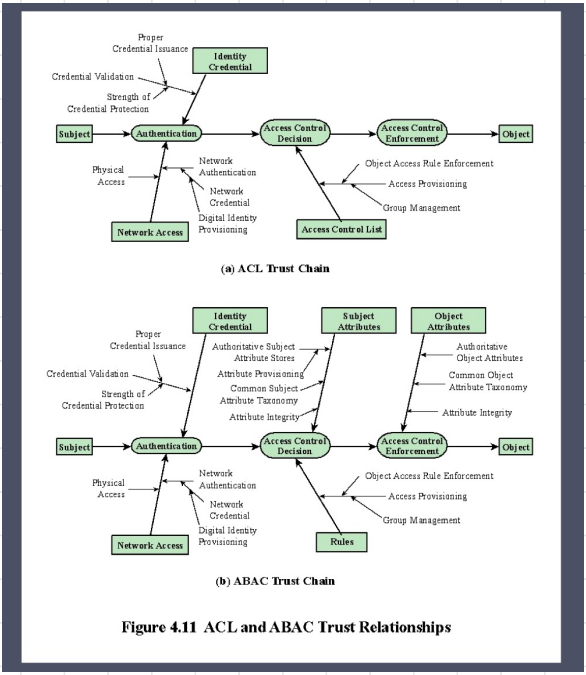


Figure 4.11 ACL and ABAC Trust Relationships

ABAC Policies

↳ a policy is a set of rules that control allowed behavior in an organization based on
  ↳ what subjects are allowed to do
  ↳ how resources are protected in certain conditions

↳ Policies focus on protecting resources and the privileges given to subjects
  aka → rights, authorizations, entitlements

ICAM

Identity, Credential and Access Management

- a system for managing
  - digital identities
  - credentials
  - access control
- developed by the US goverment
- Its goals are to
  - create secure digital identities for individuals and non person entities (NPE)
  - link these identities to credentials which act as proxies for identity in access transactions

Credential

- a secure object that connects an identity to a token controlled by the user
- allowing authorised access to an organizations resources

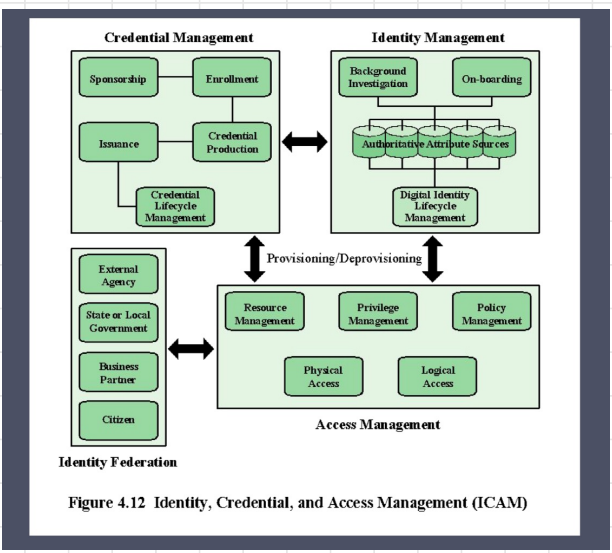
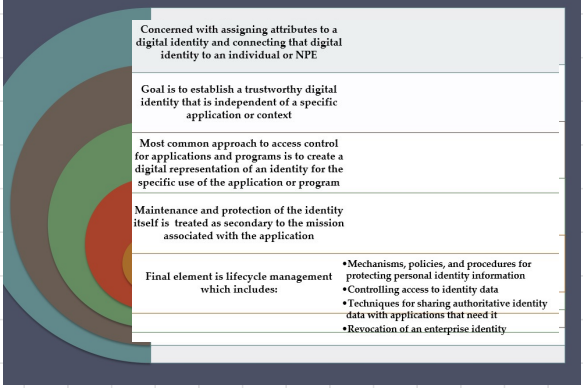


Figure 4.12 Identity, Credential, and Access Management (ICAM)

gives an overview of the logical components of an ICAM architecture

Identity Management



Access Management

Deals with the management and control of the ways entities are granted access to resources

Covers both logical and physical access

May be internal to a system or an external element

Purpose is to ensure that the proper identity verification is made when an individual attempts to access a security sensitive building, computer systems, or data

Three support elements are needed for an enterprise-wide access control facility:

- Resource management
- Privilege management
- Policy management

Credential Management

The management of the life cycle of the credential

Examples of credentials are smart cards, private/public cryptographic keys, and digital certificates

Encompasses five logical components:

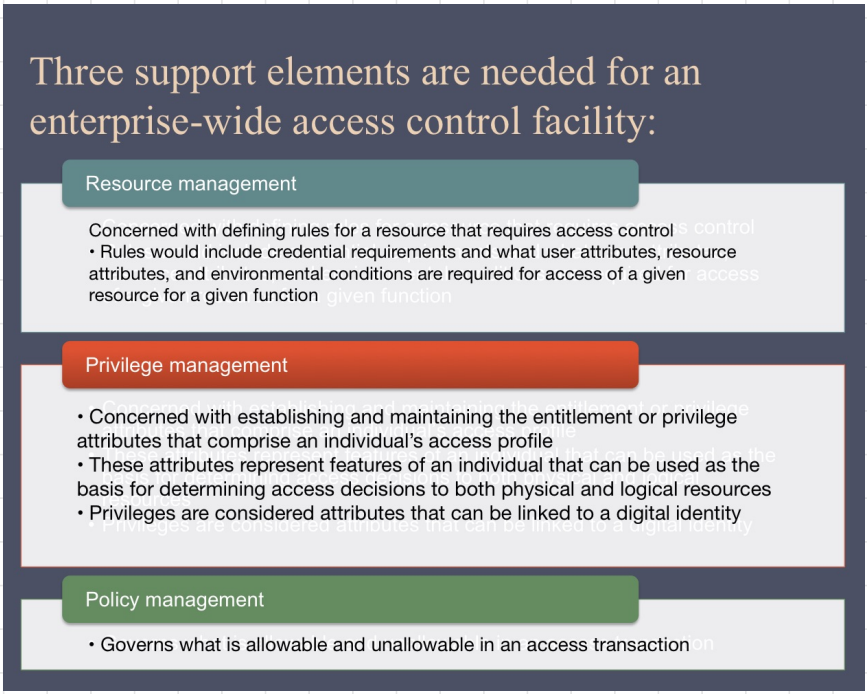
Sponsorship: An authorized person, like a department supervisor, sponsors an individual or entity for a credential to confirm the need for it.

Enrollment: The individual enrolls by verifying their identity and providing biographic and biometric data. This step may also involve linking verified attribute data from the identity management system.

Production: The credential is created, potentially involving encryption, digital signatures, or smart card production, depending on the type.

Issuance: The credential is provided to the individual or non-person entity (NPE).

Maintenance: Over its life, the credential may need actions like revocation, reissuance, reenrollment, expiration updates, PIN resets, suspension, or reinsta



Identity Federation
refers to the technology, standards, policies, and processes that enable organizations to trust digital identities, attributes, and credentials issued by other organizations.
It addresses two main questions:

How can you trust the identities of individuals from outside organizations who need access to your systems?
How can you verify the identities of individuals in your organization when they need access to external systems?

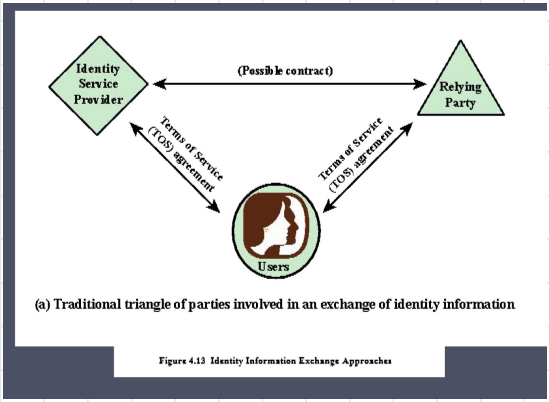
Online or network transactions between different organizations, or between an organization and an individual (like an online customer), typically require sharing identity information, which may include a variety of attributes beyond a basic name or ID.
Both the party sharing and the party receiving this information need mutual trust regarding the security and privacy of the data.

In the traditional setup (as shown in Figure 4.13a), users obtain digital identities and credentials from an identity service provider, which has arrangements with service providers who rely on these credentials for end-user services.

This arrangement has several requirements:

For the Relying Party: They need assurance that the user's identity has been authenticated to a certain standard, that the user's attributes are accurate, and that the identity service provider is authoritative for these attributes.
For the Identity Service Provider: They require assurance that user information is accurate and that the relying party will adhere to legal and contractual obligations when using this information.
For the User: They need assurance that both the identity service provider and the relying party will handle sensitive information responsibly, respect their privacy, and honor their preferences.

All parties also want confirmation that each party's stated practices align with their actual implementations and that each party is reliable.



(a) Traditional triangle of parties involved in an exchange of identity information

Figure 4.13 Identity Information Exchange Approaches

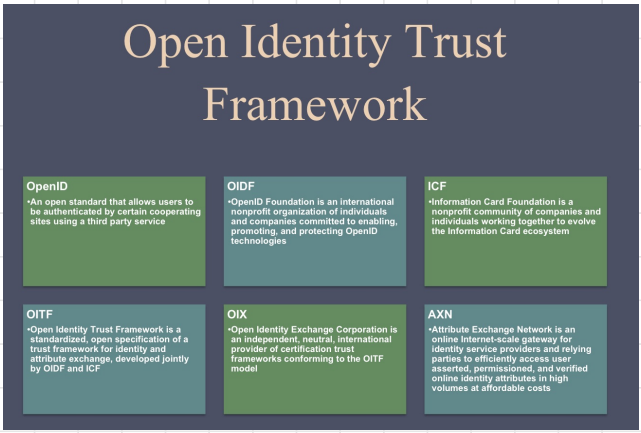


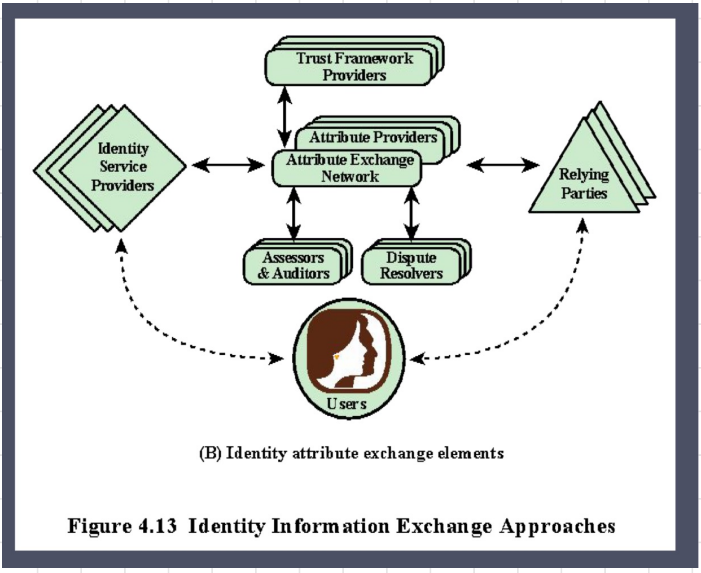
Figure 4.13b presents the components of the Open Identity Trust Framework (OITF) within an organization. Key roles and support elements include:

Core Roles:

Relying Parties (RPs): Also known as service providers, RPs deliver services to users and rely on verified identities or attributes.
Subjects: These are users, such as customers or employees, who utilize the RP's services.
Attribute Providers (APs): APs verify certain attributes about subjects and, through the Attribute Exchange Network (AXN), issue attribute credentials as per AXN rules.
APs may either be authorities on specific attributes or brokers of derived attributes.
Identity Providers (IDPs): IDPs authenticate user credentials and issue digital identities (or pseudonyms) that reference user attributes, typically via the AXN or a compatible Identity and Access Management (IDAM) system.

Support Elements in the AXN:
Assessors: Evaluate identity service providers and RPs, certifying their alignment with OITF standards.
Auditors: Ensure that practices adhere to agreed standards and requirements under the OITF.
Dispute Resolvers: Handle arbitration and dispute resolution under Open Identity Exchange (OIX) guidelines.
Trust Framework Providers: These organizations interpret policymaker requirements to create a tailored trust framework blueprint. For each industry sector or organization needing AXN compatibility, a suitable entity is selected to build this trust framework.
In Figure 4.13b, solid arrows indicate agreements with the trust framework provider for technical, operational, and legal compliance, while dashed arrows show other potentially influenced agreements.

Framework Operation:
Participating organisations define technical, operational, and legal requirements for identity exchanges.
They choose OITF providers to meet these needs. OITF providers translate requirements into a trust framework, possibly adding further conditions.
The OITF provider evaluates and contracts with identity service providers and RPs to adhere to these framework standards in identity exchanges.
Contracts include provisions for dispute resolution and audit processes to enforce and interpret the agreements.



(B) Identity attribute exchange elements

Figure 4.13 Identity Information Exchange Approaches

Table 4.5
Functions and Roles for Banking Example

(a) Functions and Official Positions

Role	Function	Official Position
A	financial analyst	Clerk
B	financial analyst	Group Manager
C	financial analyst	Head of Division
D	financial analyst	Junior
E	financial analyst	Senior
F	financial analyst	Specialist
G	financial analyst	Assistant
•••	•••	•••
X	share technician	Clerk
Y	support e-commerce	Junior
Z	office banking	Head of Division

Dresdner Bank implemented a Role-Based Access Control (RBAC) system to improve access management across various applications, which include both older mainframe-based systems and newer server-based applications. Previously, the bank used a Discretionary Access Control (DAC) system on each mainframe and server, where administrators had to manage separate access control files for each application and each user. This setup was complex, slow, and prone to errors.

The new RBAC system is applied across the entire organization, with access rights managed by three distinct administrative units to enhance security. In this RBAC model, roles are defined by a mix of an employee's official position and job function, as illustrated in Table 4.5a. This differs slightly from the NIST standard, where roles are defined strictly by job function, though this is largely a matter of terminology. This role structure naturally supports an inheritance hierarchy, where higher official positions inherit access rights from roles associated with lower ones.

Within Dresdner Bank, job roles are structured in a hierarchy that reflects levels of responsibility, such as Head of Division, Group Manager, and Clerk, in descending order. When combining a person's official position with their job function, access rights form a similar hierarchy. For instance, a "financial analyst/Group Manager" (role B) has broader access rights than a "financial analyst/Clerk" (role A), as shown in Table 4.4b. Role B not only matches or exceeds the access rights of role A in three applications but also has rights to an additional application.

However, roles in different functional areas, such as "Office Banking/Group Manager" and "financial analyst/Clerk," do not have a hierarchical relationship since they involve separate responsibilities. Thus, the role hierarchy only applies when both the position and function align. This hierarchical setup allows for efficient management of access rights, as shown in Table 4.5c, by reducing redundancy in defining permissions across similar roles.

Table 4.5
Functions and Roles for Banking Example

(b) Permission Assignments

Role	Application	Access Right
A	money market instruments	1, 2, 3, 4
	derivatives trading	1, 2, 3, 7, 10, 12
	interest instruments	1, 4, 8, 12, 14, 16
B	money market instruments	1, 2, 3, 4, 7
	derivatives trading	1, 2, 3, 7, 10, 12, 14
	interest instruments	1, 4, 8, 12, 14, 16
	private consumer instruments	1, 2, 4, 7
•••	•••	•••

(c) PA with Inheritance

Role	Application	Access Right
A	money market instruments	1, 2, 3, 4
	derivatives trading	1, 2, 3, 7, 10, 12
	interest instruments	1, 4, 8, 12, 14, 16
B	money market instruments	7
	derivatives trading	14
	private consumer instruments	1, 2, 4, 7
•••	•••	•••

In Dresdner Bank's original setup, access rights were assigned directly to users for each application individually, making management complex. The new RBAC system, however, organizes access rights by defining them at the application level and linking them with user roles. Access is granted based on a "security profile" tailored to each user's role rather than direct user-specific access assignments.

Each user is assigned a static role (or up to four roles, if needed) based on their job position and function. When a user opens an application, the application references the user's security profile to determine which access rights apply within that context. Although each role encompasses a broad set of access rights, only a subset relevant to the specific application is activated, as illustrated in Table 4.4b.

The Human Resources Department assigns each employee a unique User ID and roles based on their position and job function. This information is then sent to the Authorization Administration, which generates a security profile linking the User ID, role, and relevant access rights. These profiles are centrally managed and accessed by applications to enforce the appropriate permissions.

The bank's setup involves 65 official positions, combined with 368 job functions, resulting in the potential for 23,920 distinct roles. However, in practice, around 1,300 roles are actively used. The Authorization Administration processes an average of 42,000 security profile distributions daily, illustrating the scalability of the RBAC model in efficiently managing access rights across the bank.

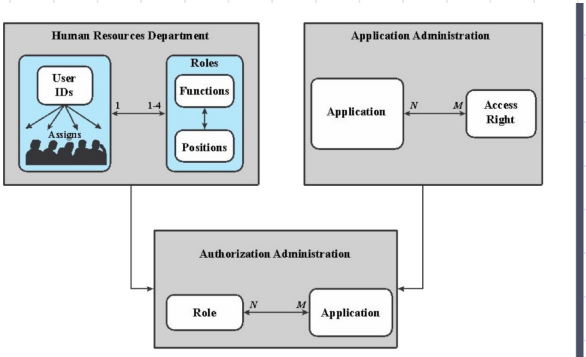
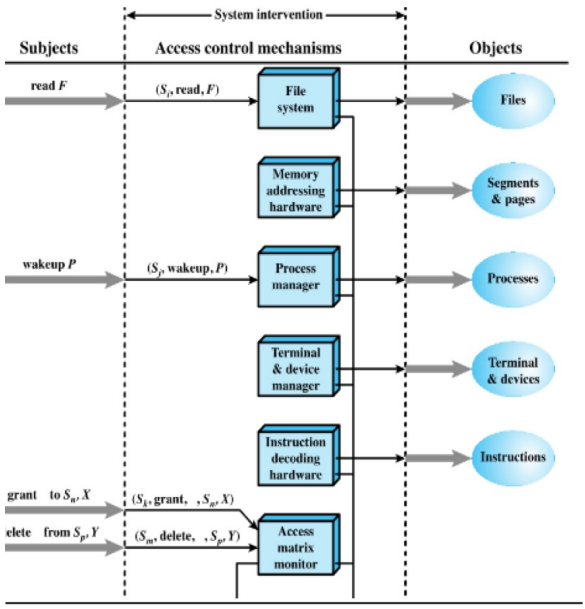


Figure 4.14 Example of Access Control Administration

Summary

- Access control principles
  - Access control context
  - Access control policies
- Subjects, objects, and access rights
- Discretionary access control
  - Access control model
  - Protection domains
- UNIX file access control
  - Traditional UNIX file access control
  - Access control lists in UNIX
- Role-based access control
  - RBAC reference models
- Attribute-based access control
  - Attributes
  - ABAC logical architecture
  - ABAC policies
- Identity, credential, and access management
  - Identity management
  - Credential management
  - Access management
  - Identity federation
- Trust frameworks
  - Traditional identity exchange approach
  - Open identity trust framework
- Bank RBAC system

e. How can an access control function be organized? Illustrate using various control mechanisms.



MALICIOUS SOFTWARE

Chp 6

malware

↳ a program that is secretly inserted into a system to compromise confidentiality, integrity, availability of data

Types of malware

Propagation mechanisms

↳ Infection
- ↳ viruses infects existing files spreading to other systems

↳ Exploitation
- ↳ worms and drive by downloads exploit software vulnerabilities that allow malware to replicate

↳ Social Engineering
- ↳ tricks users into bypassing security to install Trojans or respond to phishing

Payload Actions

↳ Data Corruption
- ↳ Damages system or data files

↳ Service Theft
- ↳ Turns the system into a bonet zombie for attacks

↳ Information Theft
- ↳ Steals data, often using key logging

↳ Stealthing
- ↳ Conceals malware's presence on the system

2 classifications of malware

1. Propogation
   - ↳ how it spreads to reach targets
2. Actions / payloads
   - ↳ what it does once it infects a target
3. Other classifications
   - ↳ Dependency
     - ↳ Host dependent
       - ↳ req a host program eg viruses
     - ↳ Independent
       - ↳ Self contained eg ·worms ·trojans ·bots
   - ↳ Replication
     - ↳ Non replicating malware eg ·trojans ·spam emails
     - ↳ Replicating malware eg ·viruses ·worms

Malware spreads through various mechanisms: infecting executables or scripts to spread viruses; exploiting software vulnerabilities through worms or drive-by downloads for replication; and using social engineering to trick users into installing Trojans or falling for phishing attacks.

Once on a target system, malware can corrupt files, steal resources to turn the system into a botnet zombie, capture information like logins and passwords with keyloggers or spyware, and hide its presence to evade detection.

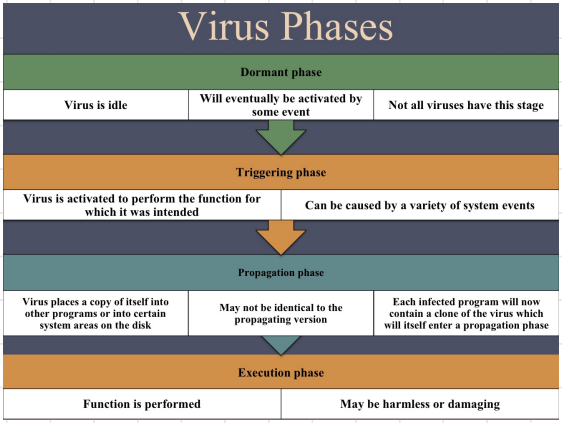
Initially, malware used a single method of spread and attack, but modern malware often combines multiple propagation and payload techniques to spread faster, stay hidden, and cause more damage. Some malware can even update itself to enhance its propagation and payload capabilities.

viruses

↳ Piece of software that infects programs
  ↳ modifies them to include a copy of the virus
  ↳ replicates and goes on to infect other content
  ↳ easily spread through network enviorments
↳ when infected can perform any action allowed by the program
  ↳ executes secretly when host program is run
↳ OS and hardware specific,
  exploits their details and vulnerabilities

virus components

↳ infection mechanism → aka infection vector
  ↳ the method a virus uses to spread
↳ Trigger → aka logic BOMB
  ↳ event/condition that determines when the payload is activated or delivered
↳ payload
  ↳ what the virus does (beside spreading)
  ↳ can be harmful or simply noticeable



worms

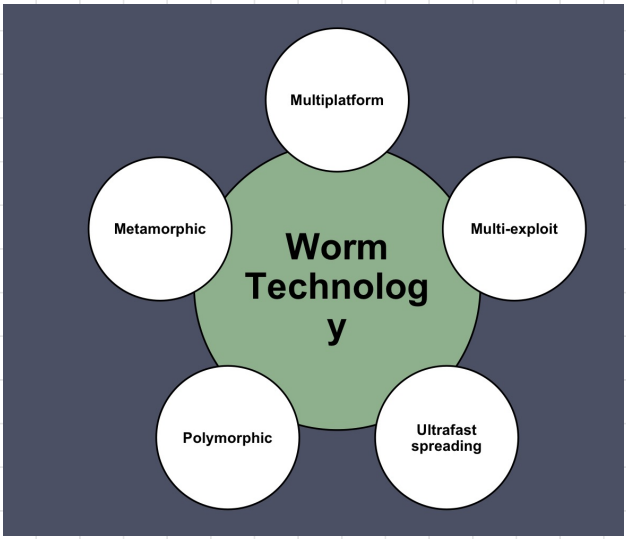
↳ Programs that actively seek and infect multiple machines
  and using those machines as automated launching pads
  for attacks on other machines
↳ exploit vulnerabilities in client or server software
↳ spreads through network, shared media (→ USB, CDs), emails, instant messenger attachments
↳ replicates and propogates upon activation, often carying a payload

Worm Replication

Electronic mail or instant messenger facility	• Worm e-mails a copy of itself to other systems • Sends itself as an attachment via an instant message service
File sharing	• Creates a copy of itself or infects a file as a virus on removable media
Remote execution capability	• Worm executes a copy of itself on another system
Remote file access or transfer capability	• Worm uses a remote file access or transfer service to copy itself from one system to the other
Remote login capability	• Worm logs onto a remote system as a user and then uses commands to copy itself from one system to the other

Recent Worm Attacks

Melissa	**1998**	**E-mail worm** **First to include virus, worm and Trojan in one package**
Code Red	**July 2001**	**Exploited Microsoft IIS bug** **Probes random IP addresses** **Consumes significant Internet capacity when active**
Code Red II	**August 2001**	**Also targeted Microsoft IIS** **Installs a backdoor for access**
Nimda	**September 2001**	**Had worm, virus and mobile code characteristics** **Spread using e-mail, Windows shares, Web servers, Web clients, backdoors**
SQL Slammer	**Early 2003**	**Exploited a buffer overflow vulnerability in SQL server** **compact and spread rapidly**
Sobig.F	**Late 2003**	**Exploited open proxy servers to turn infected machines into spam engines**
Mydoom	**2004**	**Mass-mailing e-mail worm** **Installed a backdoor in infected machines**
Warezov	**2006**	**Creates executables in system directories** **Sends itself as an e-mail attachment** **Can disable security related products**
Conficker (Downadup)	**November 2008**	**Exploits a Windows buffer overflow vulnerability** **Most widespread infection since SQL Slammer**
Stuxnet	**2010**	**Restricted rate of spread to reduce chance of detection** **Targeted industrial control systems**



The advanced features of modern worms include:

Multiplatform: New worms can attack multiple platforms, including UNIX variants, not just Windows, and may exploit macros or scripting in popular document types.
Multi-exploit: They use various methods to penetrate systems, targeting web servers, browsers, email, file sharing, and network applications, as well as shared media.
Ultrafast spreading: These worms spread quickly to maximize infection by locating vulnerable machines rapidly.
Polymorphic: Worms use polymorphic techniques, generating new code on the fly to evade detection and bypass filters.
Metamorphic: Beyond changing appearance, these worms have different behaviors that activate at various propagation stages.
Transport vehicles: Worms can carry and distribute malicious payloads, such as DDoS bots, rootkits, spam generators, and spyware.
Zero-day exploit: They often exploit unknown vulnerabilities (zero-days) to surprise and spread widely before detection. In 2015, 54 zero-day vulnerabilities were exploited, targeting common software, libraries, development tools, and even industrial systems, showing the broad range of targets.

Mobile Code

- Function
  - It is sent from a remote system to a local one where it is executed
  - often serving as a carrier for → Viruses, Trojans, worms
- Exploitation
  - uses vulnerabilities on the local system for attacks
- Common Types
  - Java applets
  - Active X
  - JavaScript
  - VBscript
- Most common Malicious Use
  - cross-site scripting
  - interactive / dynamic web sites

Mobile Phone Worms

- targets smart phones
- communicated through bluetooth or MMS
- It can
  - completely disable Phone
  - delete data on the Phone
  - force device to send costly messages

- CommWarrior replicates by means of Bluetooth to other phones, sends itself as an MMS file to contacts and as an auto reply to incoming text messages

Drive-By-Downloads

Exploits browser and plugin vulnerabilities so when the user views a webpage controlled by the attacker, it contains code that exploits the bug to download and install malware on the system without the user's knowledge or consent

In most cases the malware does not actively propagate as a worm does

Spreads when users visit the malicious Web page

Watering-Hole Attacks

- A variant of drive-by-download used in highly targeted attacks
- The attacker researches their intended victims to identify websites they are likely to visit, then scans these sites to identify those with vulnerabilities that allow their compromise
- They then wait for one of their intended victims to visit one of the compromised sites
- Attack code may even be written so that it will only infect systems belonging to the target organization and take no action for other visitors to the site
- This greatly increases the likelihood of the site compromise remaining undetected

Malvertising

Places malware on websites without actually compromising them

The attacker pays for advertisements that are highly likely to be placed on their intended target websites and incorporate malware in them

Using these malicious ads, attackers can infect visitors to sites displaying them

The malware code may be dynamically generated to either reduce the chance of detection or to only infect specific systems

Has grown rapidly in recent years because they are easy to place on desired websites with few questions asked and are hard to track

Attackers can place these ads for as little as a few hours, when they expect their intended victims could be browsing the targeted websites, greatly reducing their visibility

Clickjacking

- Also known as a user-interface (UI) redress attack
- Using a similar technique, keystrokes can also be hijacked
  - A user can be led to believe they are typing in the password to their email or bank account, but are instead typing into an invisible frame controlled by the attacker
- Vulnerability used by an attacker to collect an infected user's clicks
  - The attacker can force the user to do a variety of things from adjusting the user's computer settings to unwittingly sending the user to Web sites that might have malicious code
  - By taking advantage of Adobe Flash or JavaScript an attacker could even place a button under or over a legitimate button making it difficult for users to detect
  - A typical attack uses multiple transparent or opaque layers to trick a user into clicking on a button or link on another page when they were intending to click on the top level page
  - The attacker is hijacking clicks meant for one page and routing them to another page

Social Engineering

- "Tricking" users to assist in the compromise of their own systems

SPAM

Unsolicited bulk e-mail

Significant carrier of malware

Used for phishing attacks

Trojan horse

Program or utility containing harmful hidden code

Used to accomplish functions that the attacker could not accomplish directly

Trojans

First appeared in 2004 (Skuller)

Target is the smartphone

Payload

System Corruption

Chernobyl virus
- First seen in 1998
- Example of a destructive parasitic memory-resident Windows 95 and 98 virus
- Infects executable files when they are opened and when a trigger date is reached, the virus deletes data on the infected system by overwriting the first megabyte of the hard drive with zeroes, resulting in massive corruption of the hard drive to become empty entire file system

Klez
- Mass mailing worm infecting Windows 95 to XP systems
- First seen in October 2001
- Spreads by e-mailing copies of itself to addresses found in the address book and in files on the system
- It can stop and delete some anti-virus programs running on the system
- On trigger date causes files on

Ransomware
- Encrypts the user's data and demands payment in order to access the key needed to recover the information
- PC Cyborg Trojan (1989)
- Mid-2006 a number of worms and Trojans appeared that used public-key cryptography with incresasingly larger key sizes to encrypt data
- The user needed to pay a

WannaCry

Ransomware

- WannaCry
  - Infected a large number of systems in many countries in May 2017
  - When installed on infected systems, it encrypted a large number of files and then demanded a ransom payment in Bitcoins to recover them
  - Recovery of this information was generally only possible if the organization had good backups and an appropriate incident response and disaster recovery plan
  - Targets widened beyond personal computer systems to include mobile devices and Linux servers
  - Tactics such as threatening to publish sensitive personal information, or to permanently destroy the encryption key after a short period of time, are sometimes used to increase the pressure on the victim to pay up

Remote Control Facility

- Distinguishes a bot from a worm
  - Worm propagates itself and activates itself
  - Bot is initially controlled from some central facility
- Typical means of implementing the remote control facility is on an IRC server
  - Bots join a specific channel on this server and treat incoming messages as commands
  - More recent botnets use covert communication channels via protocols such as HTTP
  - Distributed control mechanisms use peer-to-peer protocols to avoid a single point of failure

Payload
System Corruption

- Real-world damage
  - Causes damage to physical equipment
    - Chernobyl virus rewrites BIOS code
  - Stuxnet worm
    - Targets specific industrial control system software
  - There are concerns about using sophisticated targeted malware for industrial sabotage
- Logic bomb
  - Code embedded in the malware that is set to "explode" when certain conditions are met

Payload – Attack Agents
Bots

- Takes over another Internet attached computer and uses that computer to launch or manage attacks
- *Botnet* - collection of bots capable of acting in a coordinated manner
- Uses:
  - Distributed denial-of-service (DDoS) attacks
  - Spamming
  - Sniffing traffic
  - Keylogging
  - Spreading new malware
  - Installing advertisement add-ons and browser helper objects (BHOs)
  - Attacking IRC chat networks
  - Manipulating online polls/games

Malware can take over a computer, turning it into a "bot" or "zombie" that the attacker can use for various activities, forming a coordinated network called a botnet. This botnet payload compromises the infected system's integrity and availability.

Uses of Botnets:

DDoS attacks: Overloads a target system, making it unavailable to users.
Spamming: Sends massive amounts of unsolicited emails.
Traffic sniffing: Monitors unencrypted data to capture sensitive information like usernames and passwords.
Keylogging: Captures keystrokes to steal sensitive information from encrypted channels.
Spreading malware: Uses infected computers to download and spread new malware.
Advertising fraud: Hijacks browser start pages or automates ad clicks to generate ad revenue fraudulently.
IRC network attacks: Overloads IRC networks by creating numerous bot connections, similar to DDoS.
Manipulating online polls/games: Uses bots with unique IPs to influence poll results or game outcomes.

Payload – Information Theft
Keyloggers and Spyware

Payload – Stealthing
Backdoor

- Also known as a *trapdoor*
- Secret entry point into a program allowing the attacker to gain access and bypass the security access procedures
- *Maintenance hook* is a backdoor used by Programmers to debug and test programs
- Difficult to implement operating system controls for backdoors in applications

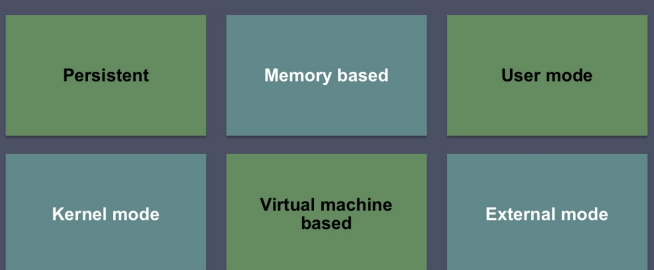
Payload – Information Theft
Phishing

- Exploits social engineering to leverage the user's trust by masquerading as communication from a trusted source
  - Include a URL in a spam e-mail that links to a fake Web site that mimics the login page of a banking, gaming, or similar site
  - Suggests that urgent action is required by the user to authenticate their account
  - Attacker exploits the account using the captured credentials
- Spear-phishing
  - Recipients are carefully researched by the attacker
  - E-mail is crafted to specifically suit its recipient, often quoting a range of information to convince them of its authenticity

Payload - Stealthing Rootkit

- Set of hidden programs installed on a system to maintain covert access to that system
- Hides by subverting (undermining) the mechanisms that monitor and report on the processes, files, and registries on a computer
- Gives administrator (or root) privileges to attacker
  - Can add or change programs and files, monitor processes, send and receive network traffic, and get backdoor access on demand

Rootkit Classification Characteristics



Rootkits can be classified based on how they operate and where they hide within a system:

Persistent: Loads each time the system boots by storing code in persistent storage (like the registry). Easier to detect because stored code can be scanned.
Memory-based: Only exists in memory and doesn't survive a reboot, making it harder to detect.
User mode: Intercepts API calls and alters returned results, such as hiding files during directory listings.
Kernel mode: Operates at the kernel level to intercept native API calls and hide malware processes from the system's active process list.
Virtual machine-based: Runs a lightweight virtual machine below the operating system, allowing it to intercept and modify system events without detection.
External mode: Hides outside normal system operations, such as in the BIOS, with direct hardware access.
Rootkit authors continue to develop stealthier techniques, moving to "layer-below" attacks that hide from detection tools operating at higher levels, resulting in an ongoing battle with security developers. Early rootkits worked in user mode, but newer ones use more advanced methods for greater invisibility.

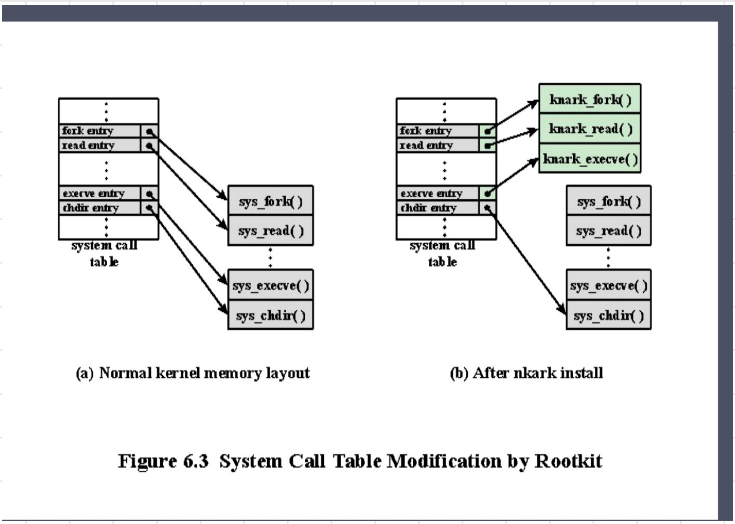


Figure 6.3 System Call Table Modification by Rootkit

The next generation of rootkits became harder to detect by embedding themselves directly in the kernel, modifying low-level code that interacts with the operating system. This makes traditional antivirus programs vulnerable to the same low-level changes used to conceal the rootkit. However, detection methods were developed to counter these tactics.

Kernel-level rootkits target system calls, which are how user-level programs interact with the kernel. In Linux, each system call has a unique number that points to an entry in the system call table, which in turn directs to the corresponding routine.

Three techniques used by rootkits to alter system calls are:

1. **Modify the system call table**: The rootkit changes specific addresses in the system call table to redirect calls from legitimate routines to its own malicious code.
2. **Modify system call table targets**: Instead of altering the table itself, the rootkit overwrites certain system call routines directly with malicious code, leaving the table unchanged.
3. **Redirect the system call table**: The rootkit replaces the reference to the original system call table with a new table at a different location in memory.

Malware Counter Measure Approaches

↳ 4 main elements of Prevention → ideal solution to the threat of malware

1. Policy
2. Awarness
3. Vulnerability mitigation
4. Threat mitigation

Technical Mechanisms for Threat Mitigation

↳ used when Prevention fails

- ↳ Detection
- ↳ Identification
- ↳ Removal

Generations of Anti virus Softwares

↳ First Generations

- ↳ req a malware signature to identify the malware
- ↳ Limited to the detection of known malware

↳ Second Generation

- ↳ uses heuristic rules to search for probable malware instances
- ↳ integrity checking

↳ Third Generation

- ↳ memory resident programs that identify malware by its actions

↳ Fourth Generation

- ↳ full featured protection
- ↳ packages consisting a variety of anti-virus techniques
- ↳ scanning and activity trap components
- ↳ Acess control capacity

Sandbox Analysis

↳ runs suspicious code in a virtual/emulated enviorment
↳ allows safe observation of the codes behavior w/o risking a real system
↳ helps detect complex malware types ⇉ encrypted, Polymorphic, metamorphic malware
↳ challange
  ↳ deciding how long to observe the codes behavior

Host Based behavior Blocking Software

↳ works within a computers OS to monitor and block malicious actions in real time
↳ Prevents harmful actions before they impact the system
↳ Limitations
  ↳ malicious code may still cause some harm as it must begin running for it behavior to be detected and blocked

Perimeter scanning Approaches

- Anti-virus software typically included in e-mail and Web proxy services running on an organization's firewall and IDS
- May also be included in the traffic analysis component of an IDS
- May include intrusion prevention measures, blocking the flow of any suspicious traffic
- Approach is limited to scanning malware

2 types of monitoring softwares

↳ they help protect the network perimeter

1. Ingress monitors

↳ Placed at the network internet border
↳ detects incomming threats — eg traffic to unused IP addresses

2. Egress monitors

↳ Placed at LAN exits and the network border
↳ stops suspicious outgoing traffic or scanning attempts

Summary

- Types of malicious software (malware)
  - Broad classification of malware
  - Attack kits
  - Attack sources
- Advanced persistent threat
- Propagation-vulnerability exploit-worms
  - Target discovery
  - Worm propagation model
  - The Morris Worm
  - Brief history of worm attacks
  - State of worm technology
  - Mobile code
  - Mobile phone worms
  - Client-side vulnerabilities
  - Drive-by-downloads
  - Clickjacking
- Payload-stealthing-backdoors, rootkits
  - Backdoor
- Propagation-social engineering-span E-mail, Trojans
  - Spam E-mail
  - Trojan horses
  - Mobile phone Trojans
- Payload-system corruption
  - Data destruction
  - Real-world damage
  - Logic bomb
- Payload-attack agent-zombie, bots
  - Uses of bots
  - Remote control facility
- Payload-information theft-keyloggers, phishing, spyware
  - Credential theft, keyloggers, and spyware
  - Phishing and identity theft
  - Reconnaissance, espionage, and data exfiltration
- Countermeasures
  - Malware countermeasure approaches
  - Host-based scanners
  - Signature-based anti-virus
  - Perimeter scanning approaches

Task # 1: You found that the code has three different ways for propagation. Briefly describe each, explaining functionality and specific propagation steps.
Need explanations of following three propagation methods in 2-3 lines each.

6.3 Propagation—Infected Content—Viruses
6.4 Propagation—Vulnerability Exploit—Worms
6.5 Propagation—Social Engineering—Spam E-mail, Trojans

Task # 2: You found that the code has four different ways to damage IT assets. Briefly describe each, explaining how systems are compromised to delete/damage/steal/ransom different digital assets.
Need explanations of following three propagation methods in 2-3 lines each.

6.6 Payload—System Corruption
6.7 Payload—Attack Agent—Zombie, Bots
6.8 Payload—Information Theft—Keyloggers, Phishing, Spyware
6.9 Payload—Stealthing—Backdoors, Rootkits