

RT 801 - Security in Computing

Module II **OS Security**

Instructor: Dr. SABU M THAMPI, Rajagiri School of Engineering and Technology, Kochi, India

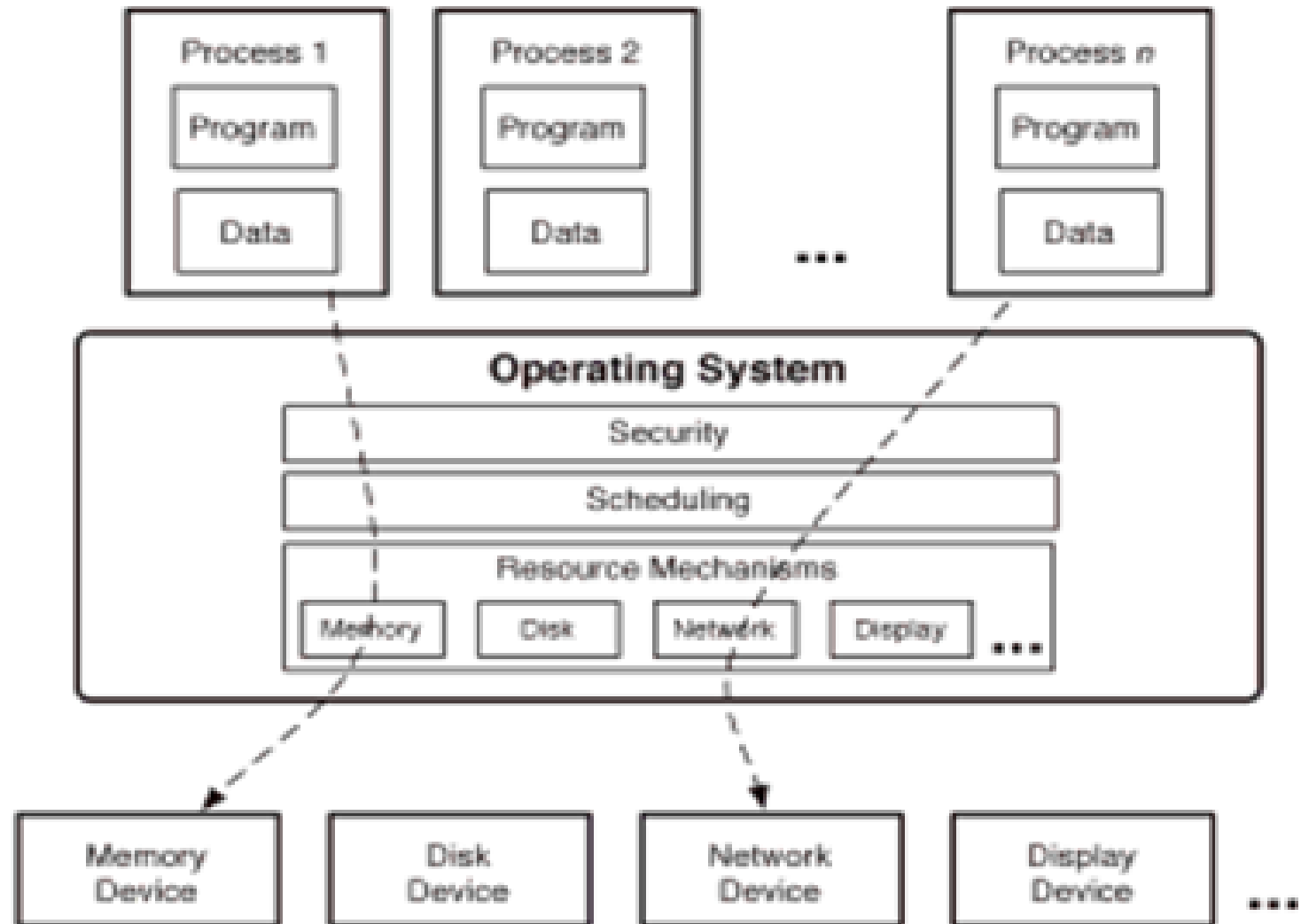
Module 2

- OS Security – Protection Mechanisms – Authentication & Access control – Discretionary and Mandatory access control – Authentication mechanisms – Official levels of computer security (DoD) - Security breaches – Concept of a hole - Types of a holes – Study of the security features for authentication, access control and remote execution in UNIX, WINDOWS 2000

Module V covers few portions of V module -- Refer Module V Notes

What is an Operating System?

- The operating system (OS) is the program which starts up when you turn on your computer and runs underneath all other programs - without it nothing would happen at all.
- In simple terms, an operating system is a *manager*. It manages all the available resources on a computer, from the CPU, to memory, to hard disk accesses.
- Tasks the operating system must perform:
 - **Control Hardware** - The operating system controls all the parts of the computer and attempts to get everything working together.
 - **Run Applications** - Another job the OS does is run application software. This would include word processors, web browsers, games, etc...
 - **Manage Data and Files** - The OS makes it easy for you to organize your computer. Through the OS you are able to do a number of things to data, including copy, move, delete, and rename it. This makes it much easier to find and organize what you have.



- To build a successful OS, three major tasks are identified:
 - **OS must provide efficient resource mechanisms**, such as file systems, memory management systems, network protocol stacks etc.. That define how processes use the hardware resources.
 - **Scheduling access to computer resources.** Switching among processes fairly such that the user experiences good performance from each process in concert with access to the computer's devices.
 - **Controlling access to resources** such that one process cannot inadvertently or maliciously impact the execution of another. This third problem is the problem of *ensuring the security of all processes run on the system.*

- The challenge in developing OS security is to design security mechanisms that protect process execution and their generated data in an environment with complex interactions.
- The ideal goal of OS security is the development of a secure OS.
 - A secure OS provides security mechanisms that ensure that the system's security goals are enforced despite the threats faced by the system.

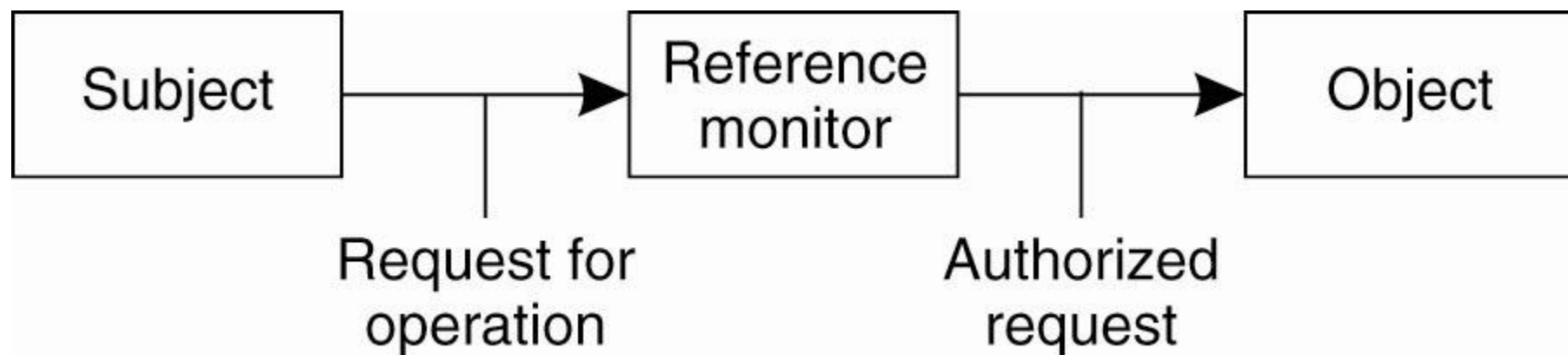
Threats

Goal	Threat
Data confidentiality	Exposure of data
Data integrity	Tampering with data
System availability	Denial of service
Exclusion of outsiders	System takeover by viruses

Security goals and threats.

Protection Mechanisms

- In some systems, protection is enforced by a program called a **reference monitor**.
- Every time an access to a potentially protected resource is attempted, the system first asks the reference monitor to check its legality.
- The reference monitor then looks at its policy tables and makes a decision.



Objects

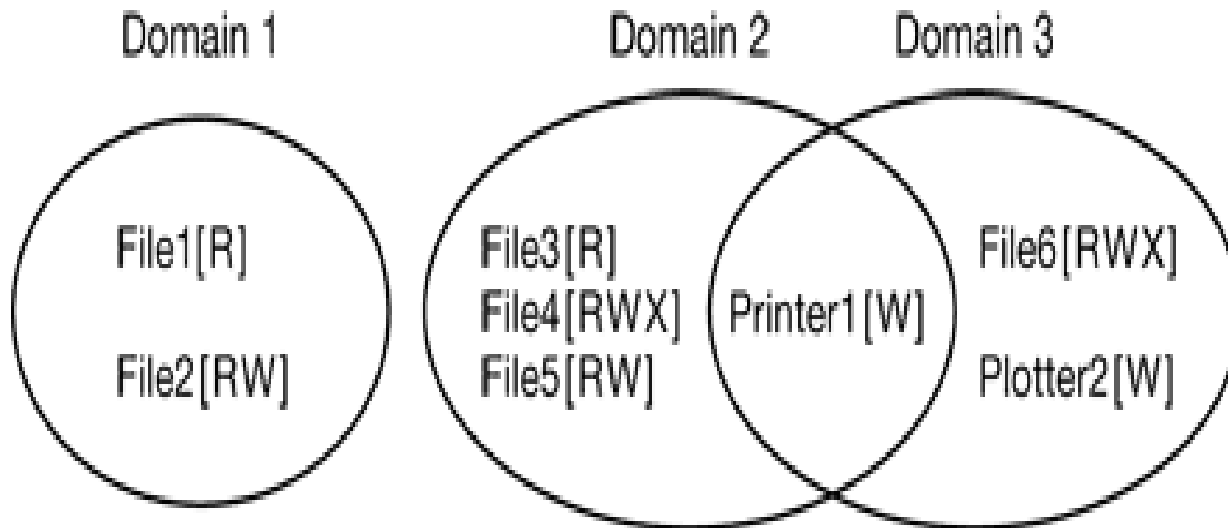
- A computer system contains many “objects” that need to be protected.
 - Objects can be hardware (e.g., CPUs, memory segments, disk drives, or printers), or they can be software (e.g., processes, files, databases...).
- Each object has a unique name by which it is referenced, and a finite set of operations that processes are allowed to carry out on it.
 - The read and write operations for a file.
- A way is needed to prohibit processes from accessing objects that they are not authorized to access.
 - Furthermore, this mechanism must restrict processes to a subset of the legal operations when that is needed.
 - For example, process *A* may be entitled to read, but not write, file *F*.

Protection Domains.....

- A **domain** is a set of (object, rights) pairs.
 - Each pair specifies an object and some subset of the operations that can be performed on it.
- A **right** in this context means permission to perform one of the operations.
- Often a domain corresponds to a single user, telling what the user can do and not do..

Three protection domains

- Three domains, showing the objects in each domain and the rights [Read, Write, eXecute] available on each object.
- Printer1 is in two domains at the same time.
- It is possible for the same object to be in multiple domains, with different rights in each one.



- At every instant of time, each process runs in some protection domain.
 - That is there is some collection of objects a process can access
 - For each object, the process has some set of rights.
- Processes can also switch from domain to domain during execution.

How the system keeps track of which object belongs to which domain

A protection matrix

		Object							
		File1	File2	File3	File4	File5	File6	Printer1	Plotter2
Domain	1	Read	Read Write						
	2			Read	Read Write Execute	Read Write		Write	
	3						Read Write Execute	Write	Write

Given this matrix and the current domain number, the system can tell if an access to a given object in a particular way from a specified domain is allowed.

A protection matrix with domains as objects.

- Domain switching itself can be easily included in the matrix model by realizing that a domain is itself an object, with the operation enter.
- Figure - with the three domains as objects themselves.

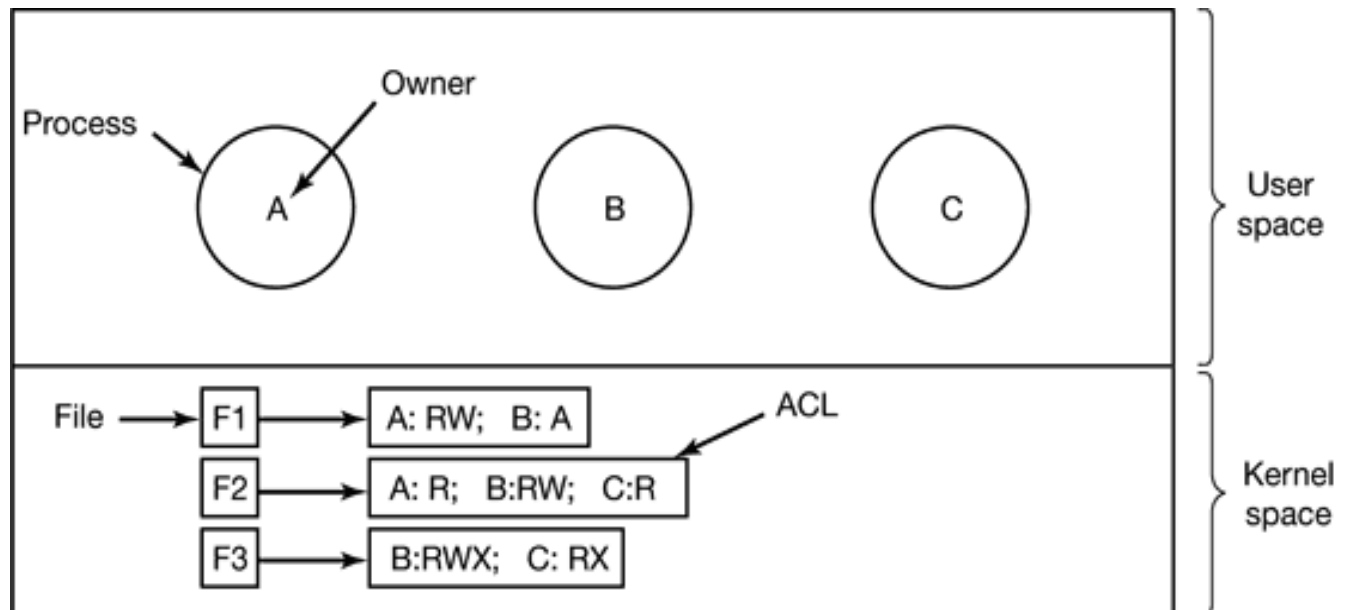
		Object										
		File1	File2	File3	File4	File5	File6	Printer1	Plotter2	Domain1	Domain2	Domain3
Domain	1	Read	Read Write								Enter	
	2			Read	Read Write Execute	Read Write		Write				
	3						Read Write Execute	Write	Write			

Access Control Lists

- Most domains have no access at all to most objects, so storing a very large, mostly empty, matrix is a waste of disk space.
- Two methods - storing the matrix by rows or by columns, and then storing only the nonempty elements.
 - Storing it by column
 - Storing it by row.

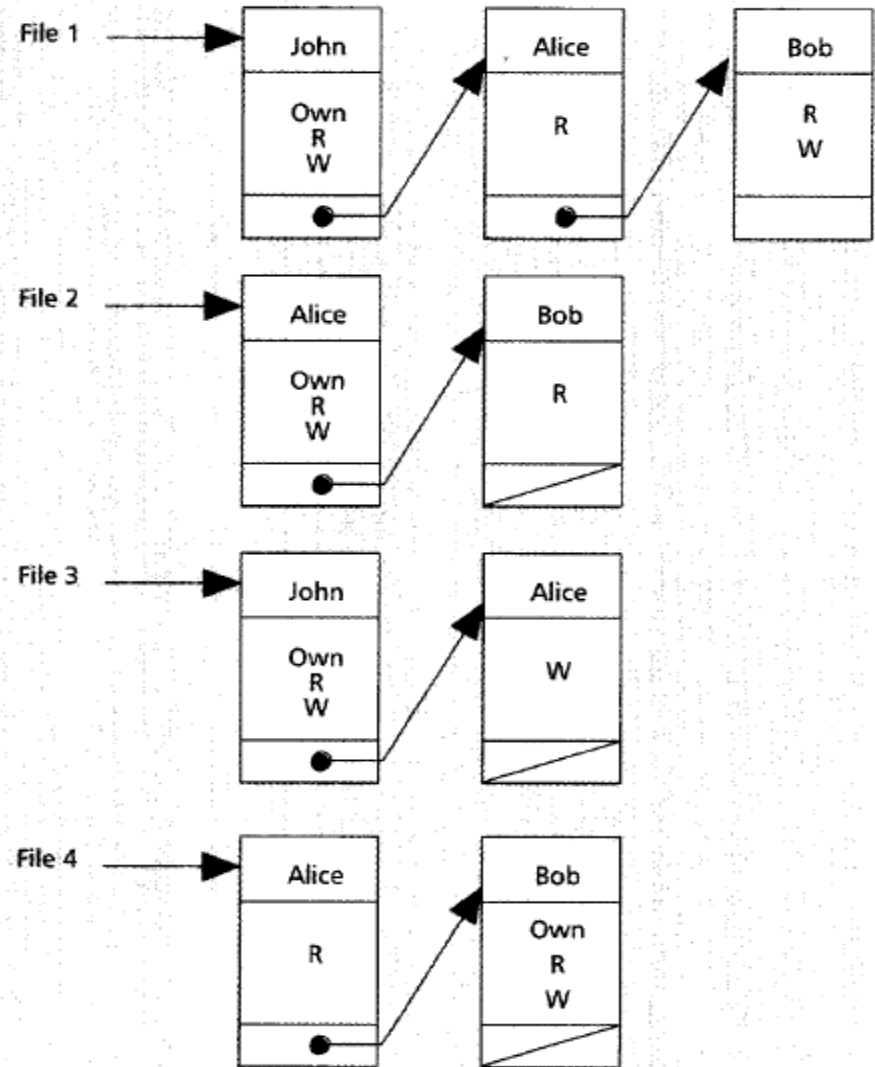
Access Control List or ACL

- Each file has an ACL associated with it.
 - File *F1* has two entries in its ACL (separated by a semicolon).
 - The first entry says that any process owned by user *A* may read and write the file.
 - The second entry says that any process owned by user *B* may read the file.
 - The second entry says that any process owned by user *B* may read the file.
 - All other accesses by these users and all accesses by other users are forbidden.

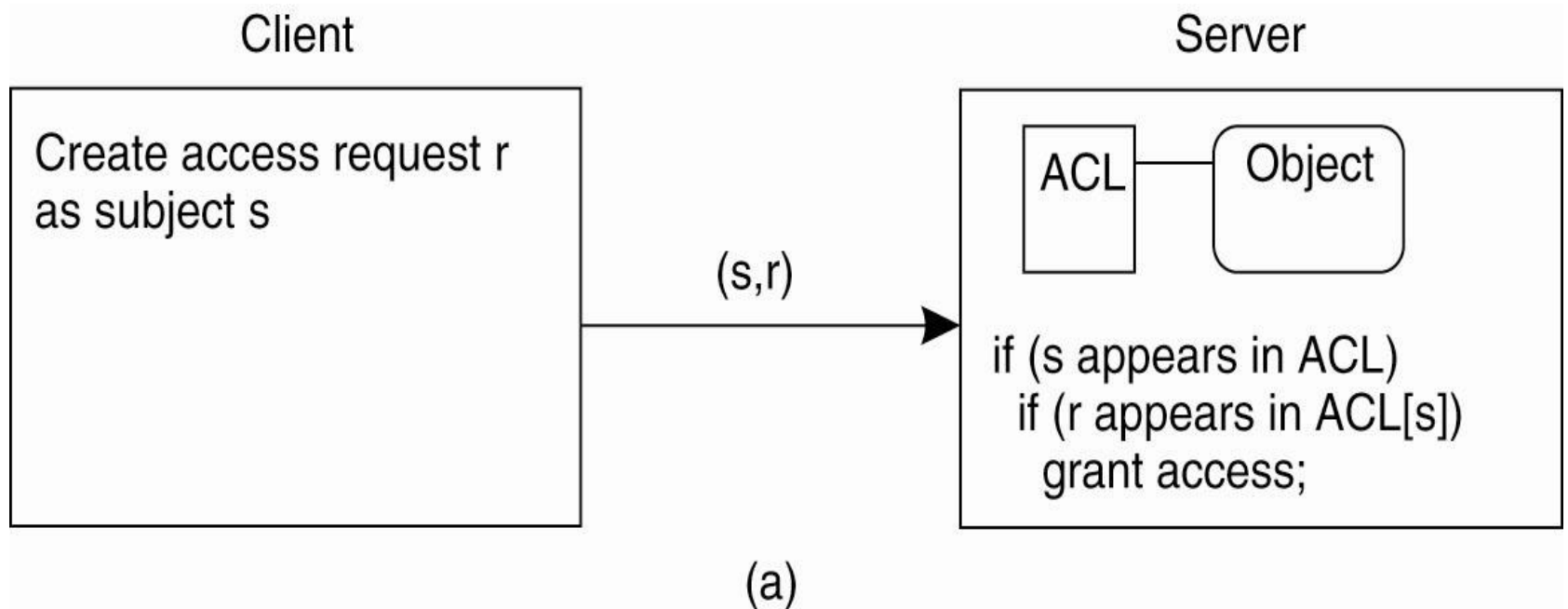


Access Control Lists

- A popular approach to implementing the access matrix
- Each object is associated with an ACL indicating for each subject in the system the accesses the subject is authorized to execute on the object.

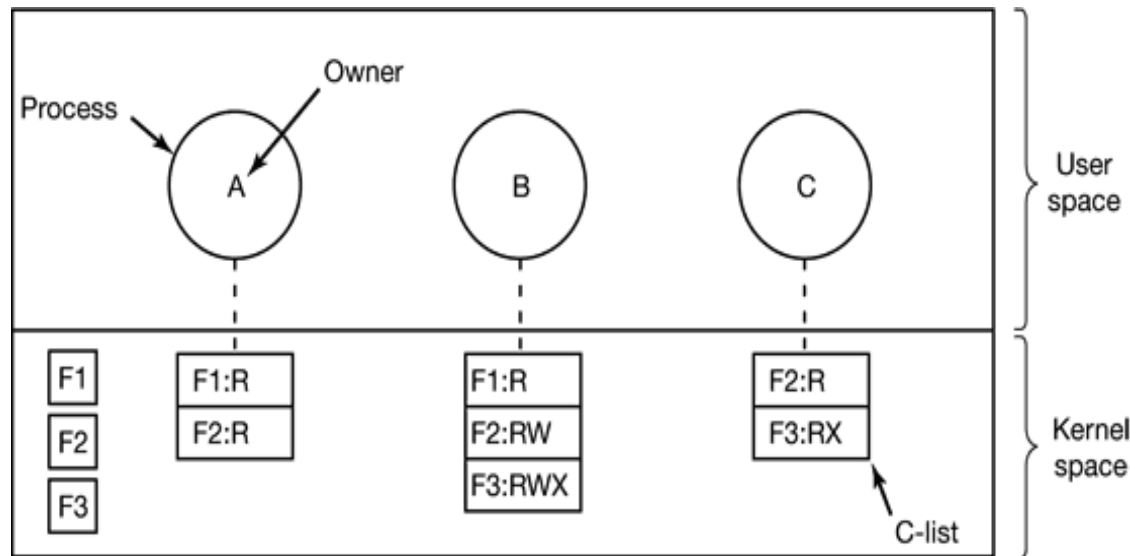


Using an ACL



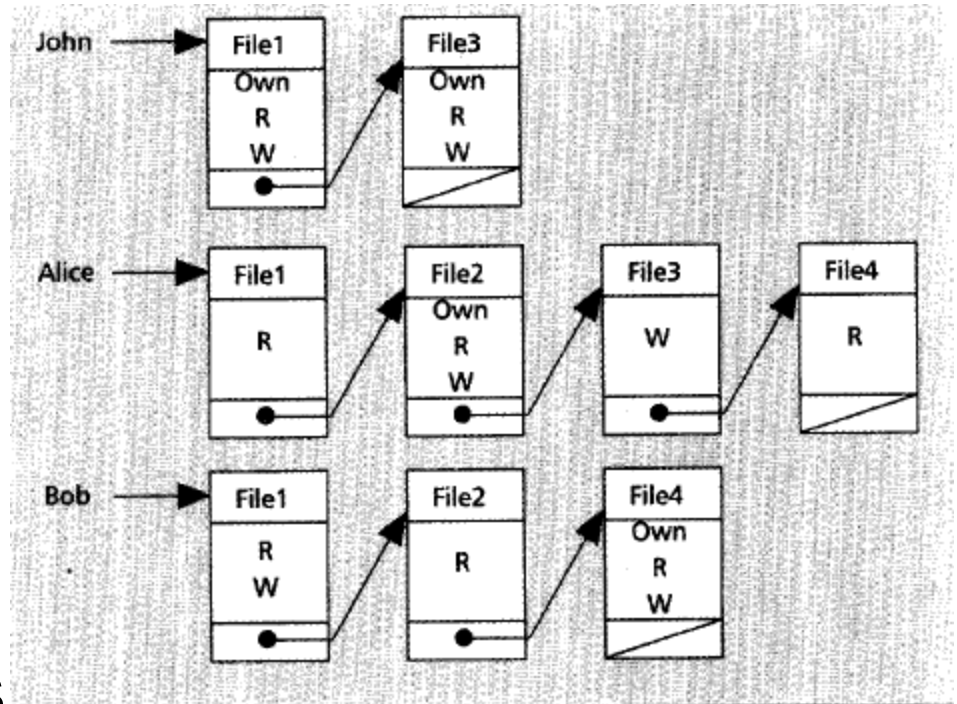
Capabilities

- The other way of slicing up the matrix is by rows.
- When this method is used, associated with each process is a list of objects that may be accessed, along with an indication of which operations are permitted on each, in other words, its domain.
- This list is called a **capability list** or **C-list** and the individual items on it are called **capabilities**
- A set of three processes and their capability lists is shown in Fig.

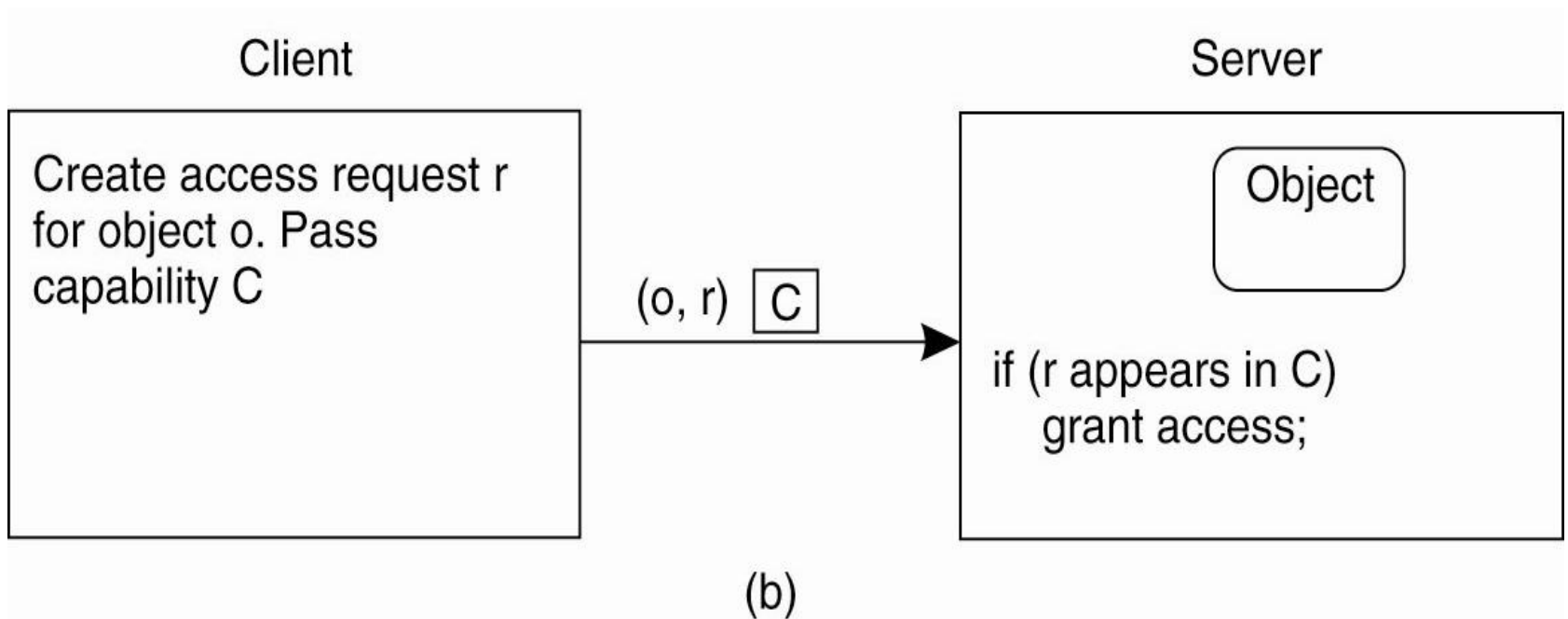


Capability Lists

- Capabilities are a dual approach to ACLs.
- Each subject is associated with a list that indicates, for each object in the system, which accesses the subject is authorized to execute on the object.



Using capabilities



Where should capabilities be stored?





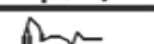

- Capabilities are critical to system security. Once a capability is issued to a user, the user should not be able to tamper with the capability.
- capabilities can be stored
 - In a protected place
 - In an unprotected place
 - Hybrid Approach

In a protected place

- Capabilities can be stored in a protected place.
 - Users cannot touch the capability; they use capabilities in an implicit manner:
- **In kernel:** The capability list is stored in the kernel.
 - Users cannot modify the contents of any capability, because they have no access to the kernel.
 - Whenever users need their capabilities, the system will go to the kernel to the capability-list.
- **Tagged architecture:** the capability can be saved in memories that are tagged as read-only and use-only.

Tagged architecture

- The capability can be saved in memories that are tagged as read-only and use-only
- Every word of machine memory has one or more extra bits (tag) to that word that tells whether the word contains a capability or not
- The bits are tested every time an instruction accesses that location
- Has been used in a few systems
- Expensive hardware design

Tag	Memory Word
R	0001
RW	0137
R	0099
X	
X	
X	
X	
X	
X	
R	4091
RW	0002

Code: R = Read-only RW = Read/Write
X = Execute-only

In an unprotected place (user space) :

- In some applications, users may have to carry their capabilities with themselves.
- When they request an access, they simply present their capability to the system.
- This is an explicit use of capabilities. Because permissions are encoded in the capability, if users can tamper with the contents of a capability, they can gain unauthorized privileges.
- The protection can be achieved using cryptographic checksum: the capability issuer can put a cryptographic checksum on the capability (e.g. digital signature).
 - Any tampering of the capability will be detected.

Hybrid Approach

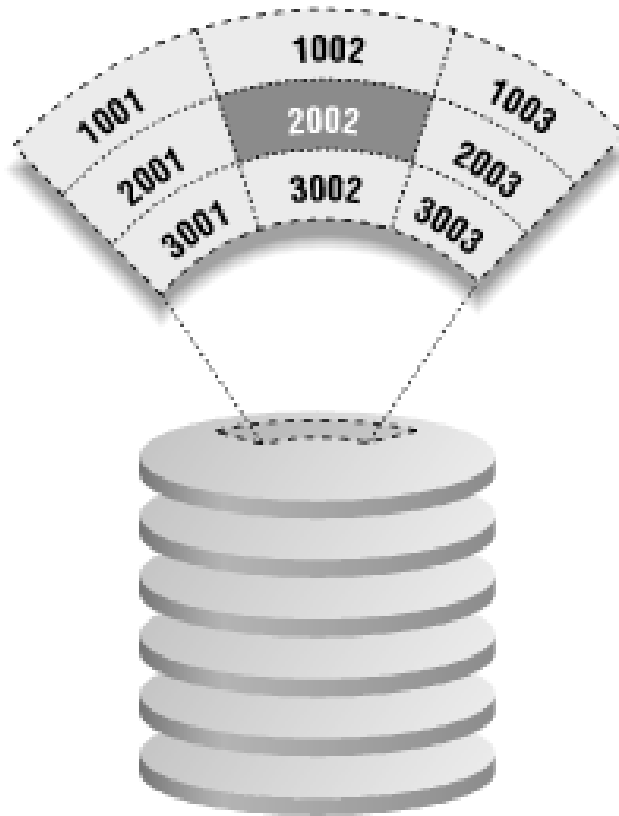
- Users can use capabilities in an explicit manner, but the capabilities are stored in a safe place.
- The real capabilities are stored in a table, which resides in a protected place (e.g. kernel).
- **Users are given the index to these capabilities.** They can present the index to the system to explicitly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

inode in Unix

- Kernel maintains file information in a structure called inode.
- All UNIX files have its description stored in this structure.
- The inode contains info about the file-size, its location, time of last access, time of last modification, permission and so on.
- Directories are also represented as files and have an associated inode.
- In addition to descriptions about the file, the inode contains pointers to the data blocks of the file.

- Each inode generally contains:
 - The location of the item's contents on the disk, if any
 - The item's type (e.g., file, directory, symbolic link)
 - The item's size, in bytes, if applicable
 - The time the file's inode was last modified (the ctime)
 - The time the file's contents were last modified (the mtime)
 - The time the file was last accessed (the atime) for read () , exec () , etc
 - A reference count: the number of names the file has
 - The file's owner (a UID)
 - The file's group (a GID)
 - The file's mode bits (also called file permissions or permission bits)

inode



inode 2002

Item Location	Item Type	Item Size (bytes)
Time Inode Modified (ctime)	Time Contents Modified (mtime)	Time File Accessed (atime)
File's Owner (UID)	File's Group (GID)	Per-missions (mode bits)
Reference Count	Location of Data on Disk	

C-list in user space (Unix)

- Manage the capabilities cryptographically so that users cannot tamper with them.
- Suited to distributed systems.
 - When a client process sends a message to a remote server, for example, a file server, to create an object for it, the server creates the object and generates a long random number - *a check field*.
 - A slot in the server's file table is reserved for the object and the check field is stored there along with the addresses of the disk blocks, etc.
 - Actually, the check field is stored on the server in the i-node.
 - It is not sent back to the user and never put on the network. The server then generates and returns a capability to the user of the form shown



Figure 9-10. A cryptographically-protected capability.

- The capability returned to the user contains the server's identifier, the object number (the index into the server's tables, essentially, the i-node number), and the rights, stored as a bitmap (digest).
- For a newly created object, all the rights bits are turned on.
- The last field consists of the concatenation of the object, rights, and check field run through a **cryptographically-secure one-way function, f**

- When the user wishes to access the object, it sends the capability to the server as part of the request.
- The server then extracts the object number to index into its tables to find the object.
- It then computes $f(\text{Object}, \text{Rights}, \text{Check})$ taking the first two parameters from the capability itself and the third one from its own tables.
- If the result agrees with the fourth field in the capability, the request is honored; otherwise, it is rejected.
- If a user tries to access someone else's object, he will not be able to fabricate the fourth field correctly since he does not know the check field, and the request will be rejected.

Basic Operations on Capabilities

- *Create capability*: a capability is created for a user (or assign to a user).
- *Delegate capability*: a subject delegates its capability to other subjects.
- *Revoke capability*: a subject revokes the capabilities it has delegated to other subjects.
- *Enable capability*: a subject enables a disabled capability.
- *Disable capability*: a subject temporarily disables a capability.
- *Delete capability*: a subject permanently deletes a capability.

Multilevel Security

Multilevel Security

- Most operating systems allow individual users to determine who may read and write their files and other objects. This policy is called *discretionary access control*.
- But for other environments such as the military, corporate patent departments, and hospitals much tighter security is required,.
 - In the such environments, the organization has stated rules about who can see what, and these may not be modified by individual soldiers, lawyers, or doctors, at least not without getting special permission from the boss.
 - These environments need *mandatory access controls* to ensure that the stated security policies are enforced by the system, in addition to the standard discretionary access controls.
 - Mandatory access controls regulate the flow of information, to make sure that it does not leak out in a way it is not supposed to.

The Bell-La Padula Model

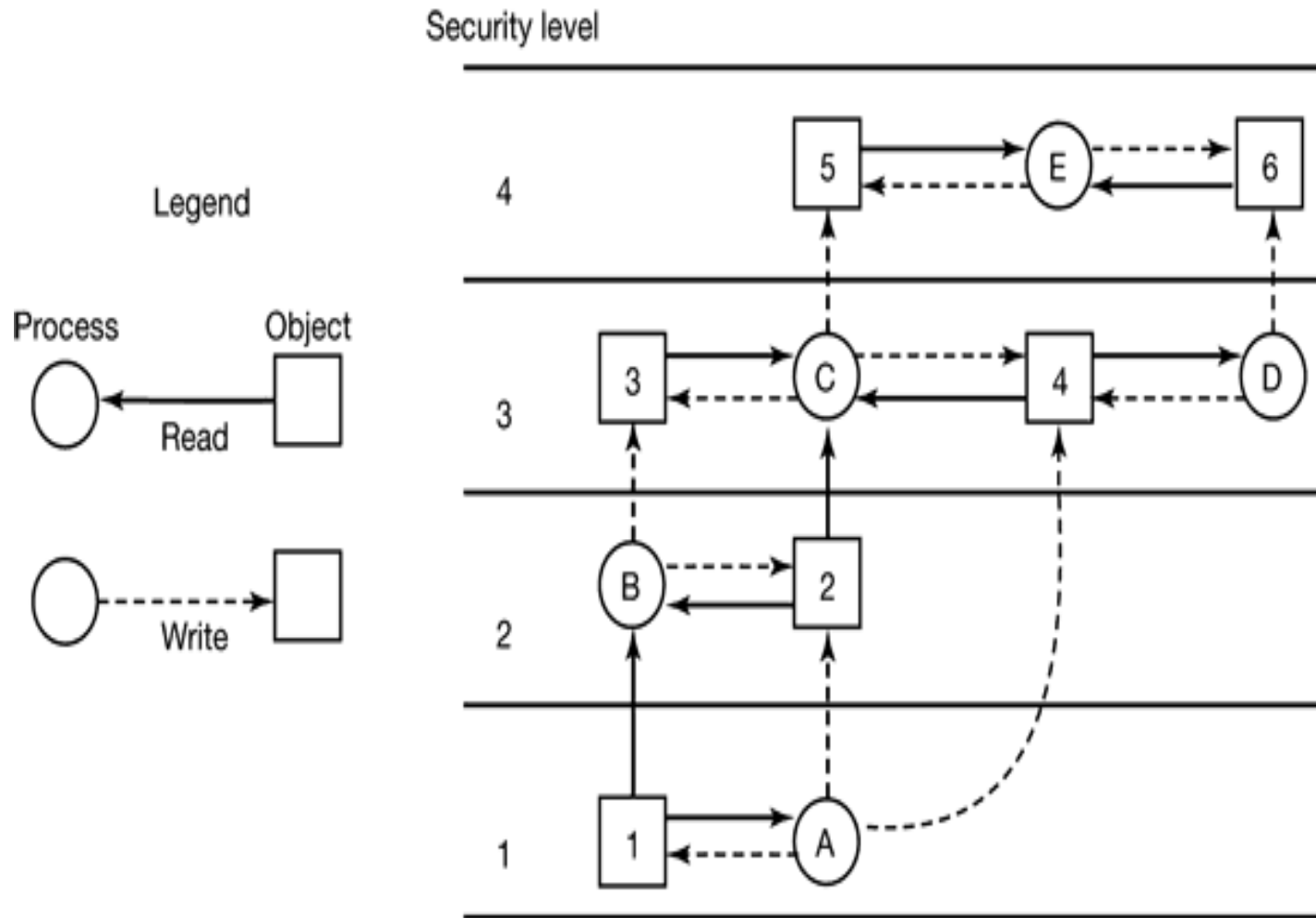
Please read PPT for Module 5

- The most widely used multilevel security model is the Bell-La Padula model.
- This model was designed for handling military security, but it is also applicable to other organizations.
- In the military world, documents (objects) can have a security level, such as unclassified, confidential, secret, and top secret.
- People are also assigned these levels, depending on which documents they are allowed to see.

- A process running on behalf of a user acquires the user's security level.
- Since there are multiple security levels, this scheme is called a multilevel security system.
- The Bell-La Padula model has rules about how information can flow:
 - **The simple security property:** A process running at security level k can read only objects at its level or lower.
 - For example, a general can read a lieutenant's documents but a lieutenant cannot read a general's documents.

- 2. The * property: A process running at security level k can write only objects at its level or higher.
 - For example, a lieutenant can append a message to a general's mailbox telling everything he knows, but a general cannot append a message to a lieutenant's mailbox telling everything he knows because the general may have seen top secret documents that may not be disclosed to a lieutenant.

The Bell-La Padula multilevel security model



- Processes can read down and write up, but not the reverse
- In the figure a (solid) arrow from an object to a process indicates that the process is reading the object
 - Information is flowing from the object to the process.
- A (dashed) arrow from a process to an object indicates that the process is writing into the object
 - Information is flowing from the process to the object.
 - For example, process B can read from object 1 but not from object 3.

- ***The Bell-La Padula model and the operating system.***
 - Assigning each user a security level, to be stored along with other user-specific data such as the UID and GID.
 - Upon login, the user's shell would acquire the user's security level and this would be inherited by all its children.
 - If a process running at security level k attempted to open a file or other object whose security level is greater than k , the operating system should reject the open attempt.
 - Similarly attempts to open any object of security level less than k for writing must fail.

The Biba Model

- The problem with the Bell-La Padula model is that it was devised to keep secrets, not guarantee the integrity of the data.
- To guarantee the integrity of the data, the following reverse properties are used:
 - ***The simple integrity principle:*** A process running at security level k can write only objects at its level or lower (no write up).
 - ***The integrity * property:*** A process running at security level k can read only objects at its level or higher (no read down).

Authentication

- Authentication Using Passwords
 - Unix passwords
- One-Time Passwords
 - A one-time password (OTP) is a password that is valid for only one login session or transaction.
- Challenge-Response Authentication
- Authentication Using a Physical Object
- Authentication Using Biometrics

Authentication

General principles of authenticating users:

- Something the user knows – passwords, PIN numbers
- Something the user has – physical keys, driving license, ID cards
- Something the user is – biometric (finger print)

Authentication Using Passwords

LOGIN: mitch
PASSWORD: FooBar!-7
SUCCESSFUL LOGIN

(a)

LOGIN: carol
INVALID LOGIN NAME
LOGIN:

(b)

LOGIN: carol
PASSWORD: Idunno
INVALID LOGIN
LOGIN:

(c)

(a) A successful login.

(b) Login rejected after name is entered.

(c) Login rejected after name and password are typed.

How Crackers Break In

```
LBL> telnet elxsi
ELXSI AT LBL
LOGIN: root
PASSWORD: root
INCORRECT PASSWORD, TRY AGAIN
LOGIN: guest
PASSWORD: guest
INCORRECT PASSWORD, TRY AGAIN
LOGIN: uucp
PASSWORD: uucp
WELCOME TO THE ELXSI COMPUTER AT LBL
```

How a cracker broke into a U.S. Department of Energy computer at LBL.

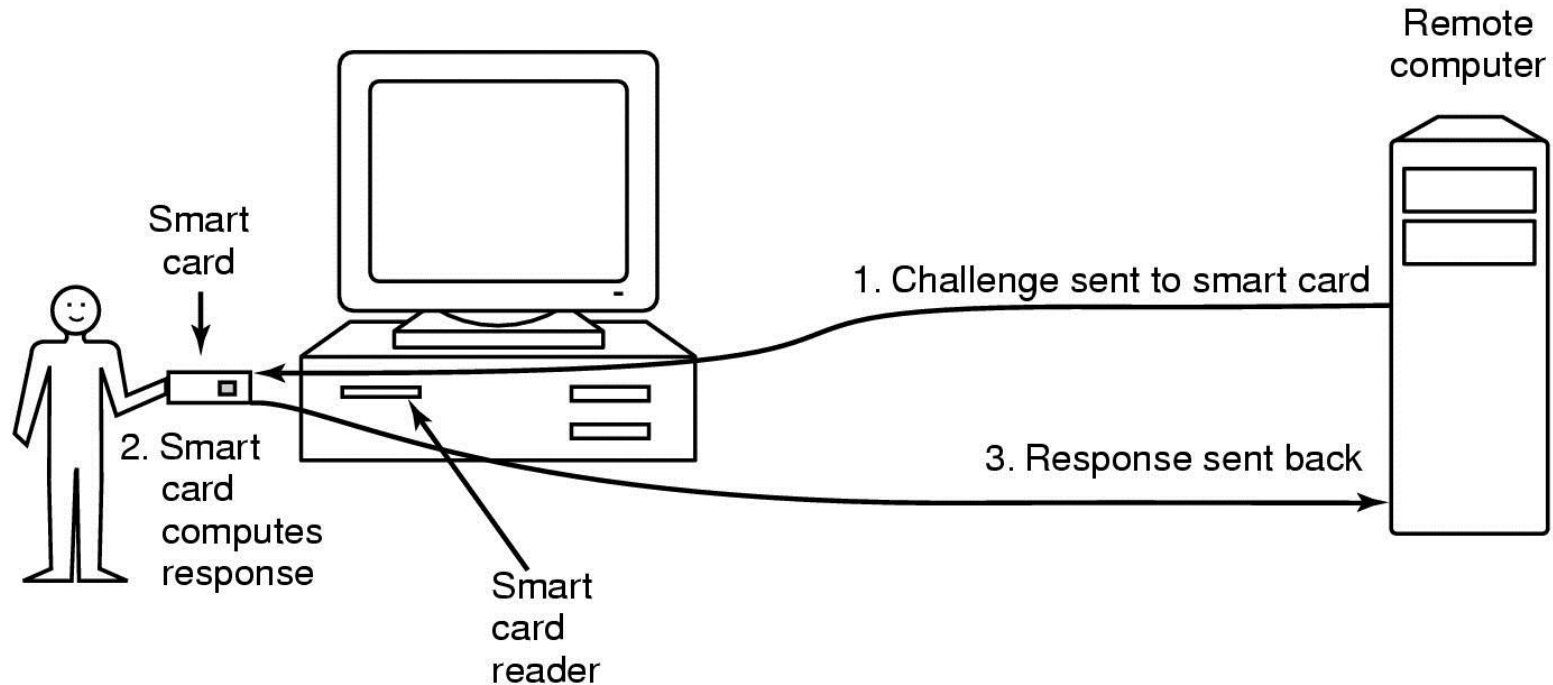
Challenge-Response Authentication

The questions should be chosen so that the user does not need to write them down.

Examples:

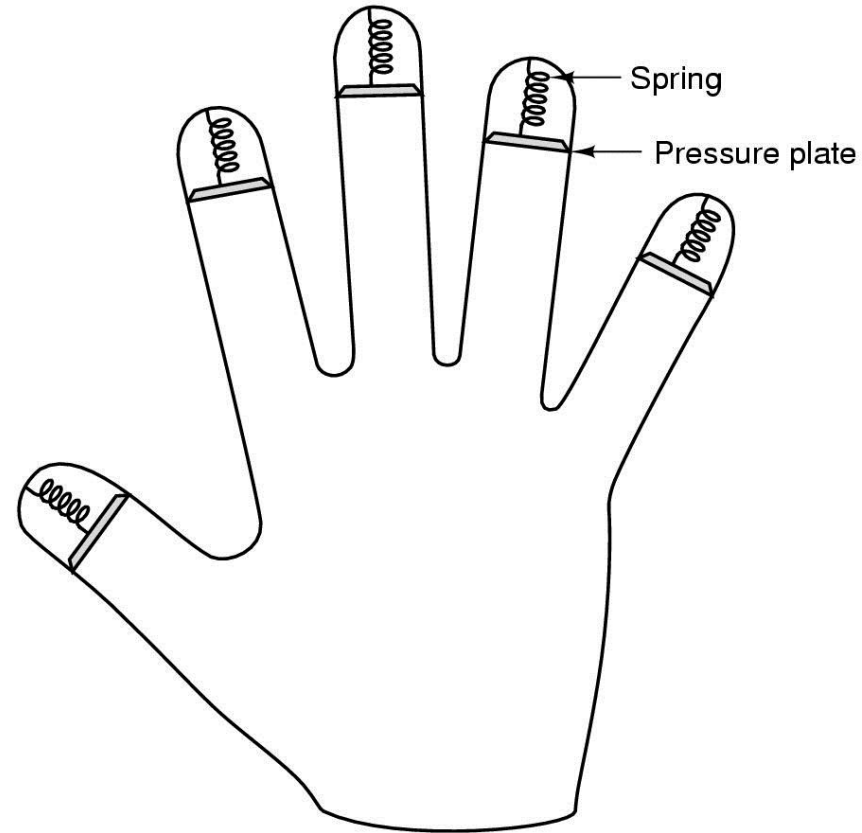
- Who is Marjolein's sister?
- On what street was your elementary school?
- What did Mrs. Woroboff teach?

Authentication Using a Physical Object



Use of a smart card for authentication.

Authentication Using Biometrics



A device for measuring finger length.

UNIX

UNIX History

- The UNIX operating system was born in the late 1960s. It originally began as a one man project led by Ken Thompson of Bell Labs, and has since grown to become the most widely used operating system.
- In the time since UNIX was first developed, it has gone through many different generations and even mutations.
 - Some differ substantially from the original version, like Berkeley Software Distribution (BSD) or Linux.
 - Others, still contain major portions that are based on the original source code.
- An interesting and rather up-to-date timeline of these variations of UNIX can be found at <http://www.levenez.com/unix/history.html>.

General Characteristics of UNIX as an OS

- **Multi-user & Multi-tasking** - most versions of UNIX are capable of allowing multiple users to log onto the system, and have each run multiple tasks. This is standard for most modern OSs.
- **Over 30 Years Old** - UNIX is over 30 years old and it's popularity and use is still high..
- **Large Number of Applications** – there are an enormous amount of applications available for UNIX operating systems. They range from commercial applications such as CAD, Maya, WordPerfect, to many free applications.

Parts of the UNIX OS

- **The Kernel** - handles memory management, input and output requests, and program scheduling.
- **The Shell and Graphical User Interfaces (GUIs)** - basic UNIX shells provides a “command line” interface which allows the user to type in commands. These commands are translated by the shell into something the kernel can comprehend, and then executed by the kernel.
- **The Built-in System Utilities** - are programs that allow a user to perform tasks which involve complex actions. Utilities provide user interface functions that are basic to an operating system, but which are too complex to be built into the shell. Examples of utilities are programs that let us see the contents of a directory, move & copy files, remove files, etc...
- **Application Software & Utilities** –They are additional programs that are bundled with the OS distribution, or available separately.

The UNIX File System

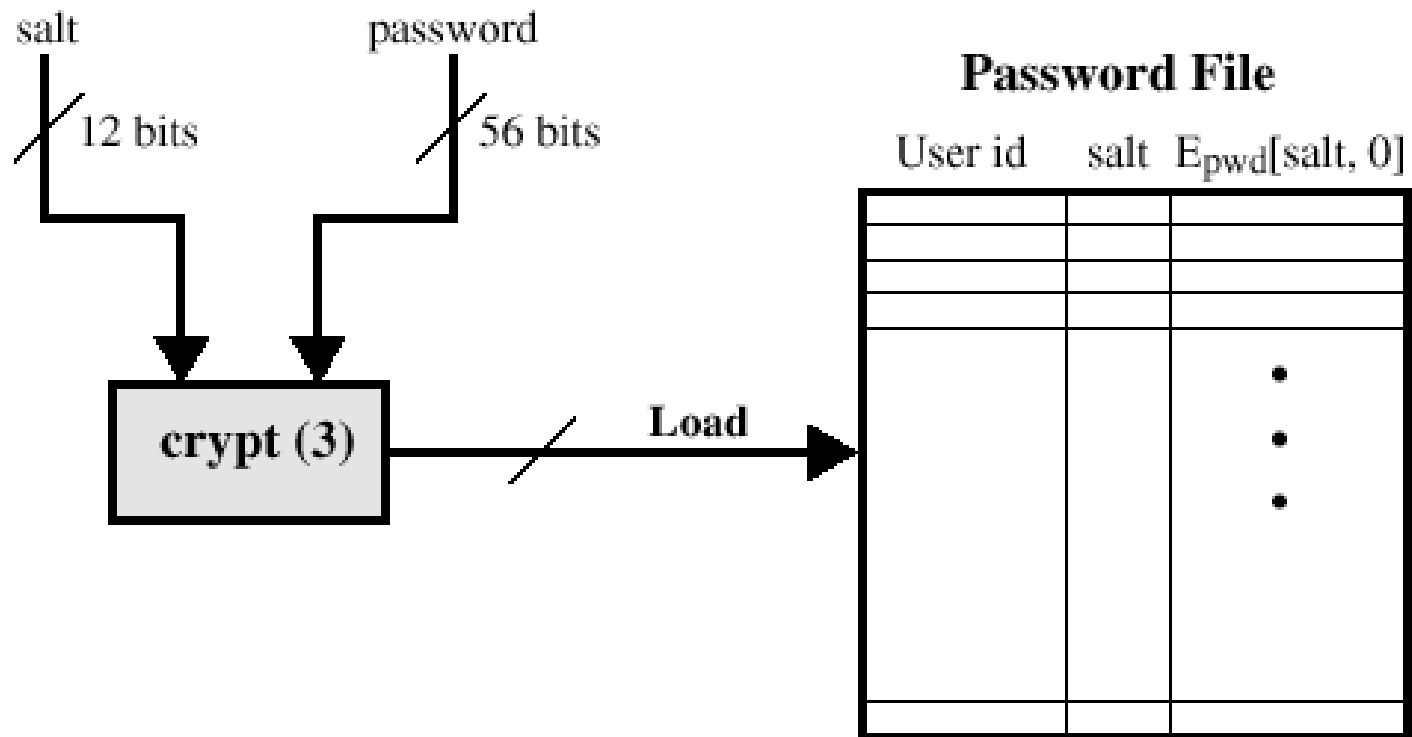
Directory	Contents
bin	Binary (executable) programs
dev	Special files for I/O devices
etc	Miscellaneous system files
lib	Libraries
usr	User directories

Some important directories found in most UNIX systems

UNIX Utility Programs

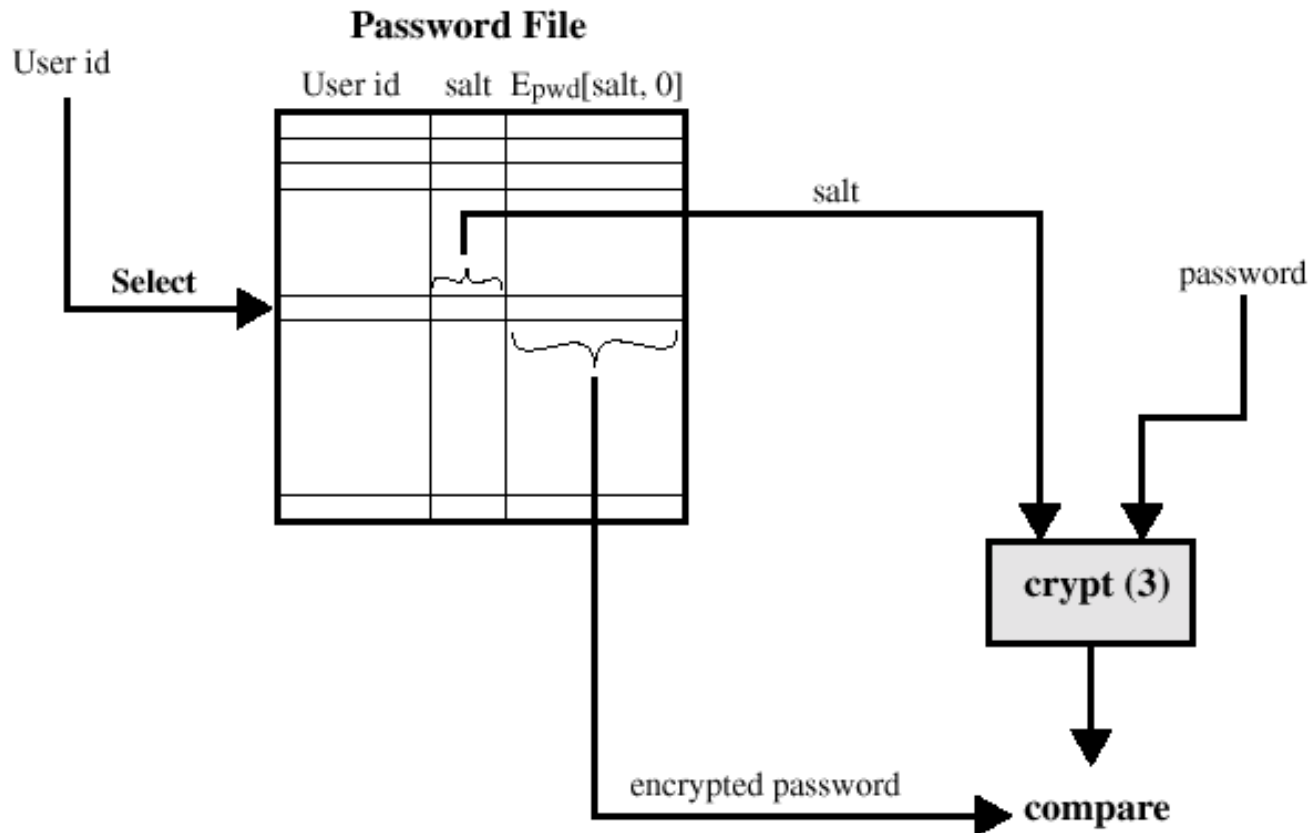
Program	Typical use
cat	Concatenate multiple files to standard output
chmod	Change file protection mode
cp	Copy one or more files
cut	Cut columns of text from a file
grep	Search a file for some pattern
head	Extract the first lines of a file
ls	List directory
make	Compile files to build a binary
mkdir	Make a directory
od	Octal dump a file
paste	Paste columns of text into a file
pr	Format a file for printing
rm	Remove one or more files
rmdir	Remove a directory
sort	Sort a file of lines alphabetically
tail	Extract the last lines of a file
tr	Translate between character sets

UNIX Password Scheme



Loading a new password

UNIX Password Scheme



Verifying a password file

"Salt"

- The salt serves two purposes:
 - Prevents duplicate passwords.
 - Effectively increases the length of the password.

How Are Passwords Stored?

- In past UNIX systems, password are stored in directory: /etc/passwd/
 - World-readable (anyone who accessed the machine would be able to copy the password file to crack at their leisure)
- In newer UNIX Systems
 - Password hashes stored in /etc/shadow directory (or similar)
 - only readable by system administrator (root)
 - Less sensitive information still in /etc/password
 - Added expiration dates for passwords

Security in UNIX

- A UID, or "user ID number", is an integer that identifies particular user .
- Users can be organized into groups, which are also numbered with 16-bit integers called GIDs (Group IDs).
- Assigning users to group is done manually by the system administrator.
- The user with UID 0 is the ***superuser***
- When a file is created, it gets the UID and GID of the creating process.
 - A UID is an integer between 0 and 65535.
 - Files (also processes and other resources) are marked with the UID of their owner
- The file also gets a set of permissions determined by the creating process.
 - These permissions specify what access the owner, the other members of the owner's group and the rest of the users have to file.

Unix Domain

- In UNIX, the domain of a process is defined by its UID and GID.
- Given any (UID, GID) combination, it is possible to make a complete list of all objects (files, including I/O devices represented by special files, etc.) that can be accessed, and whether they can be accessed for reading, writing, or executing.
- Two processes with the same (UID, GID) combination will have access to exactly the same set of objects.
- Processes with different (UID, GID) values will have access to a different set of files.

Security in UNIX

Binary	Symbolic	Allowed file accesses
111000000	rwx-----	Owner can read, write, and execute
111111000	rwrxwx---	Owner and group can read, write, and execute
110100000	rw-r-----	Owner can read and write; group can read
110100100	rw-r--r--	Owner can read and write; all others can read
111101101	rwxr-xr-x	Owner can do everything, rest can read and execute
000000000	-----	Nobody has any access
000000111	-----rwx	Only outsiders have access (strange, but legal)

Some examples of file protection modes

--- no permission

--x execute

-w- write

-wx write and execute

r-- read

r-x read and execute

rw- read and write

rw- read, write and execute

-rwxr-xr-x

- From left to right, we have: permissions, owner, group, size, date, and name.
- Every position in the 10-character permissions field is significant.
 - The first indicates the type of file. This example is of type '-', indicating that it is a normal file.
 - The next three indicate the permissions that the owner has. 'r', 'w', and 'x' are all present, meaning that the owner can read, modify, and execute the file.
 - Members of the group can read and execute the file, but not modify it .
 - The next three characters work the same way, and apply to users who neither own the file, nor are members of the file's group (others).

drwxrwx---

- **d**' for directory in the first position.
- The next three groups of three characters apply to the owner, group, and others, just like a normal file.
- However, the meanings of ' **r**', ' **w**', and ' **x**' are slightly different.
 - ' **r**' denotes permission to list files in the directory.
 - ' **w**' denotes permission to add or remove files in the directory.
 - ' **x**' denotes permission to change to the directory, which is needed to access specified files or subdirectories in the directory.

System Calls for File Protection

System call	Description
<code>s = chmod(path, mode)</code>	Change a file's protection mode
<code>s = access(path, mode)</code>	Check access using the real UID and GID
<code>uid = getuid()</code>	Get the real UID
<code>uid = geteuid()</code>	Get the effective UID
<code>gid = getgid()</code>	Get the real GID
<code>gid = getegid()</code>	Get the effective GID
<code>s = chown(path, owner, group)</code>	Change owner and group
<code>s = setuid(uid)</code>	Set the UID
<code>s = setgid(gid)</code>	Set the GID

- **s** is an error code
- **uid** and **gid** are the UID and GID, respectively

chmod command

- The *chmod* command is used to change the permissions of a file or directory.
- To change the permissions of a file, you must have write access to its directory and you must own the file or be superuser.
- ***chmod u+x filename***
 - 'u+x' means that you are granting ('+' to grant, '-' to remove) access of 'x' (execute for a file, change to directory for a directory-- can also be 'r' or 'w') to the owner ('u' is owner, 'g' is group, and 'o' is others).

- **To change permissions for a file named filename.cgi**
`chmod u=rwx,g=rx,o=rx filename.cgi`
read, execute, and write access to the user (that's you)
read and execute access to the group and
read and execute access to others
- **chmod 775** (for example)
 - When using the numeric system, the code for permissions is as follows: $r = 4$ $w = 2$ $x = 1$ $rwx = 7$
 - The first 7 of `chmod775` tells Unix to change the user's permissions to `rxw` (because $r=4 + w=2 + x=1$ adds up to 7. The second 7 applies to the group, and the last number 5, refers to others ($4+1=5$)).

- **access**

- `access()` checks whether the process would be allowed to read, write or test for existence of the file (or other file system object) whose name is `pathname`.
- If `pathname` is a symbolic link permissions of the file referred to by this symbolic link are tested. `mode` is a mask consisting of one or more of `R_OK`, `W_OK`, `X_OK` and `F_OK`.
- `R_OK`, `W_OK` and `X_OK` request checking whether the file exists and has read, write and execute permissions, respectively. `F_OK` just requests checking for the existence of the file.

- `getuid`
 - `getuid` returns the real user ID of the current process. The real user ID identifies the person who is logged in.
- `geteuid()`
 - returns the effective user ID of the calling process. The effective ID corresponds to the set ID bit on the file being executed.
- `getgid()`
 - returns the real group ID of the calling process.
- `getegid()`
 - returns the effective group ID of the calling process.

Set User Identification Attribute

- The file permissions bits include an execute permission bit for file owner, group and other.
- When the execute bit for the **owner is set to "s"** the set user ID bit is set. This causes any persons or processes that run the file to have access to system resources as though they are the owner of the file.
- When the execute bit for the **group is set to "s"**, the set group ID bit is set and the user running the program is given access based on access permission for the group the file belongs to.

The command:

```
chmod +s myfile
```

sets the user ID bit on the file "myfile".

The command:

```
chmod g+s myfile
```

sets the group ID bit on the file "myfile".

The listing below shows a listing of two files that have the group or user ID bit set.

```
-rws--x--x  1 root  root  14024 Sep  9 1999 chfn  
-rwxr-sr-x  1 root  mail  12072 Aug 16 1999 lockfile
```

- **chown**

- Changes the owner or group associated with a file.
- The chown command changes the owner of the file or directory specified by the File or Directory parameter to the user specified by the Owner parameter.
- The value of the Owner parameter can be a user name from the user database or a numeric user ID.
- Optionally, a group can also be specified. The value of the Group parameter can be a group name from the group database or a numeric group ID.
- Only the root user can change the owner of a file. You can change the group of a file only if you are a root user or if you own the file. If you own the file but are not a root user, you can change the group only to a group of which you are a member.

To change the owner of the file program.c:

chown jim program.c

The user access permissions for program.c now apply to jim. As the owner, jim can use the chmod command to permit or deny other users access to program.c.

To change the owner and group of all files in the directory /tmp/src to owner john and group build:

chown -R john:build /tmp/src

Implementation of Security in Unix

- When a user logs in, the login program, login (which is SETUID root) asks for a login name and a password.
- It hashes the password and then looks in the password file, /etc/passwd , to see if the hash matches the one there.
- If the password is correct, the login program looks in /etc/passwd to see the name of the user's preferred shell, possibly sh.
- The login program then uses setuid and setgid to give itself the user's UID and GID.
- Then it opens the keyboard for standard input, the screen for standard output , and the screen for standard error.
- Finally, it executes the preferred shell, thus terminating itself.
- At this point the preferred shell is running with the correct UID and GID and standard input, output, and error all set to their default devices.

Windows 2000

Security in Windows 2000

- Windows 2000 inherits many properties from NT 4.0.
- It is a true 32-bit multiprocessing system with individually protected processes.
- Each process has a private 32-bit demand-paged virtual address space.
- Windows 2000 is an immensely complex system, consisting of over 29 million lines of C code.
- Windows needs to keep track of a great deal of information about hardware, software, and users. This information was stored in hundreds of .ini (initialization) files spread all over the disk..
 - Nearly all the information needed for booting and configuring the system and tailoring it to the gathered in a big central database called the registry .

- Windows 2000 consists of two major pieces: the *operating system* itself, which runs in kernel and *environment subsystems*, which run in user mode.
 - The kernel is a traditional kernel in the sense process management, memory management, file systems, and so on.
 - The environment subsystems unusual in that they are separate processes that help user programs carry out certain system functions.
- The security manager enforces Windows 2000's elaborate security mechanism, which meets the Defense's Orange Book C2 requirements.
 - The Orange Book specifies a large number of rules that system must meet, starting with authenticated login through how access control is handled, to the pages must be zeroed out before being reused.

- The object manager manages all objects known to the operating system.
- The I/O manager provides a framework for managing I/O devices.
- The process manager handles processes and threads, including their creation and termination. It mechanisms used to manage them.
- The memory manager implements Windows 2000's demand-paged virtual memory architecture.
- The cache manager keeps the most recently used disk blocks in memory to speed up access to them event that they are needed again.

- The plug-and-play manager is sent all notifications of newly attached devices.
- The power manager rides herd on power usage. This consists of turning off the monitor and disks been idle for a while. On laptops, the power manager monitors battery usage and takes action when about to run dry.
- The configuration manager is in charge of the registry. It adds new entries and looks up keys when
- The local procedure call manager provides for a highly-efficient inter-process communication processes and their subsystems.

File Encryption

- Windows 2000 encrypts files, so even in the event the computer is stolen or rebooted using MS-DOS, the files will be unreadable.
- The normal way to use Windows 2000 encryption is to mark certain directories as encrypted, which causes all the files in them to be encrypted, and new files moved to them or created in them to be encrypted as well.
- The actual encryption and decryption is done by a driver called EFS (Encrypting File System), which is positioned between NTFS and the user process. In this way, application programs are unaware of encryption and NTFS itself is only partially involved in it.

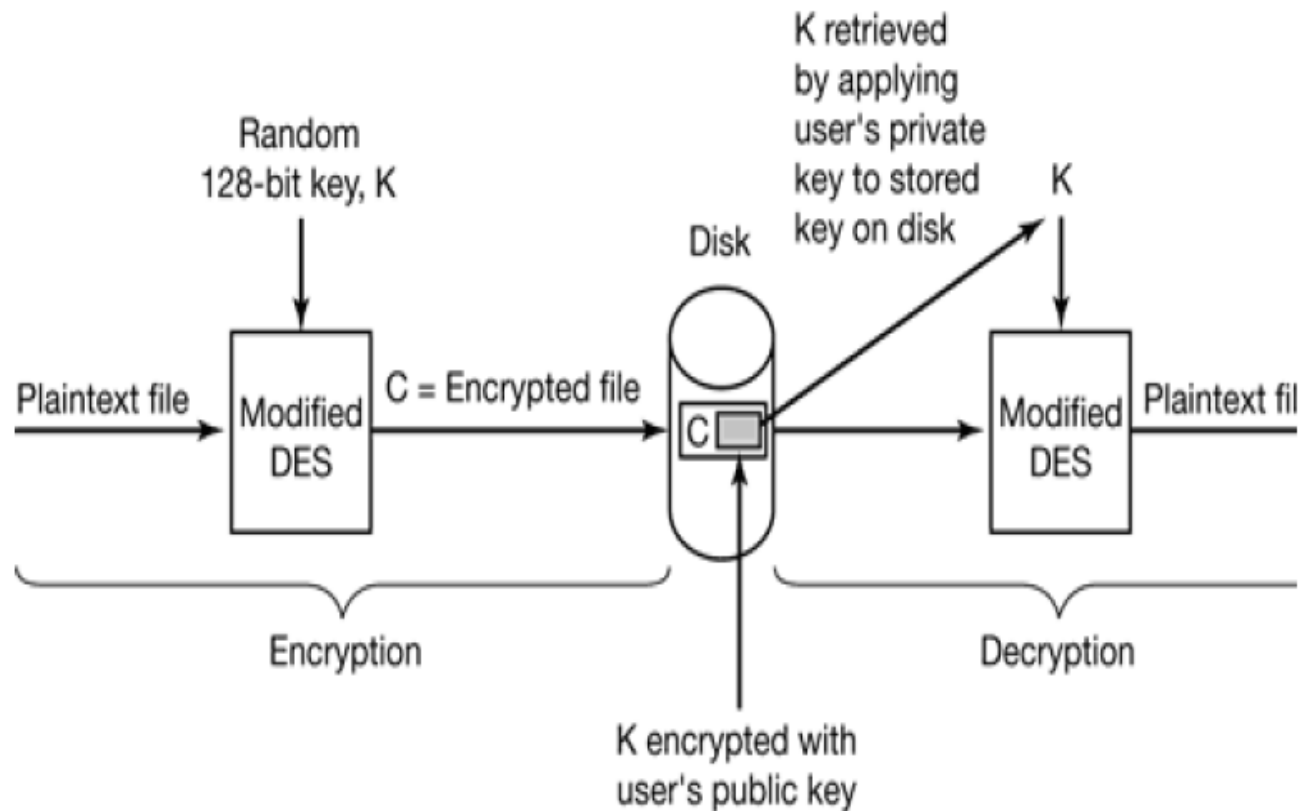


Figure 11-41. Operating of the encrypting file system.

- When the user asks a file to be encrypted, a random key is generated and used to encrypt the file block using a symmetric algorithm
- The current encryption algorithm is a variant of Data Encryption Standard (DES).
- The file keys are encrypted using public key cryptography (PKC) before they are stored on the disk.
 - After the file is encrypted, the location of the user's public key is looked up using information in the registry.
- **To decrypt a file**, the encrypted 128-bit random file key is fetched from disk. However, to decrypt it and retrieve the file key, the user must present the private key.
 - Ideally, the private key should be stored on a smart card, computer, and only inserted in a reader when a file has to be decrypted.

- Windows 2000 inherits many security properties from NT, including the following:
 - Secure login with antispooofing measures
 - Discretionary access controls
 - Privileged access controls
 - Address space protection per process
 - New pages must be zeroed before being mapped in
 - Security auditing

- Secure login means that the system administrator can require all users to have a password to login.
 - Spoofing is when a malicious user writes a program that displays the login prompt or screen and then walks away from the computer in the hope that an innocent user will sit down and enter a name and password.
 - The name and password are then written to disk and the user is told that login has failed.
- Discretionary access controls allow the owner of a file or other object to say who can use it and in what way.
- Privileged access controls allow the system administrator (superuser) to override them when needed.

- Address space protection simply means that each process has its own protected virtual address space not accessible by any unauthorized process.
- When a stack grows, the pages mapped in are initialized to zero so processes cannot find any old information put there by the previous owner.
- Security auditing allows the administrator to produce a log of certain security-related events.

Fundamental Concepts

- Every Windows 2000 user (and group) is identified by a SID (Security ID).
- SIDs are binary numbers header followed by a long random component.
- Each SID is intended to be unique worldwide.
- When a user starts up a process, the process and its threads run under the user's SID.
- Most of the security system is designed to make sure that each object can be accessed only by threads with authorized SIDs.

- Each process has an **access token** that specifies its SID and other properties. It is normally assigned at login time by winlogon and is shown in Fig.

Header	Expiration Time	Groups	Default CACL	User SID	Group SID	Restricted SIDs	Privileges
--------	-----------------	--------	--------------	----------	-----------	-----------------	------------

- The header contains some administrative information.
- The expiration time field could tell when the token ceases to be valid, but it is currently not used.
- The Groups fields specify the groups to which the process belongs.
- The default DACL (Discretionary ACL) is the access control list assigned to objects created by the process if no other ACL is specified.
- The user SID tells who owns the process.
- The restricted SIDs are to allow untrustworthy processes to take part in jobs with trustworthy processes but with less power to do damage.
- The privileges listed, if any, give the process special powers, such as the right to shut the machine down or access files to which access would otherwise be denied.

Security descriptor

- Every object has a security descriptor associated with it that tells who can perform which operations on it.
- A security descriptor consists of a header followed by a DACL with one or more ACEs (Access Control Elements).
 - The two main kinds of elements are Allow and Deny.
- An **allow element** specifies a SID and a bitmap that specifies which operations processes with that SID may perform on the object.
- A **deny clement** works the same way, except a match means the caller may not perform the operation.

Security descriptor.....

- For example, Ida has a file whose security descriptor specifies that everyone has read access, Elvis has no access.
- Cathy has read/write access, and Ida herself has full access.
- This simple example is illustrated in Fig .

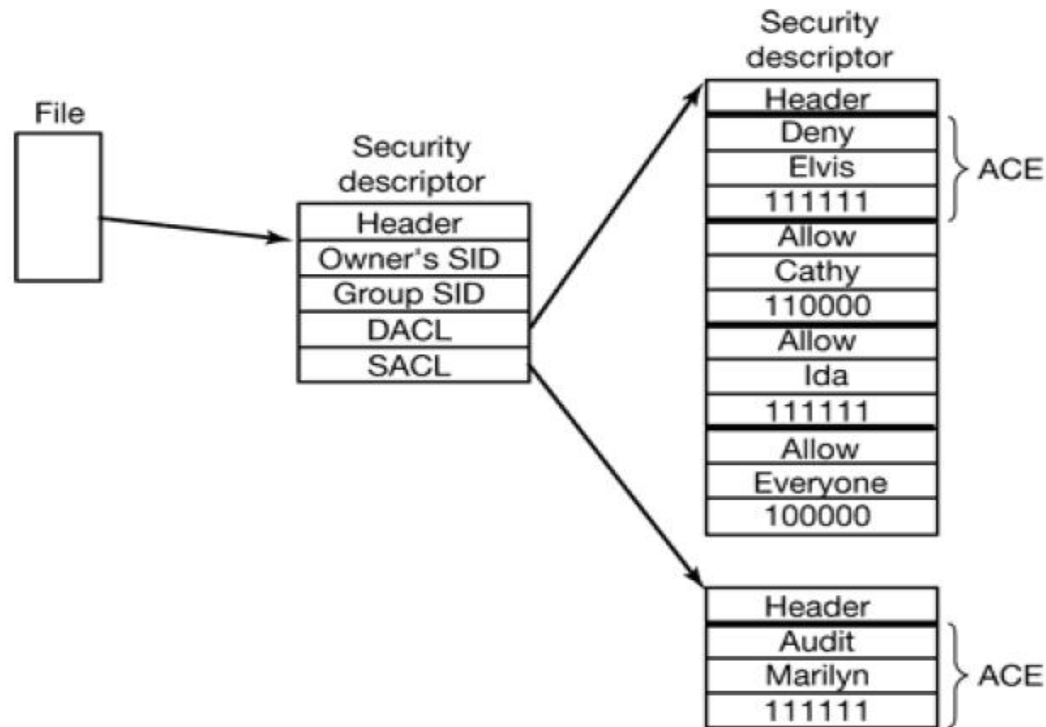


Figure 11-43. An example security descriptor for a file.

Security descriptor.....

- In addition to the DACL, a security descriptor also has a SACL (System Access Control list), which is like DACL except that it specifies not who may use the object, but which operations on the object are system-wide security event log.
- In the above figure, every operation that Marilyn performs on the file Windows 2000 provides additional **auditing** features to log sensitive accesses.

- Most of the Windows 2000 access control mechanism is based on **security descriptors**.
- The usual pattern is that when a process creates an object, it provides a security descriptor as one of the parameters to the CreateProcess, CreateFile , or other object creation call.
- If no security descriptor is provided in the object creation call, the in the caller's access token is used instead.

Security API Calls

- To create a security descriptor, storage for it is first allocated and then initialized using `InitializeSecurityDescriptor`.
 - This call fills in the header.
- If the owner SID is not known, it can be looked up by name using `LookupAccountSid`.
- Security descriptor's DACL (or SACL) can be initialized with `InitializeAcl` .
- ACL entries can be added using `AddAccessAllowedAce` , and `AddAccessDeniedAce` .
- `DeleteAce` can be used to remove an entry.
- When the ACL is ready, `SetSecurityDescriptorDacl` can be it to the security descriptor.
- When the object is created, the newly minted security descriptor as a parameter to have it attached to the object using `SetSecurityDescriptorDacl` .

Orange Book Security

(Tanenbaum - page no. 659)

- In 1985, U.S. Dept. of Defense published a document formally known as Dept. of Defense standard DoD 5200.28, but usually called the Orange Book on account of its cover, which divides operating systems into seven categories based on their security properties.
- **Level D** has no security requirements at all. It collects all the systems that have failed to pass even the minimum security tests. MS-DOS and Windows 95/98/Me are level D.

- **Level C** is intended for environments with cooperating users.
- **C1** requires a protected mode operating system, authenticated user login, and the ability for users to specify which files can be made available to other users and how (discretionary access control). Minimal security testing and documentation are also required.
- **C2** adds the requirement that discretionary access control is down to the level of the individual user. It also requires that objects (e.g., files, virtual memory pages) given to users must be initialized to all zeros, and a minimal amount of auditing is needed.
- The UNIX rwx scheme meets C1 but does not meet C2. For this, a more elaborate scheme, such as ACLs or equivalent, are needed.

- The **B** and **A** levels require all controlled users and objects to be assigned a security label, such as unclassified, secret, or top secret. The system must be capable of enforcing the Bell-La Padula information flow model.
- **B2** adds to this requirement that the system has been designed top-down in a modular way. The design must be presented in such a way that it can be verified.
- **B3** contains all of B2's features plus there must be ACLs with users and groups, a formal TCB must be presented, adequate security auditing must be present, and secure crash recovery must be included.
- **A1** requires a formal model of the protection system and a proof that the model is correct. It also requires a demonstration that the implementation conforms to the model.

The Concept of the Hole

- A hole is any feature of hardware or software that allows unauthorized users to gain access or increase their level of access without authorization.
- A hole could be virtually anything
 - For example, many peculiarities of hardware or software commonly known to all users qualify as holes.
 - CMOS passwords on IBM compatibles are lost when the CMOS battery is shorted, disabled, or removed.
 - Even the ability to boot into single-user mode on a workstation (safe mode – admin rights) could be classified as a hole.
 - This is so because it will allow a malicious user to begin entering interactive command mode, perhaps seizing control of the machine.

Types of holes

- Holes that allow denial of service
- Holes that allow local users with limited privileges to increase those privileges without authorization
- Holes that allow outside parties (on remote hosts) unauthorized access to the network.

Holes That Allow Denial of Service

- Holes that allow denial of service are in category C, and are of low priority.
- These attacks are almost always operating-system based.
 - These holes exist within the *networking portions of the operating system* itself. When such holes exist, they must generally be corrected by the authors of the software or by patches from the vendor.
 - For large networks or sites, a denial-of-service attack is of only limited significance. It amounts to a nuisance and no more.
 - Smaller sites may suffer in a denial-of-service attack. This is especially so if the site maintains only a single machine.

- Denial of Service attacks are a class of attack in which an individual or individuals exploit aspects of the Internet Protocol suite to deny other users of legitimate access to systems and information.
- The TCP SYN attack is one in which connection requests are sent to a server in high volume, causing it to become overwhelmed with requests.
- The result is a slow or unreachable server, and upset customers.

Holes That Allow Local Users Unauthorized Access

- Class B are those holes that allow local users to gain increased and unauthorized access. These types of holes are typically found within applications .
- A fine example of a hole that allows local users increased and unauthorized access is a well-known sendmail problem.
- Sendmail is the world's most popular method of transmitting electronic mail. It is the heart of the Internet's e-mail system.
- This program is initiated as a daemon at boot time and remains active as long as the machine is active.

- By manipulating the sendmail environment, the user can have sendmail execute an arbitrary program with root privileges. Thus, a local user can gain a form of root access.
- A local user is someone who has an account on the target machine or network.
 - If a local user successfully manages to exploit such a hole, the system administrator may never discover it.
 - A local user can employ basic system utilities to learn more about the local network. Therefore, a local user with even fleeting increased access can exploit that access to a much greater degree.

Holes That Allow Remote Users Unauthorized Access (Class A)

- Class A holes are the most threatening of all and not surprisingly, most of them stem from either poor system administration or misconfiguration.
- The typical example of a misconfiguration (or configuration failure) is any sample script that remains on the drive, even though the distribution docs advise that it be removed.
- One HTTP server on the Novell platform includes a sample script called `convert.bas`. The script, written in BASIC, allows remote users to read any file on the system.