Module 4
Software Security

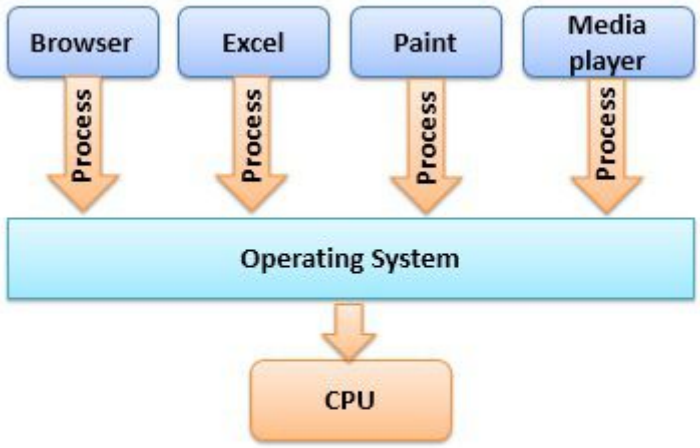
Submodule 1: Operating System Security

OS Concepts

- An **operating system (OS)** provides the interface between the users of a computer and that computer's hardware.
  - An operating system manages the ways applications access the resources in a computer, including its disk drives, CPU, main memory, input devices, output devices, and network interfaces.
  - An operating system manages multiple users.
  - An operating system manages multiple programs.

Multitasking

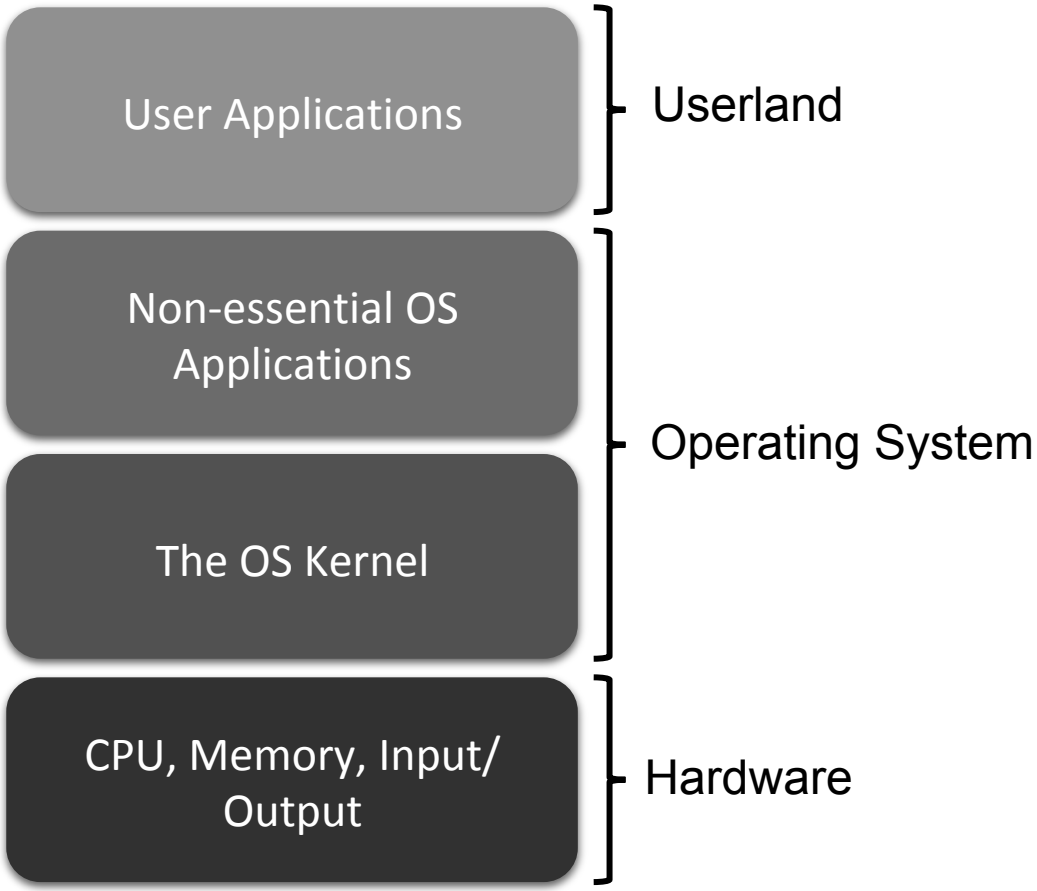
- Give each running program a "slice" of the CPU's time.
- The CPU is running so fast that to any user it appears that the computer is running all the programs simultaneously.



Multitasking

The Kernel

- The **kernel** is the core component of the operating system.
  - It handles the management of low-level hardware resources, including memory, processors, and input/output (I/O) devices, such as a keyboard, mouse, or video display.
- Most operating systems define the tasks associated with the kernel in terms of a **layer** metaphor:
  - The hardware components, such as the CPU, memory, and input/output devices being on the bottom
  - Users and applications being on the top.


User Applications
Userland
Non-essential OS Applications
Operating System
The OS Kernel
CPU, Memory, Input/ Output
Hardware

Input/Output

- Computers have input/output devices:
  - keyboard, mouse, video display, and network card, scanner, Wi-Fi interface, video camera, USB ports
- Operating system uses device driver to encapsulates the details of how interaction with that device should be done:
  - The **application programmer interface** (**API**)**,** which the device drivers present to application programs, allows those programs to interact with those devices at a fairly high level, while the operating system does the "heavy lifting" of performing the low-level interactions that make such devices actually work.

System Calls

- User applications don't communicate directly with low-level hardware components, and instead delegate such tasks to the kernel via **system calls**.

- System calls are usually contained in a collection of programs called a **library:**
  - Library provides an interface through which applications can use a predefined series of APIs that define the functions for communicating with the kernel
  - Examples of system calls include those for performing file I/O (open, close, read, write) and running application programs (exec).

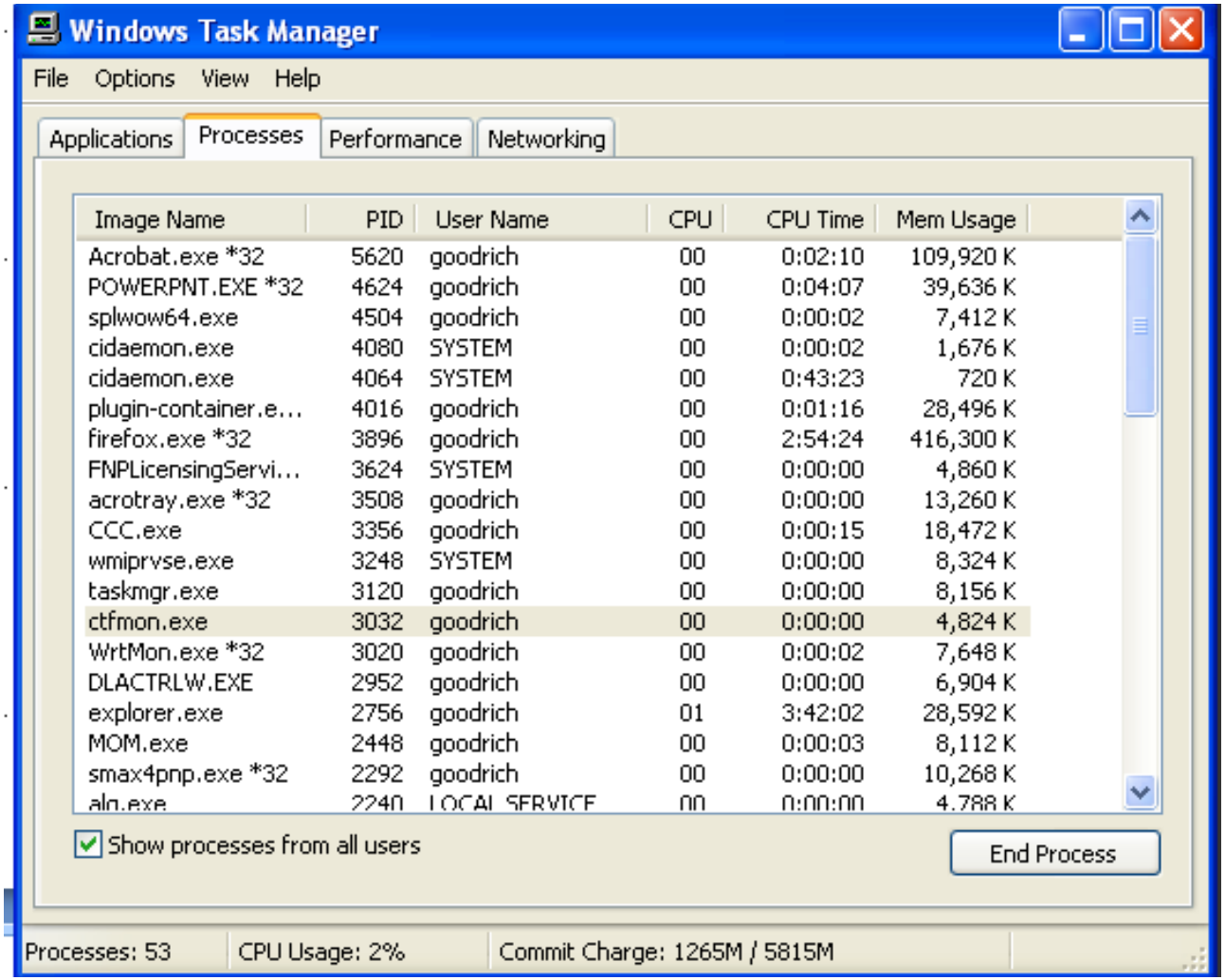
Processes

- A **process** is an instance of a program that is currently executing.
- The actual contents of all programs are initially stored in persistent storage, such as a hard drive.
- In order to be executed, a program must be loaded into random-access memory (RAM) and uniquely identified as a process.
- In this way, multiple copies of the same program can be run as different processes.
  - For example, we can have multiple copies of MS Powerpoint open at the same time.

Process IDs

- Each process running on a given computer is identified by a unique nonnegative integer, called the **process ID (PID).**

- Given the PID for a process, we can then associate its CPU time, memory usage, user ID (UID), program name, etc.


Windows Task Manager
File Options View Help
Applications Processes Performance Networking
Image Name PID User Name CPU CPU Time Mem Usage
Acrobat.exe *32 5620 goodrich 00 0:02:10 109,920 K
POWERPNT.EXE *32 4624 goodrich 00 0:04:07 39,636 K
splwow64.exe 4504 goodrich 00 0:00:02 7,412 K
cidaemon.exe 4080 SYSTEM 00 0:00:02 1,676 K
cidaemon.exe 4064 SYSTEM 00 0:43:23 720 K
plugin-container.e... 4016 goodrich 00 0:01:16 28,496 K
firefox.exe *32 3896 goodrich 00 2:54:24 416,300 K
FNPLicensingServi... 3624 SYSTEM 00 0:00:00 4,860 K
acrotray.exe *32 3508 goodrich 00 0:00:00 13,260 K
CCC.exe 3356 goodrich 00 0:00:15 18,472 K
wmiprvse.exe 3248 SYSTEM 00 0:00:00 8,324 K
taskmgr.exe 3120 goodrich 00 0:00:00 8,156 K
ctfmon.exe 3032 goodrich 00 0:00:00 4,824 K
WrtMon.exe *32 3020 goodrich 00 0:00:02 7,648 K
DLACTRLW.EXE 2952 goodrich 00 0:00:00 6,904 K
explorer.exe 2756 goodrich 01 3:42:02 28,592 K
MOM.exe 2448 goodrich 00 0:00:03 8,112 K
smax4pnp.exe *32 2292 goodrich 00 0:00:00 10,268 K
Show processes from all users
End Process
Processes: 53 CPU Usage: 2% Commit Charge: 1265M / 5815M

File Systems

- A **filesystem** is an abstraction of how the external, nonvolatile memory of the computer is organized.
- Operating systems typically organize files hierarchically into **folders,** also called **directories.**
- Each folder may contain files and/or subfolders.
- Thus, a volume, or drive, consists of a collection of nested folders that form a **tree**.
- The topmost folder is the **root** of this tree and is also called the root folder.

File Permissions

- File permissions are checked by the operating system to determine if a file is readable, writable, or executable by a user or group of users.
- In Unix-like OS's, a **file permission matrix** shows who is allowed to do what to the file.
  - Files have **owner permissions**, which show what the owner can do, and **group permissions**, which show what some group id can do, and **world permissions**, which give default access rights.

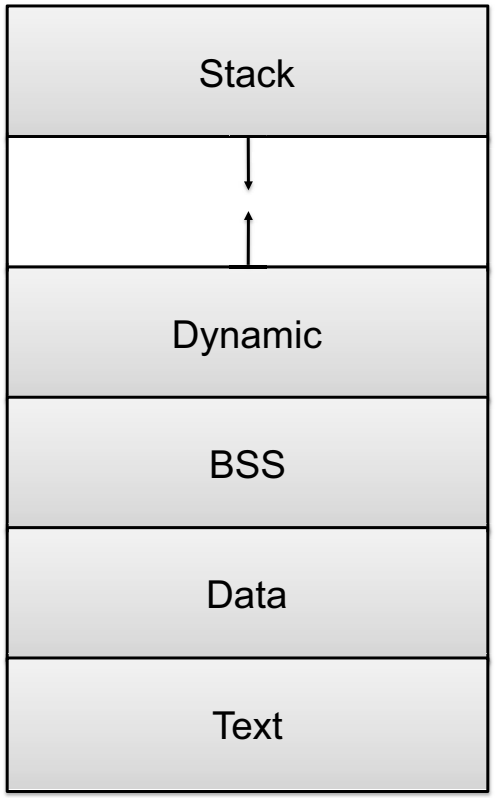
```
rodan:~/java % ls -l
total 24
-rwxrwxrwx   1 goodrich faculty     2496 Jul 27 08:43 Floats.class
-rw-r--r--   1 goodrich faculty     2723 Jul 12  2006 Floats.java
-rw-------   1 goodrich faculty      460 Feb 25  2007 Test.java
rodan:~/java %
```

Memory Management

- The RAM memory of a computer is its **address space**.
  - It contains both the code for the running program, its input data, and its working memory.
  - For any running process, it is organized into different segments, which keep the different parts of the address space separate.
  - Security concerns require that we never mix up these different segments.

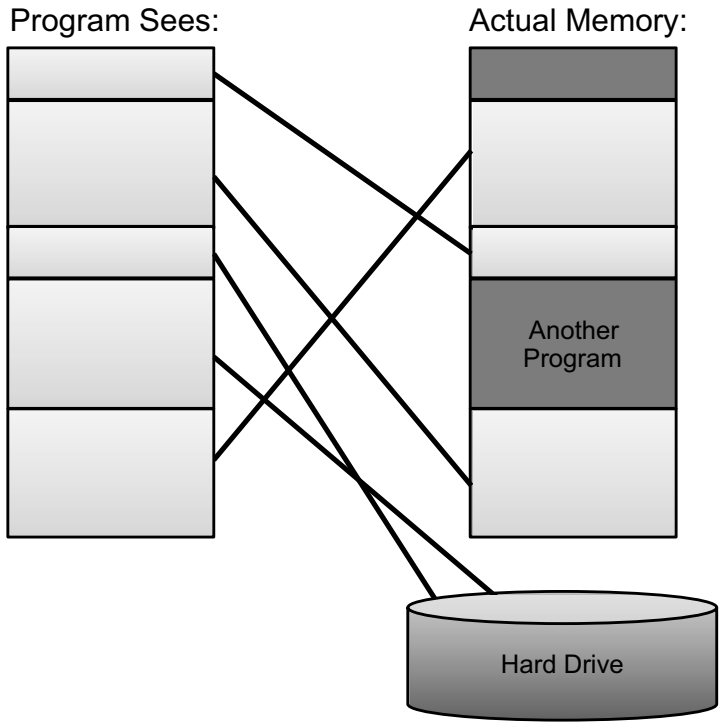
Memory Organization

- **Text.** This segment contains the actual (binary) machine code of the program.
- **Data.** This segment contains static program variables that have been initialized in the program code.
- **BSS.** This segment, which is named for an antiquated acronym for block started by symbol, contains static variables that are uninitialized.
- **Heap.** This segment, which is also known as the dynamic segment, stores data generated during the execution of a process.
- **Stack.** This segment houses a stack data structure that grows downwards and is used for keeping track of the call structure of subroutines (e.g., methods in Java and functions in C) and their arguments.


Stack
Dynamic
BSS
Data
Text

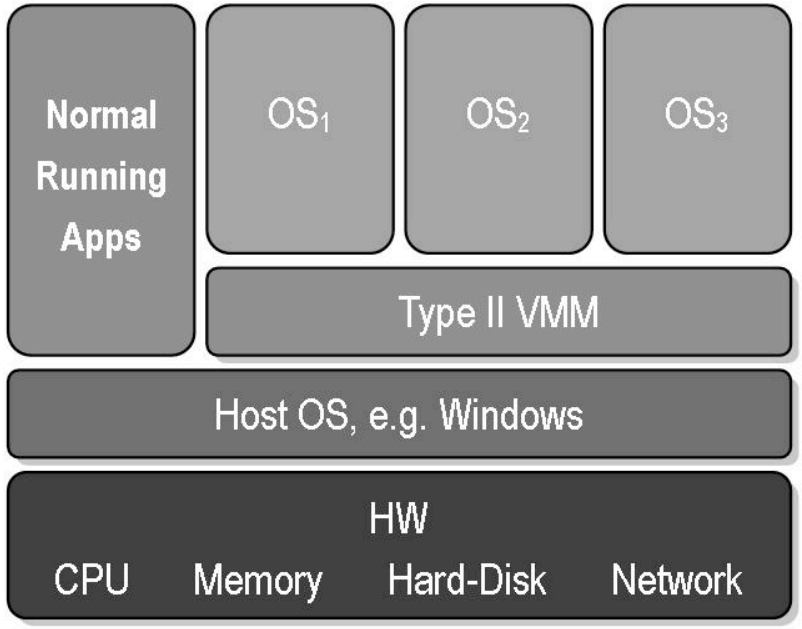
Virtual Memory

- There is generally not enough computer memory for the address spaces of all running processes.
- Nevertheless, the OS gives each running process the illusion that it has access to its complete (contiguous) address space.
- In reality, this view is **virtual**, in that the OS supports this view, but it is not really how the memory is organized.
- Instead, memory is divided into **pages**, and the OS keeps track of which ones are in memory and which ones are stored out to disk.


Program Sees:
Actual Memory:
Another Program
Hard Drive

Virtual Machines

- **Virtual machine:** A view that an OS presents that a process is running on a specific architecture and OS, when really it is something else. E.g., a windows emulator on a Mac.
- **Benefits:**
  - Hardware Efficiency
  - Portability
  - Security
  - Management

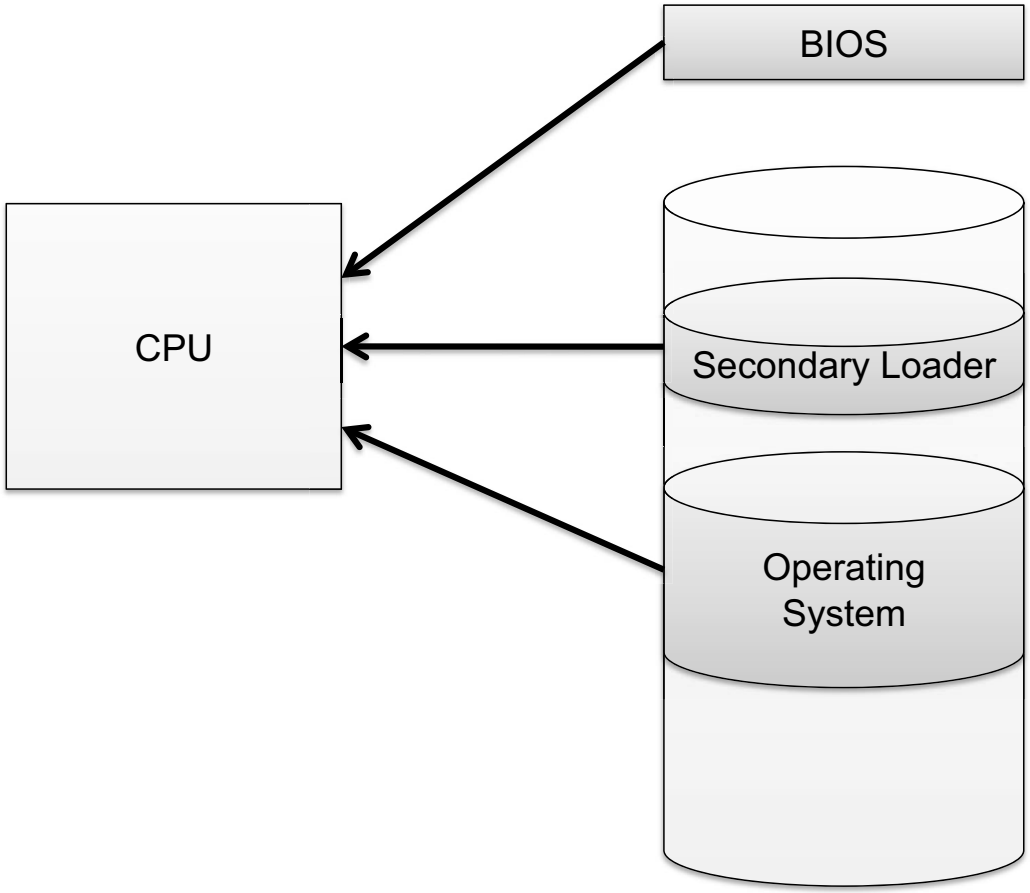


The Boot Sequence

- The action of loading an operating system into memory from a powered-off state is known as **booting** or **bootstrapping.**

- When a computer is turned on, it first executes code stored in a firmware component known as the **BIOS (basic input/output system).**

- On modern systems, the BIOS loads into memory the **second-stage boot loader,** which handles loading the rest of the operating system into memory and then passes control of execution to the operating system.


BIOS
CPU
Secondary Loader
Operating
System

BIOS Passwords

- A malicious user could potentially seize execution of a computer at several points in the boot process.

- To prevent an attacker from initiating the first stages of booting, many computers feature a **BIOS password** that does not allow a second-stage boot loader to be executed without proper authentication.

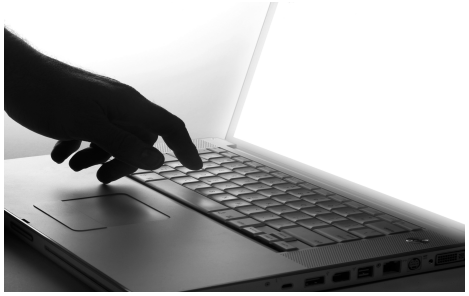
Hibernation

- Modern machines have the ability to go into a powered-off state known as **hibernation.**

- While going into hibernation, the OS stores the contents of machine's memory into a **hibernation file** (such as hiberfil.sys) on disk so the computer can be quickly restored later.

- But… without additional security precautions, hibernation exposes a machine to potentially invasive forensic investigation.

1. User closes a laptop computer, putting it into hibernation.



2. Attacker copies the hiberfil.sys file to discover any unencrypted passwords that were stored in memory when the computer was put into hibernation.

Event Logging

- Keeping track of:
  - What processes are running
  - What other machines have interacted with the system via the Internet
  - If the operating system has experienced any unexpected or suspicious behavior can often leave important clues not only for troubleshooting ordinary problems, but also for determining the cause of a security breach.

Memory and Filesystem Security

- The contents of a computer are encapsulated in its memory and filesystem.

- Thus, protection of a computer's content has to start with the protection of its memory and its filesystem.

Password Security

- The basic approach to guessing passwords from the password file is to conduct a **dictionary attack**, where each word in a dictionary is hashed and the resulting value is compared with the hashed passwords stored in the password file.

- A dictionary of 500,000 "words" is often enough to discover most passwords.

Password Salt

- One way to make the dictionary attack more difficult to launch is to use **salt**.

- Associate a random number with each userid.

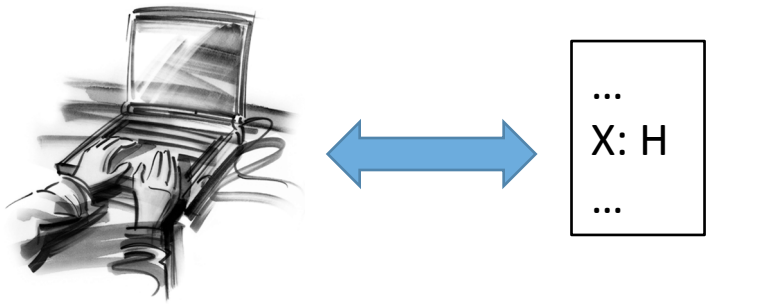
- Rather than comparing the hash of an entered password with a stored hash of a password, the system compares the hash of an entered password and the salt for the associated userid with a stored hash of the password and salt.

Without salt:

1. User types userid, X, and password, P.

2. System looks up H, the stored hash of X's password.

3. System tests whether h(P) = H.

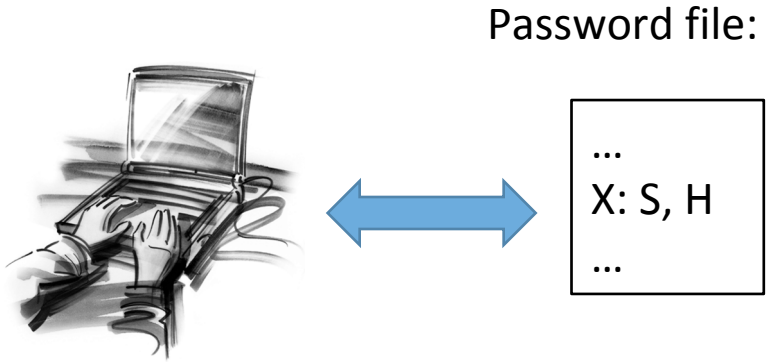


With salt:

1. User types userid, X, and password, P.

2. System looks up S and H, where S is the random salt for userid X and H is stored hash of S and X's password.

3. System tests whether h(S||P) = H.



How Salt Increases Search Space Size

- Assuming that an attacker cannot find the salt associated with a userid he is trying to compromise, then the search space for a dictionary attack on a salted password is of size

$$2^B * D,$$

  where B is the number of bits of the random salt and D is the size of the list of words for the dictionary attack.

- For example, if a system uses a 32-bit salt for each userid and its users pick passwords in a 500,000 word dictionary, then the search space for attacking salted passwords would be

$$2^{32} * 500{,}000 = 2{,}147{,}483{,}648{,}000{,}000,$$

  which is over 2 quadrillion.

- Also, even if an attacker can find a salt password for a userid, he only learns one password.

General Principles

- Files and folders are managed by the operating system
- Applications, including shells, access files through an API
- Access control entry (ACE)
  - Allow/deny a certain type of access to a file/folder by user/group
- Access control list (ACL)
  - Collection of ACEs for a file/folder
- A file handle provides an opaque identifier for a file/folder
- File operations
  - Open file: returns file handle
  - Read/write/execute file
  - Close file: invalidates file handle
- Hierarchical file organization
  - Tree (Windows)

Discretionary Access Control (DAC)

- Users can protect what they own
  - The owner may grant access to others
  - The owner may define the type of access (read/write/execute) given to others
- DAC is the standard model used in operating systems
- Mandatory Access Control (MAC)
  - Multiple levels of security for users and documents
  - Read down and write up principles

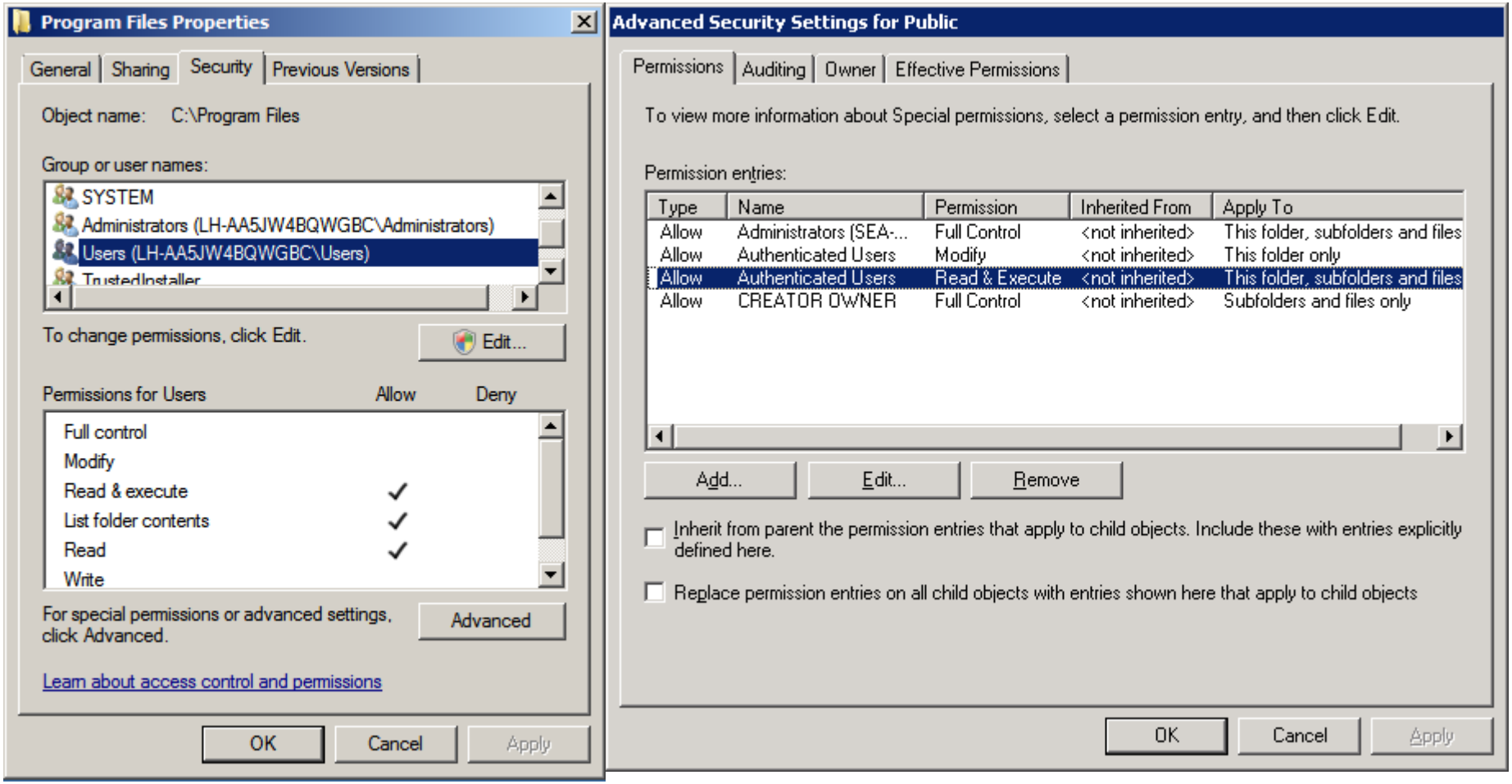
Closed vs. Open Policy

- Closed policy
  - Also called “default secure”
  - Give Tom read access to “foo”
  - Give Bob r/w access to “bar
  - Tom: I would like to read “foo”
    - Access allowed
  - Tom: I would like to read “bar”
    - Access denied
- Open Policy
  - Deny Tom read access to “foo”
  - Deny Bob r/w access to “bar”
  - Tom: I would like to read “foo”
    - Access denied
  - Tom: I would like to read “bar”
    - Access allowed

Access Control Entries and Lists

- An Access Control List (ACL) for a resource (e.g., a file or folder) is a sorted list of zero or more Access Control Entries (ACEs)

- An ACE specifies that a certain set of accesses (e.g., read, execute and write) to the resources is allowed or denied for a user or group


Program Files Properties
General
Sharing
Security
Previous Versions
Object name: C:\Program Files
Group or user names:
SYSTEM
Administrators (LH-AA5JW4BQWGBC\Administrators)
Users (LH-AA5JW4BQWGBC\Users)
TrustedInstaller
To change permissions, click Edit.
Edit...
Permissions for Users
Allow
Deny
Full control
Modify
Read & execute
List folder contents
Read
Write
For special permissions or advanced settings, click Advanced.
Advanced
Learn about access control and permissions
OK
Cancel
Apply
Advanced Security Settings for Public
Permissions
Auditing
Owner
Effective Permissions
To view more information about Special permissions, select a permission entry, and then click Edit.
Permission entries:
Type
Name
Permission
Inherited From
Apply To
Allow
Administrators (SEA-...
Full Control
<not inherited>
This folder, subfolders and files
Allow
Authenticated Users
Modify
<not inherited>
This folder only
Allow
Authenticated Users
Read & Execute
<not inherited>
This folder, subfolders and files
Allow
CREATOR OWNER
Full Control
<not inherited>
Subfolders and files only
Add...
Edit...
Remove
Inherit from parent the permission entries that apply to child objects. Include these with entries explicitly defined here.
Replace permission entries on all child objects with entries shown here that apply to child objects
OK
Cancel
Apply

Unix Permissions

- Standard for all UNIXes
- Every file is owned by a user and has an associated group
- Permissions often displayed in compact 10-character notation
- To see permissions, use `ls -l`

```
jk@sphere:~/test$ ls -l
total 0
-rw-r-----  1 jk ugrad 0 2005-10-13 07:18 file1
-rwxrwxrwx  1 jk ugrad 0 2005-10-13 07:18 file2
```

Permissions Examples (Regular Files)

-rw-r—r--	read/write for owner, read-only for everyone else
-rw-r-----	read/write for owner, read-only for group, forbidden to others
-rwx------	read/write/execute for owner, forbidden to everyone else
-r--r--r--	read-only to everyone, including owner
-rwxrwxrwx	read/write/execute to everyone

Permissions Examples (Directories)

drwxr-xr-x	all can enter and list the directory, only owner can add/delete files
drwxrwx---	full access to owner and group, forbidden to others
drwx--x---	full access to owner, group can access known filenames in directory, forbidden to others
-rwxrwxrwx	full access to everyone

Special Permission Bits

- Three other permission bits exist
  - Set-user-ID (“suid” or “setuid”) bit
  - Set-group-ID (“sgid” or “setgid”) bit
  - Sticky bit

Set-user-ID

- Set-user-ID ("suid" or "setuid") bit
  - On executable files, causes the program to run as file owner regardless of who runs it
  - Ignored for everything else
  - In 10-character display, replaces the 4th character (x or -) with s (or S if not also executable)
    - -rwsr-xr-x: setuid, executable by all
    - -rwxr-xr-x: executable by all, but not setuid
    - -rwSr--r--: setuid, but not executable - not useful

Set-group-ID

- Set-group-ID ("sgid" or "setgid") bit
  - On executable files, causes the program to run with the file's group, regardless of whether the user who runs it is in that group
  - On directories, causes files created within the directory to have the same group as the directory, useful for directories shared by multiple users with different default groups
  - Ignored for everything else
  - In 10-character display, replaces 7th character (x or -) with s (or S if not also executable)

    -rwxr-sr-x: setgid file, executable by all

    drwxrwsr-x: setgid directory; files within will have group of directory

    -rw-r-Sr--: setgid file, but not executable - not useful

Sticky Bit

- On directories, prevents users from deleting or renaming files they do not own
- Ignored for everything else
- In 10-character display, replaces 10th character (x or -) with t (or T if not also executable)
  - drwxrwxrwt: sticky bit set, full access for everyone
  - drwxrwx--T: sticky bit set, full access by user/group
  - drwxr--r-T: sticky, full owner access, others can read *(useless)*

Octal Notation

- Use three or four digits from 0 to 7

- Digits from left (most significant) to right(least significant):
  [special bits][user bits][group bits][other bits]

- Special bit digit =
  (4 if setuid) + (2 if setgid) + (1 if sticky)

- All other digits =
  (4 if readable) + (2 if writable) + (1 if executable)

644 or 0644	read/write for owner, read-only for everyone else
775 or 0775	read/write/execute for owner and group, read/execute for others
640 or 0640	read/write for owner, read-only for group, forbidden to others
2775	same as 775, plus setgid (useful for directories)
777 or 0777	read/write/execute to everyone *(dangerous!)*
1777	same as 777, plus sticky bit

Acknowledgement

- Part of the content in this document is adopted from the recommended textbook:

Michael Goodrich, Roberto Tamassia, “Introduction to Computer Security”, 1st Edition. Pearson. ISBN-13: 978-0321512949, ISBN-10: 9780321512949