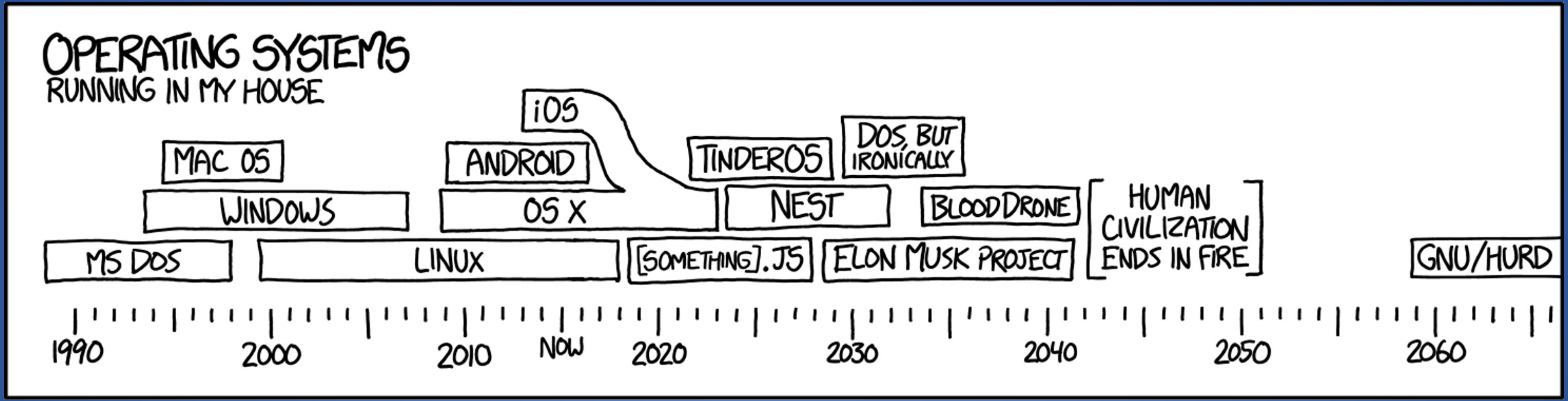
Operating Systems Security

CS 1660: Introduction to Computer Systems Security

Attribution:
Some slides from Tom Doeppner, used with permission

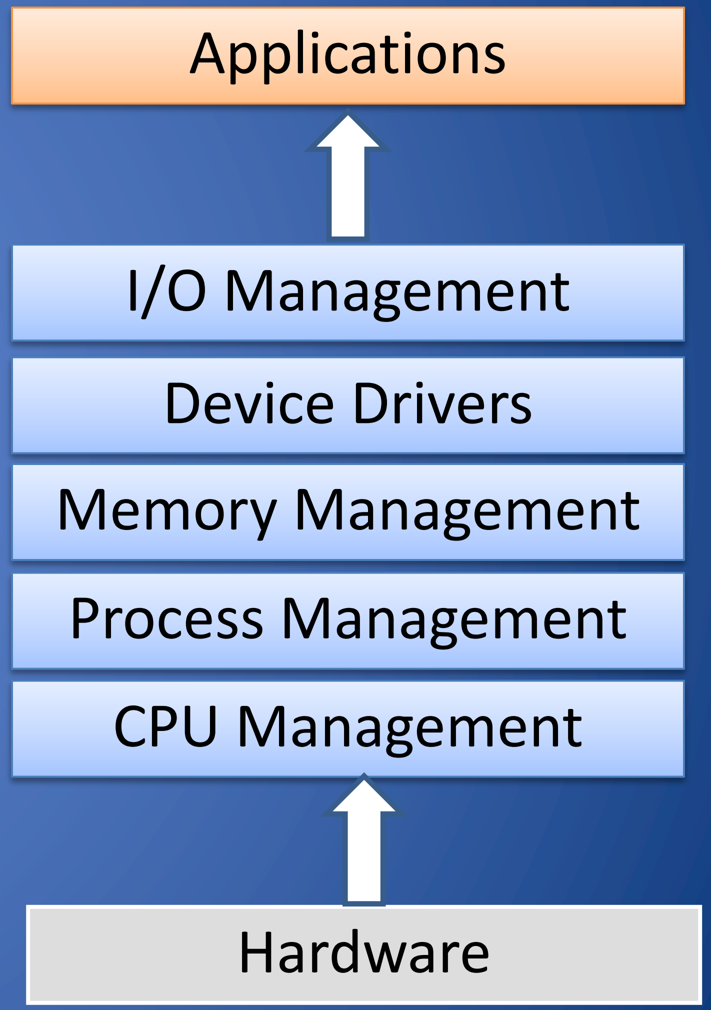

OPERATING SYSTEMS
RUNNING IN MY HOUSE
iOS
MAC OS
ANDROID
TINDEROS
DOS, BUT IRONICALLY
WINDOWS
OS X
NEST
BLOOD DRONE
HUMAN CIVILIZATION ENDS IN FIRE
MS DOS
LINUX
[SOMETHING].JS
ELON MUSK PROJECT
GNU/HURD
1990
2000
2010
NOW
2020
2030
2040
2050
2060

Source: XKCD

Operating System Layers

Many layers of abstraction:

- Kernel: core of the OS, controls hardware, resource access
  - Various subsystems (memory management, networking, storage, …)
- Execution modes:
  - user mode: access to resources mediated by the kernel
  - kernel mode: full and direct access to resources



Processes

The kernel manages applications as processes (or threads)

Every process has:

- Process ID (PID)
- Virtual memory
- Effective user

Processes

The kernel manages applications as processes (or threads)

Every process has:

- Process ID (PID)
- Virtual memory
- Effective user

Kernel provides

- Separate address space from other process
- Time/resource sharing
- Access control

Processes

```
emplisi@ubuntu:~$ pstree
systemd─┬─ModemManager───2*[{ModemManager}]
        ├─NetworkManager───2*[{NetworkManager}]
        ├─VGAuthService
        ├─accounts-daemon───2*[{accounts-daemon}]
        ├─acpid
        ├─anacron───sh───run-parts───mlocate───flock───updated
        ├─avahi-daemon───avahi-daemon
        ├─bluetoothd
        ├─boltd───2*[{boltd}]
        ├─colord───2*[{colord}]
        ├─cron
        ├─cups-browsed───2*[{cups-browsed}]
        ├─cupsd
        ├─dbus-daemon
        ├─firefox─┬─3*[Web Content───18*[{Web Content}]]
        │         ├─Web Content───19*[{Web Content}]
        │         ├─WebExtensions───18*[{WebExtensions}]
        │         ├─file:// Content───18*[{file:// Content}]
        │         └─59*[{firefox}]
        ├─fwupd───4*[{fwupd}]
        ├─gdm3─┬─gdm-session-wor─┬─gdm-wayland-ses─┬─gnome-ses
        │      │                 │                 │
```

Activity Monitor
My Processes

CPU | Memory | Energy | Disk | Network

Search

Process Name	Mem...	Threads	Ports	PID	User
Firefox	10.09 GB	162	1,387	3561	deemer
Virtual Machine Service	7.92 GB	24	83	35580	deemer
FirefoxCP Isolated Web Content	5.19 GB	39	158	3569	deemer
Docker	4.10 GB	42	333	32597	deemer
Microsoft PowerPoint	3.59 GB	42	13,784	3516	deemer
FirefoxCP Isolated Web Content	1.91 GB	33	140	3568	deemer
FirefoxCP Isolated Web Content	1.30 GB	32	136	3567	deemer
FirefoxCP Isolated Web Content	1.30 GB	33	139	3566	deemer
FirefoxCP Isolated Web Content	830.4 MB	29	114	6432	deemer
Preview	773.0 MB	6	1,713	11577	deemer
FirefoxCP Isolated Web Content	659.7 MB	29	115	66660	deemer
Open and Save Panel Service (Preview)	526.2 MB	4	2,596	11578	deemer
Terminal	519.3 MB	7	459	3476	deemer
Finder	457.7 MB	8	1,456	3292	deemer
Discord Helper (Renderer)	420.0 MB	40	805	11208	deemer
Docker Desktop	414.3 MB	29	7,939	32617	deemer
FirefoxCP Isolated Web Content	407.9 MB	29	113	63607	deemer
FirefoxCP Isolated Web Content	405.5 MB	32	137	3576	deemer

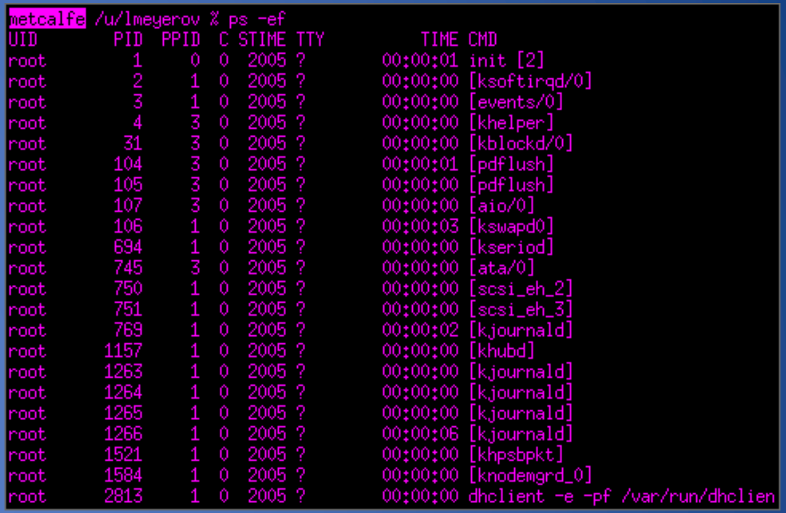
MEMORY PRESSURE

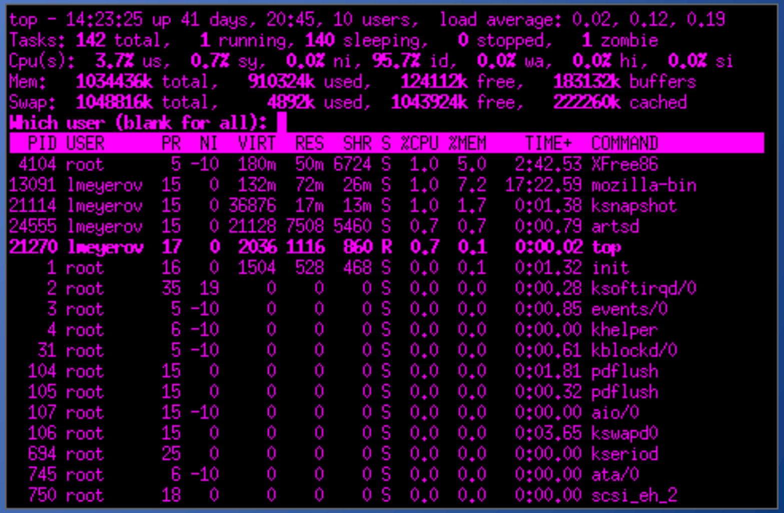
Physical Memory:	32.00 GB
Memory Used:	28.48 GB
Cached Files:	3.46 GB
Swap Used:	8.80 GB

App Memory:	9.95 GB
Wired Memory:	3.00 GB
Compressed:	14.92 GB

View Processes in Linux

- ps: displays snapshot of running processes
  - ps -ef : show all processes
  - ps -u <username>: show processes for a user
- top, htop: fancier list of processes
  - top -u <username>: filter by username
- kill <pid>: terminates a process





Process Management

- Each process has a context, which includes the user, parent process, and address space

- Kernel enforces policies to decide which resources each processes can use

System calls (syscalls)

- Primary way processes interact with kernel
- OS provides a “library” of syscalls for nearly all OS functions
  - Files: read, write, open, close, chmod, …
  - Process management: fork, clone, kill, …
  - Networking: socket, bind, connect

On syscall, process “yields” to kernel, executes in privileged kernel mode

System services (daemons)

- Background process that performs common tasks
- Started at boot time
- Could run with higher permissions than users

Typical services:

- Remote SSH connections
- Web servers
- Logging

AAA: Authentication, Authorization, Accounting

Authorization: how to specify access rights to resources

- To authorize => to define access policy

Users

- Each process is associated with a user
- Specific users can have more privileges than regular users
  - Install or remove programs
  - Change rights of other users
  - Modify the configuration of the system
- Unix: root is a “super-user” with no restrictions

Users

- Each process is associated with a user
- Specific users can have more privileges than regular users

- Unix: root is a “super-user” with no restrictions

How to we manage users?

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 (i.e. confidential, restricted, secret, top secret)
  - A user can create documents with just his level of security

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)
  - DAG (Linux)

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 refers 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
- Examples of ACEs for folder "Bob's CS166 Grades"
  - Bob; Read; Allow
  - TAs; Read; Allow
  - TWD; Read, Write; Allow
  - Bob; Write; Deny
  - TAs; Write; Allow

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

Clicker Question (1)

An ACL with no entries on a file?

A. Access Allowed to all with Open Policy
   Access Allowed to all with Closed Policy

B. Access Denied to all with Open Policy
   Access Allowed to all with Closed Policy

C. Access Allowed to all with Open Policy
   Access Denied to all with Closed Policy

D. Access Denied to all Open Policy
   Access Denied to all Closed Policy

E. It is not possible to realize

Clicker Question (1) - Answer

An ACL with no entries on a file?

A. Access Allowed to all with Open Policy
   Access Allowed to all with Closed Policy
B. Access Denied to all with Open Policy
   Access Allowed to all with Closed Policy
C. Access Allowed to all with Open Policy
   Access Denied to all with Closed Policy
D. Access Denied to all Open Policy
   Access Denied to all Closed Policy
E. It is not possible to realize

Closed Policy with Negative Authorizations and Deny Priority

- Give Tom r/w access to “bar”
- Deny Tom write access to “bar”
- Tom: I would like to read “bar”
  - Access allowed
- Tom: I would like to write “bar”
  - Access denied
- Policy is used by Windows to manage access control to the file system

Role-Based Access Control

- Within an organization roles are created for various job functions
- The permissions to perform certain operations are assigned to specific roles
- Users are assigned particular role, with which they acquire the computer authorizations
- Users are not assigned permissions directly, but only acquire them through their role

U.S. Navy image in the public domain.

Access Control: File System

Linux vs. Windows

- Linux
  - Allow-only ACEs
  - Access to file depends on ACL of file and of all its ancestor folders
  - Start at root of file system
  - Traverse path of folders
  - Each folder must have execute (cd) permission
  - Different paths to same file not equivalent
  - File's ACL must allow requested access
- Windows
  - Allow and deny ACEs
  - By default, deny ACEs precede allow ones
  - Access to file depends only on file's ACL
  - ACLs of ancestors ignored when access is requested
  - Permissions set on a folder usually propagated to descendants (inheritance)
  - System keeps track of inherited ACE's

Linux File Access Control

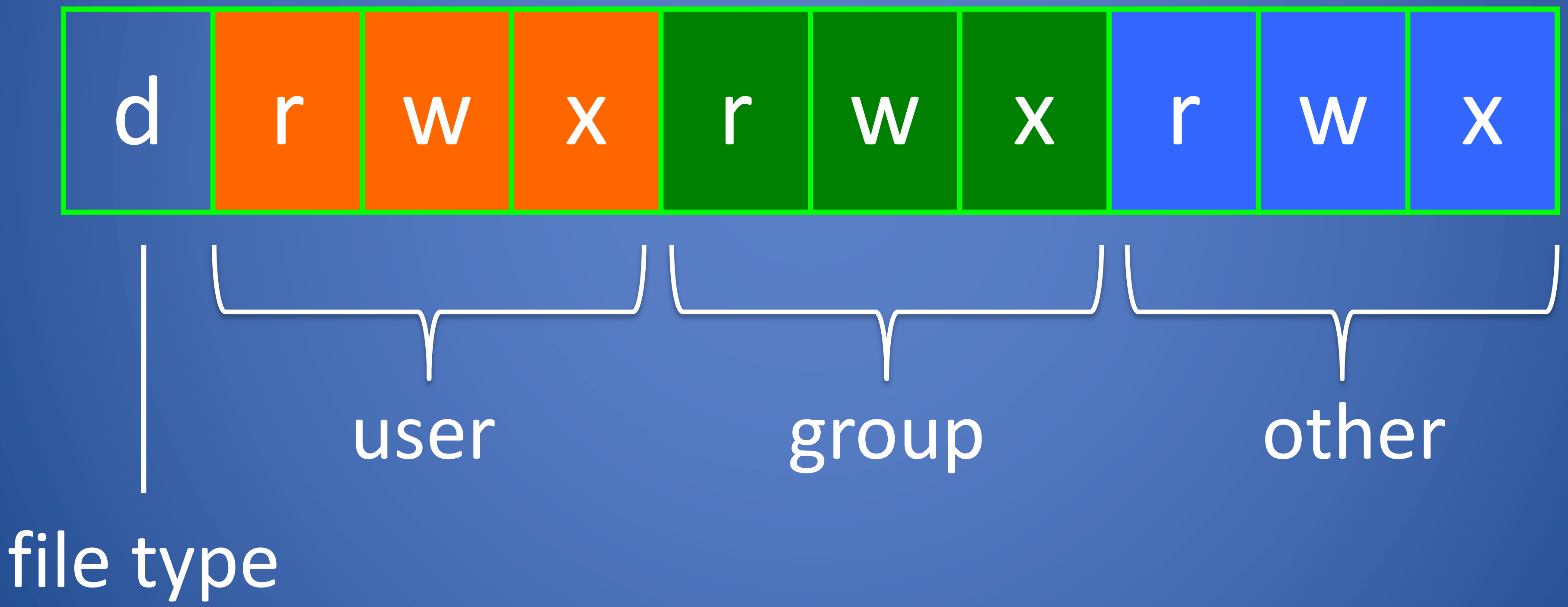
- File Access Control for:
  - Files
  - Directories
  - Therefore…
    - \dev\ : *devices*
    - \mnt\ : *mounted file systems*
    - What else? *Sockets, pipes, symbolic links…*

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
```

Unix File Types and Basic Permissions



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 for Directories

- Permissions bits interpreted differently for directories
- *Read* bit allows listing names of files in directory, but not their properties like size and permissions
- *Write* bit allows creating and deleting files within the directory
- *Execute* bit allows entering the directory and getting properties of files in the directory
- Lines for directories in `ls -l` output begin with d, as below:

```
jk@sphere:~/test$ ls -l
Total 4
drwxr-xr-x  2 jk ugrad 4096 2005-10-13 07:37 dir1
-rw-r--r--  1 jk ugrad    0 2005-10-13 07:18 file1
```

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

Octal Notation Examples

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

Becoming Root

- su
  - Changes home directory, PATH, and shell to that of root, but doesn’t touch most of environment and doesn’t run login scripts
- sudo <command>
  - Run just one command as root
- su [-] <user>
  - Become another non-root user
  - Root does not require to enter password

Examples of Changing Permissions

chown -R root dir1	Changes ownership of dir1 and everything within it to root
chmod g+w,o-rwx file1 file2	Adds group write permission to file1 and file2, denying all access to others
chmod -R g=rwX dir1	Adds group read/write permission to dir1 and everything within it, and group execute permission on files or directories where someone has execute permission
chgrp testgrp file1	Sets file1's group to testgrp, if the user is a member of that group
chmod u+s file1	Sets the setuid bit on file1. (Doesn't change execute bit.)

The /tmp Directory

- In Unix systems, directory /tmp is
  - Readable by any user
  - Writable by any user
  - Usually wiped on reboot
- Convenience
  - Place for temporary files used by applications
  - Files in /tmp are not subject to the user's space quota
- What could go wrong?
  - Sharing of resources may lead to vulnerabilities

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

Setuid Programs

- Unix processes have two user IDs:
  - real user ID: user launching the process
  - effective user ID: user whose privileges are granted to the process
- An executable file can have the set-user-ID property (setuid) enabled
- If a user A executes setuid file owned by B, then the effective user ID of the process is B and not A

Setuid Programs

- System call setuid(uid) allows a process to change its effective user ID to uid
- Some programs that access system resources are owned by root and have the setuid bit set (setuid programs)
  - e.g., passwd and su
- Writing secure setuid programs is tricky because vulnerabilities may be exploited by malicious user actions

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)*

Symbolic Link

- In Unix, a symbolic link (aka symlink) is a file that points to (stores the path of) another file
- A process accessing a symbolic link is transparently redirected to accessing the destination of the symbolic link
- Symbolic links can be chained, but not to form a cycle

- ln -s really_long_directory/even_longer_file_name myfile

Octal Notation

- Standard syntax is nice for simple cases, but bad for complex changes
  - Alternative is octal notation, i.e., 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)

Octal Notation Examples

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

Root

- “root” account is a super-user account, like Administrator on Windows
- Multiple roots possible
- File permissions do not restrict root
- This is *dangerous*, but necessary, and OK with good practices

Becoming Root

- su
  - Changes home directory, PATH, and shell to that of root, but doesn’t touch most of environment and doesn’t run login scripts
- sudo <command>
  - Run just one command as root
- su [-] <user>
  - Become another non-root user
  - Root does not require to enter password

Changing Permissions

- Permissions are changed with chmod or through a GUI like Konqueror
- Only the file owner or root can change permissions
- If a user owns a file, the user can use chgrp to set its group to any group of which the user is a member
- root can change file ownership with chown (and can optionally change group in the same command)
- chown, chmod, and chgrp can take the -R option to recur through subdirectories

Examples of Changing Permissions

chown -R root dir1	Changes ownership of dir1 and everything within it to root
chmod g+w,o-rwx file1 file2	Adds group write permission to file1 and file2, denying all access to others
chmod -R g=rwX dir1	Adds group read/write permission to dir1 and everything within it, and group execute permission on files or directories where someone has execute permission
chgrp testgrp file1	Sets file1’s group to testgrp, if the user is a member of that group
chmod u+s file1	Sets the setuid bit on file1. (Doesn’t change execute bit.)

Limitations of Unix Permissions

- Unix permissions are not perfect
  - Groups are restrictive
  - Limitations on file creation
- Linux optionally uses POSIX ACLs
  - Builds on top of traditional Unix permissions
  - Several users and groups can be named in ACLs, each with different permissions
  - Allows for finer-grained access control
- Each ACL is of the form *type*:[*name*]:*rwx*
  - Setuid, setgid, and sticky bits are outside the ACL system

Gone for Ten Seconds

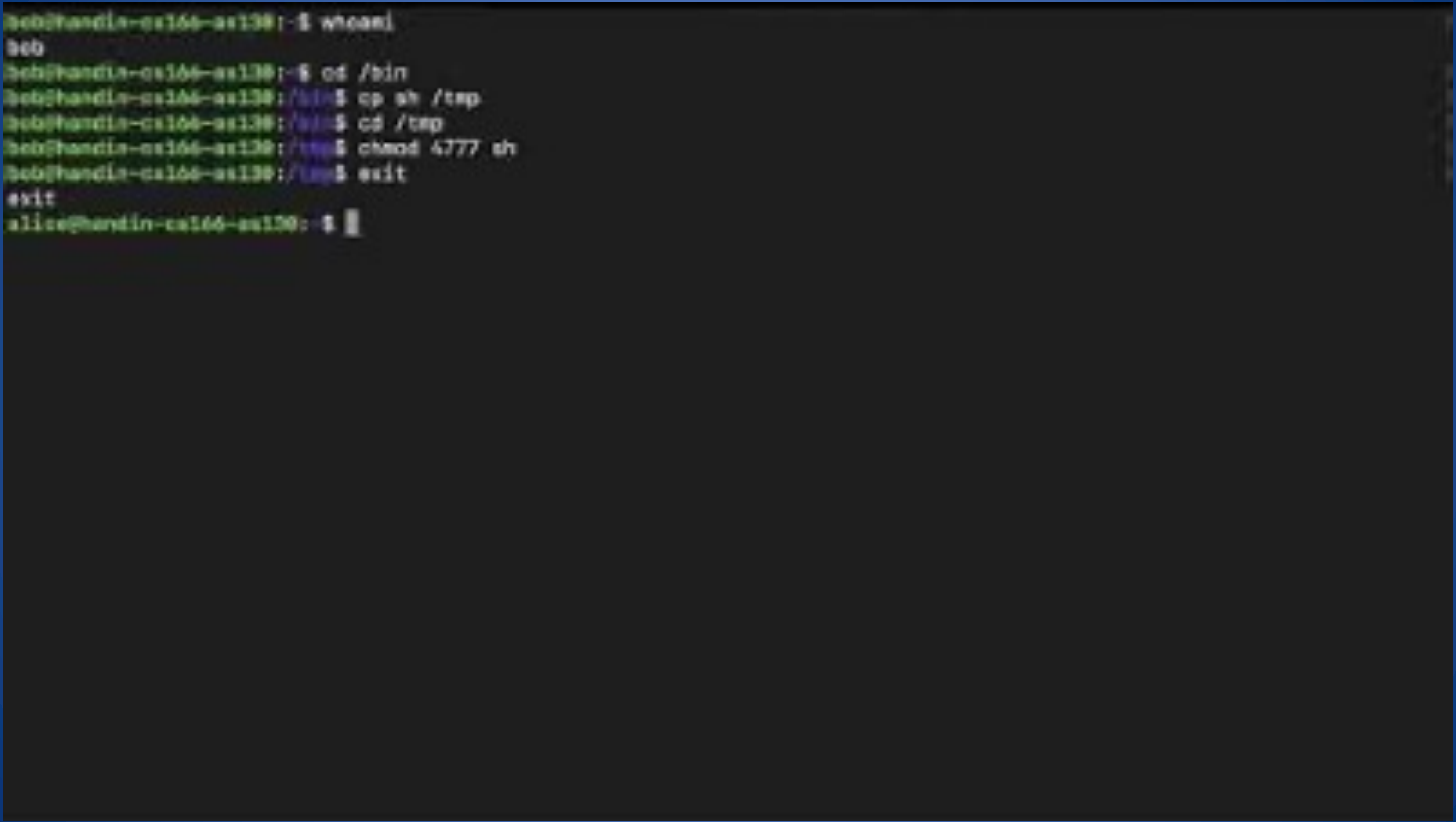
- You leave your desk for 10 seconds without locking your machine
- The attacker sits at your desk and types:

```
% cp /bin/sh /tmp
% chmod 4777 /tmp/sh
```

- The first command makes a copy of shell sh
- The second command makes sh a setuid program

- What happens next?
- The attacker can run the copy of the shell with your privileges
- For example:
  - Can read your files
  - Can change your files

Gone for Ten Seconds: Demo



Historical setuid Unix Vulnerabilities: lpr

- Command lpr
  - running as root setuid
  - copied file to print, or symbolic link to it, to spool file named with 3-digit job number (e.g., print954.spool) in /tmp
  - Did not check if file already existed
  - Random sequence was predictable and repeated after 1,000 times
- How can we exploit this?

- Attack
  - A dangerous combination: setuid, /tmp, symlinks, …
  - Create new password file newpasswd
  - Print a very large file
  - lpr –s /etc/passwd
  - Print a small file 999 times
  - lpr newpasswd
  - The password file is overwritten with newpasswd

Beyond Setuid and Files

- Writing setuid programs is tricky
  - Easy to inadvertently create security vulnerabilities
  - Unix variants have subtle different behaviors in setuid-related calls
- Access control to files is tricky
  - A user file can be accessed by any user process
  - Shared folders and predictable file names create security vulnerabilities

- Consider alternatives
  - Manage system resources via services
  - Use databases instead of files and shared folders
  - Use RPCs (including database queries) to request access to system resources

What We Have Learned

- What is an operating system
- Processes, users, services
- Access control models (DAC and RBAC)
- Setuid programs
- Dangers of symlinks, setuid, and shared directories
  - A demo if you are "Gone for Ten Seconds"