Protection is essential to security

• Protection
  – The **mechanism** that provides controlled access of resources to processes
  – A protection mechanism **enforces** security policies

• Protection includes:
  – User privileges: access rights to files, devices, and other system resources
  – Resource scheduling & allocation
    • Process scheduling & memory allocation – Which processes get priority?
  – Quotas (sometimes) – set limits on disk space, CPU, network, memory, …

• And relies on
  – Mechanisms for user accounts & user authentication – identify who we’re dealing with
  – Policies – defining who should be allowed do what
  – Auditing: generate audit logs for certain events
Co-located resources

• **Earliest computers** – *1945+*
  – Single-user batch processing – no shared resources
  – No need for access control – access control was physical

• **Then … batch processing … but no shared storage** – *1950s*
  – Per-process allocation of tape drives, printers, punched card machines, …

• **Later … shared storage & timesharing systems** – *1960s-now*
  – Multiple users share the same computer
  – User accounts & access control important

• **Even later … PCs** – *1974 to now*
  – Back to single-user systems
  – … but software & media became less trusted by the 1990s

• **Now: networked PCs + mobile devices + IoT devices + …**
  – Shared access: cloud computing, file servers, university systems
  – Need to enforce **access control**
Access control

• Ensure that authorized users can do what they are permitted to do … and no more

• Real world
  – Keys, badges, guards, policies

• Computer world
  – Hardware
  – Operating systems
  – Web servers, databases & other multi-access software
  – Policies
Goals

• **OS Gives us access to resources on a computer:**
  – CPU
  – Memory
  – Files & devices
  – Network

• **We need to:**
  – Protect the operating system from applications
  – Protect applications from each other
  – Allow the OS to stay in control

The OS and hardware are the fundamental parts of the Trusted Computing Base (TCB)
Regaining control: hardware timer

• OS kernel requests timer interrupts

• One of several timer devices:
  – Programmable Interval Timer (PIT)
  – High Precision Event Timer (HPET)
  – or Advanced Programmable Interrupt Controller (APIC timer, one per CPU)

• Most current Intel Linux systems use APIC

• Applications cannot disable this

Ensures that the OS can always regain control
Timer interrupts allow the OS to examine processes while they are running

**OS Process Scheduler**
- Decides whether a process had enough CPU time, and it is time for another process to run
- Prioritizes threads
  - Based on user, user-defined priorities, interactivity, deadlines, “fairness”
  - One process should not adversely affect others
- Avoid **starvation**: ensure all processes will get a chance to run
  - This would be an **availability** attack
Memory Protection: Memory Management Unit

- All modern CPUs have a Memory Management Unit (MMU)
- OS provides each process with virtual memory
- Gives each process the illusion that it has the entire address space
- One process cannot see another process’ address space
- Enforce memory access rights
  - Read-only (code)
  - Read-write (program’s data)
  - Execute (code)
  - Unmapped
Page translation

Virtual memory address

Page number, $p$

Displacement (offset), $d$

$f = page\_table[p]$
Logical vs. physical views of memory

Logical Memory – Process 0

Logical Memory – Process 1

Page Table 0

Page Table 1

Physical Memory

Page 3
page 2
page 1
page 0

Page 3
page 2
page 1
page 0

Page 3
page 2
page 1
page 0

Page 3
page 2
page 1
page 0

Page 3
page 2
page 1
page 0

Page 3
page 2
page 1
page 0

Page 3
page 2
page 1
page 0

Page 3
page 2
page 1
page 0
User & kernel mode

**Kernel mode** = privileged, system, or supervisor mode
- Access restricted regions of memory
- Modify the memory management unit by changing the page table register
- Set timers
- Define interrupt vectors
- Halt the processor
- Etc.

**Getting into kernel mode**
- **Trap**: explicit instruction
  - Intel architecture: *INT* instruction (interrupt)
  - ARM architecture: *SWI* instruction (software interrupt)
  - System call instructions
- **Violation** (e.g., access unmapped memory, illegal instruction)
- Hardware **interrupt** (e.g., receipt of network data or timer)
Protection Rings

• All modern operating systems support two modes of operation: user & kernel

• Multics defined a ring structure with 6 different privilege levels
  – Each ring is protected from higher numbered rings
  – Special call (call gates) to cross rings: jump to predefined locations
  – Most of the OS did not run in ring 0

• Intel x86, IA-32 and IA-64 support 4 rings

• Today’s OSes only use
  – Ring 0: kernel
  – Ring 3: user

Note: hypervisors (virtual machine monitors) run at a 3rd privilege level
  – In many systems, this is ring -1 for the hypervisor, 0 for the kernel and 3 for user programs

https://en.wikipedia.org/wiki/Protection_ring
Subjects, Principals, and Objects

**Subject:** the thing that needs to access resources

- **Principal:** unique identity for a user
  - Subjects may have multiple identities and be associated with a set of principals

- **User:** a human (generally)

**Object:** the resource the subject may access

- Typically, files and devices – they do not perform operations

**Subjects access objects:** they perform actions on objects

**Access control**

- Define what operations subjects can perform on objects

**Most operating systems control who can do what to each object**
  (permissions are associated with each object)
User authentication

Must be done before we can do access control

- Establish user identity – determine the *subject*
  - Operating system privileges are granted based on user identity

Steps

1. Get user credentials (e.g., name, password)
2. Authenticate user by validating the credentials
   - Get user ID(s), group ID(s)
3. Access control: grant further access based on user ID
Domains of Protection
Domains of protection

• **Subjects** (users running processes) interact with **objects**
  – Process runs with the authority of the subject (user)
  – Objects include:
    hardware (CPU, memory, I/O devices)
    software: files, processes, semaphores, messages, signals

• A process should be allowed to access only objects that it is authorized to access
  – A process operates in a **protection domain**
  – It’s part of the **context of the process**
  – Protection domain **defines the objects the process may access** and how it may access them
Modeling Protection: Access Control Matrix

Rows: **domains**
(subjects or groups of subjects)

Columns: **objects**

Each entry in the matrix represents an access right of a domain on an object.

An **Access Control Matrix** is the primary abstraction for protection in computer security.
We may need some more controls

- **Domain transfers**
  - Allow a process to run under another domain’s permissions

- **Copy rights**
  - Allow a user to grant certain access rights for an object

- **Owner rights**
  - Identify a subject as the owner of an object
  - Can change access rights on that object for any domain

- **Domain control**
  - A process running in one domain can change any access rights for another domain
Switching from one domain to another is a configurable policy

**Domain transfers**
Allow a process to run under another domain’s permissions

*Why?* Log a user in – how would you run the first user’s process?

<table>
<thead>
<tr>
<th>Subjects</th>
<th>F0</th>
<th>F1</th>
<th>Printer</th>
<th>D0</th>
<th>D1</th>
<th>D2</th>
<th>D3</th>
<th>D4</th>
</tr>
</thead>
<tbody>
<tr>
<td>D0</td>
<td>read</td>
<td>read-write</td>
<td>print</td>
<td>–</td>
<td>switch</td>
<td>switch</td>
<td></td>
<td></td>
</tr>
<tr>
<td>D1</td>
<td>read-write-execute</td>
<td>read</td>
<td>–</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D2</td>
<td>read-execute</td>
<td></td>
<td>switch</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D3</td>
<td>read</td>
<td>print</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D4</td>
<td></td>
<td>print</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

A process in $D_0$ can switch to running in domain $D_1$.
Copy rights: allow a user to grant certain rights to others

- Copy a specific access right on an object from one domain to another

<table>
<thead>
<tr>
<th>Subjects</th>
<th>F₀</th>
<th>F₁</th>
<th>Printer</th>
<th>D₀</th>
<th>D₁</th>
<th>D₂</th>
<th>D₃</th>
<th>D₄</th>
</tr>
</thead>
<tbody>
<tr>
<td>D₀</td>
<td>read</td>
<td>read-write</td>
<td>print</td>
<td>–</td>
<td>switch</td>
<td>switch</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>D₁</td>
<td>read-write</td>
<td>read*</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D₂</td>
<td>read-execute</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D₃</td>
<td>read</td>
<td>print</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D₄</td>
<td></td>
<td>print</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

A process executing in D₁ can give a read right on F₁ to another domain.
# Access Control Matrix: Object Owner

**Owner:** allow new rights to be added or removed

Identify a subject as the owner of an object
Can change access rights on that object for any domain (column)

<table>
<thead>
<tr>
<th>Subjects domains of protection</th>
<th>F₀</th>
<th>F₁</th>
<th>Printer</th>
<th>D₀</th>
<th>D₁</th>
<th>D₂</th>
<th>D₃</th>
<th>D₄</th>
</tr>
</thead>
<tbody>
<tr>
<td>D₀ read owner</td>
<td>read</td>
<td>write</td>
<td>print</td>
<td>–</td>
<td>switch</td>
<td>swtich</td>
<td></td>
<td></td>
</tr>
<tr>
<td>D₁ read-write-execute</td>
<td>read*</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D₂ read-execute</td>
<td>read</td>
<td></td>
<td></td>
<td></td>
<td>swtich</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D₃</td>
<td>read</td>
<td>print</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D₄</td>
<td></td>
<td>print</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

A process executing in $D₀$ owns $F₀$, so it can give a *read* right on $F₀$ to domain $D₃$ and remove the *execute* right from $D₁$. 

---

*January 30, 2022 CS 419 © 2022 Paul Krzyzanowski*
Access Matrix: Domain Control

- A process running in one domain can change any access rights for another domain
- Change entries in a row (all objects)

<table>
<thead>
<tr>
<th>Subjects</th>
<th>F₀</th>
<th>F₁</th>
<th>Printer</th>
<th>D₀</th>
<th>D₁</th>
<th>D₂</th>
<th>D₃</th>
<th>D₄</th>
</tr>
</thead>
<tbody>
<tr>
<td>D₀</td>
<td>read owner</td>
<td>read-write</td>
<td>print</td>
<td>–</td>
<td>switch</td>
<td>switch</td>
<td></td>
<td></td>
</tr>
<tr>
<td>D₁</td>
<td>read-write-execute</td>
<td>read*</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>control</td>
<td></td>
</tr>
<tr>
<td>D₂</td>
<td>read-execute</td>
<td>–</td>
<td>switch</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D₃</td>
<td>read</td>
<td>print</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D₄</td>
<td>print</td>
<td>–</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

A process executing in D₇ can modify any rights in domain D₄
This gets messy!

• An access control matrix does not address everything we may want

• Processes execute with the rights of the user (domain)
  – But sometimes they need extra privileges
    • Read configuration files
    • Read/write from/to a restricted device
    • Append to a queue

• We don’t want the user to be able to access these objects
  – Adding domains to the row of objects is not sufficient
  – We may need a 3-D access control matrix: (subjects, objects, processes)

• This gets messy!
  – One solution is to give an executable file a temporary domain transfer
    • Assumption is this is a trusted application that can access these resources
    – When run, it assumes the privileges of another domain
Implementing an access matrix

A single table to store an access matrix is impractical

• Big size: # domains (users) × # objects (files)
• Objects may come and go frequently
• Lookup needs to be efficient
Implementing an access matrix

Access Control List

- Associate a column of the table with each object

<table>
<thead>
<tr>
<th>Subjects</th>
<th>D0</th>
<th>D1</th>
<th>D2</th>
<th>D3</th>
<th>D4</th>
</tr>
</thead>
<tbody>
<tr>
<td>domains of protection</td>
<td>read owner</td>
<td>read-write-execute</td>
<td>read-execute</td>
<td>read-execute</td>
<td>read-execute</td>
</tr>
<tr>
<td>F0</td>
<td>read</td>
<td>read-write</td>
<td>read</td>
<td></td>
<td></td>
</tr>
<tr>
<td>F1</td>
<td>read-write</td>
<td>read</td>
<td>read</td>
<td></td>
<td></td>
</tr>
<tr>
<td>F2</td>
<td>read</td>
<td>read-execute</td>
<td>read</td>
<td></td>
<td></td>
</tr>
<tr>
<td>F3</td>
<td>read</td>
<td>read-execute</td>
<td>write</td>
<td></td>
<td></td>
</tr>
<tr>
<td>F4</td>
<td>read</td>
<td>read-execute</td>
<td>write</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Printer</td>
<td>print</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Capability List

- Associate a row of the table with each domain

<table>
<thead>
<tr>
<th>Subjects</th>
<th>D₀</th>
<th>D₁</th>
<th>D₂</th>
<th>D₃</th>
<th>D₄</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>F₀</td>
<td>F₁</td>
<td>F₂</td>
<td>F₃</td>
<td>F₃</td>
</tr>
<tr>
<td>D₀</td>
<td>read owner</td>
<td>read-write</td>
<td>read execute</td>
<td>read</td>
<td></td>
</tr>
<tr>
<td>D₁</td>
<td>read-write-execute</td>
<td>read</td>
<td>read execute</td>
<td>read</td>
<td>write</td>
</tr>
<tr>
<td>D₂</td>
<td>read-execute</td>
<td></td>
<td>read-execute</td>
<td></td>
<td>write</td>
</tr>
<tr>
<td>D₃</td>
<td></td>
<td>read-execute</td>
<td></td>
<td></td>
<td>print</td>
</tr>
<tr>
<td>D₄</td>
<td></td>
<td>read-execute</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Capability list for domain D₁**
Capability Lists

**Capability list** = list of objects together with the operations a specific subject can perform on the objects

- Each item in the list is a *capability*: the operations allowed on a specific object
  - Also known as a *ticket* or *access token*

- A process presents the capability to the OS along with a request
  - Possessing the capability means that access is allowed

- The capability is a protected object
  - A process cannot modify its capability list
Capability Lists

• **Advantages**
  – Run-time checking is more efficient
  – Delegating rights is easy

• **Disadvantages**
  – Creating or deleting files means updating all capability lists
  – Changing a file’s permissions is hard
  – Hard to find all users that have access to a resource
  – Lists can be huge – the system might have millions of objects

• **Not used in mainstream systems in place of ACLs**
  – Limited implementations: Cambridge CAP, IBM AS/400

• **Capability lists are rarely used but capabilities are used**
  – Used in single sign-on services and other authorization services such as OAuth and Kerberos (sort of)
  – **Access Tokens**
    • Identifies a user’s identity and the access rights permitted on the requested service (not objects!)
The End