Access Control / Authorization

- Controlling which subject (or principal) has what kind of access to what objects (or resources)

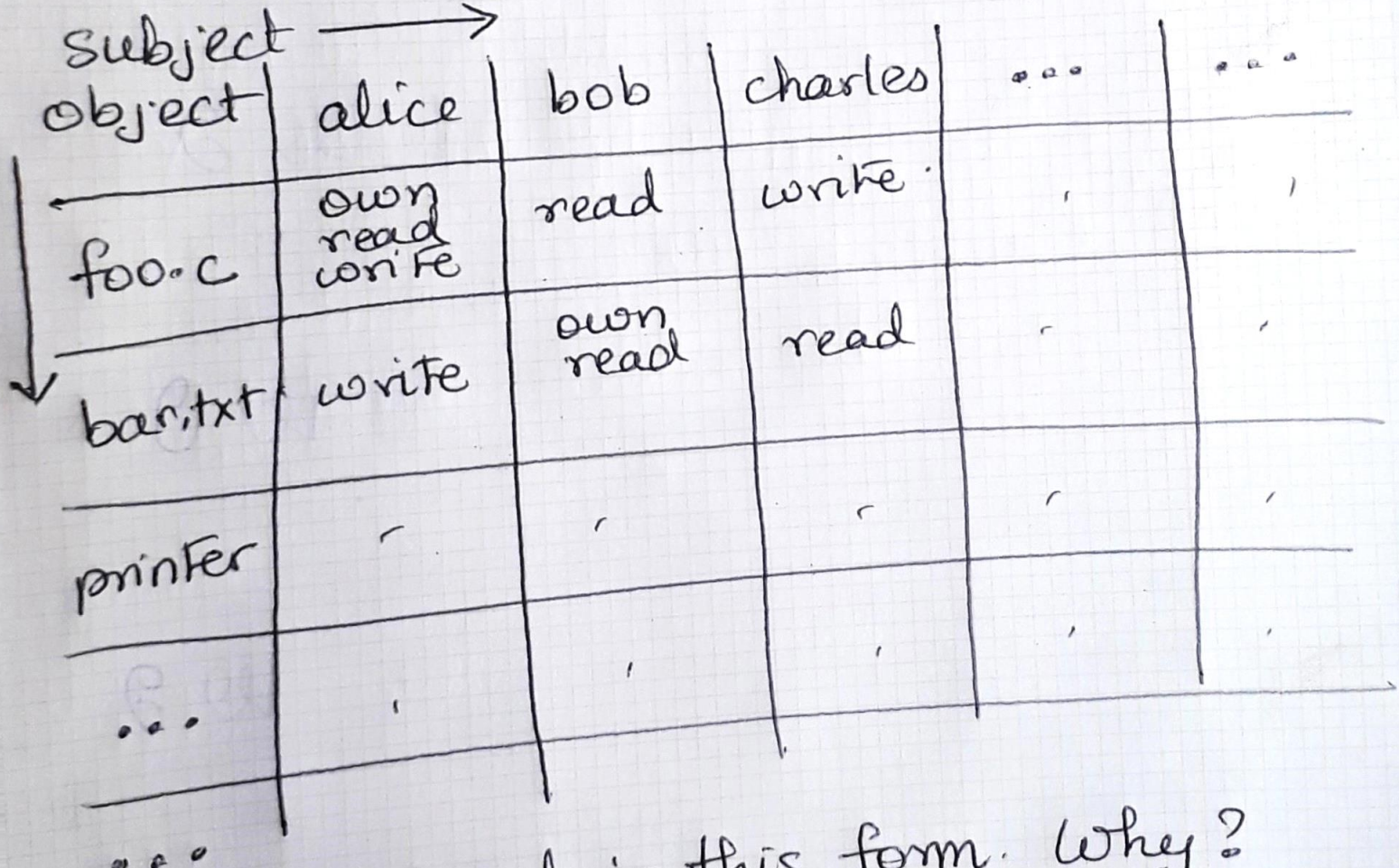
- Two kinds of access control
  - → Mandatory Access Control (MAC)
  - → Discretionary Access Control (DAC)

- MAC → system-wide policies, typically set by sysadmin & enforced on all users.
  (e.g., SELinux, SEAndroid & also Bell-Lapadula, Biba)

- DAC → set by resource owner
  (e.g., rwx perms on Linux)

Be it MAC or DAC, we need an enforcement mechanism.

That mechanism is an access control matrix



Rarely stored in this form. Why?

Two ways of "sharding" this access control matrix:

- By row: store perms with the object

  ACCESS CONTROL LISTS (ACL)

  rwx rwx rwx foo.txt.
  self group world.

- By column: store perms with the subject

  CAPABILITIES.

  We have seen an example of capabilities in class. Any guesses?

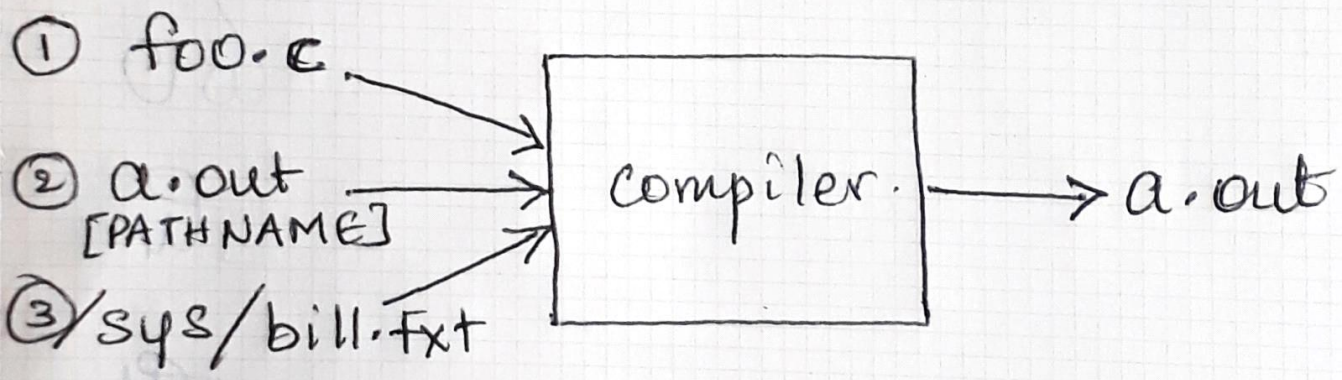
  - ownership of the capability grants access rights to the holder.
  - Must be non-forgeable.

  We will revisit capabilities in Web security as well

Both ACL and capabilities are equal in expressive power, Yet, there are certain advantages to capabilities.

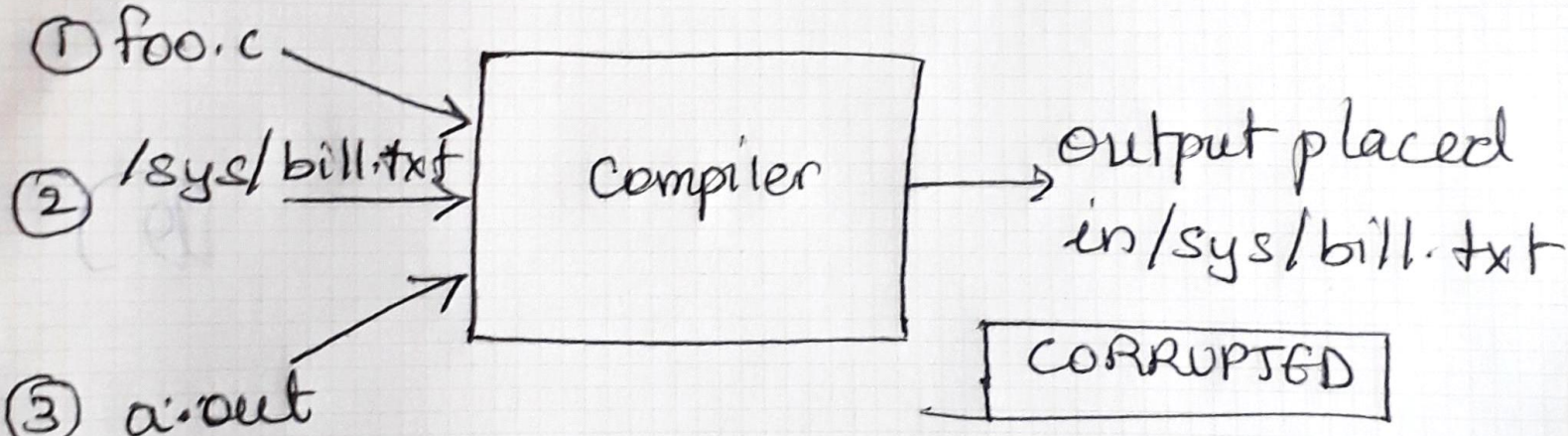
"The confused deputy problem" Noam Hardy.

[THE SETTING FOR HARDY'S PAPER]



- Compiler puts output of compilation in a.out [PATHNAME given as an input].
- Places bill in /sys/bill.txt.
- Compiler must have privileges to write to a.out (file owner) and sys/bill.txt.

- /sys/bill.txt requires special (admin) privileges to write to.
- Compiler must have both sets of privileges
  → Remember setuid?
- Can a malicious user trick compiler and force it to corrupt /sys/bill.txt,



- How can we solve this problem?
- The compiler is a confused deputy

CONFUSED DEPUTY? ⇒  CAPABILITIES!

- Why is it a confused deputy?
  → It has 2 sets of access rights
  [access to write to a.out in ~/
  and access to write to /sys/bill.txt].
  → It does not know when to use which permission.
- The set of perms this confused deputy has is its AMBIENT AUTHORITY.
- Attacker misused this ambient authority.
- Solution → Use Capabilities!
  For example, file descriptors

Here is how it works:

→ Strip the compiler of all its access rights.
i.e., it cannot open anything on its own.

→ Invoker of the program (i.e. invoker of compiler) opens a.out for writing foo.c for reading and passes these file descriptors to the compiler.

→ Root/Admin opens /sys/bill.txt for appending and passes it to compiler.

→ Compiler or invoker themselves cannot access /sys/bill.txt with open. Any such accesses will fail.

Compiler has no ambient authority.
Invoker or admin are obtaining required capabilities and handing it over to compiler, thus "blessing" it to do those ops.

A second example of everyday use of capabilities vs. ACLs.

- $ cat input.txt.
  - ↳ Here, ~~the shell~~ cat opens input.txt for reading

- $ cat <input.txt >output.txt.
  - ↳ Here, the shell opens input.txt for reading, output.txt for writing and passes these capabilities (ie fds) to cat. Cat does not open the files on its own.

So far, we have considered access control, but not how the information flows once it is accessed.

Can we control how information is used?

- Two very influential policy models for info flow.
- Developed in the era of multi-level access control
- Modern incarnations in SELinux & SEAndroid
- Policy models:
  - ↳ Bell LaPadula (BLP) – confidentiality
  - ↳ Biba – for integrity.

We will look at BLP first.

Basic setup:

- Files have confidentiality levels that determine how sensitive they are.
- Subjects have clearances that say what kind of data they can access.

Example:

(clearances)	Users	Files	(level)
(secret)	SUNDAR	HIRINGFIRING.txt	(secret)
(public)	PICHAI	LUNCHMENU.txt	(public)

Assume you want to protect confidentiality of secret files from public. How?

determine who can read & who can write

Read

	HiringFiring	Lunchmenu
Sundar.	✓	✓
Pichai	X ←	✓

"No Read up"

Write

	Hiring Firing	Lunchmenu.
Sundar	✓	X ←
Pichai	✓	✓

No write down.

Bottom left entry (Pichai/Hiring Firing) is counter-intuitive. Why?

→ This is a confidentiality model we haven't yet discussed in Fegrity

> Bell LaPadula = "No Reads up"
> "No Writes down"

Biba – an integrity policy model.

Similar, yet dual concept.

Subjects & objects have integrity **labels** & **clearances**

USERS		FILES	
High	SUNDAR	ACCOUNTS.XLS	High
Low	PICHAI	LUNCHMENU.txt	Low

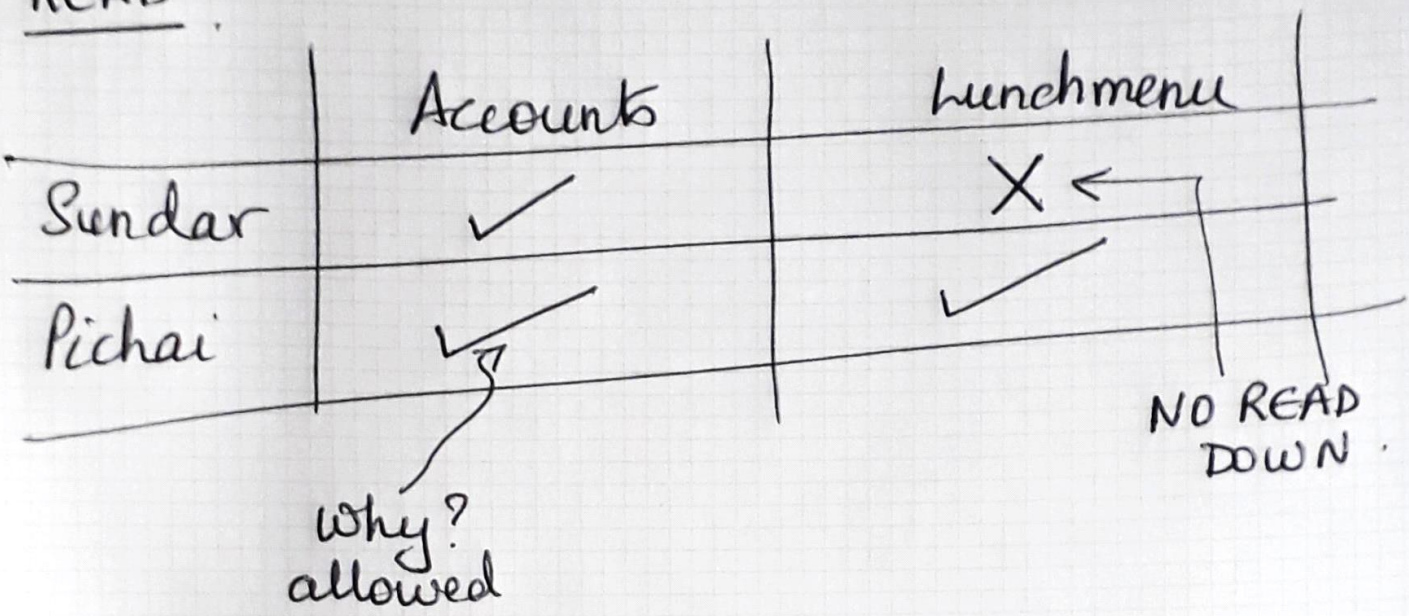
Again, study who can read & who can write.

~~READ~~ WRITE

	Accounts	lunchmenu
Sundar	✓	✓
Pichai	✗	✓

"No writes up"

READ



Thus, Biba Integrity is
" No writes up, No reads down"

So, aren't confidentiality and integrity at odds? Can we achieve both in the same system?

Yes. In real systems, there are 2 sets of labels, a confidentiality label & an integrity label. For conf labels, BLP is used. For int labels, Biba is used.