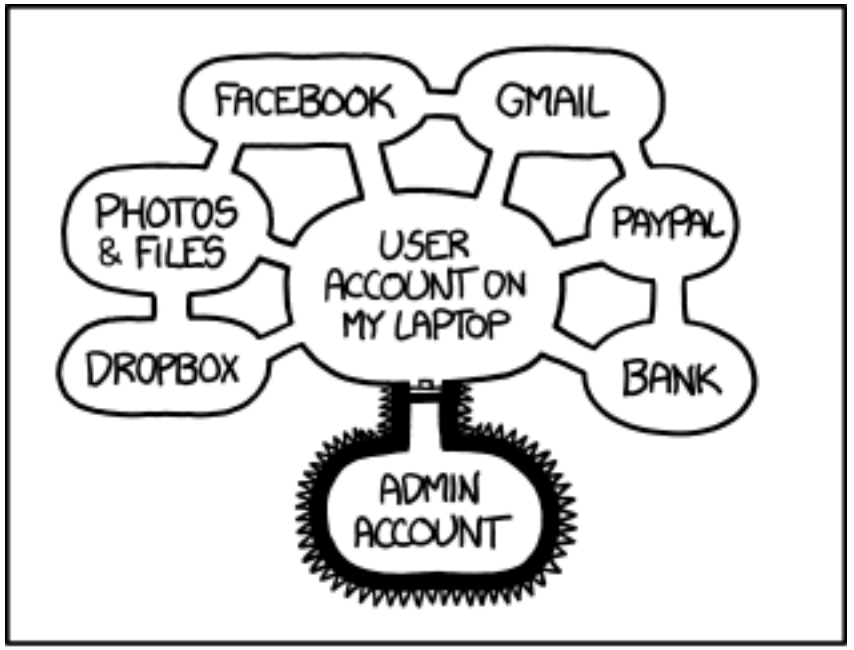
Protection

Kevin Webb

Swarthmore College

April 23, 2024

[xkcd #1200]



Before you say anything, no, I know not to leave my computer sitting out logged in to all my accounts. I have it set up so after a few minutes of inactivity it automatically switches to my brother's.

Today’s Goals

- Meaning of ‘protection’, and contrast with ‘security’.
- Models for representing protection in an OS.
- Real examples in Unix for non-trivial file protection.

Protection

- Protection: a mechanism for controlling access to the resources provided by a computer system.

- Why is protection needed?
  - Prevent unauthorized users from accessing data / resources.
  - Prevent buggy programs from wreaking havoc on the system (e.g., memory).

Protection vs. Security

Protection

- Mechanisms
  - Bits set? You can access.
  - Simple(r) expression.
- Users play by the rules.
- The system is implemented correctly.

Security

- Policies
  - Only the members of the development team can read this file, and only senior developers and managers can edit it.
- Users are evil and untrusted!
- The system has flaws, and users will exploit them!

OS Kernel Enforces Protection

- To protect resources, let OS “own” them
  - OS can allow access to other actors (temporarily).
- To access a resource, a process must ask for it
  - OS can test whether access should be given.
- Once a process is given access…
  - OS can prevent others for gaining access (mutual exclusion).
  - OS may or may not be able to take away access (revocation).
- This assumes the kernel operates correctly.

Protecting the Kernel

- The kernel itself must be protected! We've seen this stuff already!

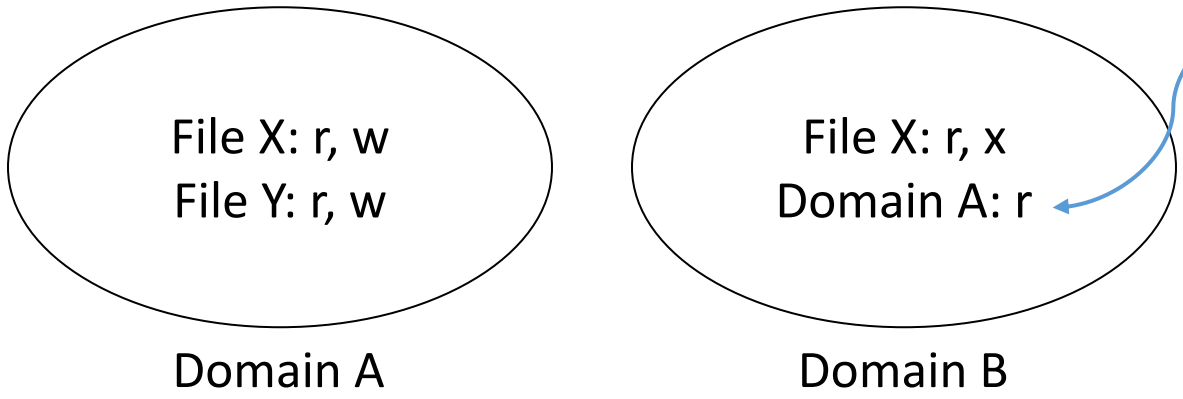
- Mechanisms:
  - Memory protection.
  - Protected mode of operation: kernel vs. user and CPU rings.
  - Clock interrupt, so kernel eventually gets control back from processes.

- Note: these mechanisms are all hardware supported.

Thought experiment…

- You have a room with a door.
- You want to protect the room so that only certain people can enter.
- How might you do that? What mechanisms might you employ?
- Things to consider:
  - How difficult will it be to verify if someone is / isn't allowed to enter?
  - How difficult will be to grant / revoke entry permission? How often will that happen?
  - Can we formalize the idea of protecting the door?

A Formal Model of Protection

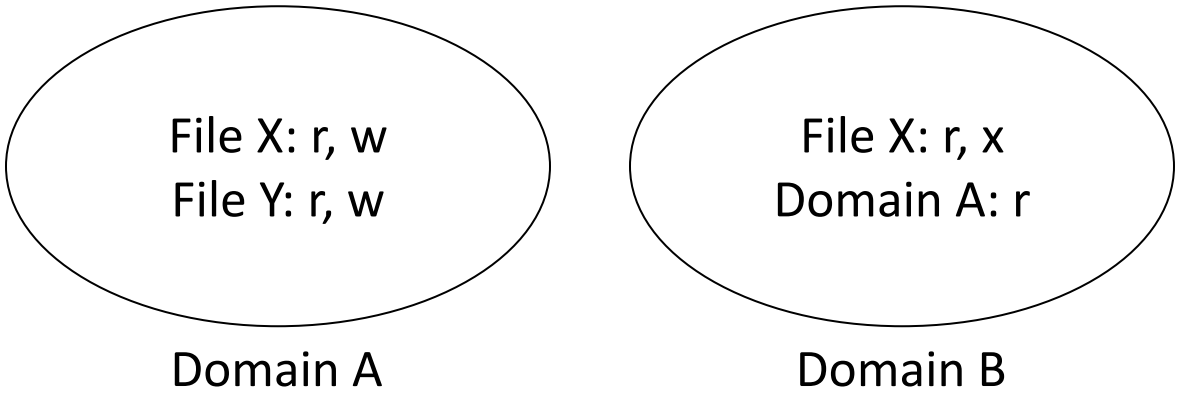


Note: Domains can also be resources!

This says that Domain B can read Domain A's protection settings.

- Domain: identity of an actor in the system.
  - Could be a user, process, procedure.
  - Contains a set of (resource, permission) pairs.
- Resource: object (device, file, data) that requires protection.
- For now, let's assume that a process executes within the context of a protection domain.

A Formal Model of Protection



- Protection goal: if a process requests access to a resource, check the domain it's operating within to see if access is allowed.

- We need: some way to represent domains and their permissions.

Protection Matrix

Resources

Domains	X	Y	A	B
A	r, w	r, w		
B	r, x		r	

- Possibility: describe all permissions as a matrix.
  - Rows are domains
  - Columns are resources
  - Matrix entry [d, r] contains permissions/rights

Is there a problem with storing permissions in a matrix? (e.g., correctness? other concerns?)

A. Yes – Why?

B. No – Why not?

C. It depends. On what?

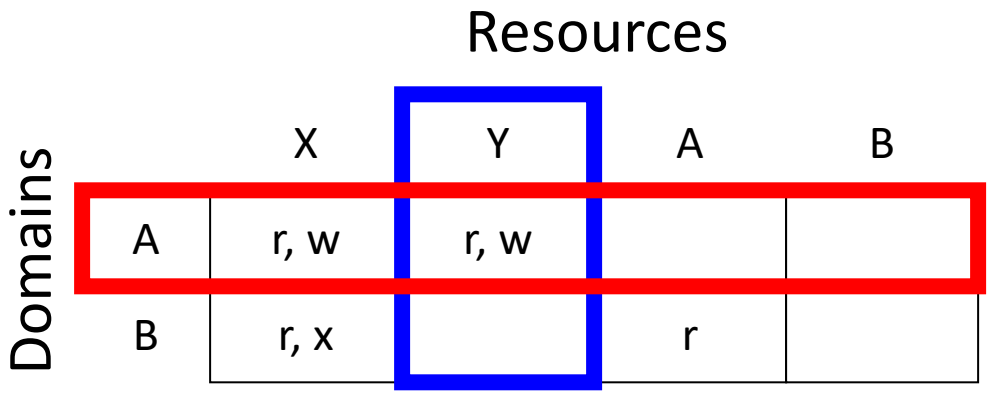
Resources

Domains	X	Y	A	B
A	r, w	r, w		
B	r, x		r	

Efficient Representations

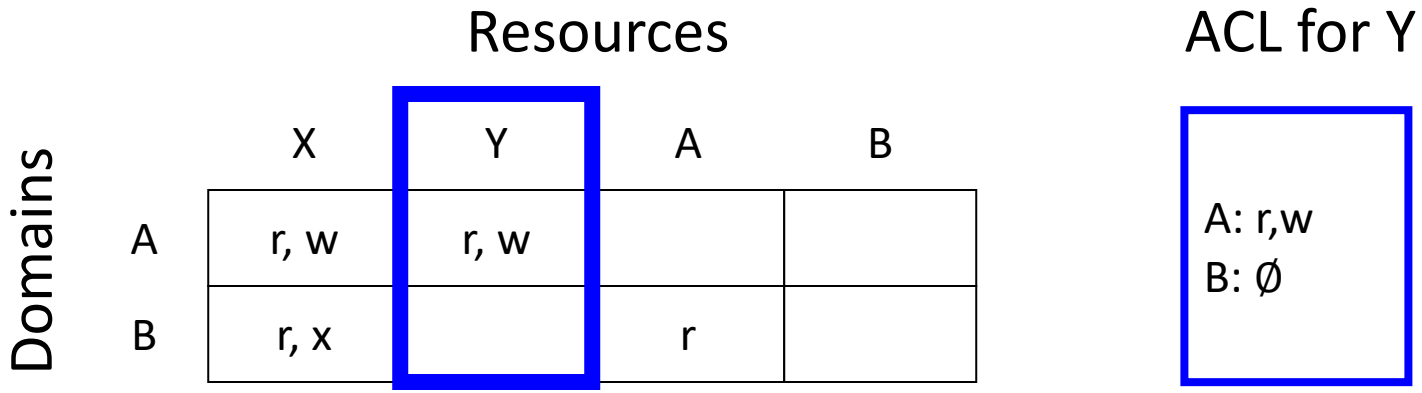
- Matrix idea is too costly
- What should our implementation provide?
  - Low storage overhead
  - Manageability
    - Add permission easily
    - Revoke permission easily
  - Performance
    - Verify permission quickly

Efficient Representations



- Access Control Lists
  - For each resource, list (domain, permissions) pairs
- Capability Lists
  - For each domain, list (resource, permissions) pairs

Access Control Lists (ACLs)



- ACL is associated with **resource**.
- When process tries to access, check for its domain on the list.

- Analogy: bouncer at the door checking names on a list.
  If you re-enter, you need to get checked again.

How do we think ACL's perform? Why?

Resources

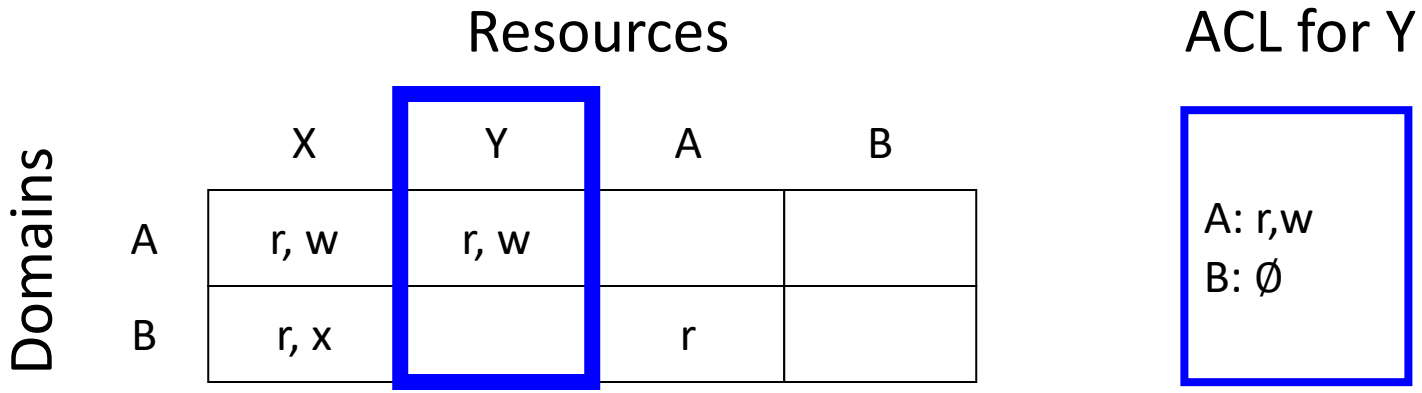
Domains	X	Y	A	B
A	r, w	r, w		
B	r, x		r	

ACL for Y

A: r,w
B: Ø

Answer Choice	Manageability	Performance
A	Easy to add/revoke permissions	Quick to verify
B	Easy to add/revoke permissions	Slow to verify
C	Hard to add/revoke permissions	Quick to verify
D	Hard to add/revoke permissions	Slow to verify

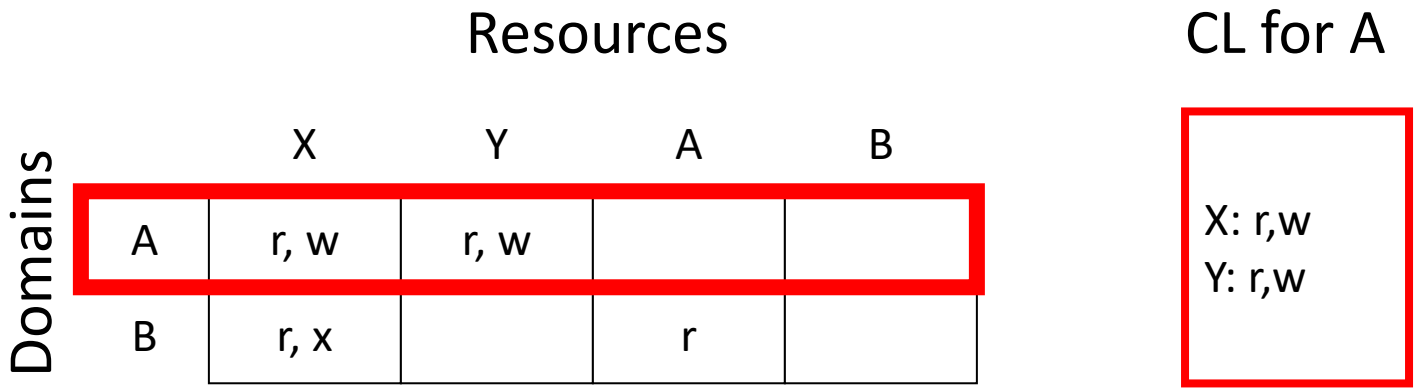
Access Control Lists



- ACL is associated with **resource**.
- When process tries to access, check for its domain on the list.

- Can be inefficient: must lookup on each access.
- Revocation is easy, just remove from list.

Capability Lists



CL for A

X: r,w
Y: r,w

- Capability list associated with each domain.
- When process tries to access, validate that is has the capability.

- Analogy: process gets a key it can “present” to verify access.

How do we think capabilities perform? Why?

Resources

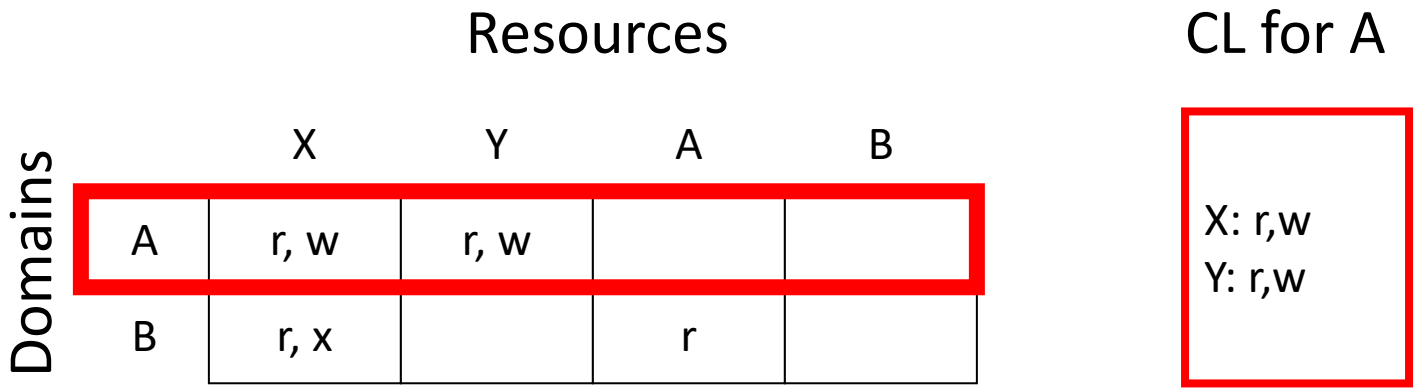
Domains	X	Y	A	B
A	r, w	r, w		
B	r, x		r	

CL for A

X: r,w
Y: r,w

Answer Choice	Manageability	Performance
A	Easy to add/revoke permissions	Quick to verify
B	Easy to add/revoke permissions	Slow to verify
C	Hard to add/revoke permissions	Quick to verify
D	Hard to add/revoke permissions	Slow to verify

Capability Lists



- Capability list associated with each domain.
- When process tries to access, validate that is has the capability.

- Efficient: on access, just produce capability.
- Difficult to revoke.

Which protection mechanism would you expect to find in a modern OS? Which would you use? Why?

A. Capabilities

B. Access control lists

C. Both

D. Neither (some other mechanism)

Concrete Examples

- When you open a file, you get back a file descriptor. If you get a descriptor, you can access the file, even if the file is changed:
  - The file's access permissions change.
  - The file gets renamed / deleted.
  - The FD is a capability!

- At the time you open the file, the permission checks are done. On Unix systems, this is an ACL check.

Unix File Permissions

- Every file (regular, directory, FIFO, link, etc.) has three sets of bits:
  - What can the file's **owner** do with the file (exactly one owner)?
  - What can users in the file's **group** do with the file? (groups contain many users)
  - What can everyone else in the **world** do with the file?

- Examples, from my home dir:

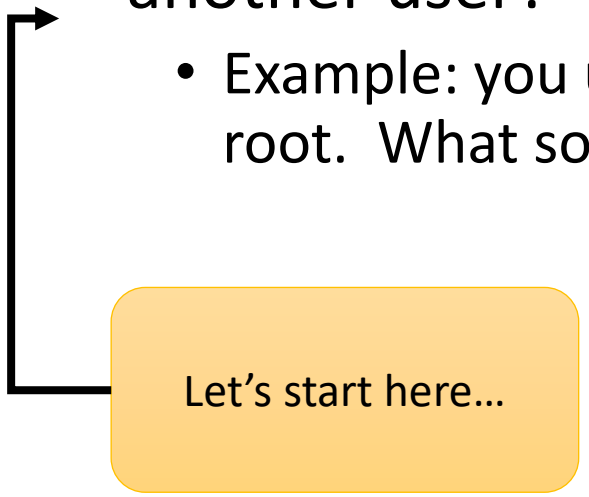
```
drwxr-xr-x   8 kwebb users        4096 Apr  2 14:20 public/
drwxr-xr-x  14 kwebb users        4096 Mar 27 17:00 public_html/
```

```
drwx------  15 kwebb users        4096 Dec  4 11:49 Teaching/
```

What about…?

- What if we want one group of users to be able to read/write, and another group to be read-only, and everyone else gets no access?

- What if we want a program to execute with the permissions of another user?
  - Example: you use ‘sudo’ to execute commands as the root user, but you’re not root. What sorcery is sudo using to make that happen?



The setuid / setgid bits.

- The Unix file permissions model has more bits:
  - setuid: when executing this program, inherit the owner's permissions
  - setgid: when executing this program, inherit the group's permissions
  - sticky bit: only the owner can rename/delete the file, even if others have write access (e.g., through group permissions)

- 'sudo' is setuid to the 'root' user:

```
-rwsr-xr-x 1 root root 136808 Jul  4  2017 /usr/bin/sudo*
```

Anything sudo does is happening with elevated privileges. It better be careful when checking authorized users!

What about…?

- What if we want one group of users to be able to read/write, and another group to be read-only, and everyone else gets no access?

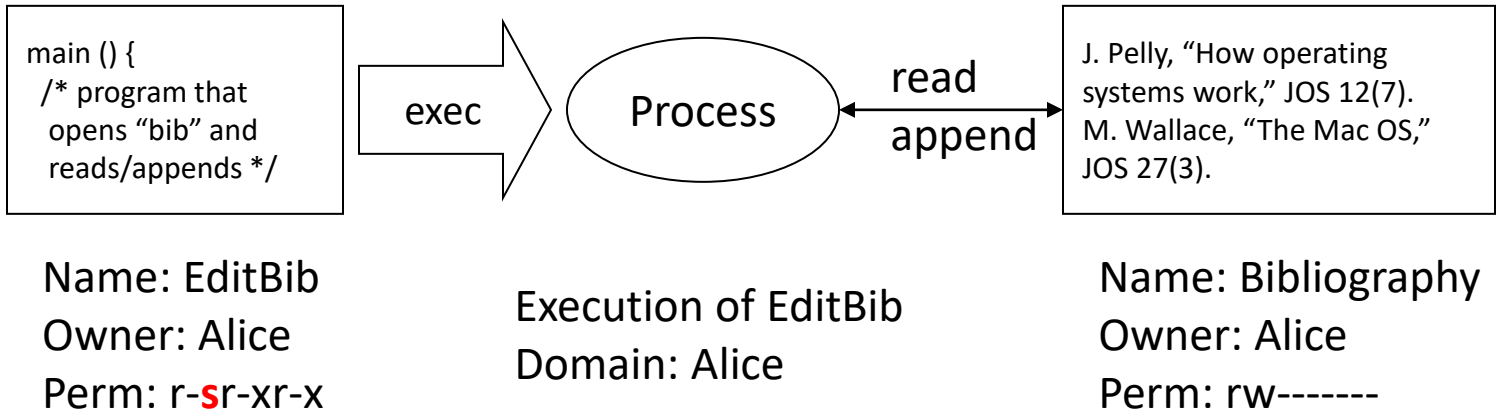
- Suppose Alice is keeping a bibliography file. She wants:
  - Bob and Carol to be able to contribute to the bibliography (append entries), but NOT arbitrarily write (e.g., delete) entries.
  - Dave and Erin to be able to read the bibliography.
  - Nobody else should access the bibliography (read or write).

What **doesn't** work…

- Make bibliography file writable by group.
  - Which group, readers or appenders?
  - For appenders, write is too much power – it means they can delete too.

- Make bibliography readable by group.
  - Now appenders can't modify the file, only the owner (Alice) can.

Solution



- Alice provides “EditBib” program: only reads/appends
- Alice sets permissions…
  - of EditBib program: execute, and setuid (it runs with Alice’s credentials)
  - of Bibliography file: read/write only for Alice, nobody else
- EditBib: look at which user is running it, allow/deny permissions accordingly

Alternative: POSIX “facls”

- Extended “file access control list” functionality
  - setfacl: change a file’s ACL
  - getfacl: read a file’s ACL
- ACL defines a list of checks that determine what sort of access (read/write/execute) a file access should get, depending on who’s making the request.
- Flexible and expressive, but not really fun to use: man 5 acl

Summary

- Protection is an enforcement mechanism. Often low-level (e.g., checking whether certain bits are set).

- Protection can be expressed in the OS using different structures, with tradeoffs:
  - Access control list: easy to modify/revoke, slower to check
  - Capabilities: difficult to modify/revoke, easy to verify

- Unix has several advanced protection mechanisms for files: setuid/setgid and POSIX facls.