Overlayfs And Containers

Miklos Szeredi, Red Hat
Vivek Goyal, Red Hat

Introduction to overlayfs

Union or…?

- Union: all layers made equal
- How do you take the union of two files?
- Or a file and a directory?
- **NO!** Layers can’t be treated equal

…overlay!

- Layer upon layer upon layer…
- Only upper layer can be modified
  - copy-up (exception: directory contents)
- Objects in one layer cover up objects with the same name in layer(s) below
- Exception: directories, which are merged
- Exception for the exception: opaque directories
- One more exception: whiteout
  - covers up anything and makes it look like nothing

Design

- Userspace API (most important!)
    - No new object types
        - Whiteout -> char dev with 0/0 device number
        - Opaque dir -> xattr
- Make it as simple as possible (and not a bit simpler)
    - Most of the logic is in a separate filesystem module
    - Some VFS impact but not much; some FS impact but not much
- Upstream early
    - It doesn’t have to do *everything* right; features can be added later…

Implementation

- Separate cache for the overlay directory tree
  - Allows less impact on VFS/FS
  - **BUT** bad for memory use
- Shared cache for the file contents
  - Copy-up when opened for write (may be too early)
  - Ugliness when copy-up happens while file is already open read-only
  - **BUT** great for performance and memory use
- Limitations
  - modifying lower layer -> don't care
  - Not (yet) a "POSIX" filesystem (st_dev/ino quirks, directory rename, hard link copy-up, etc)

Features added later

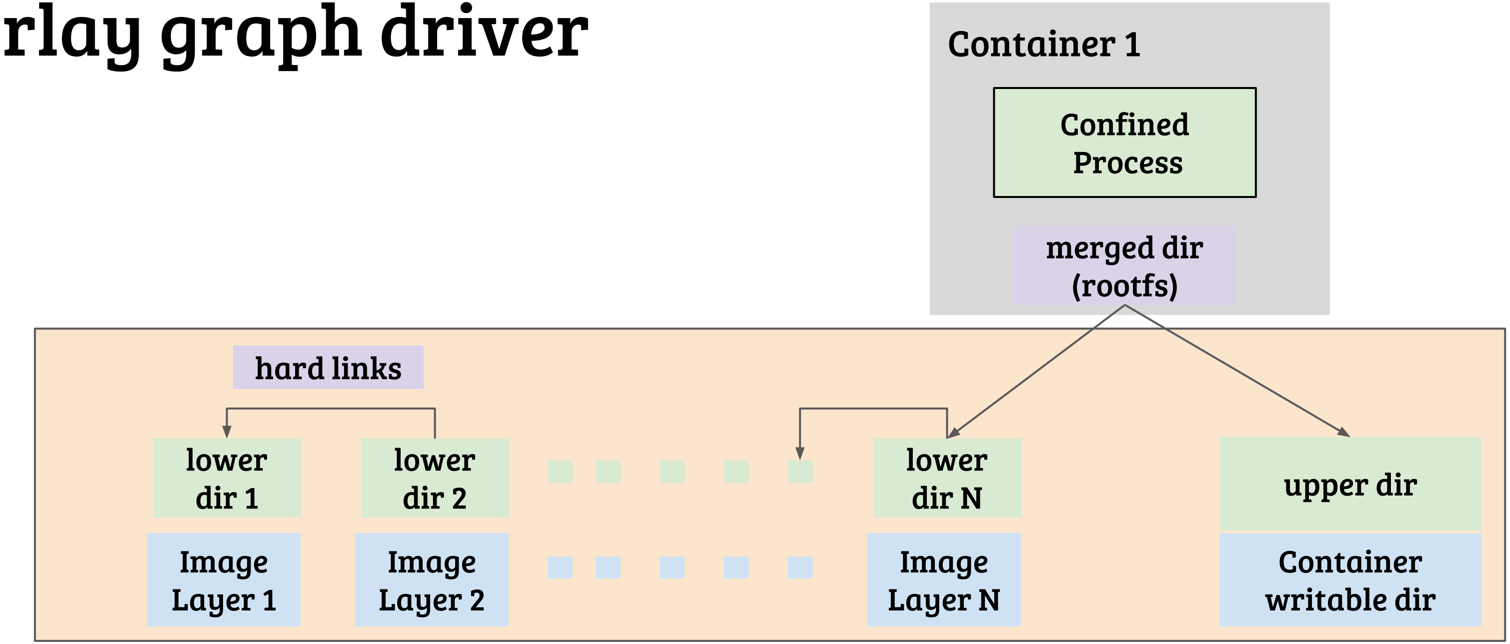
- Multiple lower layers
- Renaming directories
- SELinux
- POSIX ACL
- File locking

Features (work in progress)

- RW-RO file consistency after copy-up
  - Just need to fix this case up in VFS
- Fix st_dev, constant st_ino/d_ino
  - Store inode number for copied up files
  - Finding a common ino space for different underlying filesystems
- Hard link copy up
  - Should be very rare
  - Can use a global database for storing inode numbers of copied up hard links

overlayfs usage in docker

overlay graph driver



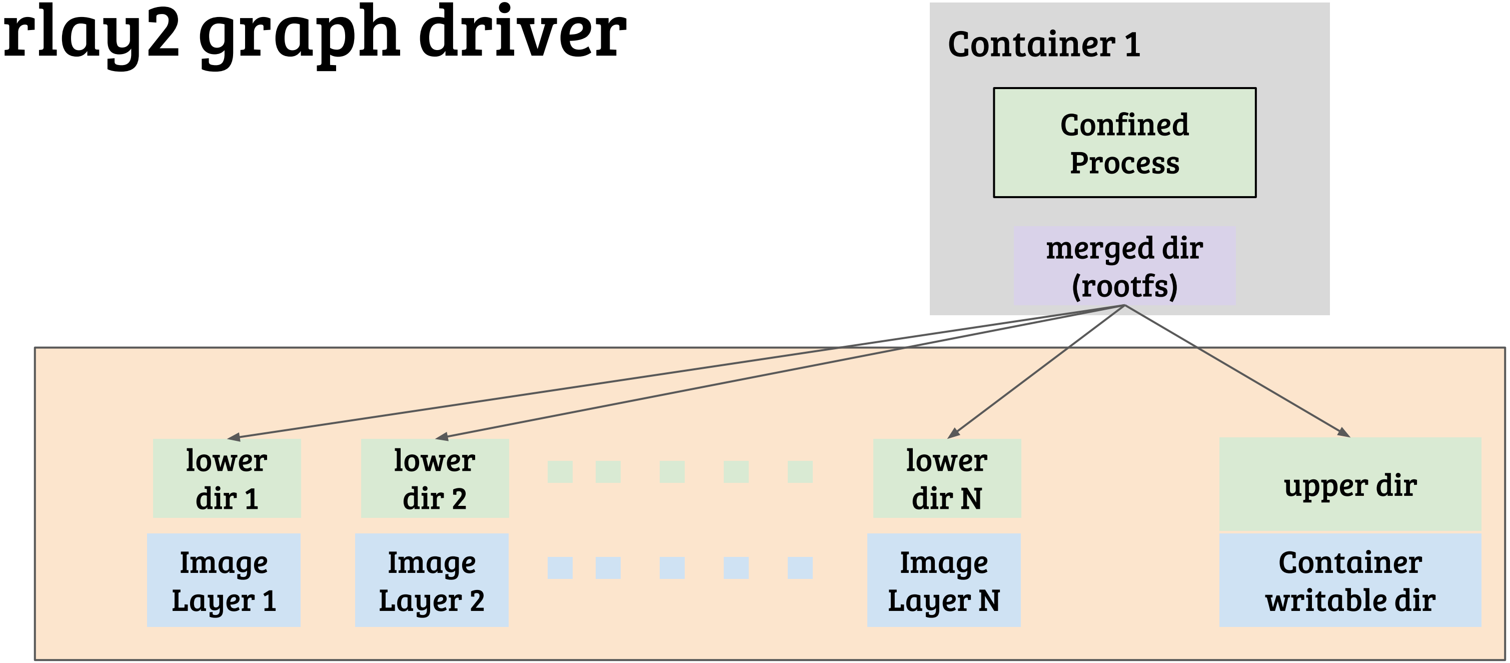
docker daemon option --storage-driver=overlay

Overlay supported single lower directory

Hard links created between image layers

Higher inode utilization

overlay2 graph driver



docker daemon option --storage-driver=overlay2

overlayfs should support multiple lower dirs

No hardlinks and dir creation in every layer

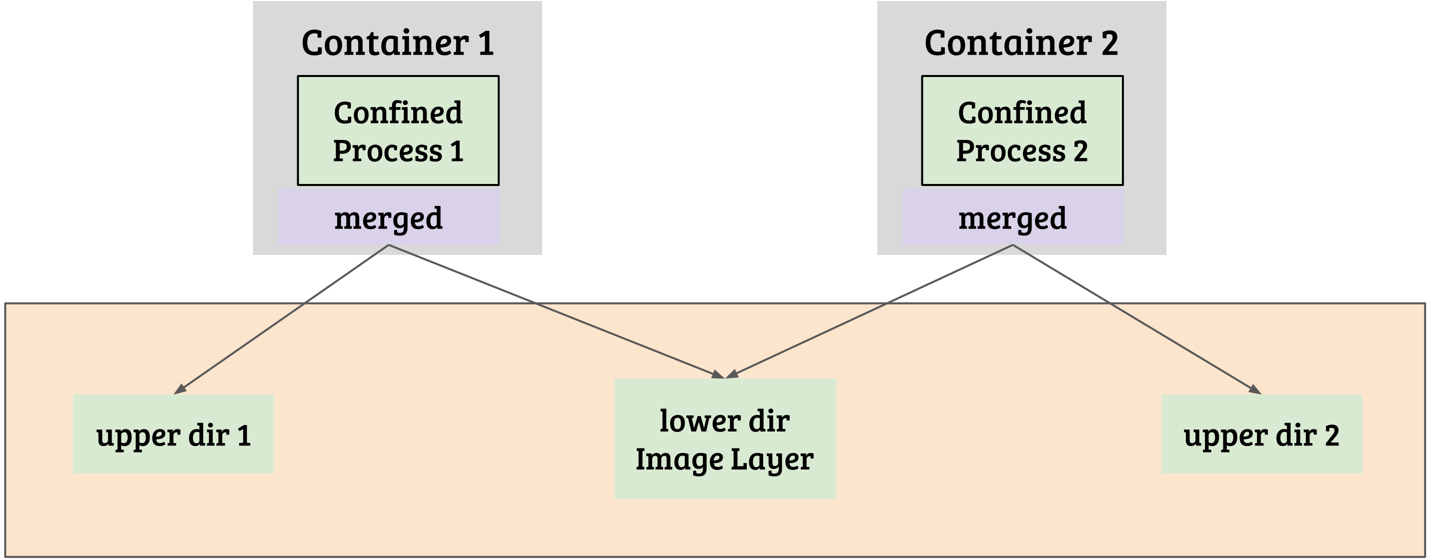
Better inode utilization

Container security and overlayfs

How do we handle access permissions?

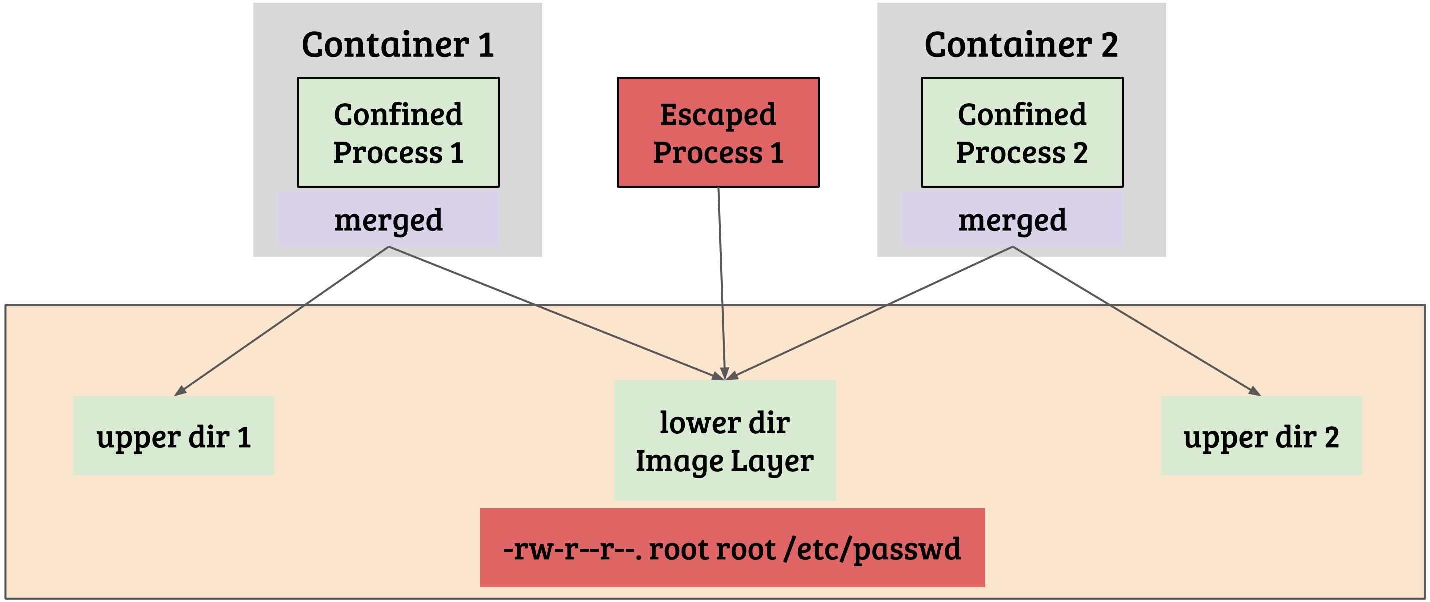
DAC(Ownership/Permissions)
MAC (SELinux)

An example setup



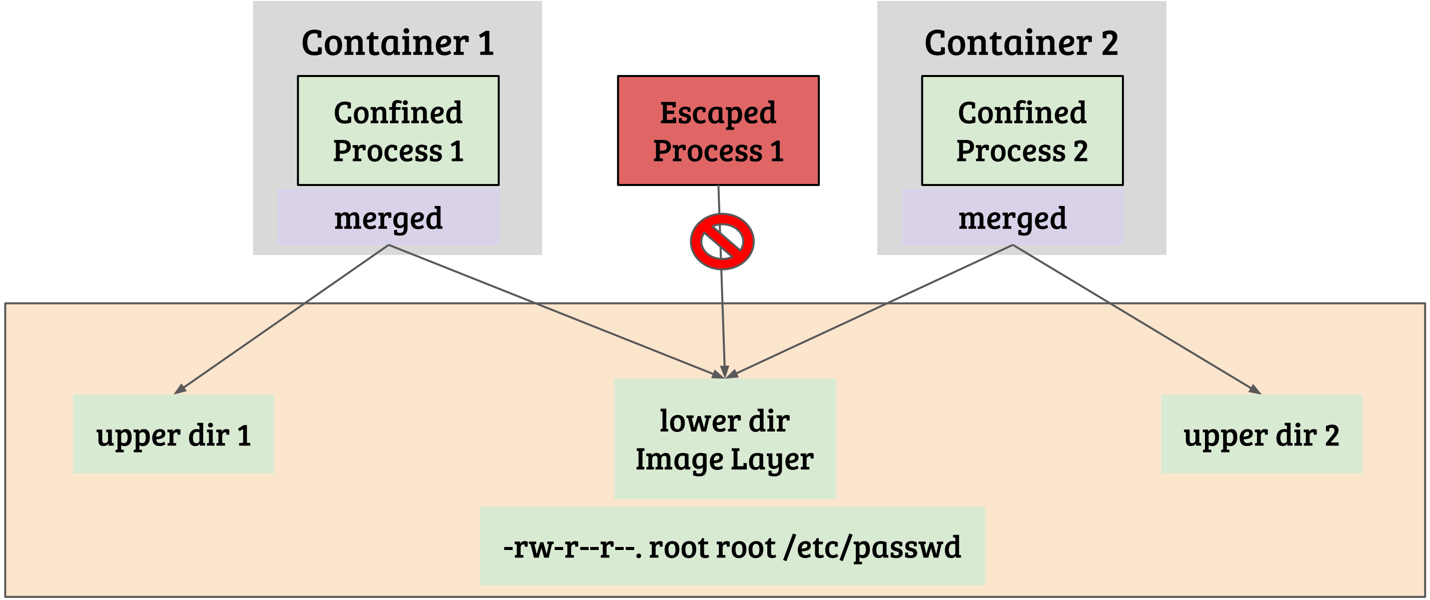
Two containers sharing lower dir with separate upper dir

Escaped container process writes to image dir/files



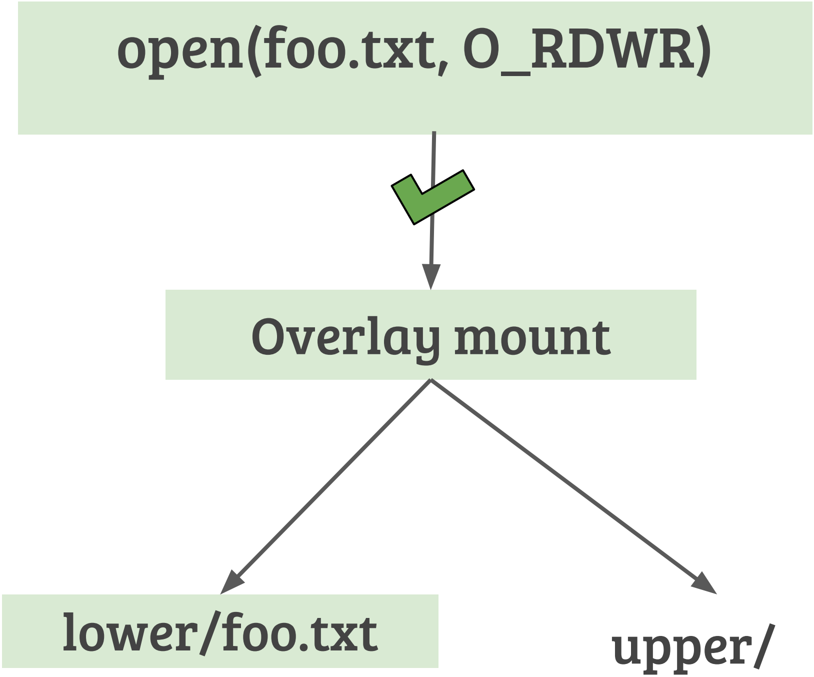
DAC allows writing to /etc/passwd

Security goal 1



Do not allow writing to image dir/files

Allow access through overlay mount point



open(foo.txt, O_RDWR)

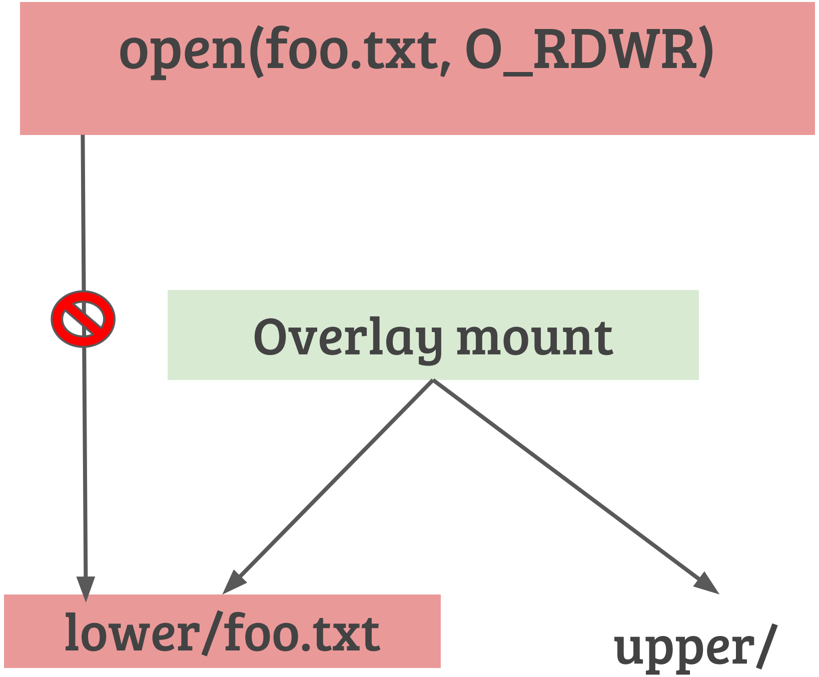
✓

Overlay mount

lower/foo.txt

upper/

Deny write access on underlying file



open(foo.txt, O_RDWR)

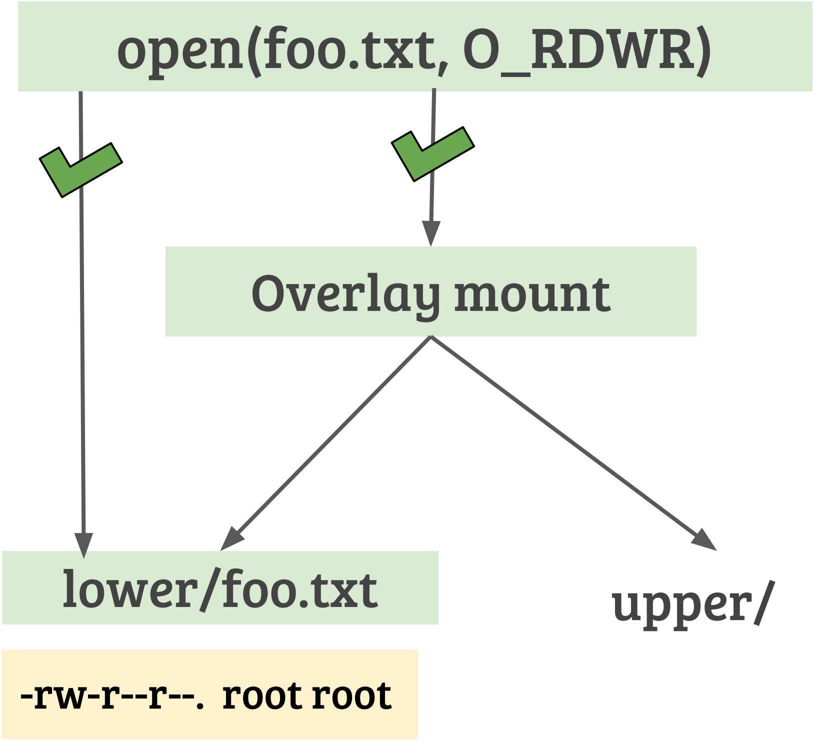
Overlay mount

lower/foo.txt

upper/

DAC allows access through both paths

(When root inside container is root outside)



open(foo.txt, O_RDWR)

✓

✓

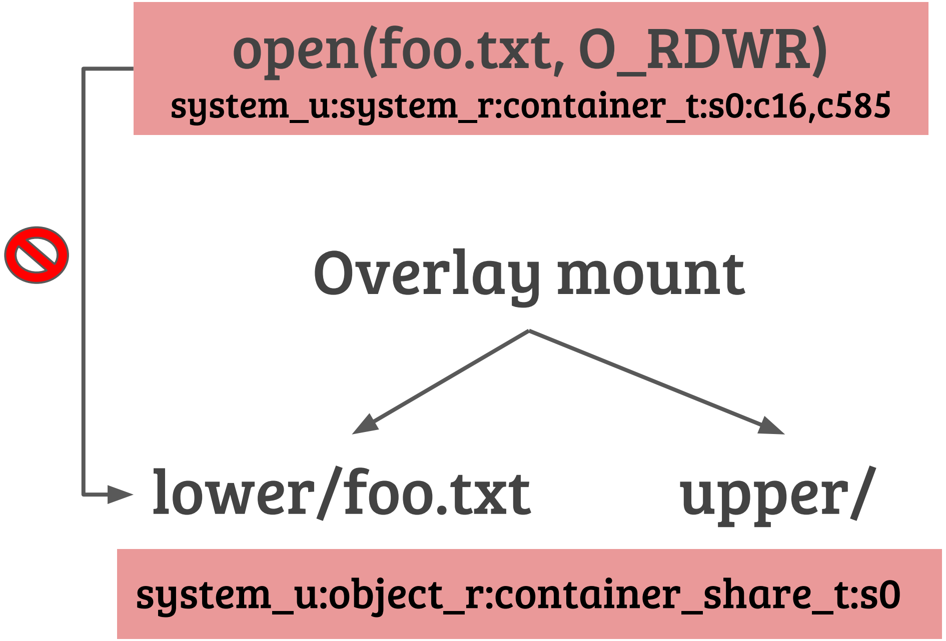
Overlay mount

lower/foo.txt

upper/

-rw-r--r--. root root

Read only label on lower files



open(foo.txt, O_RDWR)

system_u:system_r:container_t:s0:c16,c585

🚫

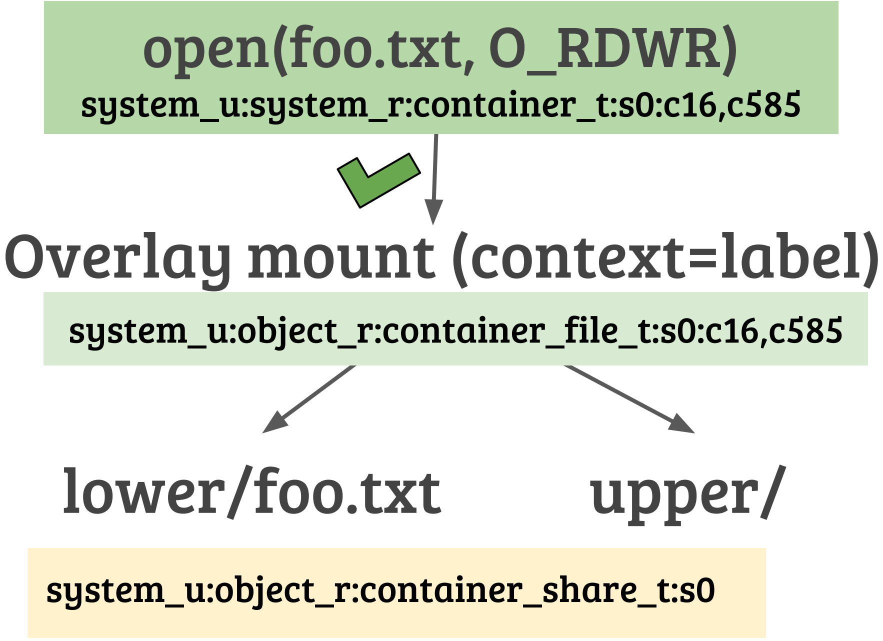
Overlay mount

lower/foo.txt

upper/

system_u:object_r:container_share_t:s0

Use context mount option for overlay



open(foo.txt, O_RDWR)

system_u:system_r:container_t:s0:c16,c585

✓

Overlay mount (context=label)

system_u:object_r:container_file_t:s0:c16,c585

lower/foo.txt

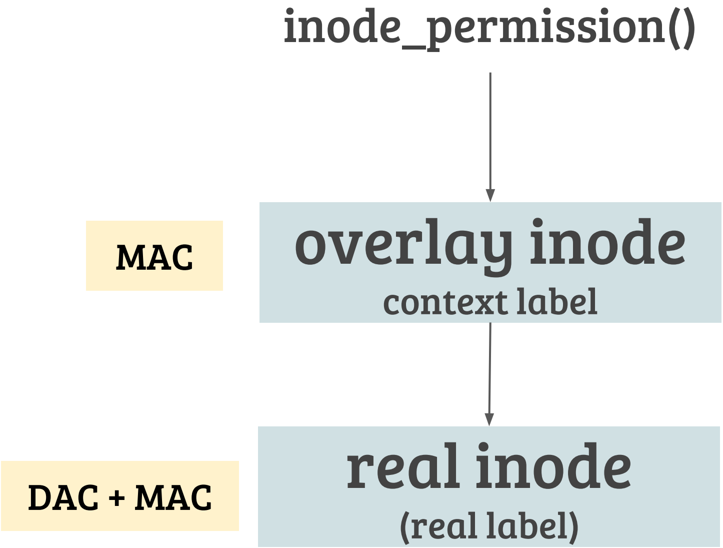
upper/

system_u:object_r:container_share_t:s0

```
mount -t overlay -o context="system_u:object_r:container_file_t:s0:c16,c585".... merged/
```

That did not work

Access permission checks in overlay



inode_permission()

MAC

overlay inode

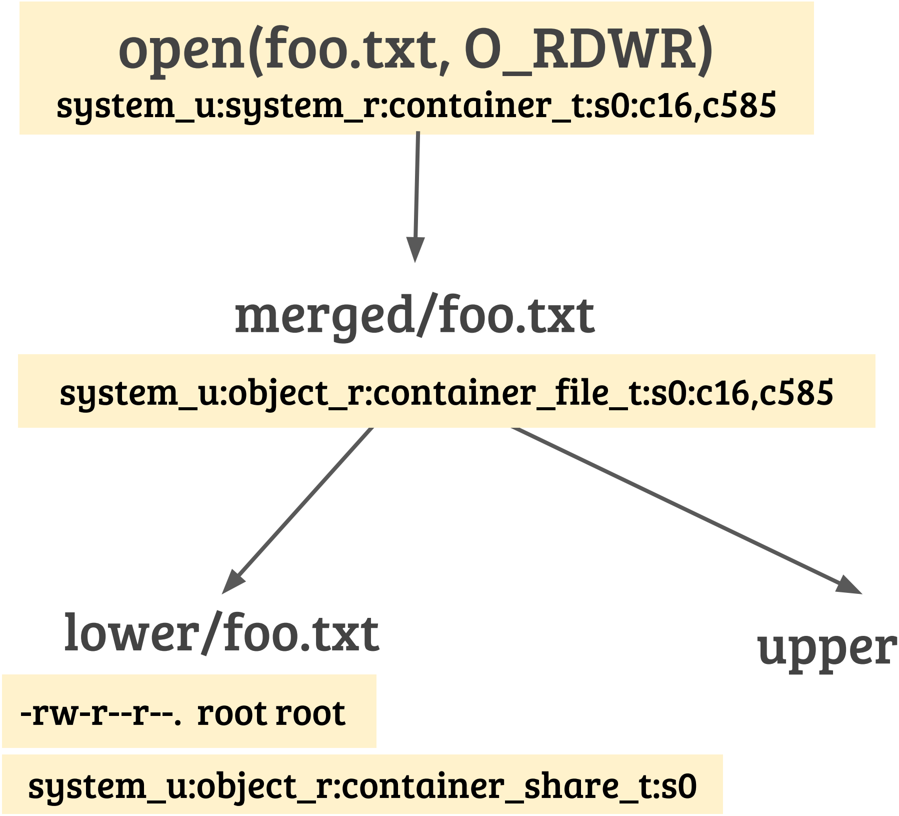
context label

DAC + MAC

real inode

(real label)

Read only label on lower file



open(foo.txt, O_RDWR)

system_u:system_r:container_t:s0:c16,c585

merged/foo.txt

system_u:object_r:container_file_t:s0:c16,c585

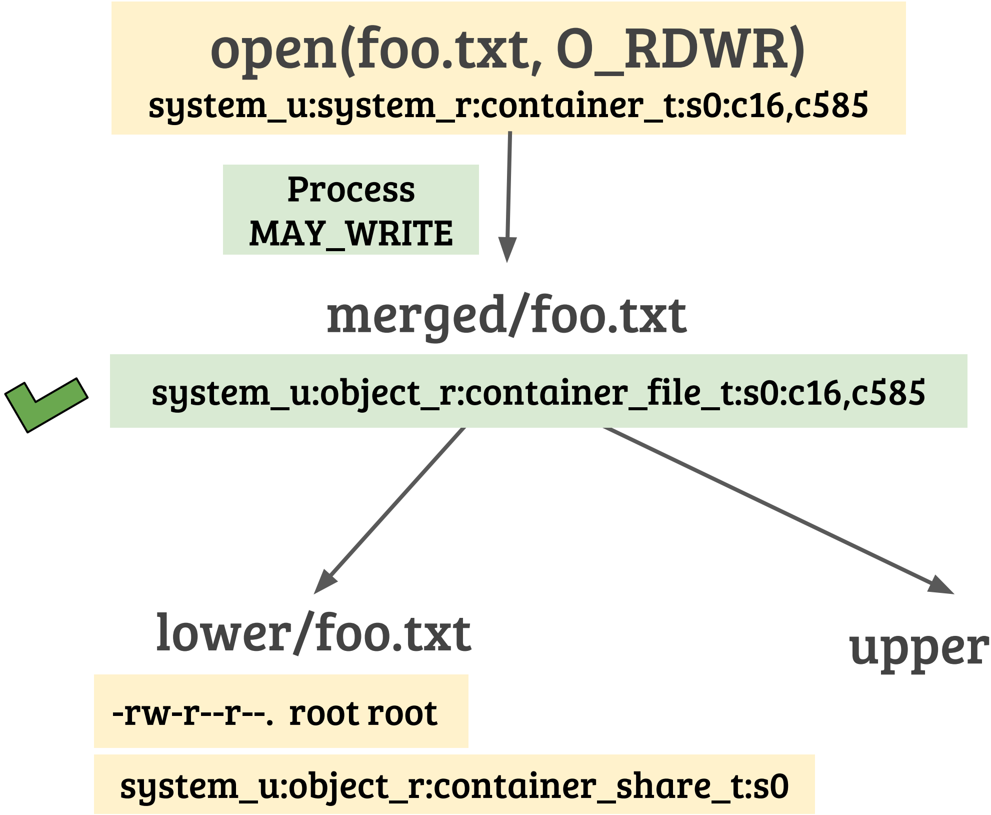
lower/foo.txt

-rw-r--r--. root root

system_u:object_r:container_share_t:s0

upper/

Process Overlay inode check



open(foo.txt, O_RDWR)

system_u:system_r:container_t:s0:c16,c585

Process MAY_WRITE

merged/foo.txt

✓ system_u:object_r:container_file_t:s0:c16,c585

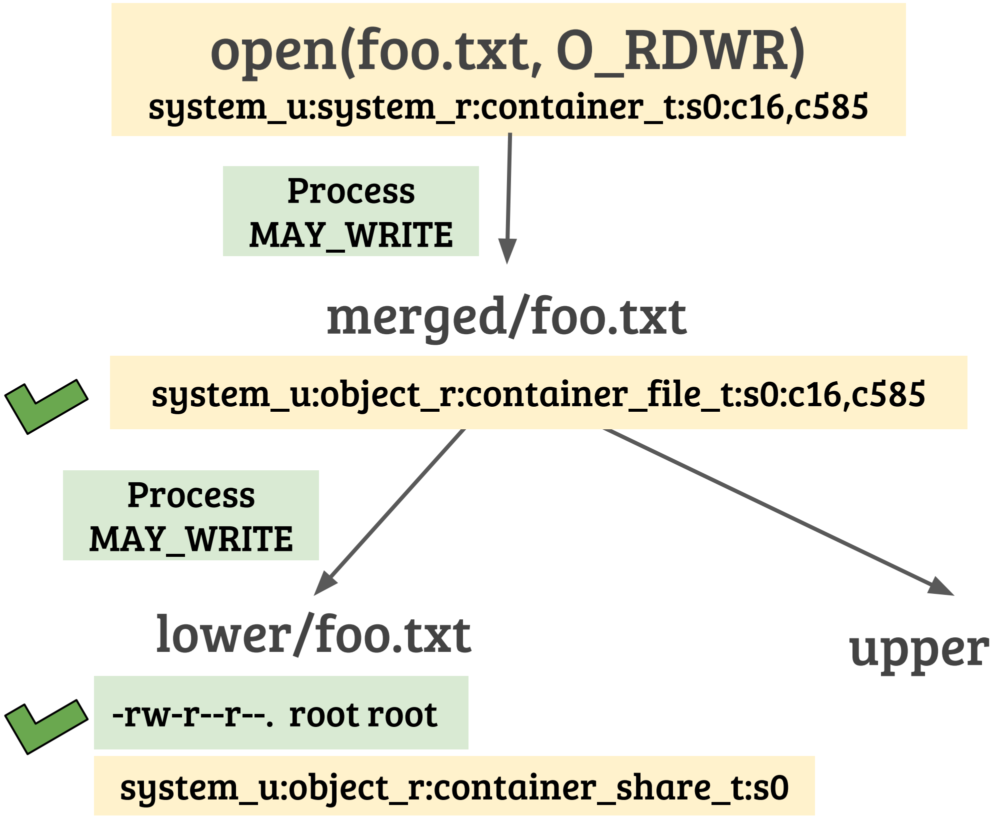
lower/foo.txt

-rw-r--r--. root root

system_u:object_r:container_share_t:s0

upper/

Process lower inode DAC check



open(foo.txt, O_RDWR)

system_u:system_r:container_t:s0:c16,c585

Process
MAY_WRITE

merged/foo.txt

✓ system_u:object_r:container_file_t:s0:c16,c585

Process
MAY_WRITE

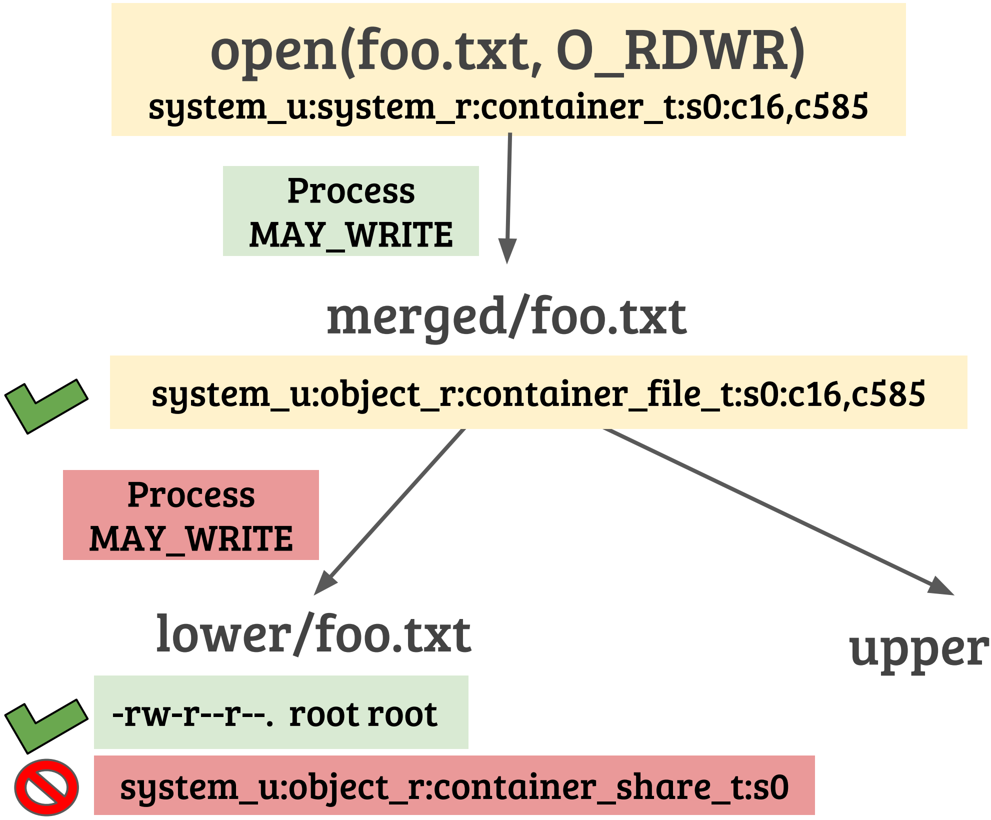
lower/foo.txt

upper/

✓ -rw-r--r--. root root

system_u:object_r:container_share_t:s0

Process lower inode MAC check



open(foo.txt, O_RDWR)

system_u:system_r:container_t:s0:c16,c585

Process MAY_WRITE

merged/foo.txt

✓ system_u:object_r:container_file_t:s0:c16,c585

Process MAY_WRITE

lower/foo.txt

upper/

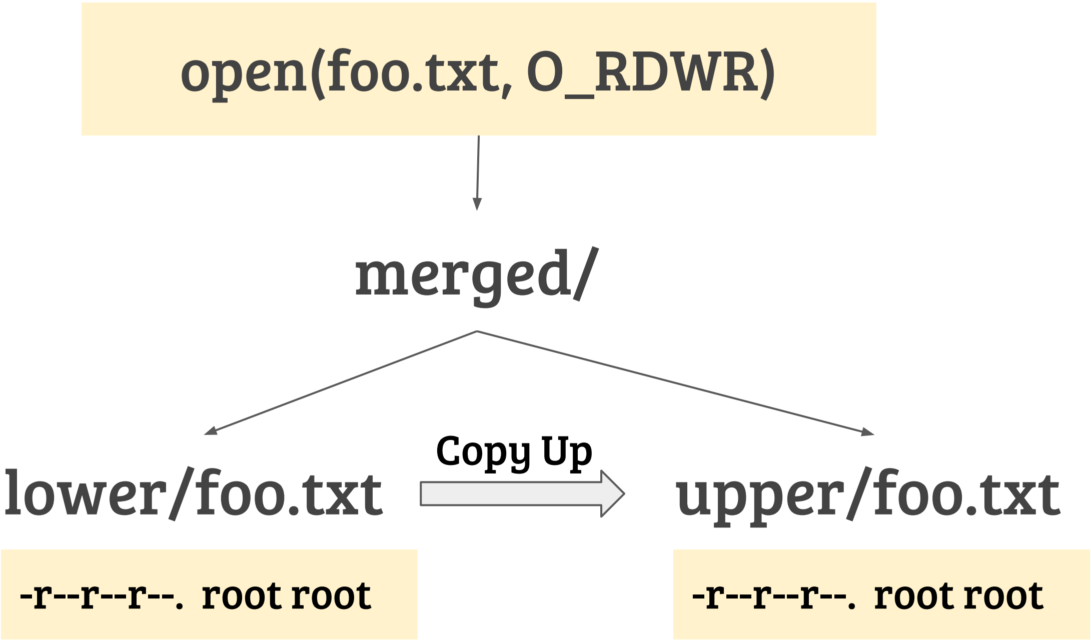
✓ -rw-r--r--. root root

🚫 system_u:object_r:container_share_t:s0

What if we don’t do WRITE checks on lower inode

But that will break DAC

DAC checks happen only at real inode



open(foo.txt, O_RDWR)

merged/

lower/foo.txt

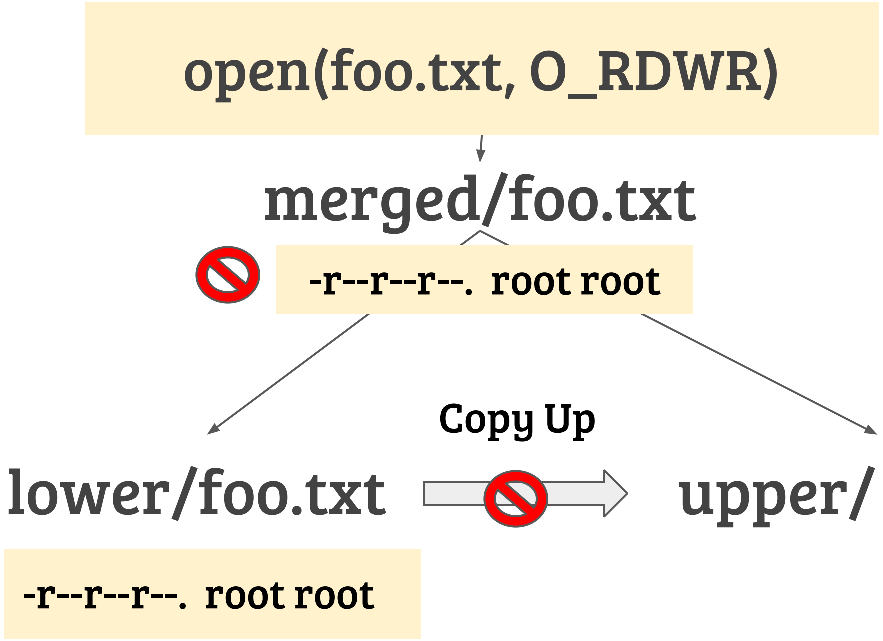
Copy Up

upper/foo.txt

-r--r--r--. root root

-r--r--r--. root root

Why not do DAC checks on both inodes



open(foo.txt, O_RDWR)

merged/foo.txt

-r--r--r--. root root

Copy Up

lower/foo.txt

upper/

-r--r--r--. root root

That kind of worked but…

Certain overlayfs operations failed MAC checks

Certain overlayfs operations fail MAC checks

File creation over whiteout

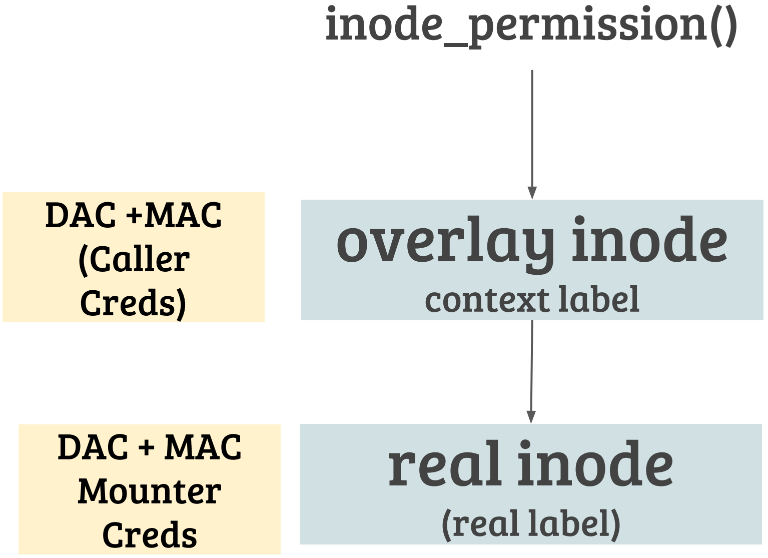
Certain overlayfs operations fail MAC checks

File creation over whiteout

Use mounter’s creds for privileged operations

Two Levels of Permission Checks

- Overlay inode is checked with creds of task
- Underlying inode is checked with creds of mounter
- Certain privileged operations are done with the creds of mounter



inode_permission()

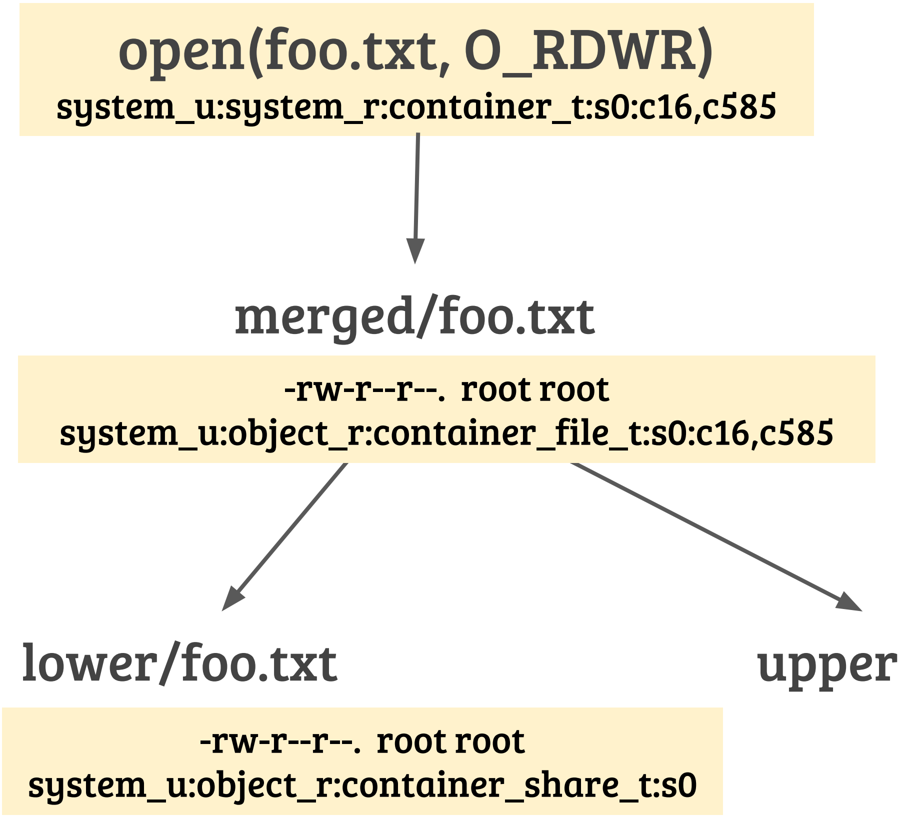
DAC +MAC (Caller Creds)

overlay inode
context label

DAC + MAC Mounter Creds

real inode
(real label)

Two levels of checks



open(foo.txt, O_RDWR)

system_u:system_r:container_t:s0:c16,c585

merged/foo.txt

-rw-r--r--. root root

system_u:object_r:container_file_t:s0:c16,c585

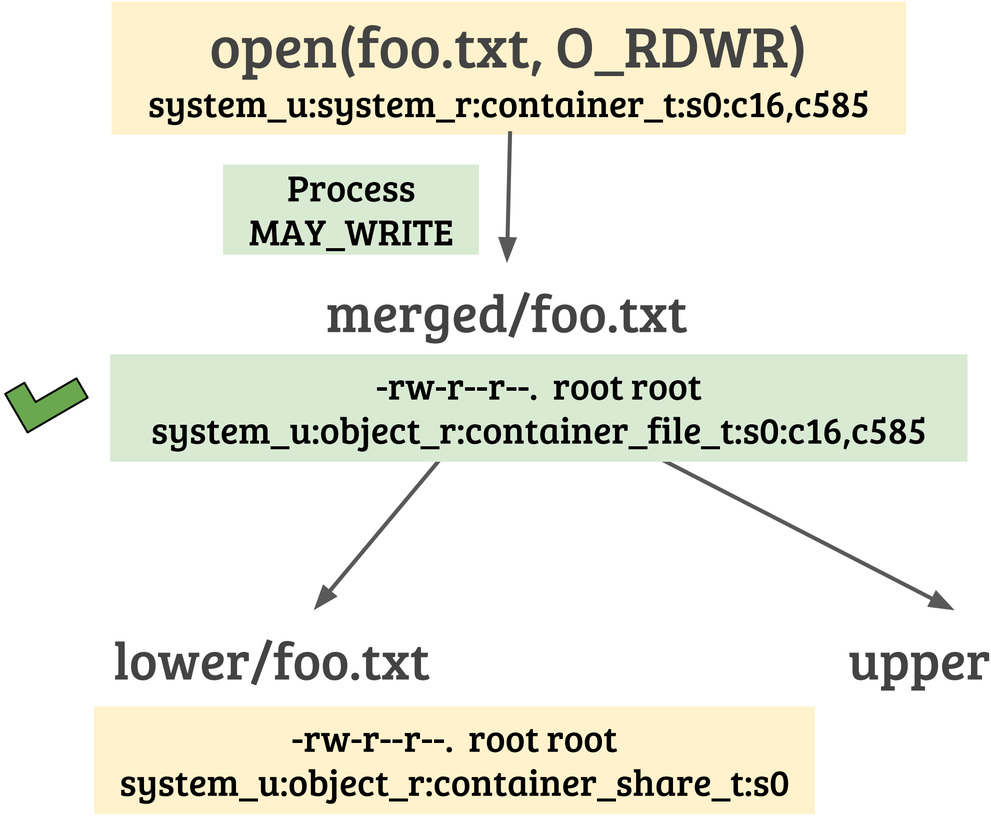
lower/foo.txt

-rw-r--r--. root root

system_u:object_r:container_share_t:s0

upper/

Process Overlay inode check



open(foo.txt, O_RDWR)
system_u:system_r:container_t:s0:c16,c585

Process
MAY_WRITE

merged/foo.txt

✓

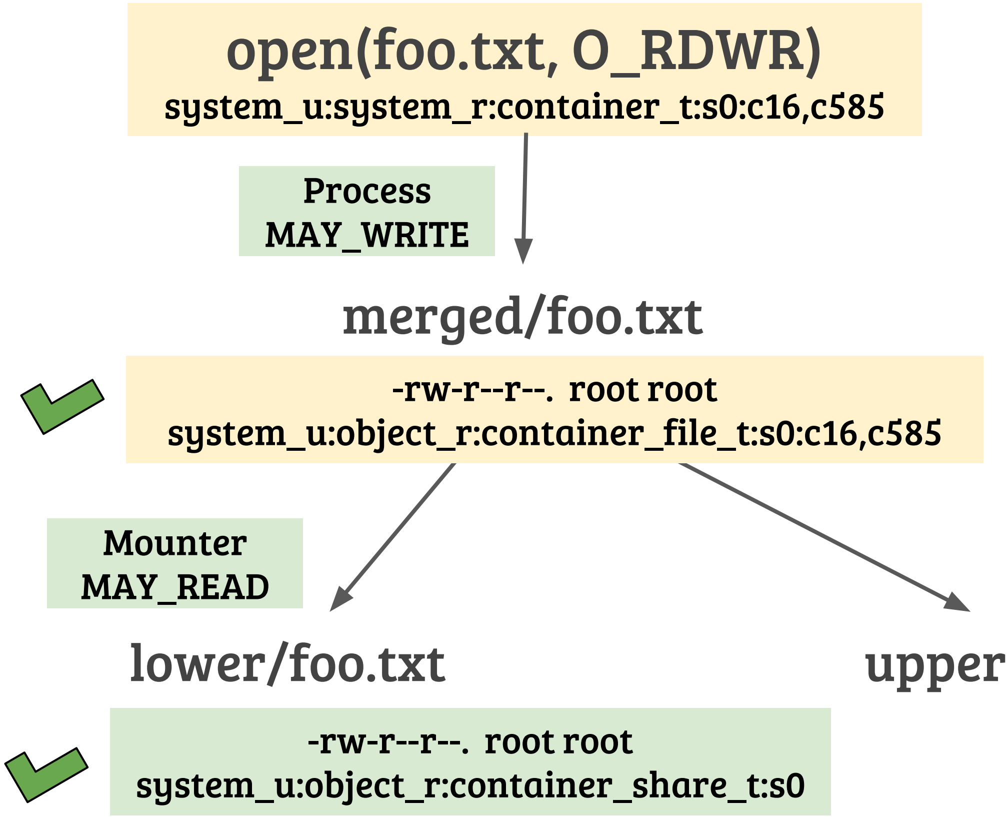
-rw-r--r--. root root
system_u:object_r:container_file_t:s0:c16,c585

lower/foo.txt

-rw-r--r--. root root
system_u:object_r:container_share_t:s0

upper/

Mounter real lower inode check



open(foo.txt, O_RDWR)
system_u:system_r:container_t:s0:c16,c585

Process
MAY_WRITE

merged/foo.txt

✓ -rw-r--r--. root root
system_u:object_r:container_file_t:s0:c16,c585

Mounter
MAY_READ

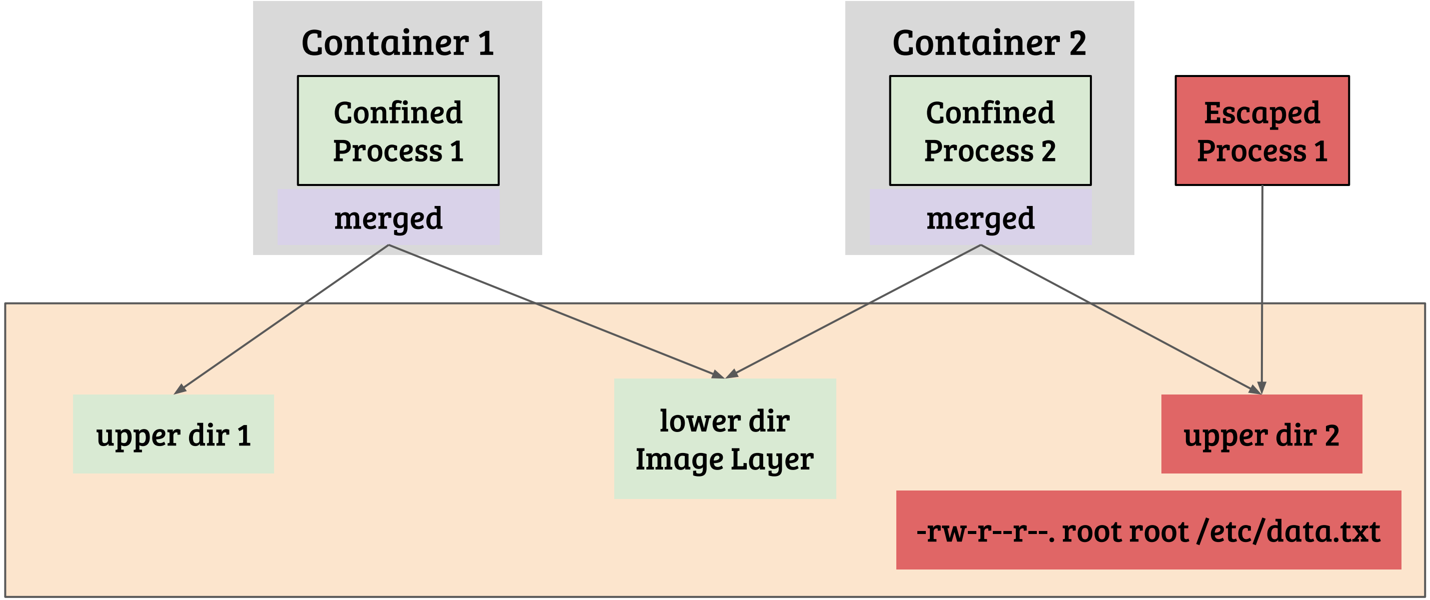
lower/foo.txt

upper/

✓ -rw-r--r--. root root
system_u:object_r:container_share_t:s0

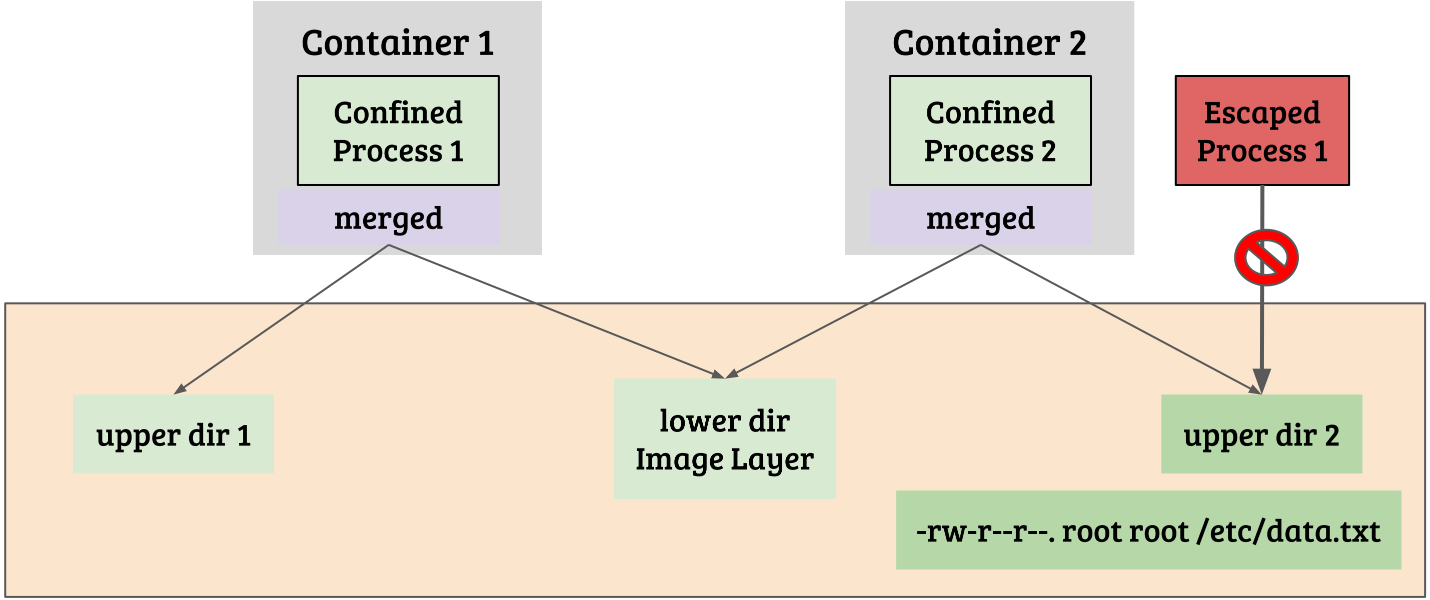
First requirement met

Escaped process accesses other container’s data



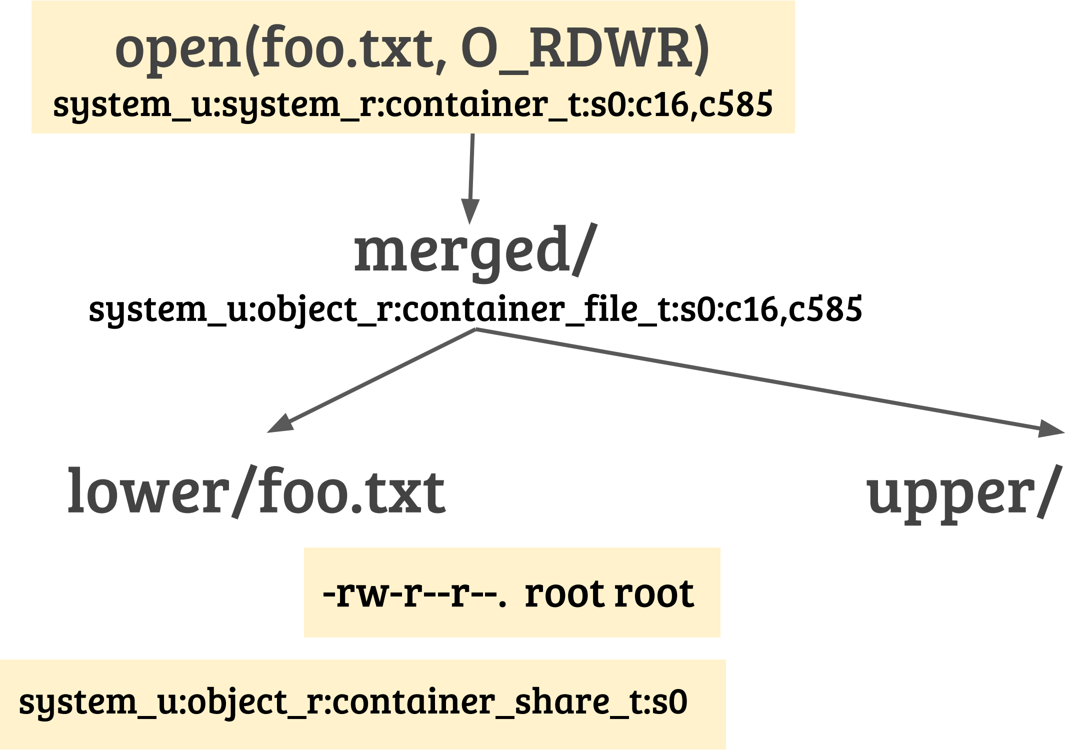
Container1 accesses container2’s data

Security goal 2



One container should not be able to access other container's data

Label upper files for container access only



open(foo.txt, O_RDWR)

system_u:system_r:container_t:s0:c16,c585

merged/

system_u:object_r:container_file_t:s0:c16,c585

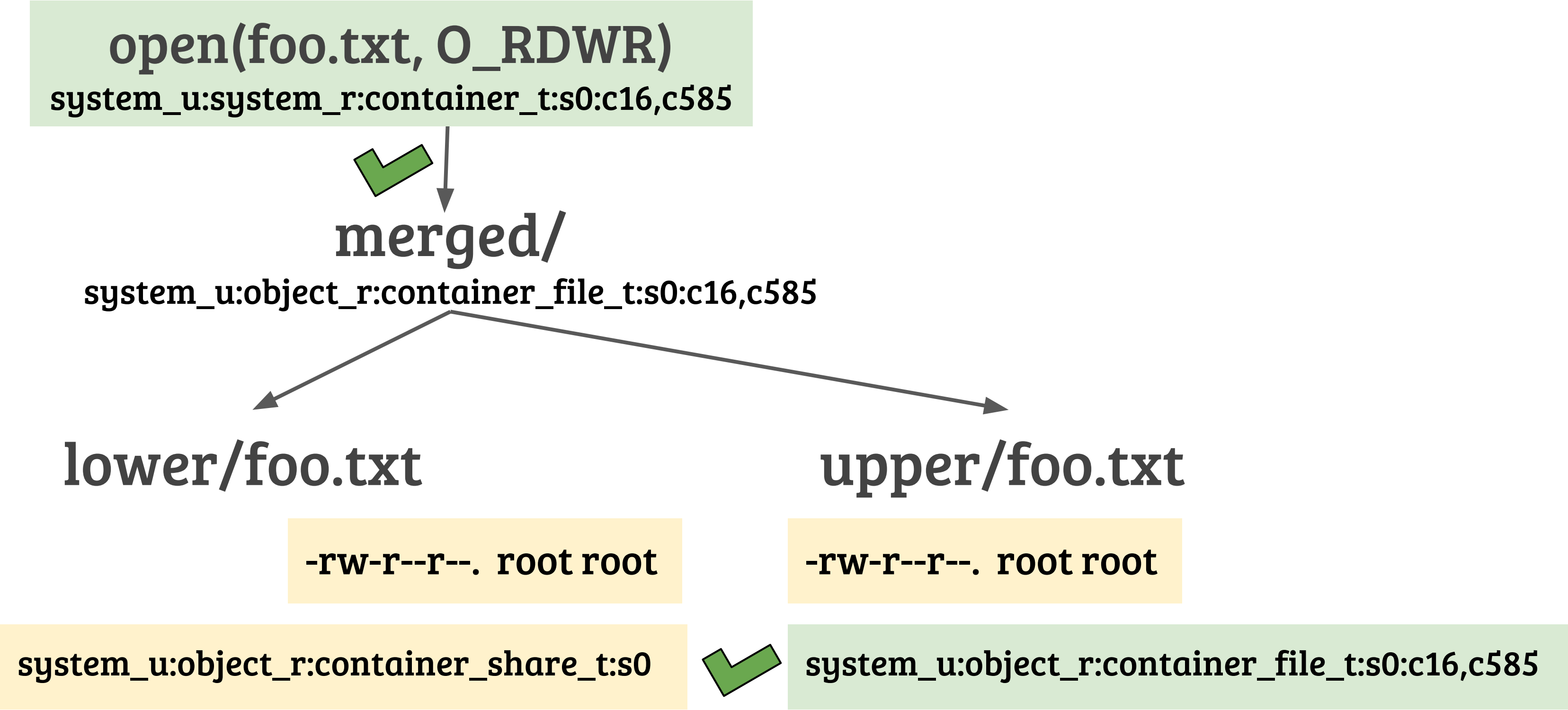
lower/foo.txt

upper/

-rw-r--r--. root root

system_u:object_r:container_share_t:s0

Label upper files for container access only



open(foo.txt, O_RDWR)

system_u:system_r:container_t:s0:c16,c585

✓

merged/

system_u:object_r:container_file_t:s0:c16,c585

lower/foo.txt

-rw-r--r--. root root

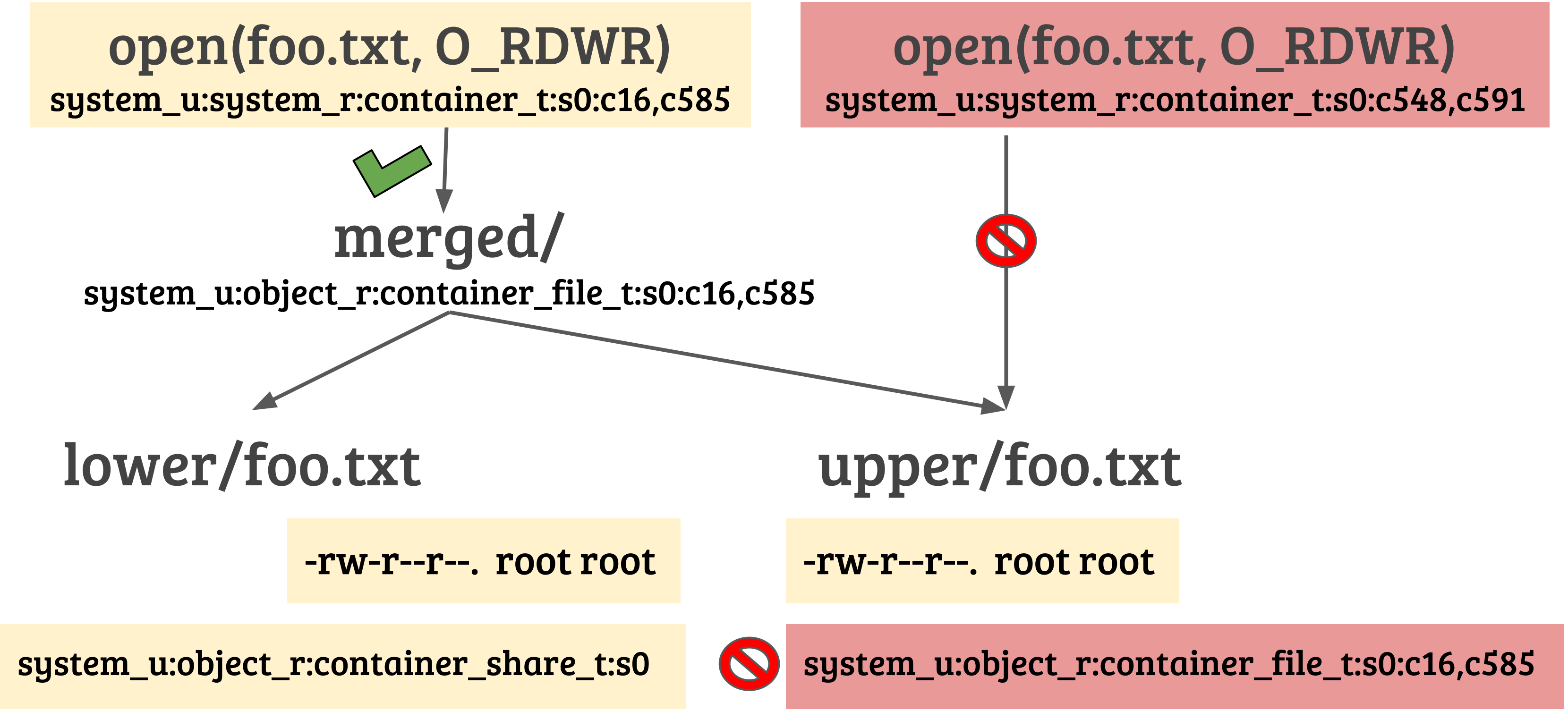
system_u:object_r:container_share_t:s0

upper/foo.txt

-rw-r--r--. root root

✓ system_u:object_r:container_file_t:s0:c16,c585

One container can’t access data of another container



open(foo.txt, O_RDWR)
system_u:system_r:container_t:s0:c16,c585

open(foo.txt, O_RDWR)
system_u:system_r:container_t:s0:c548,c591

merged/
system_u:object_r:container_file_t:s0:c16,c585

lower/foo.txt

-rw-r--r--. root root

system_u:object_r:container_share_t:s0

upper/foo.txt

-rw-r--r--. root root

system_u:object_r:container_file_t:s0:c16,c585

New LSM Hooks

- inode_copy_up()
  - Called during copy up. Returns new set of creds for file creation.
  - For context mounts, file is created with label specified in context= option.
- inode_copy_up_xattr()
  - Called during copy up of xattrs. SELinux blocks copying up of SELinux xattr.
- dentry_create_files_as()
  - Called during creation of new file. Returns new set of creds for file creation.
  - For context mounts, file is created with label specified in context= option.

Overlayfs vs. devicemapper

- In general, faster than devicemapper
  - Page cache sharing
- Not fully POSIX compliant, yet
  - So some workloads might experience issues
- Fedora 26 will have overlay2 as default graph driver
  - Switch back to devicemapper if you face issues

DAC and container security

- DAC will solve these issues if containers run in user namespaces with different mappings
- Needing to do a chown on image continues to be a issue
- shiftfs or something else?

Thank You