

Subvolume Quarantine

CephFS ransomware containment —
implementation and support handoff

Igor Golikov — SW Engineer, IBM, Ceph Days Zurich 2026

Session plan

1. Why / use case

Ransomware response
Operational goals

2. How it works

State model
MDS enforcement
Client behavior

3. Demo / ops

Enable / disable
Expected behavior
Troubleshooting

4. Test + status

Regression matrix
Thrash coverage

Feature use case

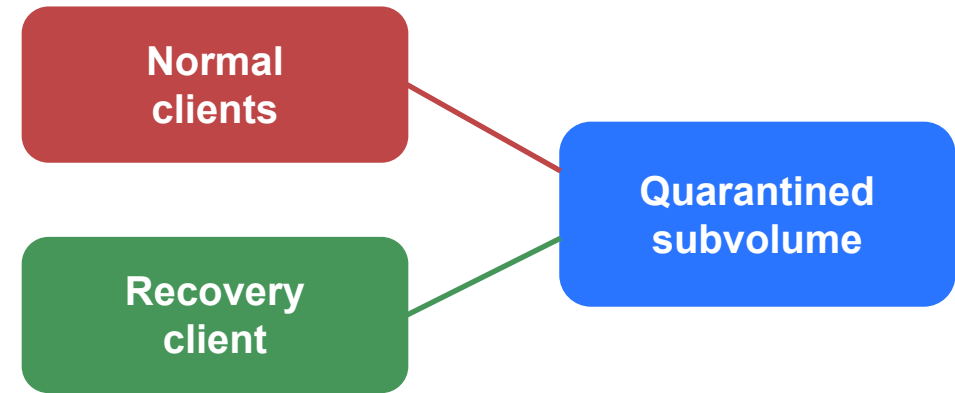
As an admin, I need to immediately isolate a compromised CephFS subvolume while keeping recovery/admin access available.

Contain suspected ransomware or destructive workload quickly

Prevent normal clients from reading/writing/removing data under that subvolume

Allow recovery clients to inspect, copy, or repair data using explicit quarantine caps

Keep the quarantine state persistent across MDS restart/failover



Default is fail-closed: normal clients are blocked unless explicitly authorized.

Design principles

Persistent state

Inode opmetadata is the source of truth
No separate per-subvolume status DB
Survives MDS restart

Ephemeral tracking

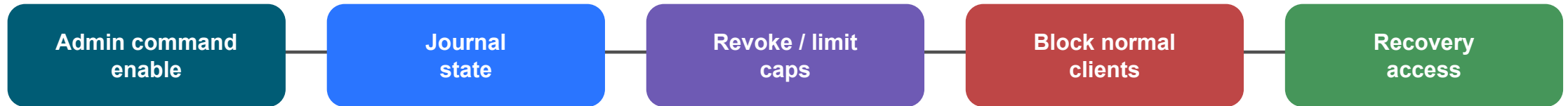
Track only in-flight enable/disable
Completion, timeout, and multi-MDS coordination
Remove tracking after operation completes

Explicit authority

q = quarantine-aware access
Q = quarantine admin / recovery access
`allow \*` does not imply quarantine access

Core rule: the inode flag represents desired state; MDS ranks coordinate work needed to enforce it.

Quarantine lifecycle



Enable writes quarantine optmetadata on the subvolume root inode

MDS walks the subtree path to identify affected inodes

Caps are REVOKED for non-q clients; operations get EQUARANTINED / EACCES

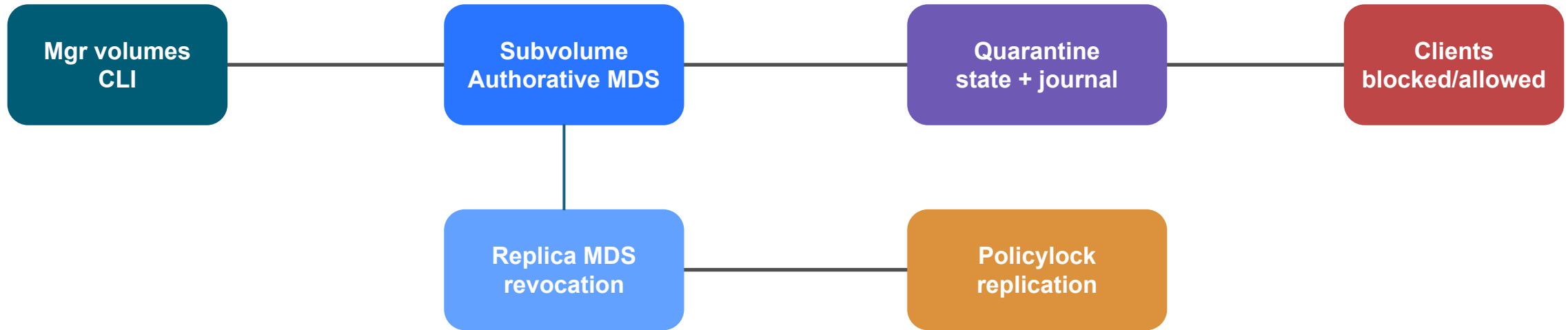
Disable clears the flag and re-issues caps

State is durable; operation tracking is not durable by design

Concurrent enable/disable is guarded to avoid racing admin commands

Timeouts prevent admin commands from hanging on missing replica replies

Implementation architecture



- Mgr/volumes sends enable/disable to an MDS (admin path, not the data path)
- Auth MDS for the subvolume root owns quarantine state
- State is persisted on the subvolume-root inode and journaled
- Auth MDS revokes caps under that tree
- Replica ranks learn via policylock and revoke locally
- Clients are then blocked or allowed by caps (q / Q)

Where access is enforced

1. Cap restriction

``get_caps_quarantine_mask()``
restricts caps

Normal client loses
read/write/create/remove ability
Recovery client with q/Q keeps
access

2. Operation checks

``Server::check_access()`` validates
operations under quarantine
Subvolume rm, resize, snapshot
create/list, getpath are blocked
Migrator export path checks
quarantine before movement

3. Client behavior

MDS sends ``-EQUARANTINED`` in
cap messages
Client marks inode as quarantined
Kernel clients block/resume;
FUSE/libcephfs return access errors
where expected

Result: data under the subvolume is protected at both capability and operation boundaries.

Auth caps and client behavior

Client / cap	During quarantine	Expected behavior
Normal client	No q / Q	Blocked under subvolume
Recovery client	rwq	Allowed where caps match path
Admin recovery	rwQ	Allowed for any quarantined subvolume
`allow *` client	No explicit q/Q	Still blocked — no implicit bypass
Old client	No feature support	Blocked safely

Quarantine access is intentionally explicit; broad existing caps should not accidentally bypass isolation.

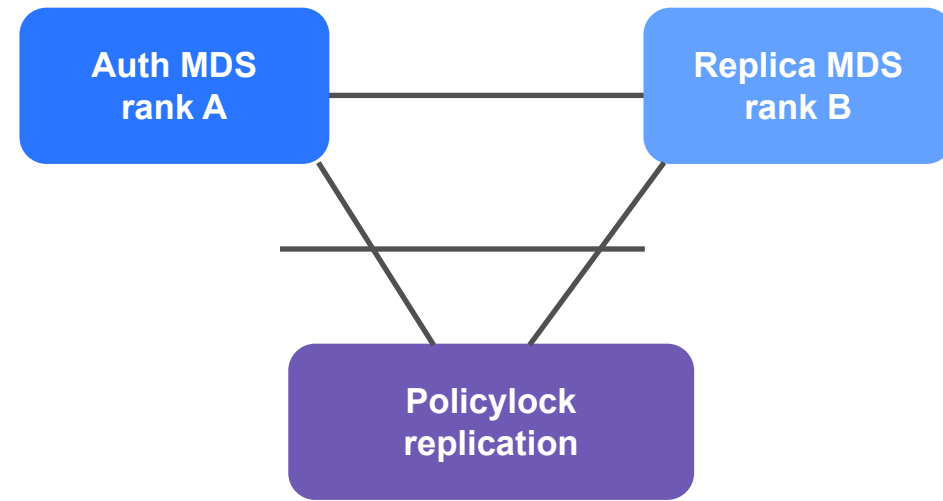
Multi-MDS coordination

Initial design used custom `MMDSQuarantine` messages

Review and testing exposed dispatch, decoding, replica, and hang risks

Final direction moved to policylock replication where possible

Replica MDS learns quarantine through inode policy state and triggers cap revocation



Coverage includes pinned subdirectories, snapshot paths, non-auth rank reads, concurrent quarantine, and recovery clients across ranks.

Mgr/volumes behavior

Operational behavior is chosen so admins can still discover and manage quarantined subvolumes safely.

Operation	During quarantine	Reason
`subvolume ls`	Allowed	Lists all subvolumes; quarantined entry is not dropped
`subvolume info`	Allowed	Returns quarantine status; minimal info when access is blocked
`subvolume getpath`	Blocked	Requires `.meta` access inside the quarantined subvolume
`snapshot ls`	Blocked	Snapshot content is protected by quarantine
`subvolume rm/resize`	Blocked	Destructive/metadata-changing operations are denied

Demo flow



```
# 1. Enable quarantine
ceph fs subvolume quarantine enable cephfs <subvol>

# 2. Verify status
ceph fs subvolume info cephfs <subvol>

# 3. Normal client: access is blocked
ls /mnt/cephfs/volumes/<group>/<subvol>

# 4. Disable quarantine
ceph fs subvolume quarantine disable cephfs <subvol>
```

Show behavior from normal client first
Then repeat from recovery client with q/Q caps
Show `subvolume ls` and `info` remain useful
for admins
Finish with disable restoring access

Regression and functional coverage

Blocking semantics

Read/write/create/delete
Snapshot access
Mgr operations
New client after enable

Authorized access

rwq recovery client
rwQ admin recovery
`allow \*` does not bypass
Disable restores access

Distributed cases

Multi-MDS both ranks
Pinned subdirectories
Replica reads
Concurrent quarantine

Resilience

Quarantine
thrasher

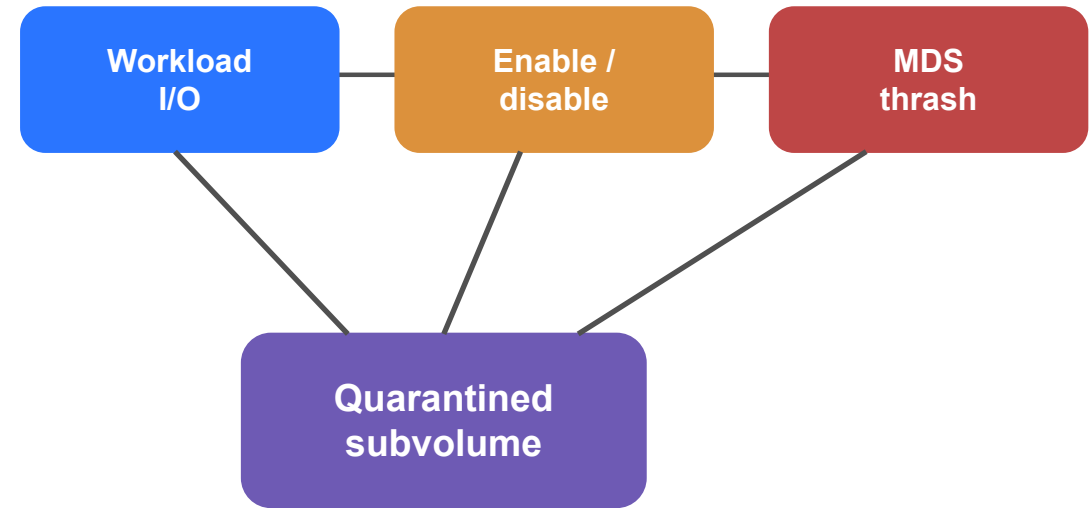
25+ focused tests plus thrash coverage for enable/disable during MDS failover.

MDS failover / thrash coverage

New `quarantine\_thrasher.py` background greenlet
Runs with existing `mds\_thrash` and real workloads
such as fsstress

Loops enable → hold → disable → release while
MDS daemons are killed/restarted

Retries transient failover errors and verifies state
after each transition



**Purpose: prove quarantine remains enforced
and recoverable when failover happens
mid-operation.**

Troubleshooting checklist

What to collect

MDS logs around enable/disable
Client logs for EQUARANTINED /
EACCES
MDS map / ranks / failover timeline
Subvolume info and auth caps

What to check first

Is the subvolume root flag set?
Does the client have explicit q/Q?
Are multiple MDS ranks involved?
Did cap revocation finish or timeout?

Common symptoms

Admin command hang → replica
reply/timeout path
Unexpected allow → q/Q cap or
stale client
Unexpected block → cache miss
fail-closed / old client
`ls` incomplete → readdir/getattr
issue

Debug pattern: validate state first, then authorization, then cap revocation/multi-MDS propagation.

Documentation and code map

MDS / protocol

- `MDSRankQuarantine.cc`
- `QuarantineManager.\*`
- `MDCache.cc/h`
- `Server.cc/h`
- `Locker.cc/h`
- `CInode.cc/h`

Client / mgr

- `Client.cc/h`
- `Inode.cc/h`
- `MQuarantineDisable.h`
- `volumes/module.py`
- `volumes/fs/volume.py`

Tests / docs

- `test\_quarantine.py`
- `test\_quarantine\_kclient.py`
- `quarantine\_thrasher.py`
- `qa/suites/fs/.../quarantine.yaml`
- `doc/cephfs/fs-volumes.rst`

PR: [ceph/ceph#68772](https://github.com/ceph/ceph/pull/68772) — cephfs: support subvolume quarantine

Key takeaways

Subvolume quarantine provides fast, persistent containment for ransomware response.

The inode optmetadata flag is the durable source of truth; managers coordinate enforcement work only.

Normal clients are blocked; q/Q recovery clients are explicitly authorized; `allow \*` does not bypass quarantine.

Multi-MDS, snapshots, mgr operations, kernel clients, restart/failover, and thrash scenarios are covered by tests.

Troubleshooting should start from state → caps → cap revocation / MDS propagation.

Thank You

Questions ?

