CMD3 Security Layering

Nikita Danilov <nikita@clusterfs.com>

2006.09.29

1 Introduction

This document describes how security infrastructure (capabilities, remote users, and remote acls) are integrated into new md server layering. Understanding of security code and new md stack is assumed.

2 Requirements

Capabilities, remote users and remote acls have to be integrated into new md stack. Specifically:

- The OST / OSD shall export a capability based authorization mechanism. When the OSD is used by the MDS such a scheme will not introduce any overhead.
- Future support for NFSv4 ACL's will be possible
- If any VFS context or similar context is used by a server, the server will setuid/setgid/segroups to the user context for which it is executing and chroot to the root of a file set in which it is operating for maximum reslience in case of un-resolved security problems. (So - don't run as root and chroot before you do anything.)
- Upcalls will not be made from a layer below any layer that can initiate transactions.
- All storage management software that is planned for use with an OSD, such as replay/replication logs, punching inodes, hot migrations etc will equally well work if that OSD is used by an MDS.
- Storage management software (backup, migration etc) designed for MDS use will only use OSD interfaces. Note that in particular this means that things like fid -> path reconstruction need to be considered carefully before they are used, because they may violate this.

- If an OSD is normally used by an MDS its state is not made inconsistent if it is temporarily used as an OST. I.e. OST functionality will avoid trashing MDD maintained state.
- The device method tables will be small.
- Security will be concentrated to be easily auditable.

3 Functional specification

Access control is implemented in two layers:

- fid-capability based mechanism at the level of osd, and
- uid/gid/acl based mechanism at the level of mdd.

Capability is a cryptographically protected datum that authorizes particular operation (or a set of operations) against given object. Servers generate capabilities according to their security policies and send them to other nodes (clients). Clients store capabilities, and can send them back to the originating server or other servers to perform operation authorized by capability. Lustre capabilities are "repeatable" that is, the same capability may be used multiple times (this is in contrast with some other security models where capability is automatically invalidated after corresponding operation is performed). Capability has expiration, after it expires it is no longer valid (will be refused by server).

Object, capability authorizes an operation against, is usually identified by fid. Alternatively object may be whole device (for operation of object creation and for administrative actions) identified by some other means.

Overall, fid-capability access control is based on [0], including mechanisms of capability and key expiration, key exchange, and capability verification.

uid/gid/acl access control machanism implements traditional POSIX style DAC. Following issues have to be considered:

- remote uid mapping. Clients and servers may assign different numerical identifiers to user and group principals. This implies that server has to be able to detect such situation (on a per-client basis), and to perform run-time uid conversion. This conversion is handled through user level upcall (i.e., performed by some user daemon of unspecified nature), and results of this conversion are cached on the server (see remote_uid_HLD.lyx, for details.)
- interoperability with fid-capabilitiy based control. Under certain conditions mdd has to perform an operation on object for which client provided no capability. For example, to handle an unlink, server (given only a capability for parent directory)

has to lookup child object and to decrease its hard link count. To implement this, mdd is granted special “bypass” capability which authorizes unlimited access to all objects. To achieve “least privilege” goal, finer grained “read-bypass”, “read-write-bypass”, etc., might be introduced.

4 Use cases

4.1 getattr

Client sends to mds a fid of target object together with META_READ capability. After performing POSIX DAC (which requires usage of bypass-capability), mdd calls ->do_attr_get() method of osd, pasing capability as an argument. osd performs capability verification.

4.2 getattr_name

Client sends to mds a fid of parent object, together with INDEX_LOOKUP capability and a name of target object. mdd uses INDEX_LOOKUP capability to lookup a name within parent. After POSIX DAC is performed on found target object, bypass capability is used to obtain its attributes.

4.3 setattr

Set same as getattr, except that META_WRITE capability is used.

4.4 link

Client passes fid of (new) parent directory together with INDEX_INSERT and META_WRITE capabilities, new name and fid of child object together with META_WRITE capability. mdd uses INDEX_INSERT capability to insert new name, and META_WRITE to adjust target hard link count.

4.5 intent and open

When handling intent, server gets INDEX_LOOKUP capability for the parent directory. After finding target object, server used bypass capability to obtain object attributes, and requests new capability from osd. This capability is returned to the client. mdd requests capability for operations determied by combination of user, on whose behalf intent is executed, and POSIX permission bits (and acl) of the target object. I.e., if POSIX would allows read access to the target, BODY_READ or INDEX_ITERATE operation is requested (depending on whether object is directory or not), similarly for write. META_READ and META_WRITE capabilities are returned all the time (because under POSIX semantics permission bits do not protect from modification of inode attributes).

Similarly, capability authorizing appropriate operations is returned by open.

4.6 object creation

mdd creates new object as part of open, create, mknod, and mkdir. Parent directory with INDEX_INSERT and INDEX_LOOKUP capabilities is received from client. Additionally, in mkdir case, META_WRITE capability on parent is required. Also, client passes device-wide OBJECT_ALLOC capability (alternatively this capability can be acquired by mdd at startup). mdd uses OBJECT_ALLOC capability to authorize object (inode) creation, INDEX_INSERT capability to insert new name into parent directory, and META_WRITE on parent to update hard link count in case of mkdir. Bypass capability is used to initialize new object (insert dot and dotdot in directory, initialize name in symlink, etc.)

4.7 unlink

Client sends parent directory together with INDEX_LOOKUP and META_WRITE capabilities, and a name to unlink. Server uses INDEX_LOOKUP to obtain target object. If target object is a directory, META_WRITE is used to decrease parent hard link count. Bypass capability is used to update target hard link count, and to destroy its body on last unlink.

4.8 recovery

During recovery, server recreates capabilities as necessary and sends them back to the client.

5 Logic specification

5.1 capability operations

Fid-capabilities:

- BODY_READ
- BODY_WRITE

- BODY_TRUNCATE
- META_READ
- META_WRITE
- INDEX_LOOKUP
- INDEX_INSERT
- INDEX_DELETE

Device-capabilities:

- OBJECT_ALLOC
- SYNC

5.2 changes to the interfaces

Data types for capability and an enumeration of operations are introduced.

dt_device_operations and dt_object_operations are modified to take capability.

New method ->do_capability_make() is added that creates new capability for a given object, and given set of operations (this method uses current working key for the current server).

6 State management

6.1 State invariants

Capability caches on server.

6.2 Scalability & performance

Remote-local uid mapping introduces some overhead. Clusters and clients that do not need this featuer do not pay for it.

6.3 Recovery changes

See Use Cases.

6.4 Locking changes

State transition in capability and key state machine should be serialized. This implies that key expiration, and capability verification have to protected by the same lock.

6.5 Disk format changes

If AV counter (see Alternatives section) is used, it has to be stored persistently for every object. AV counter can be added dynamically, and assumed to be 0 if absent, thus assuring seamless upgrade and downgrade.

6.6 Wire format changes

Capabilities are passed in client requests and server replies. Format changes.

6.7 Protocol changes

See previous section.

6.8 API changes

See Functional Specification section.

6.9 RPCs order changes

Not envisaged at this point.

7 Alternatives

- fid-capability based access control may be implemented as a separate level just above osd.
- POSIX DAC access control may be implemented as a separate level just above mdd.
- [0] mentions AV counter that can be used to forcible invalidate all capabilities for a given object.

8 Focus for inspections

- should OBJECT_ALLOC capability be passed from client?
- what operations should be allowed by capability returned by intent processing? This list should be carefully minimized.
- in particular, always granting META_WRITE assumes that _all_ inode attrubutes are mutable by client. Which is subtle for inode flags (APPEND, IMMUTABLE, etc.).
- do we want capability checking on oss nodes? This implies that our mds and oss capability formats should be compatible.
- how server failures interact with key expiration?

9 References

[0] *Security for Network Attached Storage Devices* Howard Gobioff, Garth Gibson, Doug Tygar (http://www.pdl.cmu.edu/PDL-FTP/NASD/CMU-CS-97-185.pdf)