1 MDS and OSS Capabilities

1.1 Introduction

Capabilities are pieces of data generated by one service, the **master service**, passed to a client and presented by the client to another service, the **slave service,** to authorize an action.

1.2 Specification

1.2.1 Feature control

For each of the MDS and OST the feature of capability checking and generation can be turned on or off with an lconf parameter and a /proc/sys variable.

The duration of validity can be tuned through /proc (important for acceptance tests).

1.2.2 Special user

The security context for a user, both on the MDS and OSS may indicate that the user has special priviliges for the target and can access fids and/or objects in the target without capabilities. If the MDS detects this, a 0 capability is generated.

1.2.3 MDS capabilities (not required for cmd2 - skip initially)

There are a few MDS capabilities:

1. A fid capability: this proves that a client obtained access to a fid
2. Replay capability for an operation associated with a fid.
   (a) At present only open for a fid (together with the opening mode) are important
   (b) Chdir and mount may be additional capabilities
3. Each request to the MDS will include the capability or capabilities associated with the fid(s) in that request.
4. Each reply to a lookup request or open request from the MDS contains a capability.

1.2.4 OSS capabilities

1. When objects for a file are created the MDS id and storage id of the file inode is stored in an EA in the object.
2. When a file is opened successfully a capability is sent to the client allowing create, read/write or read access to the objects.
3. The capability is verified by the OSS.

1.2.5 Capability Renewal

1. the client keeps capabilities in a list. When they are close (like 90%) to expiring they are renewed with the MDS. Notice that renewal of a capability is based on the original open, not on current permissions of a file.

1.2.6 Capability cryptography

HMAC-SHA1 function A MAC is used to prove that a capability was not modified since it was generated by a master. The size of the MAC is 128 bits.

We need a kernel level implementation of this function to compute hashes of capabilities, to protect their integrity. It seems that HMAC-MD5 or HMAC-SHA1 would be acceptable for this purpose.

capability cache Signing the capabilities is probably time consuming. A cache of fixed size (maybe some 1000's) of signed capabilities should be retained. To increase cache hits, timeouts in capabilities should be rounded to the nearest 1000 seconds.

Encryption Capabilities need to be encrypted on the wire so that they cannot be snooped. There are two mechanisms for encrypting capabilities:

1. Use the GSS infrastructure - this is the simplest solution, good for cmd2.
2. Build a special purpose encryption mechanism for capabilities.

When OSS capabilities are enabled both client - MDS and client - OSS connections need to use privacy on the GSS connection.

1.2.7 Managing and setting shared keys with slaves

The master and the slave need to share a key to verify the authenticity of capabilities, this key is called the **base key** and the base key must be identified so that it can be found by the slave. Slaves may have base keys from different masters. The capability will identify what key was used, but not contain the key.

The master will be an MDS, the slave will be an MDS (possibly the same server as the master) or an OSS.

1. The master will connect using GSS to the slave using a special principal allowing administrative actions.
2. At any time a slave will have one or two keys available for any master, labeled as red and black. Keys expire after a hard coded amount of hours, after which a new key will be generated by the master. To avoid a storm of new requests, the key that expired remains valid for another time interval and is then discarded. Hence, when a new key arrives on the slave:

   (a) the black key is discarded, with all its associated cache information
   (b) the red key becomes black
   (c) the new key is given the red label

3. The keys will be identified by the a master server number and an increasing key sequence number.
4. Each time a key expires, a new key is generated by the master the sequence number is increased in the MDS disk data.
5. The keys generated by the master service and sent, encrypted using GSS to the slave service.
6. This network command is an obd-ioctl or set info command that tells the OSS:
   (a) the new key
   (b) the identifier of the key
   (c) the duration for which the key will be used
7. When capabilities are verified both the red and the black key can be used.
8. When keys need to be found, they are stored in a hash table, based on the key identifier.
9. The keys should be stored in an on disk directory to avoid having to renew keys at recovery.

1.3 Logic

The design falls into natural parts

1. Capabilities without encryption for the OSS
   (a) capabilities are somehow shared between the mdc's and osc's
      i. mdc is responsible for renewal of the capability
      ii. osc is responsible to include capability in further request(s). The capability can be used many times.
   (b) the logical location of capability code is probably in the class driver: they might / should be used all over the place. Let's consider very carefully in the DLD how a capability is found, referenced and released, in conjunction with the cache of signed capabilities on the server nodes.
   (c) structure of the capability

```
struct lustre_capa {
```

```
u64  lcapa_objid[3];   // object/inode on what server
        u32  lcapa_operation;  // operation allowed
        u32  lcapa_expiry;     // expiry time: servers have clocks
        u32  lcapa_keyid;      // key used for the capability
        u32  lcapa_secflags    // security features for capa
}
```

1. Capabilities without encryption for the MDS
2. Renewing nearly expired capabilities
   (a) maintain a list for each user of capabilities
   (b) manage capabilities in the MDC / OSC, not
3. Signing capabilities and a cache of signed capabilities
4. Shared key management between master and slave capability nodes

1.4 Future work

Possibly the use of GSS connections is associated with serious overhead. This needs to be profiled.