HLD of Remote UID/GID Handling

Peter Braam, Eric Mei

Feb 10, 2005

1 From the ERS (Engineering Requirements Spec, formerly Architecture)

- Perform uid/gid translation between remote clients and local user database.
- Handling client program calling setuid/setgid/setgroups syscalls to get unusual previlege .
- Handling supplementary groups membership.
- Various security policies in situations with/without strong authentication like Kerberos V5.

NOTE:

- remote clients may have different user database from that of MDS's.
- The remote ACL issues is addressed by a separate module.
- Most content of this document has been described in Lustre Book.

The architecture prescribes a translation mechanism at the MDS: the MDS will translate a locally found uid/gid, which is obtained through the kerberos principal.

2 Functional Specification

2.1 Determine local/remote clients

- "local" client is the client node which is supposed to share the same user database with MDS's.
- "remote" client is the client node which is supposed to have different user database from MDS's.

The MDS's will be able to determine that a client node is a local or remote one, upon the client's first connection time to the MDS, and reply back it's decision to client. Later both MDS and client will make different operation decision according to this flag. This remote flag is per-client, not per user. Once MDS made the decision, it will keep unchanged until client leave the cluster membership (umount or so).

MDS will do many conversion (mostly uid/gid mapping) for users on remote clients because of the user database mismatch, and due to the nature of this mismatch we have to put some limitation on users of remote clients, compare to local clients. Following sections have the details description.

2.2 Mapping uid/gid from clients

For local client, obviously we don't need do any uid/gid mapping. For remote clients, we need translate uid/gid in each request into one which lives in local user database; and vice versa: translate uid/gid in reply into the one in remote user database. This translation affects the uid/gid's found in the inode as owner/group, the security context which describes under what uid the MDS is executing and in some cases (chown is a good example) the arguments of calls.

Each MDS will have to access a uid-mapping database, which prescribed that: which principal from which nid/netid should be mapped to which local uid. The mapping database must be the same to every MDS to get consistent result. During runtime, when a remote user authenticated with the MDS, the corresponding mapping entry will be read from the on-disk database and cached in the kernel via an upcall. Note the same principal from different clients might be mapped to different local user, according to the mapping database. So on each MDS there's a per-client structure which maintained the uid mapping cache.

Each remote client must have nllu/nllg installed. 'nllu' is for "Non Local Lustre User", while 'nllg' for "Non Local Lustre Group". When client firstly mount a lustre fileset, it should notify MDS which local uid/gid act as nllu/nllg. MDS will translate those unrecognized uid/gid to this before send reply to client. Thus from client's perspect of view, those files which belong to unauthorized users will be shown as belonging to nllu/nllg.

2.3 Lustre security description (LSD)

There's a security configure database on each MDS, which describes who(uid) from where(nid/netid) have permission to setuid/setgid/setgroups. Later we might add more into it. the database must be the same to every MDS to get consistent result.

LSD refers to the in-kernel data structure which describe an user's security property on the MDS. It roughly be defined as:

```
struct lustre_sec_desc {
    uid_t           uid;
```

```
        gid_t           gid;
        supp_grp_t      supp_grp;
        setxid_desc     setxid;
        /* more security tags added here */
    };
```

In the future we'll add more special security tag into it. Each LSD entry correspond to an user in the local user database. The 'setxid_desc' must have the ability to describe setuid/setgid/setgroups permission for different clients respectively. In any cases remote user can not setgroups.

LSD cache is populated via an upcall during runtime. The user-level helper will be feed in uid as a parameter, and found out this uid's principal gid and supplementary groups from local user database, and find setxid permission bits and other security tags from on-disk security database.

Each LSD entry have limited expiration time, and will be flushed out when expired. Next request come from this user will result in the LSD be populated again, with the uptodate security settings if changed. System administrator also could choose to flush certain user's LSD forcely.

Every filesystem access request from client need go through checking of LSD. This checking is uid based, for those request coming from remote client, uid will be mapped at first as described above, and then go to LSD.

2.4 The MDS security context

All kernel-level service threads running on MDS are running as root, waiting request from other nodes, and provide services. But for those request to access filesystem for certain user, those threads must act as the user, running as its identities. Thus such a request comes in, we firstly collect the identity information for this user as above described, include uid, gid, etc., then switch the identity in the process context before really execute the filesystem operation; we also need switch the root directory of process to the root of MDS's backend filesystem. after it finished, we switch back to the original context, prepare to the next service.

For some request for special service like llog handling, special interaction between MDSs, which don't represent any certain user, and require keeping the root privilege. In those situation we don't need do such context switch, also user identity preparation.

2.5 Remote client cache flushing

For a remote client, it should realize that those locally cached file's owner information, e.g. owner, group, is ever translated by server side, some mapping might be stale as time goes on. for example: a user newly authenticated, while some cached file which should be owned by him still shows owner is "nllu". client must choose the proper time to flush those stale owner informations, to give user a consistent view. All attribute locks held by clients must be given a revocation callback when a new user connects.

3 Use Cases

3.1 Connect rpc from local realm (case 1)

1. Alice doing 'mount'
2. Alice sends the first ptlrpc request (MDS_CONNECT) without GSS security to MDS;
3. mds_handle() will initialize per-client structure, clear the remote flag in it;
4. After successful connection done, the MDS send the remote flag back to client for future usage in client side.

3.2 Connect rpc from local realm (case 2)

1. Alice doing 'mount'
2. Alice from a MDS local realm sends the first ptlrpc request (MDS_CONNECT) with GSS security to MDS;
3. MDS svcgssd will determine it's from a local realm client;
4. mds_handle() will initialize per-client structure, clear the remote flag in it;
5. After successful connection done, MDS will send the remote flag back to client for future usage in client side.

3.3 Connect rpc from remote realm

1. Alice from a MDS remote realm sends the first ptlrpc request (MDS_CONNECT) with GSS security to MDS, along with its nllu/nllg id number;
2. MDS svcgssd will determine it's from a remote realm client;
3. mds_handle() logic will initialize per-client structure:
   (a) Set the remote flag in it;
   (b) Fill in the nllu/nllg ids obtained from client rpc request;
4. After successful connection done, the MDS will send the remote flag back to client for future usage in client side.

3.4 Filesystem access request

1. Alice (from local or remote client) try to access a file in lustre
2. If Alice is from remote client, MDS do uid/gid mapping; otherwise do nothing
3. MDS obtain LSD item for Alice
4. MDS perform permission check, based on LSD policies.
5. MDS service process switch to this user's context
6. MDS finish the file operation on behave of Alice.
7. MDS service process switch back original context
8. If Alice is from remote client, MDS do uid/gid reserve mapping if needed.
9. MDS send reply.

3.5 Rpc after setuid/setgid/setgroups from local clients

1. Alice calls setuid/setgid/setgroups to change her identity to Bob in local client node X;
2. Bob (Alice in fact) tries to access a lustre file which belongs to Bob;
3. MDS will verify the permission of Bob through local cached LSD configuration;
4. MDS turns down or accept the file access request;

3.6 Rpc after setuid/setgid/setgroups from remote clients

1. Alice calls setuid/setgid/setgroups to change her identity to Bob in remote client node Y;
2. Bob (Alice in fact) tries to access a lustre file which belongs to Bob;
3. MDS will find Bob is from the remote realm and in fact he is not real Bob;
4. MDS turns down the file access request;

3.7 Update LSD configuration in MDS

1. Lustre system administrator hopes to update current LSD option;
2. The sysadmin uses the lsd update utility which will update the on-disk security database, and notify the changes of the LSD configuration to MDS;
3. MDS re-fresh the cached LSD info through an upcall.

3.8 Revoke a local user

1. Bob is able to access lustre filesystem
2. Sysadmin remove Bob from the MDS's local user database, and flush in-kernel LSD cache for Bob.
3. Bob will not be able to access MDS immediately

3.9 Revoke a remote user

1. Alice of a remote client is mapped to MDS local user Bob.
2. Alice is able to access lustre filesystem
3. Sysadmin remove the mapping "Alice->Bob" from mapping database, and flush in-kernel mapping entry.
4. Alice will not be able to access MDS immediately.
5. If the mapping "anyone else -> Carol" exist in the mapping database, Alice could reconnect to MDS and then will be mapped to Carol.

3.10 Revoke a remote user (2)

1. Alice of a remote client is mapped to MDS local user Bob.
2. Alice is able to access lustre filesystem
3. Sysadmin remove Bob from the MDS's local user database, and flush in-kernel LSD cache for Bob.
4. Alice will not be able to access MDS immediately.
5. If the mapping "anyone else -> Carol" exist in the mapping database, Alice could reconnect to MDS and then will be mapped to Carol.

3.11 'ls -l' on remote client

1. Suppose on a remote client, Alice's pricinpal group is AliceGrp; Bob's principal groups is BobGrp.
2. there's several files on lustre: file_1 belongs to Alice:AliceGrp; file_2 belongs to Alice:BobGrp; file_3 belongs to Bob:AliceGrp; file_4 belongs to Bob:BobGrp; file_5 belongs to Bob:nllg;
3. Alice do 'ls -l', output like this: file_1 belongs to Alice:AliceGrp; file_2 belongs to Alice:nllg; file_3 belongs to nllu:AliceGrp; file_4 belongs to nllu:nllg; file_5 belongs to nllu:nllg;

4. Bob just login the client system, also do a 'ls -l', output like this: file_1 belongs to Alice:AliceGrp; file_2 belongs to Alice:Bobgrp; file_3 belongs to Bob:AliceGrp; file_4 belongs to Bob:BobGrp; file_5 belongs to Bob:nllg;
5. Alice do 'ls -l' again, output is the same as Bob's list.
6. Alice logout, then Bob do a 'ls -l' again, output like this: file_1 belongs to nllu:nllg; file_2 belongs to nllu:Bobgrp; file_3 belongs to Bob:nllg; file_4 belongs to Bob:BogGrp; file_5 belongs to Bob:nllg;

3.12 Chown on remote client

1. Root user on a remote client want to change the owner of a file to Bob, while Bob didn't login(authenticated with lustre) yet.
2. MDS can't find the mapping for the destinated uid, so return error.
3. Bob login at that time.
4. Root do the same chown again.
5. MDS will grant the request, no matter what the original owner of this file is.

3.13 Chgrp on remote client

1. Triditional chgrp on remote client is not allowed, since there's no clear group id mapping between local and remote database. so the group id on the remote client is not meaningful on the MDS.

4 Logic Specification

4.1 Specify nllu/nllg

When client do mount, in addition to other parameter, user need supply with the IDs of nllu/nllg on this client, which will be sent to the MDS at connecting time. If no nllu/nllg explicitly supplied, default values will be used.

4.2 Determine local or remote client

Under GSS protection, user could explicitly supply the remote flag during mount time. MDS make decision as following order:

- All permitted connections without GSS security are from local realm clients.
- All connections with GSS security, if user supplied remote flag during mount, MDS will grant the flag as requested.

- All connections with GSS/local_realm_kerberos are from local realm clients.
- All connections with GSS/remote_realm_kerberos are from remote realm clients.

Here we made the assumption that: kerberos's local/remote realm == lustre's local/remote realm. Later we might bring in more factors into this dicision making.

GSS/Kerberos module is responsible to provide the information that the initial connect request whether has strong security; whether from remote kerberos realm.

On MDS's, the per-client export structure has a flag to indicate local/remote of this client. Accordingly, each client has a similar flag, which is send back by MDS's after initial connection.

4.3 Handle local rpc request

For each filesystem access request from client, we will get LSD for this uid at first. We then lookup in the cache, if not found or already invalid, issue a upcall to get it. If finally failed to get LSD(timeout or got an error), we simply deny this request.

After obtained LSD, we also check whether the client intend to do setuid/setgid/setgroups. If yes, check the permission bits in LSD, if not allow we also deny this request. The intention of setuid/setgid could be detected by compare the uid, gid, fsuid, fsgid sent by client, and the local authorized uid/gid.

If setgroups is permitted: for root we'll directly use the supplementary groups array sent by client; for normal user we compare those sent by client with those in LSD, guarantee client only could reduce the array (can't add new ids which is not part of group array in LSD).

If setgroups is not permitted, we simply use the supplementary group array provided by LSD.

After all security context prepared as above, we switch it into process context, perform the actual filesystem operation. after finished, switch back the original context. send reply out to client.

Later an special security policy is needed to allow RAW access by FID without a capability. This is used for analyzing audit logs, finding pathnames from fids (for recovery) etc.

4.4 Remote user mapping database

There will be a user mapping configuration file on MDS, already defined in "functional specification". MDS kernel will also maintain a cache of this mapping information. It is populated by upcall to server side gss daemon, along with the gss credential information.

- The on-disk mapping database only described how user(principal) is mapped to an local uid, and don't need specify the gid mapping.
- Both on-disk mapping database and kernel mapping cache should be able to allow map all other remote users to a certain local user.
- On the MDS, the per-client structure will maintain this mapping cache. When a user from remote client get authenticated, we check the on-disk mapping database. If no mapping items for this user found, we'll deny this user. otherwise we record the target uid.
- When a fs access request come from remote client, it contains the user's uid, gid on the remote client. Here we can establish mapping for uid and target uid. With target uid we can find the target gid from local user database (from LSD), thus we can also establish the mapping for gid and target gid.
- With mapping we established above, we now do the mapping: replace the uid/gid in the rpc request with target uid/gid. If it request chown we also check & map the new owner id.
- When reply populated and about to send back, we again check the mapping cache, and do the reverse mapping if in the case which return file attributes to clients. For those can't find the matched items, map them to nllu/nllg of this remote client.
- For the reverse mapping, if we map a owner/group of a inode to nllu/nllg, we should also adjust the mode bits. Namely the permission bits of "other" category will be apply to "user" or "group" category. For inode which contains ACL, we may not need special adjustment since real checking will be based on cached permission on client instead of simply by mode bits.

This approach is simple, but have some drawbacks here. Following situation will cause confusion on MDS's id mapping:

- Several principals on a remote client mapped to a single local user.
- Multiple users on a remote client logon lustre using single principal.
- A user on remote client logon lustre using several principals simultanously.
- Not well choosed mapping database cause gid mapping confusion.

Currently we simply assume that it is sysadmin's responsibility to prevent all above situation from happen, by carefully choosing user mapping. And we also assume that even the remote client is "trustable" to some extend, i.e. remote users can't change uid/gid settings arbitrarily, and each user only could logon lustre via their own kerberos account. Some good practice of user mapping choosing is like:

- A user on remote client belongs has his own group (no other users share same principal group with him), then we choose a similar local user to map to.
- A groups of users on remote client has the same principal group id, then we create a similar group of users locally, and map remote users to this one by one.
- Don't map more than one remote users to a single local user.
- Map "all other users" to one local user is OK (if allowed).

4.5 Handle remote rpc request

The overall process of handle remote rpc request is the same as for local user, except following:

- For incoming request, firstly do the uid/gid mapping for the requestor; and do reserve mapping for the reply, as described above.
- No setuid/setgid/setgroups intention is permitted, except we explicitly allow setuid-root in setxid database.
- No supplementary groups will participate in the permission checking process. We ignore the supplementary groups sent by remote client, even ignore the one in LSD. So remote user only have uid and one gid as their identity.
- For chown request, we also do translation for the new owner id (already described above) according to the in-kernel mapping cache. It means the root user on remote client can't change owner of a file to a user which is not login yet.
- Deny all chgrp request, since the group on remote client has no clear mapping on MDS's local user database (We also could choose allow this when the new group id showup in the in-kernel mapping cache, but it seems dosen't make much sense). So we probably need a special tool like "lfs chgrp" to perform chgrp on remote client, which will send out text name instead of translate to id locally.

4.6 Remote client cache flushing

Anytime there might be inodes cached and their owner belongs to nllu/nllg. If a new user Alice get authenticated and she happens to be the owner of those inodes, we need to refresh those inode even if it's cache status is correct, otherwise Alice will find her files belong to others. Since we don't know whether a inode with nllu/nllg belongs to Alice or not, we must flush all of them.

On MDS, a callback or similar event notification mechanism should be hooked into gss module. When a user authenticated at the first time, we should iterate through all the granted lock corresponding to this client, and revoke them selectively. Strictly speaking we only want to revoke those inodebits lock and the owner/group of their resource (inode) not show up in the in-kernel mapping database, but here we just flush all the inodebits locks, a cache is quickly re-populated - there are a maximum of 20-100 cached locks on clients at the moment.

When Alice logs out of the client system, we also do the similar things: iterate through all the granted inodebits locks corresponding to this client and revoke them.

4.7 LSD upcall

There is a general upcall-cache code which do upcall into user space, and cache data passed down in kernel, and also implemented timeout invalidation. Kernel LSD could simply be implemented as a instance of it. So it will be quite simple.

A user-space tools should provide following functionality:

- Accept uid as parameter
- Obtian gid and supplementary groups id array which the uid belongs to, if failed just return error.
- Obtian the setxid permission bits for this user on this NID from database. If not found a default bitset will be applied: (1) for local client: setuid/setgid is off, setgroups for root is off, setgroups for normal user is on; (2) for remote client: all of setuid/setgid/setgroups is off.
- Pass all the collected information back to kernel by /proc.

Since the upcall could happen concurrently, and admin could modified it at anytime, so a kind of read-write lock need to be done on the database file.

4.8 Recovery consideration

All the code here should have minimal effect on recovery. After MDS's crash, security context will be established during connection time in recovery; and uid-mapping cache and LSD actually are "adaptive", they will also be re-populated when handling related user's replay request during/after recovery.

5 State Management

5.1 configuration states

- Client has a remote flag at mount time.
- Remote clients must have nllu:nllg installed. it could simply be nobody:nobody.

- MDS could have a remote-user mapping database which contains which principal at with client should be mapped to which local user. Without the database no remote client is allowed to connect.
- MDS could have a security database which contains setxid permissions along with other security setting for each affected user. No such database then a default setting will be applied.

5.2 LSD entry states transition

1. NEW: generated and submit to upcall
2. READY: ready to serve
3. INVALID: expired or error

Requestor will initiate an NEW LSD entry; after upcall successfully fill in data it change to READY; if timeout or some error happen (e.g. not found in user database) during upcall it change to INVALID; a READY LSD will change to INVALID when expired, or flushed forcely by sysadmin, or MDS shutdown; an INVALID LSD will be soon destroied.

No disk format changed. When a large number of users access lustre from all kinds of local/remote clients at the same time, MDS will have more CPU and memory overhead, especially for remote users. No special recovery consideration.

6 Alternatives

6.1 NFSv4

NFSv4 sends user and groups by name.

7 Focus of Inspection

- Could this pass HP acceptance test?
- Any is not reasonable? Any security hole?
- Everything recoverable from MDS/client crash?