A lightweight access control mechanism for Lustre in wide area domains

Thomas Stibor
t.stibor@gsi.de

High Performance Computing
GSI Helmholtz Centre for Heavy Ion Research
Darmstadt, Germany

Monday 16th September, 2013

LAD’13 Workshop, Paris, France
Motivation

Lustre employed in wide area networks (WAN) can result in UID/GID conflicts (overlaps) and thus in uncontrolled data modification and deletion.

For the sake of simplicity GID is omitted here and in some remaining slides.
• Users of UID-space \( A \) access data in UID-space \( B \) under their UID of \( A \).

• Use reserved UID-space, \textit{however} we have \textit{no} mechanism to control whether reserved UID-space is \textit{truly employed}.  

reserved UID-space
Lustre UID Mapping (Problem)

What happens when frequently users are deleted, added, etc. and the mapping is not updated (keep synchronized)?

Suppose each domain $A, C, D, \ldots$ have $\geq 1000$ many Lustre users. We have to maintain in central domain (here $B$) $|A| + |C| + |D| + \ldots$ many mappings (e.g. $a_2 \mapsto b_7, a_6 \mapsto b_9, \ldots$).

- This can be problematic in large scale environments.
Lustre Reserved UID-space Access Control

Control data access directly in Lustre MDT-Layer based on:

- Network address (range) e.g. 10.[1-8].1.[1-128]
- Network type e.g. tcp0 or ib0
- UID/GID (range) e.g. [0-32000] [100-500]
Lustre Reserved UID-space Access Control (cont.)

- Do not have to specify a mapping for every single UID and GID. Use ranges e.g. [40000-50000].
- Enforce that UID’s and GID’s of reserved space are taken only.

**Summary:** Access control based on:

- Network address,
- Network type,
- UID, GID

For realizing this approach a Lustre kernel module called `LustreUsrGrpAccessControl` (short `lugac.ko`) is developed.
LUGAC Kernel Module Usage

Load/unload module (is automatically loaded by mdt.ko dependency):

>insmod ./lugac.ko
[12778.295442] GSI Lustre UID/GID access control module lugac.ko version 0.3beta loaded
>rmmod lugac
[12793.754416] GSI Lustre UID/GID access control module lugac.ko version 0.3beta unloaded

Write rules:

>echo "192.168.[67-70].[1-16]@tcp0 [500-600] 1012" > /proc/lugac
>echo "10.10.1.1@tcp5 [100-200] [100-200]" > /proc/lugac

Read rules:

>cat /proc/lugac
[13297.755041] Listing GSI Lustre UID/GID access rules:
[13297.755049] 10.10.1.1@tcp5 [100-200] [100-200]
[13297.755059] 192.168.[67-70].[1-16]@tcp0 [500-600] 1012

Flush (delete) all rules:

>echo "flush" > /proc/lugac
[13402.185049] Deleting all GSI Lustre UID/GID rules.
>cat /proc/lugac
Listing GSI Lustre UID/GID access rules:
LUGAC Kernel Module Details

- Access control information are represented as C structs and stored as nodes in a linked-list. Linux kernel provides linked-list data-structure for “free” (see `#include <linux/list.h>`).

```c
typedef struct {
    gid_t from;
    l_network_t l_network;
    gid_t to;
    struct uid_interval_t uid_iv;
} gid_interval_t;

struct l_network_uid_gid_t {
    ...}

struct l_network_uid_gid_t {
    ...}

... struct l_network_uid_gid_t {...}
```

```c
struct l_network_uid_gid_t {
    l_network_t l_network;
    gid_interval_t uid_iv;
    gid_interval_t gid_iv;
    struct list_head next;
};
```
Kernel module `lugac.ko`

- parses input strings via `/proc/lugac` and sets fields in `struct l_network_uid_gid_t`,
- iterates over linked-list and outputs fields in `struct l_network_uid_gid_t` (cat `/proc/lugac`),
- adds `struct l_network_uid_gid_t` into linked-list,
- deletes `struct l_network_uid_gid_t` from linked-list,
- exports a function (`allow_access_nugid`) which tells whether:
  - Network address,
  - Network type,
  - UID, GID
  is a member (of the interval/range) of the linked list.
- is documented with doxygen `lugac_module.c#doxygen`
LUGAC Kernel Module Integration into Lustre

Only tiny patches in MDT-Layer are required, e.g.

```c
/* lustre/mdt/mdt_open.c */
int mdt_reint_open(struct mdt_thread_info *info, struct mdt_lock_handle *lhc)
{
    ...
    /* Lugac access control based on nid, uid and gid. */
    if (!allow_access_nugid(libcfs_nid2str(mdt_info_req(info)->rq_peer.nid),
                           uc->uc_uid, uc->uc_gid)) {
        CDEBUG(D_INFO, "Deny access for %s, uid: %d, gid: %d due to
               missing entry in access control list\n",
               libcfs_nid2str(mdt_info_req(info)->rq_peer.nid), uc->uc_uid, uc->uc_gid);
        GOTO(out, result = -EPERM);
    }
    ...
}

/* lustre/mdt/mdt_reint.c */
static int mdt_md_create(struct mdt_thread_info *info)
{
    ...
    /* Lugac access control based on nid, uid and gid. */
    if (!allow_access_nugid(libcfs_nid2str(mdt_info_req(info)->rq_peer.nid),
                           uc->uc_uid, uc->uc_gid)) {
        CDEBUG(D_INFO, "Deny access for %s, uid: %d, gid: %d due to
               missing entry in access control list\n",
               libcfs_nid2str(mdt_info_req(info)->rq_peer.nid), uc->uc_uid, uc->uc_gid);
        GOTO(out_put_parent, rc = -EPERM);
    }
    ...
}
```

See [http://www.stibor.net/lugac/](http://www.stibor.net/lugac/) for documentation and more details.
Other Approaches (UID Mapping)


- An extended version will be available in Lustre 2.6.0 (see also LAD’13 Developing UID Mapping and a Stand Alone Security Mechanism for Lustre: Challenges and Successes).
Other Approaches (Kerberos Realm Mapping)


In Lustre code:

```c
lustre/utils/gss/lsupport.h:#define MAPPING_DATABASE_FILE "/etc/lustre/idmap.conf"

/* lustre/utils/gss/lsupport.c */
static int read_mapping_db(void) {
    char princ[MAX_LINE_LEN];
    char nid_str[MAX_LINE_LEN];
    char dest[MAX_LINE_LEN];
    char linebuf[MAX_LINE_LEN];
    char *line;
    lnet_nid_t nid;
    uid_t dest_uid;
    FILE *f;
    ...
    /* copernicus@ANDROMEDA.GALAXY 10.67.75.100@o2ib 1001 */
    if (sscanf(line, "%s %s %s", princ, nid_str, dest) != 3) {
        printerr(0, "mapping db: syntax error\n");
        continue;
    }
    ...
}
```

See userland tool lustre/utils/gss/l_idmap.c
1) Lustre Kerberos code needs to be cleaned up and improved:

```bash
thomas@lxdv65:~/lustre> grep -r "XXX Hack alert"
lustre/utils/gss/context_mit.c: * XXX Hack alert! XXX Do NOT submit upstream!
lustre/utils/gss/context_mit.c: * XXX Hack alert! XXX Do NOT submit upstream!
lustre/utils/gss/context_mit.c: * XXX Hack alert! XXX Do NOT submit upstream!
lustre/utils/gss/context_mit.c: * XXX Hack alert! XXX Do NOT submit upstream!
lustre/utils/gss/context_mit.c: * XXX Hack alert! XXX Do NOT submit upstream!
lustre/utils/gss/context_mit.c: * XXX Hack alert! XXX Do NOT submit upstream!
lustre/utils/gss/context_mit.c: * XXX Hack alert! XXX Do NOT submit upstream!
lustre/utils/gss/context_mit.c: * XXX Hack alert! XXX Do NOT submit upstream!
lustre/utils/gss/context_mit.c: * XXX Hack alert! XXX Do NOT submit upstream!
lustre/utils/gss/context_mit.c: * XXX Hack alert! XXX Do NOT submit upstream!
lustre/utils/gss/context_mit.c: * XXX Hack alert! XXX Do NOT submit upstream!
lustre/utils/gss/context_mit.c: * XXX Hack alert! XXX Do NOT submit upstream!
lustre/utils/gss/context_mit.c: * XXX Hack alert! XXX Do NOT submit upstream!

lustre/utils/gss/context_lucid.c:/* XXX Hack alert! XXX Do NOT submit upstream! XXX */
lustre/utils/gss/context_lucid.c:/* XXX Hack alert! XXX Do NOT submit upstream! XXX */
lustre/utils/gss/context_lucid.c:/* XXX Hack alert! XXX Do NOT submit upstream! XXX */
lustre/utils/gss/context_lucid.c:/* XXX Hack alert! XXX Do NOT submit upstream! XXX */
lustre/utils/gss/context_lucid.c:/* XXX Hack alert! XXX Do NOT submit upstream! XXX */
lustre/utils/gss/context_lucid.c:/* XXX Hack alert! XXX Do NOT submit upstream! XXX */
lustre/utils/gss/context_lucid.c:/* XXX Hack alert! XXX Do NOT submit upstream! XXX */
lustre/utils/gss/context_lucid.c:/* XXX Hack alert! XXX Do NOT submit upstream! XXX */
lustre/utils/gss/context_lucid.c:/* XXX Hack alert! XXX Do NOT submit upstream! XXX */
lustre/utils/gss/context_lucid.c:/* XXX Hack alert! XXX Do NOT submit upstream! XXX */
lustre/utils/gss/context_lucid.c:/* XXX Hack alert! XXX Do NOT submit upstream! XXX */
lustre/utils/gss/context_lucid.c:/* XXX Hack alert! XXX Do NOT submit upstream! XXX */
lustre/utils/gss/context_lucid.c:/* XXX Hack alert! XXX Do NOT submit upstream! XXX */
lustre/utils/gss/context_lucid.c:/* XXX Hack alert! XXX Do NOT submit upstream! XXX */
```

2) With hardware accelerated crypto instruction set (such as AES-NI) Lustre Kerberos bottlenecks cf. [1,2] can be attacked.
Summary

A lightweight access control mechanism for Lustre in wide area domains based on

- Network address,
- Network type,
- UID, GID

is developed.

Drawbacks, constraints and improvements:

- Force WAN domains to use predefined UID/GID spaces,
- From perspective of information security (plain IP) not secure (use IP-Sec as underlying protocol for securing IP).
- Employ efficient data-structures such as red-black trees (#include <linux/rbtree.h>) or hashing functions.
- Integrate /proc/lugac into proper Lustre proc namespace.
Outlook

Demand for employing Lustre in WAN is growing!

My personal view to tackle this demand: Cleanup Kerberos code to supply:

- Strong authentication and encryption by means of Kerberos.
- Kerberos Cross Realm UID/GID mapping and UID/GID access control.

Thank you & Questions?