LUSTRE USER GROUP 2023

Secure Namespace Isolation in Lustre

A Customer Use Case

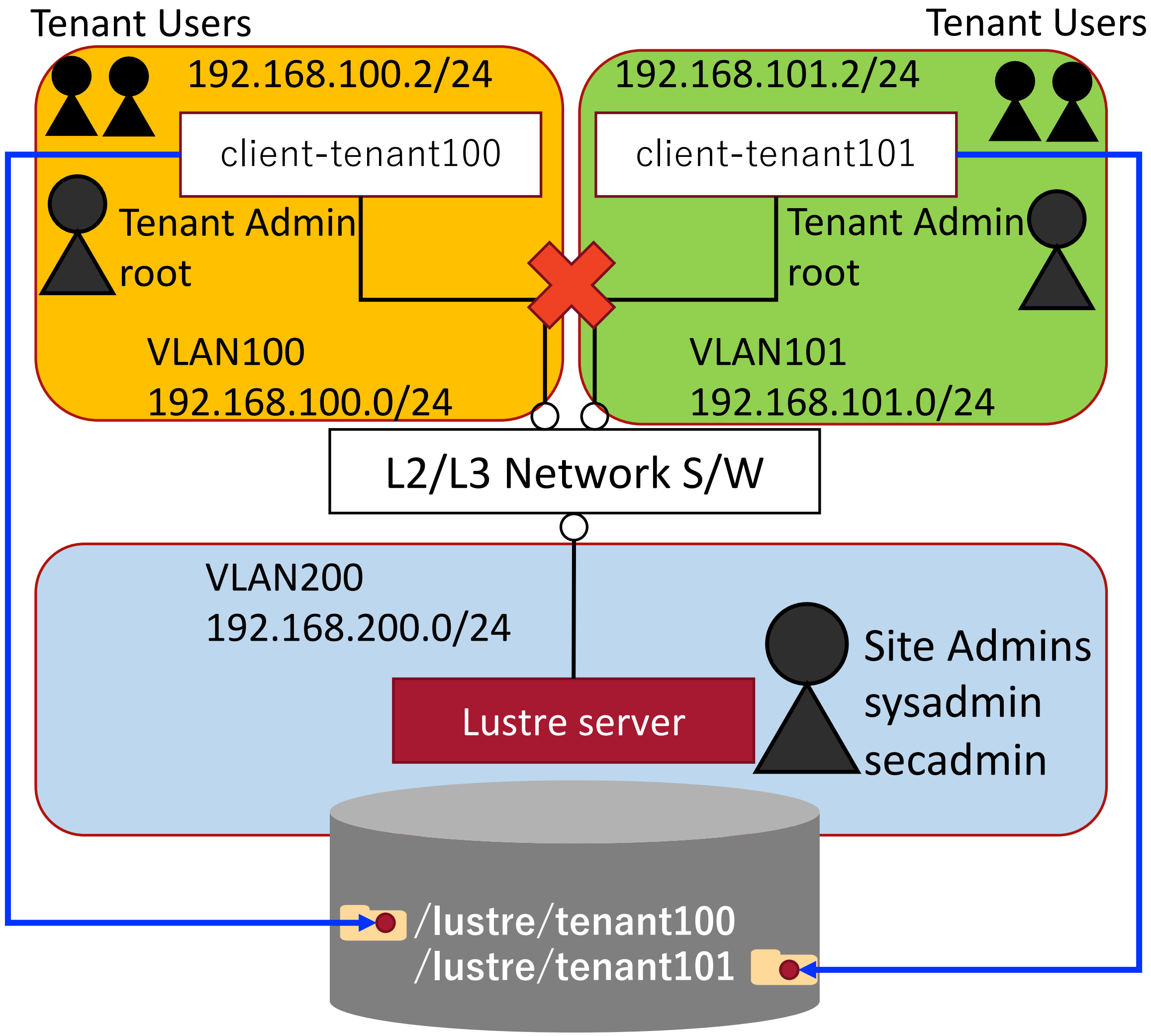
Shuichi Ihara

Background and Motivations

- File system consolidation
  - Merge multiple Lustre for organizations (or groups) into a single file system for efficiency
  - Keep data separation among organizations and groups
- Client, network, and storage isolation
  - Create isolated and limited storage space in a single Lustre file system for groups/projects
  - Restricted clients only access to a dedicated storage space of Lustre through isolated network
  - A storage space needs to be secure (restricted, isolated and encrypted) and controllable capacity
- Lustre provides multiple unique security features
  - Feature can be enabled individually as well as in conjunction with other security features

Provide a step-by-step process to a comprehensive secure namespace solution leveraging multiple Lustre features.

Requirements and Demonstration



Assumption: Two independent "Organizations" use a shared Lustre from two different VM groups (called "Tenant")

- Administrator Roles
  - System, Security and "Tenant" administrator
  - Restricted commands with sudo or RBAC
- Server and network isolation
  - Done by network configurations
- Storage isolation
  - Allocate storage capacity per "Tenant"
  - No filesystem ROOT directory access from clients
    - Instead, each tenant has separate subdirectory as sub namespace
    - Restricted clients only access to dedicated sub namespace
  - UID/GID management
    - Tenant admin manages user/group per tenant with own UID/GID policy
  - All files and directories are encrypted

Step1(Separation) : Create storage space per tenant

- Create sub-directories for each tenant

```
[sysadmin@mgmt ~]$ sudo mkdir /lustre/tenant100
[sysadmin@mgmt ~]$ sudo mkdir /lustre/tenant101
[sysadmin@mgmt ~]$ sudo touch /lustre/tenant100/welcome-tenant100
[sysadmin@mgmt ~]$ sudo touch /lustre/tenant101/welcome-tenant101
```

- Assign project ID to created directories and set project quota for capacity limit

```
[sysadmin@mgmt ~]$ sudo lfs project -srp 100 /lustre/tenant100
[sysadmin@mgmt ~]$ sudo lfs project -srp 101 /lustre/tenant101
[sysadmin@mgmt ~]$ sudo lfs setquota -p 100 -B 1T /lustre
[sysadmin@mgmt ~]$ sudo lfs setquota -p 101 -B 2T /lustre
```

- Would also create small scripts for tenant storage management by sysadmin
  - e.g.) `sudo mkdir.sh <tenant name>`
  - e.g.) `sudo lfs-set-projectid.sh <PID> <tenant_SUBDIR>`

  Instead of allowing native commands in sudo

Consideration: Capacity management for UID=0

- No root squash works in this case
  - Tenant administrator requires UID=0 privileges on tenant to manage of own users/groups
  - In the end, UID=0's files exist in "Tenant" directory (e.g. `/lustre/tenant100`)
- No USER/GROUP/PROJ Quota available for "root" account
  - Quota accounting always enables, but no quota enforcement on servers for UID=0 (No send -EDQUOT)
  - Regardless UID=0, project quota works in XFS, but isn't supported in EXT4/Lustre
- Patch "`LU-16415 quota: enforce project quota for root`" to the rescue
  - New "`osd-ldiskfs.*.quota_slave.root_prj_enable=1`" (default 0) enforces project quota for UID=0

```
[root@mgs ~]# lctl set_param -P osd-ldiskfs.*.quota_slave.root_prj_enable=1
```

  - Capacity management per tenant can be done by project quota for all files include UID=0
  - Landed patch in master for lustre-2.16

Step2(Restriction): Apply required client attributes

- Lustre Nodemap configures file system specific attributes per NIDs
  - (no) root squash and UID/GID mapping
  - Squash mode (UID, GID or PRJID or all), squash id (include UID/GID mapping) and squash behavior
  - Fileset
  - etc.
- Key nodemap configurations for clients in tenant

```
[root@mgs ~]# lctl nodemap_add tenant100
[root@mgs ~]# lctl nodemap_add_range --name tenant100 --range '192.168.100.[1-254]@tcp’
[root@mgs ~]# lctl nodemap_modify --name tenant100 --property admin --value 1
[root@mgs ~]# lctl nodemap_modify --name tenant100 --property trusted --value 1
[root@mgs ~]# lctl nodemap_modify --name tenant100 --property squash_projid --value 100
[root@mgs ~]# lctl nodemap_modify --name tenant100 --property map_mode --value projid
[root@mgs ~]# lctl nodemap_set_fileset --name tenant100 --fileset /tenant100
```

Full nodemap configurations described in Appendix.

Consideration : Minimize root privileges for Lustre

- Prevent Lustre commands by root user from nodemap clients
  - All root required Lustre commands not allowed If UID=0 is squashed
  - Privileged Lustre commands by tenant admin (no root squashed) need to be prohibited
- A new nodemap Role Based Admin Control property “`rbac`” introduced
  - `rbac` property is a mask to allow RBAC capability on nodemap clients
  - Supported roles currently cover main functional areas:
    - `byfid_ops` (FID), `chlg_ops` (Changelogs), `dne_ops` (DNE), `quota_ops` (Quota)
    - `fscrypt_admin` (Encryption) and `file_perms` (File Permission)

```
[root@mgs ~]# lctl nodemap_modify --name tenant100 --property rbac --value file_perms
```

  - Landed patch in master for lustre-2.16

Step3: Mount Lustre from clients on the tenant

Mount Lustre from clients at tenants with same syntax, but it’s isolated namespace by Fileset

- Clients at tenant100

```
 [root@tenant100-client ~]# mount -t lustre 192.168.200.2@tcp:/lustre /lustre
 [root@tenant100-client ~]# ls /lustre
welcome-tenant100
 [root@tenant100-client ~]# df -t lustre -h
Filesystem                   Size  Used Avail Use% Mounted on
192.168.200.2@tcp:/lustre  1.0T  8.0K  1.0T   1% /lustre
```

- Clients at tenant101

```
 [root@tenant101-client ~]# mount -t lustre 192.168.200.2@tcp:/lustre /lustre
 [root@tenant101-client ~]# ls /lustre
welcome-tenant101
 [root@tenant101-client ~]# df -t lustre -h
Filesystem                   Size  Used Avail Use% Mounted on
192.168.200.2@tcp:/lustre  2.0T  8.0K  2.0T   1% /lustre
```

“`df /path/to/dir`” `statfs()` returns project quota usage (LU-9555 in Lustre-2.14)

Clients on tenants can see own storage limits and usages.

Step4: Encryption

- Workflows for Encryption operations
  - Multiple administrators are involved
    1. "`secadmin`" account needs to be enabled by "`sysadmin`" to issue "`fscrypt`" command.
    2. Encryption keys need to be managed per tenant and generated by "secadmin".
    3. "`secadmin`" passes generated keys to "tenant admin".
    4. Disable "`secadmin`" account by "`sysadmin`" after encryption operations are done.
- Lustre Encryption requirements
  - User tool "fscrypt" configures encryption (Lustre supported in `fscrypt-0.2.9-`, `fscrypt-0.3.4+`)
  - Modify `/etc/fscrypt.conf` and change policy_version=2 after "fscrypt setup"

Step4-1: Apply Lustre client encryption per tenant

- Enable ”secadmin” account and initialize encryption on Lustre

```
[sysadmin@mgmt ~]$ sudo mount -t lustre 192.168.200.2@tcp:/lustre /lustre
[sysadmin@mgmt ~]$ sudo passwd -u secadmin
[secadmin@mgmt ~]$ sudo fscrypt setup /lustre
```

- Generates encryption protectors and policies for clients

```
[secadmin@mgmt ~]$ sudo fscrypt metadata create protector /lustre --name=tenant100
                   --source=custom_passphrase
[secadmin@mgmt ~]$ sudo fscrypt metadata create policy /lustre
                   --protector=/lustre:d532907a74c3326f
[secadmin@mgmt ~]$ sudo fscrypt status /lustre
lustre filesystem "/lustre" has 1 protectors and 1 policies.
Only root can create fscrypt metadata on this filesystem.

PROTECTOR         LINKED  DESCRIPTION
d532907a74c3326f  No      custom protector "tenant100"

POLICY                            UNLOCKED  PROTECTORS
44dcf8450a01a2df094f3c97e7028ab0  No        d532907a74c3326f
```

Step4-2: Enable encryption to tenant’s directories

- Enforce encryption to tenant’s directories with generated policy

```
[secadmin@mgmt ~]$ sudo fscrypt encrypt /lustre/tenant100
                   --policy=/lustre:44dcf8450a01a2df094f3c97e7028ab0
                   --source=custom_passphrase --user=root
```

- Lock directory

```
[secadmin@mgmt ~]$ sudo fscrypt lock /lustre/tenant100 --user=root
```

- Disable ”secadmin” account by “sysadmin”

```
[sysadmin@mgmt ~]$ sudo passwd -l secadmin
```

Ready to use encrypted tenant directories

Run through same steps 1 to 4 for other clients and tenants (e.g. `tenant101`)

Final step: Client access to isolated and encrypted directory

▶ Mount Lustre from clients on tenant

```
[root@tenant100-client ~]# mount -t lustre 192.168.200.2@tcp:/lustre /lustre
[root@tenant100-client ~]# dd if=/dev/zero of=/lustre/file bs=1M count=1
dd: failed to open '/lustre/file': Required key not available
```

Directory is now encrypted

▶ Unlock directory to access

```
[root@tenant100-client ~]# fscrypt unlock /lustre --user=root
Enter custom passphrase for protector "tenant100":
"/lustre" is now unlocked and ready for use.
[root@tenant100-client ~]# dd if=/dev/zero of=/lustre/file bs=1M count=1
1+0 records in
1+0 records out
1048576 bytes (1.0 MB, 1.0 MiB) copied, 0.00426131 s, 246 MB/s
```

▶ Files are not accessible even by root (“`sysadmin`”) on management nodes

```
[sysadmin@mgmt ~]$ sudo cat /lustre/tenant100/file
cat: /lustre/tenant100/file: Required key not available
```

Conclusions

Whamcloud

- Lustre has been steadily adding unique security features and capabilities
  - Namespace isolation is consisted of multiple Lustre security features.
  - Other features (e.g. project quota) improve storage manageability for tenants.
- Demonstrated building secure namespace isolation
  - Less complexity, but it’s flexible and powerful.
  - Configurations are already enabled in production in the field.
- System-wide security is still required
  - Hardware, OS, and Network isolation and security need to be enabled along with Lustre security features.

Whamcloud

Appendix: Nodemap Configuration

```
lctl nodemap_activate 0

lctl nodemap_del trusted
lctl nodemap_del tenant100
lctl nodemap_del tenant101

lctl nodemap_modify --name default --property trusted --value 0
lctl nodemap_modify --name default --property admin --value 0
lctl nodemap_modify --name default --property deny_unknown --value 1
lctl nodemap_set_fileset --name default --fileset /null

lctl nodemap_add trusted
lctl nodemap_add_range --name trusted --range 192.168.200.[1-254]@tcp
lctl nodemap_modify --name trusted --property trusted --value 1
lctl nodemap_modify --name trusted --property admin --value 1
lctl nodemap_modify --name trusted --property deny_unknown --value 0

lctl nodemap_add tenant100
lctl nodemap_add_range --name tenant100 --range '192.168.100.[1-254]@tcp'
lctl nodemap_modify --name tenant100 --property admin --value 1
lctl nodemap_modify --name tenant100 --property trusted --value 1
lctl nodemap_modify --name tenant100 --property squash_projid --value 100
lctl nodemap_modify --name tenant100 --property map_mode --value projid
lctl nodemap_set_fileset --name tenant100 --fileset /tenant100
```

```
lctl nodemap_add tenant101
lctl nodemap_add_range --name tenant101 --range '192.168.101.[1-254]@tcp'
lctl nodemap_modify --name tenant101 --property admin --value 1
lctl nodemap_modify --name tenant101 --property trusted --value 1
lctl nodemap_modify --name tenant101 --property squash_projid --value 101
lctl nodemap_modify --name tenant101 --property map_mode --value projid
lctl nodemap_set_fileset --name tenant101 --fileset /tenant101

lctl nodemap_activate 1
```