Whamcloud

Changing nodemap membership paradigms

LAD 2025

sbuisson@whamcloud.com

ddn

Changing nodemap membership paradigms

Whamcloud

- What is Nodemap?
- How to implement nodemap isolation?
  - Legacy method
  - New method based on GSS identification
- Further improvements for ease of use

What is Nodemap?

Whamcloud

- Nodemap is a long-established feature…
  - Supported from Lustre 2.9 (2015)
  - But introduced in Lustre 2.7 as a technology preview!
- … initially developed for ID mapping
  - Multiple sites with conflicting user and group ids can operate on a single Lustre file system without collisions in UID or GID space

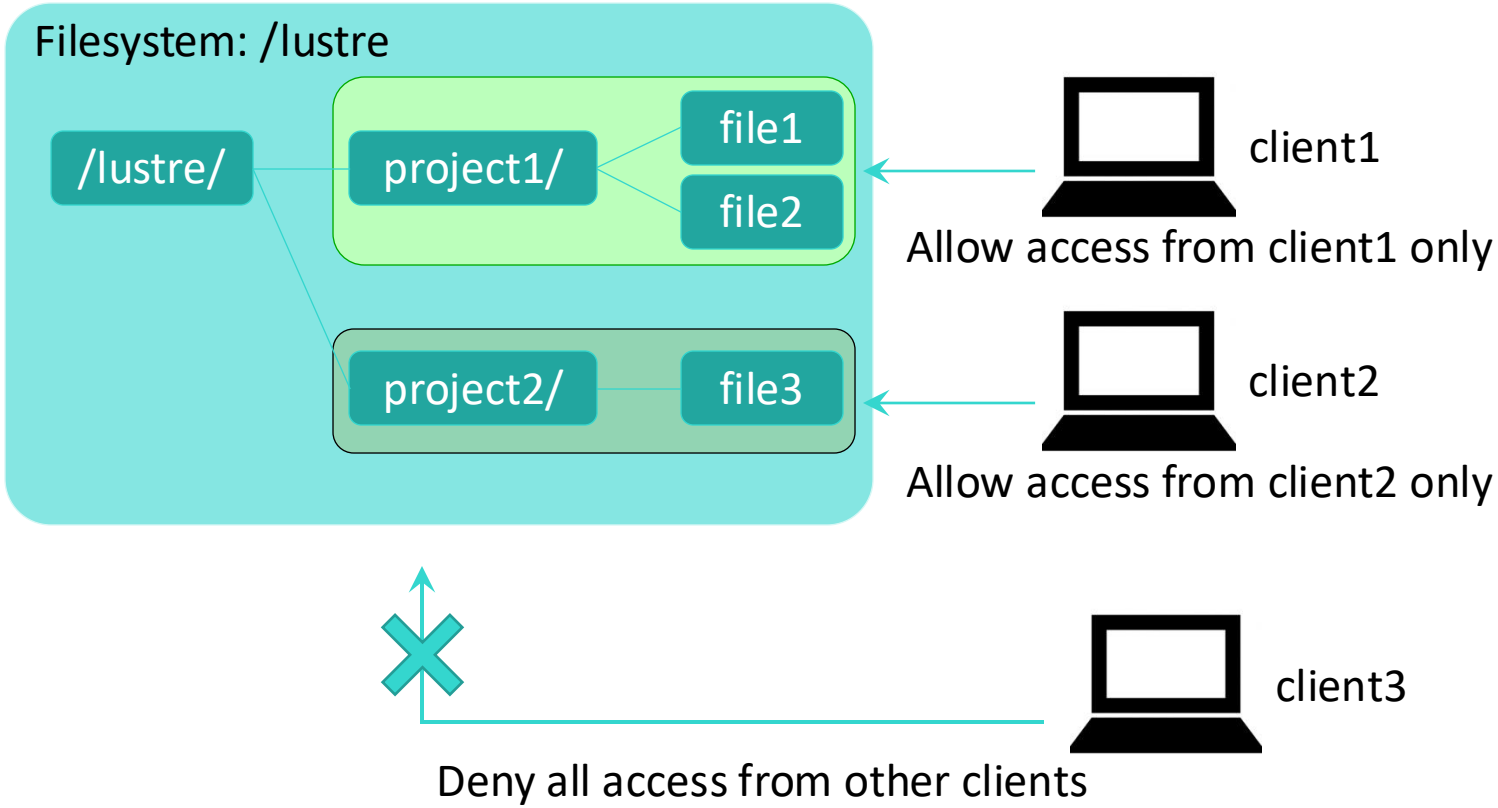
How does Nodemap work?

Whamcloud

- Nodemap distinguishes clients based on their Network IDentifier (NID)
- File system access filtered through Nodemap identity mapping policy engine

- When a connection is made from a NID:
  - Lustre decides if that NID is part of a *nodemap*: a policy group made of NID ranges
    - NID mapping is done only once at filesystem connection time
  - A collection of identity mappings or idmaps is kept for each policy group
    - idmaps translate client UIDs, GIDs, and PROJIDs into canonical filesystem IDs
  - Each policy group also has properties, governing access conditions

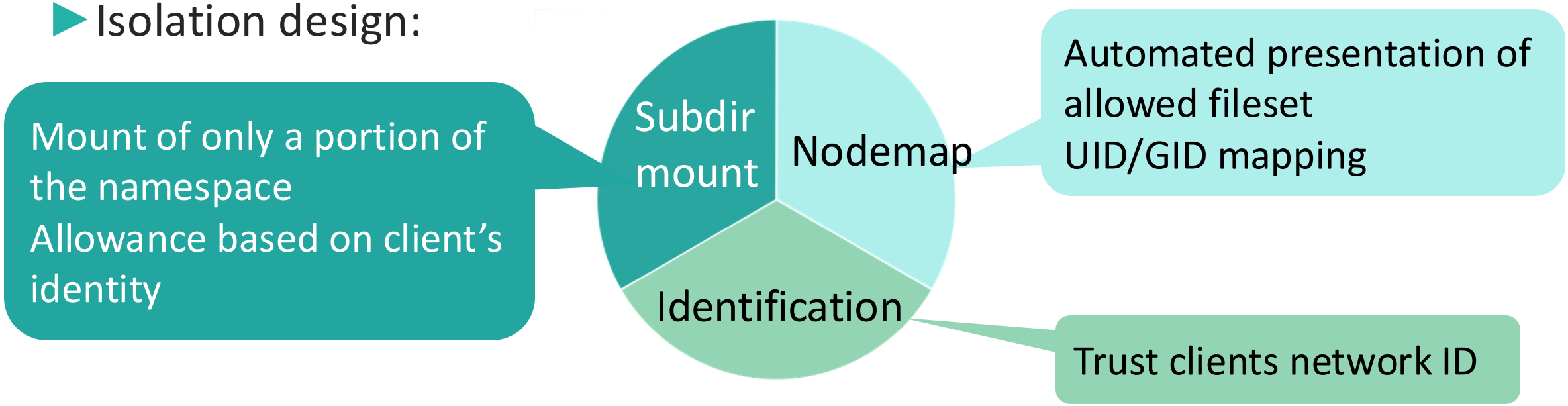
Nodemap isolation: Rough Idea



Nodemap isolation: Concept

Whamcloud

- Isolation design:



- Isolation enables:
  - Different populations of users on the same file system

How to implement nodemap isolation

Whamcloud

- Declare a nodemap dedicated to a tenant, with properties:

```
(1) mgs# lctl nodemap_add Tenant1
(2) mgs# lctl nodemap_modify --name Tenant1 --property admin=1
(3) mgs# lctl nodemap_modify --name Tenant1 --property trusted=1
(4) mgs# lctl nodemap_add_offset --name Tenant1 --offset 100000 --limit 70000
(5) mgs# lctl nodemap_modify --name Tenant1 --property rbac=local_admin
(6) mgs# lctl nodemap_fileset_add --name Tenant1 --fileset /dir_Tenant1
```

How to implement nodemap isolation

Whamcloud

- Identify clients used by tenant
- Assign clients to tenant nodemap, with their NIDs

```
(1) mgs# lctl nodemap_add_range --name Tenant1 --range 192.168.1.[100-200]@tcp
(2) mgs# lctl nodemap_add_range --name Tenant1 --range 192.168.2.[0-50]@tcp
```

- NID ranges must not overlap, **globally** for all nodemaps
- NID ranges must be updated when clients are added to/removed from tenant

Securing nodemap isolation

Whamcloud

- How can client NIDs be trusted?
  1. Users do not have root access on clients

     ⇒ Cannot modify Lustre configuration
  2. Make use of strong authentication to prevent address spoofing
     - Kerberos
     - Shared-Secret Key (SSK)
     - How does it work?
       - Maliciously modified client NID will not match client's key
       - Lustre servers will refuse connection

Securing nodemap isolation

Whamcloud

- Nodemap and SSK are tightly coupled:

```
Version:          1
Type:             server
HMAC alg:         sha256
Crypto alg:       ctr(aes)
Ctx Expiration:   604800 seconds
Shared keylen:    256 bits
Prime length:     2048 bits
File system:      lustrefs
MGS NIDs:
Nodemap name:     Tenant1
Shared key:
  0000: 1c04 c13e db15 dfc0 f557 9ea0 a254 a61b  ...>.....W...T..
  0010: 0619 daab 3ed1 e206 9c51 aa63 bd19 eeca  ....>....Q.c....
```

Securing nodemap isolation

Whamcloud

- Nodemap and SSK are tightly coupled:
  - Server looks up nodemap from client NID
  - Server uses SSK key associated with nodemap to carry out authentication
    - Proceeds to HMAC verification

⇒ Prevents SSK key forgery

⇒ Prevents NID spoofing

Securing nodemap isolation

Whamcloud

- And to be comprehensive:
  - *Client sends sha256 of nodemap name (read from SSK key) in auth request*
  - Server looks up nodemap from client NID
  - Server uses SSK key associated with nodemap to carry out authentication
    - Proceeds to HMAC verification, *that includes sha256 of nodemap name*

⇒ Prevents SSK key forgery

⇒ Prevents NID spoofing

Changing nodemap membership paradigm

Whamcloud

- Wait, what?
  - Client sends sha256 of nodemap name (read from SSK key) in auth request
  - Server proceeds to HMAC verification, that includes sha256 of nodemap name

- Could we not use nodemap name sent by client?

Changing nodemap membership paradigm

Whamcloud

- New paradigm:
  - On server side, look up nodemap from sha256 received from client
  - Then use SSK key associated with nodemap to carry out authentication

- Safe to use sha256 from client:
  - Protected from forgery thanks to HMAC
  - If client fakes nodemap sha256, HMAC verification fails (wrong SSK key)

Changing nodemap membership paradigm

Whamcloud

- New nodemap property: *gss_identification*
  - Clients for this nodemap are no longer identified with their NIDs
    - Instead, they are identified by their GSS token
  - Nodemap with *gss_identification* cannot have any NID ranges defined
    - Any client presenting a GSS token associated with this nodemap is accepted

New nodemap isolation implementation

Whamcloud

- Identify clients used by tenant
  - Install SSK key associated with tenant
- Enable GSS identification on nodemap for tenant:

```
(1) mgs# lctl nodemap_modify --name Tenant1 --property gss_identification=1
```

⇒No need to update nodemap definitions when tenant clients are moved across nodes, only SSK key is needed

New nodemap isolation implementation

Whamcloud

- Much more flexible approach:
  - **New tenant** workflow:
    - Generate SSK key, with Prime for clients
    - Create associated nodemap with *gss_identification* property
    - Load SSK key on all servers
    - Load SSK key on tenant clients
  - **Revoke tenant** workflow:
    - Unload SSK key on all servers
    - Remove associated nodemap

New nodemap isolation implementation

Whamcloud

- GSS identification vs. nodemap banlist:
  - Nodemap banlist prevents access from client NIDs in the banlist
  - However nodemaps with *gss_identification* do not accept NID ranges

- We can still support banlists for nodemaps with *gss_identification*:
  - Once clients identified by GSS context, check NID against nodemap banlist

  ⇒Prevent client access if banned

New nodemap isolation implementation

Whamcloud

- Further improvements for more flexibility and ease of use:
  - LU-19073 lgss_sk should handle ASCII-encoded SSK keys
    - Allows transferring SSK key via copy-paste
  - LU-19069 mount.lustre should automatically generate SSK prime number
    - Avoids the need to explicitly generate client key from server key
  - LU-19326 Load SSK key from mount point
    - mount.lustre automatically uses SSK key stored in **.lgss** file in mount point.
  - Support multiple SSK keys on client side
    - Would allow a client to be part of different parties

New nodemap isolation implementation

Whamcloud

- ***Even*** more flexible approach:
  - **New tenant** workflow:
    - Generate SSK *server* key
    - Create associated nodemap with *gss_identification* property
    - Load SSK key on all servers
    - Copy-paste SSK *server* key to tenant clients as **ASCII**
      - as `MOUNT_DIR/.lgss` file
  - **Revoke tenant** workflow:
    - Unload SSK key on all servers
    - Remove associated nodemap

New nodemap membership paradigms – wrap-up

Whamcloud

- Nodemap relies on NIDs for client identification
- But GSS auth can be leveraged to provide an alternative approach
  - Simplifies nodemap isolation implementation
  - Provides more flexibility

- Work is still in progress:
  - Need to support Kerberos in addition to SSK
  - Will be pushed to master once done

Whamcloud

Thank you!

sbuisson@whamcloud.com

ddn

Id mapping flow chart