--enable-gss

Strong authentication in Lustre&friends

Daniel Kobras
science + computing ag
IT-Dienstleistungen und Software für anspruchsvolle Rechnernetze
Tübingen | München | Berlin | Düsseldorf
science+computing

<table>
<thead>
<tr>
<th>Founded in</th>
<th>1989</th>
</tr>
</thead>
<tbody>
<tr>
<td>Offices</td>
<td>Tuebingen</td>
</tr>
<tr>
<td></td>
<td>Munich</td>
</tr>
<tr>
<td></td>
<td>Duesseldorf</td>
</tr>
<tr>
<td></td>
<td>Berlin</td>
</tr>
<tr>
<td>Employees</td>
<td>275</td>
</tr>
<tr>
<td>Shareholder</td>
<td>Bull S.A. (100%)</td>
</tr>
<tr>
<td>Turnover 2012</td>
<td>30.2 Mio. EUR</td>
</tr>
</tbody>
</table>

Portfolio

**IT Service** for complex computing environments
Complete solutions for Linux- and Windows-based **HPC**

**scVENUS** System management software for efficient administration of homogeneous and heterogeneous environments
Outline

Practical look at current status of Lustre with GSS feature from an admin perspective:

• Why bother?
• What is it good for?
• Is it any good?
Authentication in Lustre

• User authentication
  • **Null** authentication by default
  • MDS **trusts** uid sent by **client**
  • Root access on any Lustre client means full access to all data in filesystem

• Client authentication
  • Anything that can reach port 988 is allowed as a client
  • Firewalls can impose access restrictions
  • Still, full access is only an IP address away

• Security on par with NFSv3 and earlier
• Since Lustre 2.0: compile time option **--enable-gss** for increased filesystem security with **Generic Security Service** (GSS)
Why secure your (Lustre) filesystems?

• Because there's always someone to hide your data from (students, competitors, „Chinese hackers“, NSA, ...)
• Because stolen data makes you lose money, reputation, or a court case
• Because secure infrastructure keeps admins happy

… and also allows more options in hardware/network configuration
Configuration example without GSS

Insecure filesystems limited to secure(TM) environments (eg. Cluster-internal network)
Configuration example without GSS

Insecure filesystems limited to secure(TM) environments (eg. Cluster-internal network)
Outside access requires data copies or transition to different access protocol (NFS, CIFS)
Configuration example with GSS
Configuration example with GSS

If desired, overhead can be avoided if filesystem allows secure access via untusted networks (WAN, campus, outside datacenter)
Why GSS/Kerberos in filesystems?

- Secure communication needs to provide **privacy**, **integrity**, strong **authentication**
- Two major options
  - SSL/TLS – based on certificates/passphrases
  - Kerberos – (usually) based on shared secrets (passwords/keys)
- Passwords don't require persistent storage
  → common network filesystems (NFS, CIFS, AFS, Lustre) use Kerberos authentication
- Kerberos protocol standard, but different implementations use different APIs
- Standard GSSAPI providing Kerberos (and other) authentication systems (initial RFC 1508 published Sept. 1993)
Kerberos in a nutshell

• Kerberos authentication service issues **tickets** as **short-lived** identification credentials for specific services

• Kerberization: Services outsource authentication to KDC as **trusted third party**

• KDC itself can be kerberized to provide **Single Sign-On** (identify with TGT instead of password)

• Kerberos authenticates users and services (**principals**) 

• Kerberos can provide **mutual authentication** between communication endpoints

• Kerberos protocol **never sends passwords** across the wire (not even encrypted)

• Kerberos needs sane DNS and synchronized clocks
Headaches with kerberized filesystems

• Initial setup well documented (required principals, keytab creation, configuration options, etc.)

• Flow of credentials usually isn't – but that's where admins have to look when debugging:
  • Filesystem clients live in kernel space
  • Kerberos lives in user space
  • Kerberized filesystem client needs access to Kerberos credentials of calling user

• Solutions:
  • Require user to push credentials into filesystem
  • Implement dedicated upcalls and lookup heuristic to automagically match uid with credentials cache
  • Use kernel keyring infrastructure with generic upcall dispatcher
Credential passing: AFS

Userland

Kernel

Client

ccache

aklog

cache manager

Server

Kernel

Userland

fileserver

AFS/RX
Credential passing: NFSv4

- Userland:
  - ccache
  - magic
  - gssd

- Kernel:
  - uid

- Client:
  - NFS client
  - NFS

- Server:
  - NFS server
  - keytab
  - svcgsssd
  - lookup
  - (un)seal
Credential passing: Lustre

Kernel keyring/
keyutils
voodoo

Lustre client

Lustre

Lustre server

ccache

keytab

lkvcgssd

lookup

(un)seal

Client

Server
Credential passing: Lustre

Client

Kernel

cache

Userland

Credential passing:

- keyid
- type
- desc

( ... )

current()
Credential passing: Lustre

```
/sbin/request-key
```

```
current()
```

```
keyid
type
desc
(...)
```

```
keyid
type: lgssc
```

```
Userland
```

```
Kernel
```

```
cache
```
Credential passing: Lustre

/etc/request-key.d/lgssc.conf:

```
(...)
lgss_keyring
```

```
/sbin/request-key
```

```
/usr/sbin/lgss_keyring
```

```
/sbin/request-key
```

```
/kern/[...]
```

```
keyid
```

```
typedesc
```

```
type: lgssc
```

```
current()
```

Userland

ccache

Kernel

Client
Credentail passing: Lustre

```
/proc/self/ccache
```

```
/etc/request-key.d/lgssc.conf:

(...)
lgss_keyring

/sbin/request-key

```

```
/usr/sbin/lgss_keyring

```

```
/keyid

type: lgssc

desc: cred.
```

```
Client

Kernel

current()

Userland

cache
```
Credential passing: Lustre

```
/etc/request-key.d/lgssc.conf:

    (...)  
    lgss_keyring

/sbin/request-key

/usr/sbin/lgss_keyring

/sbin/request-key

/etc/request-key.d/lgssc.conf:

    /sbin/request-key

/user/sbin/lgss_keyring

GCC context

type  

desc  

cred.

keyid

type: lgssc

keyid

type: lgssc

current()

ccache

Userland

Kernel

Client
```
Lustre service principals

• Each filesystem access needs to be authenticated with Kerberos credentials

• Lustre service principals
  
  • MGS: `lustre_mgs/<hostname>`, eg. `lustre_mgs/mds01.example.org`
  
  • MDS: `lustre_mds/<hostname>`, eg. `lustre_mds/mds01.example.org`
  
  • OSS: `lustre_oss/<hostname>`, eg. `lustre_oss/oss01.example.org`
  
  • Client: `lustre_root/<hostname>`, eg. `lustre_root/client001.example.org`

• Client principal used for mount and access by root user, normal users need their own principals
Lustre/GSS configuration

- Configuration of request-key infrastructure for lgssc key type (/etc/request-key.d/lgssc.conf):
  ```
  create lgssc * * /usr/sbin/lgss_keyring \n  %o %k %t %d %c %u %g %T %P %S
  ```

- Start `lsvcgssd` on Lustre servers

- Configure security flavor with `lctl conf_param <fs>.srpc.flavor.default`:
  - `null`: no authentication (default)
  - `krb5n/krb5a`: authentication only
  - `krb5i`: additional integrity check of bulk data
  - `krb5p`: additional encryption of bulk data (DES, 3DES, RC4, AES128, AES256)

- Security flavor can be refined for different LNETs and directions
Lustre/GSS test setup

• Requirements:
  • Kerberos infrastructure provided by ActiveDirectory (Win2k8r2)
  • Evaluate GSS feature for different Lustre branches
  • GSS feature should not require obsolete versions of Linux kernel or other software, ie. work with latest available Lustre-patched server kernels for each branch, and latest available CentOS6 kernel for Lustre clients (currently CentOS 6.4)
  • Should work with strongest available security flavor (krb5p)
  • Should be able to work securely for all communication directions, especially all client-server communication
# Lustre/GSS availability

<table>
<thead>
<tr>
<th>Lustre branch</th>
<th>feature present</th>
</tr>
</thead>
<tbody>
<tr>
<td>b1_8</td>
<td>no</td>
</tr>
<tr>
<td>b2_1</td>
<td>yes</td>
</tr>
<tr>
<td>b2_3</td>
<td>yes</td>
</tr>
<tr>
<td>b2_4</td>
<td>yes</td>
</tr>
</tbody>
</table>
## Lustre/GSS availability

<table>
<thead>
<tr>
<th>Lustre branch</th>
<th>feature present</th>
<th>feature documented (*)</th>
</tr>
</thead>
<tbody>
<tr>
<td>b1_8</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>b2_1</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td>b2_3</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td>b2_4</td>
<td>yes</td>
<td>no</td>
</tr>
</tbody>
</table>

(*) in the Lustre manual
## Lustre/GSS availability

<table>
<thead>
<tr>
<th>Lustre branch</th>
<th>feature present</th>
<th>feature documented (*)</th>
<th>feature builds (#)</th>
</tr>
</thead>
<tbody>
<tr>
<td>b1_8</td>
<td>no</td>
<td>yes</td>
<td>N/A</td>
</tr>
<tr>
<td>b2_1</td>
<td>yes</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>b2_3</td>
<td>yes</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>b2_4</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
</tr>
</tbody>
</table>

(*) in the Lustre manual

(#) with latest Lustre-patched kernel (server) in branch and latest RHEL6.4 kernel (client)
## Lustre/GSS availability

<table>
<thead>
<tr>
<th>Lustre branch</th>
<th>feature present</th>
<th>feature documented (*)</th>
<th>feature builds (#)</th>
<th>feature works (+)</th>
</tr>
</thead>
<tbody>
<tr>
<td>b1_8</td>
<td>no</td>
<td>yes</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>b2_1</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>N/A</td>
</tr>
<tr>
<td>b2_3</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>N/A</td>
</tr>
<tr>
<td>b2_4</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
<td>no</td>
</tr>
</tbody>
</table>

(*) in the Lustre manual  
(#) with latest Lustre-patched kernel (server) in branch and latest RHEL6.4 kernel (client)  
(+) Client-server communication with krb5p authentication
Practical experiences with Lustre/GSS

• Code quality
  • Error checking through LBUG/LASSERT
    \[\text{lctl set_param panic_on_lbug}=0\] is a must for sane debugging
  • Changes to in-kernel API break GSS build in b2_1 and b2_3 (both fixed in b2_4)
  • b2_4 introduces new proxy subsystems (lwp, osp) that GSS code doesn't know about (\[\rightarrow\] instant LBUG on service startup, LU-3778)
Practical experiences with Lustre/GSS

- Verbose logs
  - `lsvcgssd`: groks option `-v`
  - `request-key`: groks option `-d`, but cannot change kernel upcall → add shell wrapper (logs go to authpriv, ie. `/var/log/secure`!)
  - `lgss_keyring`: debug level configured in `/proc/fs/lustre/sptlrpc/gss/lgss_keyring/debug_level` (no kidding)
Practical experiences with Lustre/GSS

• Handy debugging aids
  • Traces from GSS code: `lctl set_param debug=+sec`
  • Query security flavor in use for each subsystem
    `lctl get_param *.*.srpc_info`
  • Query state of kernel keyring
    `cat /proc/keys` and `keyctl show`
  • Change hard coded ccache `MEMORY:/self` of `lgss_keyring` to accessible location like `FILE:/tmp/krb5cc_...`
Practical experiences with Lustre/GSS

- ActiveDirectory (AD) specific problems
  - AD issues tickets with authorization data (PAC) by default
  - Leads to larger ticket sizes that Lustre/GSS cannot handle (→ LBUG, LU-3855)
  - Workaround: Set flag `NO_AUTH_DATA_REQUIRED` (0x2000000) in `userAccountControl` attribute of Lustre service accounts
Conclusion

• GSS support present in Lustre code base for several years already
• Adoption hasn't gained momentum (abstract for this presentation ranks on Google page 1 when searching for „Lustre GSS“ ;-) )
• Issues with GSS feature on any Lustre branch with current kernels
• Lack of comprehensive support
• GSS still a core requirement to enable use of Lustre outside of secure islands
References

- https://jira.hpdd.intel.com/browse/LU-3490 – enable GSS builds by default
Thank you!

Daniel Kobras
science + computing ag
www.science-computing.de
www.hpc-wissen.de
Telefon 07071 9457-0
info@science-computing.de