Configuring Security Type for Lustre

Setting the type of encryption used can be specified on the NID and direction (to/from) for traffic. To default all interfaces to a specific settings use the “default” word.

Usage:

```
lctl conf_param fsname.srpc.flavor.default=flavor
lctl conf_param fsname.srpc.flavor.NID=flavor
lctl conf_param fsname.srpc.flavor.NID.SP2SP=flavor
```

fsname - The file system name

flavor - The security flavor (see: `sptlrpc_name2flavor_base()` function in `lustre/ptlrpc/sec.c`)

- null - Null mechanism (no integrity or privacy)
- plain - Plaintext with a hash on RPC
- GSS Keyring Mech Types
    - gssnull - Null mechanism that uses the GSSAPI module
    - krb5n - Kerberos 5 with authentication only
    - krb5a - Kerberos 5 with authentication only
    - krb5i - Kerberos 5 with authentication and integrity checks for all bulk data
    - krb5p - Kerberos 5 with authentication, integrity, and privacy.
        - This encrypts bulk data using one of the ciphers available
    - ski - Shared key with integrity
    - skpi - Shared key with integrity and privacy

NID - Lustre Networking Identifier

- lnd - Where lnd is the lustre network driver (tcp, o2ib, or gnld)
- <lnd:lnd>num
- lndN - Where N is the network number (0-65536)

SP - Lustre Secure PTLRPC service

- cli - Lustre client
- mdt - Lustre MDT
- ost - Lustre OST
- mgc - Lustre MGC
- mgs - Lustre MGS
- any - Any Lustre service

Examples:

```
lctl conf_param FSNAME.srpc.flavor.default=gssnull
lctl conf_param FSNAME.srpc.flavor.tcp1=krb
lctl conf_param FSNAME.srpc.flavor.o2ib.cli2ost=skpi
```

Setting up Lustre Clients for GSS Keyring Types

The Lustre GSS keyring types of flavors utilize the linux kernel keyring infrastructure to maintain keys as well as perform the upcall from kernel space to user space for key negotiation/establishment. The GSS keyring establishes a key type (see "`request-key(8)`") named "lgssc" when the Lustre ptlrpc_gss kernel module is loaded. When a security context needs to be established for Lustre it will create a key and use the request-key binary in an upcall to establish the key. This key will look for the configuration file in /etc/request-key.d with the name *keytype*.conf, for Lustre this is lgssc.conf. The /etc/request-key.d/lgssc.conf for Lustre should look like the following:

```
create lgssc * * /usr/sbin/lgss_keyring %o %k %t %d %c %u %g %T %P %S
```

The request-key binary will call /usr/sbin/lgss_keyring with the arguments following it with their substituted values (see "`request-key.conf(5)`").

Note: As part of the process to verify a host the lgss_keyring program requires that the NID's IPv4 address portion resolve to a hostname using a reverse lookup. This is true for ko2iblnd and ksocklnd LNDs.

Setting up Lustre Servers for GSS Keyring Types

Lustre servers do not use the linux kernel keyring infrastructure as clients do. Instead they run a daemon that uses a pipefs with the kernel to trigger events based on read/write to a file descriptor. The server side binary is the `/usr/sbin/lsvcgssd`. This can be ran by hand in the foreground for extra debugging information or as a daemon. Future versions of Lustre will support an init script to run this as a service. Below is the usage for the `lsvcgssd` script. As part of the shared key work this daemon has been extended to require various security flavors (gssnull, krb, sk) to be enabled explicitly so only required functionality is enabled.

```
usage: /usr/sbin/lsvcgssd [ -nfvrmog ]
-f    - Run in foreground
-n    - Don't establish kerberos credentials
-v    - Verbosity
-m    - Service MDS
-o    - Service OSS
-g    - Service MGS
-k    - Enable kerberos support
-s    - Enable shared key support
-z    - Enable gssnull support
```

Debugging GSS Keyring

Lustre client and server support several debug levels which can be seen below.

Debug levels:

- 0 - Error
- 1 - Warn
- 2 - Info
- 3 - Debug
- 4 - Trace

To set the debug level on the client use the Lustre parameter: `sptlrpc.gss.lgss_keyring.debug_level`. For example to set the debug level to debug:

```
lctl set_param sptlrpc.gss.lgss_keyring.debug_level=3
```

On the server side verbosity is increased by adding additional verbose flags to the command line arguments for the daemon. The following would run the `lsvcgssd` daemon in the foreground with debug verbosity supporting gssnull and shared key security flavors.

```
/usr/sbin/lsvcgssd -f -vvv -z -s
```

Since `lgss_keyring` is called as part of the request-key upcall there is no standard output for the process so logging will be done to the syslog. The server side logging with `lsvcgssd` is done to standard output in the foreground mount and to syslog in daemon mode.

GSS Shared Keys

Usage

Shared key generation is done using the lgss_sk utility that comes with lustre when built with the --enable-gss option. The `lgss_sk` utility can be used to read the contents of a keyfile, load the keyfile into the kernel keyring or generate (write) a key file. See the `lgss_sk` usage below:

```
Usage lgss_sk [OPTIONS] [-r <file> | -l <file> | -w <file>]
-r|--read   <file>Show file's key attributes
-w|--write  <file>Generate key file
-l|--load   <file>Load key from file into user's session keyring

Load Options:
-t|--type   <type>Key type (server or client)

Write Options:
-c|--crypt  <num>       Cipher for encryption (Default: AES Counter mode)
-h|--hmac   <num>       Hash alg for HMAC (Default: SHA256)
-e|--expire <num>       Seconds before key expiration (Default:
2147483647 seconds)
-f|--fsname <name>File system name for key
-n|--name   <name>Nodemap name for key (Default: "")
-s|--session<len>       Session key length in bits (Default: 1024)
-k|--shared <len>       Shared key length in bits (Default: 256)
```

```
-d|--data    <file>Shared key data source (Default: /dev/random)
```

Generating Keys

To first use the shared key you will need to generate a key for clients. Keys can be generated by specifying the options for the key on the command line follow by the "`-w`" option and the filename to write to. By default the `lgss_sk` utility will not overwrite files so the name must be unique. You will want need to set the file system name for all keys generated but all other parameters are optional. By default the nodemap is blank which is the default configuration for Lustre when nodemap is not in use. You can provide the data file for the key or by default the utility will read from `/dev/random`. Since `/dev/random` can block if there is not sufficient entropy available this command can take a while to run.

Supported crypt algorithms are:

0 - AES Counter mode

Supported HMAC algorithms are:

0 - MD5
1 - SHA1
2 - SHA256
3 - SHA512

For example to create a key for file system tank for a client in the nodemap biology you would run:

```
[root@server ~]# lgss_sk -f tank -n biology -w tank.biology.key
```

Reading Keys

To read the keys generated by the lgss_sk utility you use the "-r" flag. The usage is straightforward and will dump up the relevant contents of the key. Contents of the key will be dumped as hex and ASCII to standard output.

> **Note the key files by the GSS shared keys are a set size but unused bytes in the shared key are not output.**

Reading the above key looks like:

```
[root@server ~]# lgss_sk -r tank.biology.key
Version:          1
HMAC alg:         0
Crypt alg:        0
Expire:           2147483647 seconds
Shared keylen:    256 bits
Session keylen:   1024 bits
File system:      tank
Nodemap name:     biology
Shared key:
  0000: 9dc3 15dc 227a 68b1 dc78 0628 dbf4 b286  ...."zh..x.(....
```

```
0010: 014a 2262 921c 00b4 8032 c3c6 1d7b b11d  .J"b.....2...{..
```

Loading Keys

Loading shared keys can be done with the lgss_sk utility or at mount time by adding an "--skpath" parameter to mount (mount.lustre). Using the lgss_sk utility loads keys into the kernel keyring and takes one mandatory parameter "-t" which specifies the type of key as server or client. The purpose of the type is because clients use a different name for the key description to find keys when needed then servers do. The format is lustre:fsname for clients and lustre:fsname:nodemap for servers. Fsname will be the file system name and is derived from the target name that Lustre is attempting to establish the security context for. Nodemap is the nodemap name that the client's NID belongs to. When nodemap is unused this will have a blank value, otherwise this will be populated. All keys for Lustre use the "user" type for keys and are attached to the user's keyring. This is not configurable today. Below is an example showing how to list the user's keyring, load the key using the lgss_sk utility, show the key load, read the key and clear the key from the kernel keyring.

```
[root@mds ~]# keyctl show
Session Keyring
      -3 --alswrv       0     0  keyring: _ses
 150619626 --alswrv     0     -1   \_ keyring: _uid.0
[root@mds ~]# lgss_sk -t server -l tank.biology.key
[root@mds ~]# keyctl show
Session Keyring
      -3 --alswrv       0     0  keyring: _ses
 150619626 --alswrv     0     -1   \_ keyring: _uid.0
 385054140 --alswrv     0     0     \_ user: lustre:tank:biology
[root@mds ~]# keyctl read 385054140
230 bytes of data in key:
01000000 00000000 ffffff7f 00010000 00040000 74616e6b 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 0062696f 6c6f6779 00000000
00000000 00009dc3 15dc227a 68b1dc78 0628dbf4 b286014a 2262921c 00b48032
c3c61d7b b11d0000 00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
00000000 0000
[root@mds ~]# keyctl clear @u
[root@mds ~]# keyctl show
Session Keyring
      -3 --alswrv       0     0  keyring: _ses
 150619626 --alswrv     0     -1   \_ keyring: _uid.0
```

Keys loaded using the --skpath option to the mount command support loading a whole directory of keys for servers to conveniently load multiple keys since a it's possible several nodemaps are

in use for the same file system. If the skpath fails to load any key during the mount command it will return a failure to the mount command. However, keys that have already been loaded will remain in the kernel keyring.

> **Note: When loading keys that already exist both the `lgss_sk` and `mount` command will ignore any files that have a key that matches the description matching it in the kernel. However, the contents of the key are not actually retrieved and verified so you must clear the keyring yourself if you update a key for a specific description.**