MyWebUniversity
python
Red Hat Enterprise Linux Manual Pages
FPDF Library PDF generator
FPDF

Red Hat Enterprise Linux Release 9.2 Manual Pages on 'luksmeta.8' command

$ man luksmeta.8

LUKSMETA(8) LUKSMETA(8)

NAME

luksmeta - Utility for storing metadata in a LUKSv1 header

SYNOPSIS

luksmeta test -d DEVICE

luksmeta nuke -d DEVICE [-f]

luksmeta init -d DEVICE [-f] [-n]

luksmeta show -d DEVICE [-s SLOT]

luksmeta save -d DEVICE [-s SLOT] -u UUID < DATA

luksmeta load -d DEVICE -s SLOT [-u UUID] > DATA

luksmeta wipe -d DEVICE -s SLOT [-u UUID] [-f]

OVERVIEW

The luksmeta utility enables an administrator to store metadata in the gap between the end of the LUKSv1 header and the start of the encrypted data. This is useful for storing data that is available before the volume is unlocked, usually for use during the volume unlock process.

The metadata is stored in a series of UUID-typed slots, allowing multiple blocks of metadata. Although the luksmeta slots are inspired by the LUKS slots, they are functionally independent and share only a casual relationship. Slots merely provide a hint that a given chunk of metadata is associated with a specific LUKSv1 password (in a slot with the same number). However, luksmeta itself is indifferent to the relationship between a LUKSv1 slot and the correspondly numbered luksmeta slot, with one exception (detailed below).

After a LUKSv1 volume is initialized using cryptsetup(8), it must also be initialized for metadata storage by luksmeta init. Once this is complete, the device is ready to store medata.

Data can be written to a slot using luksmeta save or read from a slot using luksmeta load. You can also erase the data in an existing slot using luksmeta wipe or query the slots using luksmeta show.

UUID GENERATION

It is often presumed that saving metadata to a slot requires a specific UUID or that there is an official registry of UUID types. This is incorrect.

UUID stands for Universally Unique IDentifier. UUIDs are a standardized, widely-used data type used for identification without a central registry. For the relevant standards, see ISO 9834-8:2005 and RFC 4122.

UUIDs are large enough that collision is practically impossible. So if your application wants to store data in a luksmeta slot, just generate your own UUID and use it consistently to refer to your type of data. If you have multiple types of data, feel free to generate multiple UUIDs.

The easiest way to generate a UUID is to use uuidgen(1). However, any compliant UUID generator will suffice.

INITIALIZATION

Before reading or writing metadata, the LUKSv1 block device must be initialized for metadata storage. Three commands help with this process: luksmeta test, luksmeta nuke and luksmeta init.

The luksmeta test command simply checks an existing block device to see if it is initialized for metadata storage. This command does not provide any output, so be sure to check its return code (see below).

The luksmeta nuke command will zero (erase) the entire LUKSv1 header gap. Since this operation is destructive, user confirmation will be required before clearing the gap unless the -f option is supplied.

The luksmeta init command initializes the LUKSv1 block device for metadata storage. This process will wipe out any data in the LUKSv1 header gap. For this reason, this command will require user confirmation before any data is written unless the -f option is supplied. Note that this command succeeds without any modification if the device is already initialized. If you would like to force the creation of clean initialization state, you can specify the -n option to nuke the LUKSv1 header gap before initialization (but after user confirmation).

METADATA STATE

The luksmeta show command displays the current state of slots on the LUKSv1 block device. If no slot is specified, it prints a table consisting of the slot number, the corresponding LUKSv1 slot state and the UUID of the data stored in the luksmeta slot (or "empty" if no data is stored). If a slot is specified, this command simply prints out the UUID of the data in the slot. If the slot does not contain data, it prints nothing.

MANAGING METADATA

Managing the metadata in the slots is performed with three commands: luksmeta save, luksmeta load and luksmeta wipe. These commands write metadata to a slot, read metadata from a slot and erase metadata in a slot, respectively.

The luksmeta save command reads metadata on standard input and writes it to the specified slot using the specified UUID. If no slot is specified, luksmeta will search for the first slot number for which the LUKSv1 slot is inactive and the luksmeta slot is empty. This represents the only official correlation between LUKSv1 slots and luksmeta slots. In this case, the metadata is written to the first applicable slot using the specified UUID and the slot number is printed to standard output. In either case, this command will never overwrite existing data. To replace data in a slot you will need to execute luksmeta wipe before luksmeta save.

The luksmeta load command reads data from the specified slot and writes it to standard output. If a UUID is specified, the command will verify that the UUID associated with the metadata in the slot matches the specified UUID. This type check helps to ensure that you always receive the type of data you are expecting as output. If the UUIDs do not match, the command will fail.

The luksmeta wipe command erases the data from the given slot. If a UUID is specified, the command will verify that the UUID associated with the metadata in the slot matches the specified UUID. This type check helps to ensure that you only erase the data you intended to erase. Because this is a destructive operation, this command will require user confirmation before any data is erased, unless the -f option is supplied. Note that this command succeeds if you attempt to wipe a slot that is already empty.

CAVEATS

The amount of storage in the LUKSv1 header gap is extremely limited. It also varies based upon the configuration used by LUKSv1 at device initialization time. In some LUKSv1 configurations, there is not even enough space for all the metadata slots even at the smallest possible slot size.

During the design of this utility, we considered it likely that users would want to reduce the number of usable slots in exchange for more storage space in the slots used. In order to provide this flexibility, the amount of storage available per-slot is dynamic. Put simply, slots are not a fixed size. This means that it is possible (and even somewhat likely) to encounter an error during luksmeta save indicating that there is insufficient space.

This error is not a programming bug. If you encounter this error it likely means that either all space is being consumed by the already-written slots or that the metadata you are attempting to write simply does not fit.

You can attempt to resolve this problem by calling luksmeta wipe on slots that are no longer in use. This will release the storage space for use by other slots. Note that luksmeta does not, however, currently perform defragmentation since the number of usable blocks is rather limited. You can attempt to manually get around this by extracting all slot data, wiping the slots and reloading them in order. However, this operation is potentially dangerous and should be undertaken with great care.

OPTIONS

? -d DEVICE, --device=DEVICE : The device on which to perform the operation.

? -s SLOT, --slot=SLOT : The slot number on which to perform the operation.

? -u UUID, --uuid=UUID : The UUID to associate with the operation.

? -f, --force : Forcibly suppress all user prompting.

RETURN VALUES

This command uses the return values as defined by sysexit.h. The following are general errors whose meaning is shared by all luksmeta commands:

? EX_OK : The operation was successful.

? EX_OSERR : An undefined operating system error occurred.

? EX_USAGE : The program was called with invalid parameters.

? EX_IOERR : An IO error occurred when writing to the device.

? EX_OSFILE : The device is not initialized or is corrupted.

? EX_NOPERM : The user did not grant permission during confirmation.

? EX_NOINPUT : An error occurred while reading from standard input.

? EX_DATAERR : The specified UUID does not match the slot UUID.

? EX_CANTCREAT : There is insufficient space in LUKSv1 header.

Additionally, luksmeta save will return EX_UNAVAILABLE when you attempt to save data into a slot that is already used. Likewise, luksmeta load will return EX_UNAVAILABLE when you attempt to read from an empty slot.

EXAMPLES

Destroy all data (including LUKSMeta data) in the LUKSv1 header gap and initialize the gap for LUKSMeta storage:

```
$ luksmeta init -n -f -d /dev/sdz
```

If already initialized, do nothing. Otherwise, destroy all non-LUKSMeta data in the LUKSv1 header gap and initialize the gap for LUKSMeta storage:

```
    $ luksmeta init -f -d /dev/sdz
```

Write some data to a slot:

```
    $ UUID=*uuidgen*

    $ echo $UUID

    31c25e3b-b8e2-4eaa-a427-23aa882feef2

    $ echo "Hello, World" | luksmeta save -d /dev/sdz -s 0 -u $UUID
```

Read the data back:

```
    $ luksmeta load -d /dev/sdz -s 0 -u $UUID

    Hello, World
```

Wipe the data from the slot:

```
    $ luksmeta wipe -d /dev/sdz -s 0 -u $UUID
```

Erase all trace of LUKSMeta:

```
    $ luksmeta nuke -f -d /dev/sdz
```

AUTHOR

Nathaniel McCallum <npmccallum@redhat.com>

SEE ALSO

cryptsetup(8), uuidgen(1)