LB-CSI Client Side Encryption

Oct 20, 2021

Rev 1.0

Table of Contents

1. Proposal

Add support for encryption of LightOS volumes in LightOS-CSI with `Linux Unified Key Setup` (LUKS) version 2.

2. Overview Of LUKS

Linux Unified Key Setup (LUKS) is a disk encryption specification originally intended for Linux. LUKS uses device mapper crypt (dm-crypt) as a kernel module to handle encryption on the block device level.

2.1. What LUKS does

- LUKS encrypts entire block devices and is therefore well-suited for protecting the contents of mobile devices such as removable storage media or laptop disk drives.
- The underlying contents of the encrypted block device are arbitrary. This makes it useful for encrypting swap devices. This can also be useful with certain databases that use specially formatted block devices for data storage.
- LUKS uses the existing device mapper kernel subsystem.
- LUKS provides passphrase strengthening which protects against dictionary attacks.
- LUKS devices contain multiple key slots, allowing users to add backup keys or passphrases.

2.2. What LUKS does not do

- LUKS is not well-suited for scenarios requiring many (more than eight) users to have distinct access keys to the same device.
- LUKS is not well-suited for applications requiring file-level encryption.

Note

- Disk-encryption solutions like LUKS only protect the data when your system is off. Once the system is on and LUKS has decrypted the disk, the files on that disk are available to anyone who would normally have access to them.
- Compression should be disabled when encryption is enabled. There is no point in compressing encrypted data, which will gain no storage saving - only CPU costs.

2.3. LUKS workflow outside CSI

Let's go through the basic LUKS workflow:

- Installing prerequisite
- Creating encrypted device
- Opening encrypted device
- Closing encrypted device
- Detecting if a device is encrypted
- etc...

Say we have a machine with the following disk layout:

```
lsblk
NAME    MAJ:MIN RM   SIZE RO TYPE MOUNTPOINT
vda     252:0    0   128G  0 disk
├─vda1  252:1    0   487M  0 part /boot
├─vda2  252:2    0   1.9G  0 part [SWAP]
└─vda3  252:3    0 125.6G  0 part /
nvme1n1 259:0    0     1G  0 disk
nvme0n1 259:1    0     1G  0 disk
```

2.3.1 Install Prerequisites

```
sudo apt install -y cryptsetup
```

2.3.2 Formatting the volume with LUKS

The following process encrypts `/dev/nvme0n1`.

In order to proceed, you need to enter YES in capitals and provide the password twice:

```
sudo cryptsetup -y -v luksFormat /dev/nvme0n1

WARNING!
========
This will overwrite data on /dev/nvme0n1 irrevocably.

Are you sure? (Type uppercase yes): YES
Enter passphrase for /dev/nvme0n1:
Verify passphrase:
Key slot 0 created.
Command successful.
```

2.3.3 Opening the volume with LUKS

Then, we need a target to open the encrypted volume.

I used luks-pvc-d6edd9ca-fb24-480d-a577-1a29e7516df5 as my target, for a reason.

lb-csi-plugin uses the **NGUID** value to lookup the matching LightOS block-device. By keeping the naming convention luks-pvc-<**NGUID**> we can do simple reverse lookup of the opened encrypted volume.

```
sudo cryptsetup -v luksOpen /dev/nvme0n1
luks-pvc-d6edd9ca-fb24-480d-a577-1a29e7516df5
Enter passphrase for /dev/nvme0n1:
Key slot 0 unlocked.
Command successful.
```

Now that we have opened the encrypted volume we can see it here:

```
lsblk
NAME                                                MAJ:MIN RM   SIZE RO TYPE
MOUNTPOINT
vda                                                 252:0    0   128G  0 disk
├─vda1                                              252:1    0   487M  0 part
/boot
├─vda2                                              252:2    0   1.9G  0 part
[SWAP]
└─vda3                                              252:3    0 125.6G  0 part  /
nvme1n1                                             259:0    0     1G  0 disk
```

```
nvme0n1                                           259:1    0     1G  0 disk
└─luks-pvc-d6edd9ca-fb24-480d-a577-1a29e7516df5 253:0    0  1008M  0 crypt
```

Notice the new block device is of type **`crypt`** and is placed under:

```
ls /dev/mapper/luks-pvc-d6edd9ca-fb24-480d-a577-1a29e7516df5
lrwxrwxrwx 1 root root 7 Oct 20 10:52
/dev/mapper/luks-pvc-d6edd9ca-fb24-480d-a577-1a29e7516df5
```

2.3.4 Working With Encrypted Device

Now we can work with the device as we would with any other device:

2.3.4.1 Create a filesystem

```
sudo mkfs.ext4 /dev/mapper/luks-pvc-d6edd9ca-fb24-480d-a577-1a29e7516df5
mke2fs 1.45.5 (07-Jan-2020)
Creating filesystem with 258048 4k blocks and 64512 inodes
Filesystem UUID: db440b33-7ff6-4165-8d5d-f81cf93951e5
Superblock backups stored on blocks:
        32768, 98304, 163840, 229376

Allocating group tables: done
Writing inode tables: done
Creating journal (4096 blocks): done
Writing superblocks and filesystem accounting information: done
```

2.3.4.2 Mount the filesystem

```
sudo mkdir -p /mnt/encrypted_volume
sudo mount -v /dev/mapper/luks-pvc-d6edd9ca-fb24-480d-a577-1a29e7516df5
/mnt/encrypted_volume
mount: /dev/mapper/luks-pvc-d6edd9ca-fb24-480d-a577-1a29e7516df5 mounted on
/mnt/encrypted_volume
```

Note

For RH distro we might get an SELinux warning, we might need to run the following relabel command:

```
restorecon -vvRF /mnt/encrypted_volume
```

Now we see that we have the mounted filesystem:

```
lsblk
NAME                                          MAJ:MIN RM   SIZE RO TYPE
MOUNTPOINT
vda                                           252:0    0   128G  0 disk
├─vda1                                        252:1    0   487M  0 part
/boot
├─vda2                                        252:2    0   1.9G  0 part
[SWAP]
└─vda3                                        252:3    0 125.6G  0 part  /
nvme1n1                                       259:0    0     1G  0 disk
nvme0n1                                       259:1    0     1G  0 disk
└─luks-pvc-d6edd9ca-fb24-480d-a577-1a29e7516df5 253:0    0  1008M  0 crypt
/mnt/encrypted_volume
```

2.3.4.3 UMount the filesystem

```
sudo umount /mnt/encrypted_volume
```

2.3.5 Closing Encrypted Block Device

```
sudo cryptsetup -v luksClose
luks-pvc-d6edd9ca-fb24-480d-a577-1a29e7516df5
Command successful.
```

2.3.6 Detecting rather Device is LUKS formatted

We will use the following command on unmounted block device to detect whether it is formatted already:

```
sudo cryptsetup luksDump /dev/nvme0n1
LUKS header information
Version:        2
Epoch:          3
Metadata area:  16384 [bytes]
Keyslots area:  16744448 [bytes]
UUID:           78d2777a-ed46-4e3b-84e7-3fae1fd6ae00
Label:          (no label)
Subsystem:      (no subsystem)
Flags:          (no flags)

Data segments:
  0: crypt
        offset: 16777216 [bytes]
        length: (whole device)
        cipher: aes-xts-plain64
        sector: 512 [bytes]

Keyslots:
  0: luks2
        Key:        512 bits
        Priority:   normal
        Cipher:     aes-xts-plain64
        Cipher key: 512 bits
        PBKDF:      argon2i
        Time cost:  4
        Memory:     714840
        Threads:    4
        Salt:       39 d6 f6 e6 18 07 17 01 cf 74 b1 00 23 05 15 ec
                    6c 09 d4 bc 63 cc 5e fc d4 22 cd 55 83 b7 18 a3
        AF stripes: 4000
        AF hash:    sha256
        Area offset:32768 [bytes]
```

```
        Area length:258048 [bytes]
        Digest ID:  0
Tokens:
Digests:
  0: pbkdf2
        Hash:       sha256
        Iterations: 56790
        Salt:       00 e8 47 0c af 21 77 fa 16 bf 7d ef 3b 71 e8 a5
                    d0 59 da cf 70 00 cf a4 17 cd 90 c8 2e 58 03 e7
        Digest:     3c 32 f2 64 c7 4f 27 4e 4f 55 9f b5 39 91 6b ea
                    53 5a 48 35 78 9f 8d 46 85 19 3d a8 a4 2d 82 7e
```

For reference when running this command on a non-formatted drive we will get:

```
sudo cryptsetup luksDump /dev/nvme0n2
Device /dev/nvme0n2 doesn't exist or access denied.
```

We can ask on the target device as well using the following command:

```
sudo cryptsetup status luks-pvc-d6edd9ca-fb24-480d-a577-1a29e7516df5
```

3. Changes to lb-csi-plugin

In order to make this work we need to add the following capabilities to the plugin:

1. Install prerequisites in the plugin image.
2. Instruct the plugin that this volume should be encrypted.
3. Pass in the secret to use.
4. Teach the plugin how to encrypt, open, close the encrypted volume.

3.1. StorageClass Modifications

3.1.1 Secret Injection

There are many standard ways to pass a secret to a CSI plugin.

One of the common ways is using the StorageClass to inject the secret.

The only place we would need the secret is when formatting and opening the volume. These two steps are taken on the NodeStageVolume API call and the NodeStageVolumeRequest.Secrets will get this value.

Look here for more information:
https://kubernetes-csi.github.io/docs/secrets-and-credentials-storage-class.html#node-stage-secret

Example StorageClass:

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
    name: example-sc
provisioner: csi.lightbitslabs.com
allowVolumeExpansion: true
parameters:
     mgmt-endpoint: 10.10.0.2:443,10.10.0.3:443,10.10.0.4:443
     compression: disabled
     project-name: default
     csi.storage.k8s.io/node-stage-secret-name: example-secret
     csi.storage.k8s.io/node-stage-secret-namespace: default
```

The External-Provisioner will handle these secrets and due to the template support it has we can achieve any secret to volume mapping:

`csi.storage.k8s.io/node-stage-secret-name`

- `${pv.name}`
    - Replaced with name of the `PersistentVolume` object being provisioned.

- `${pvc.namespace}`
    - Replaced with namespace of the `PersistentVolumeClaim` object that triggered provisioning.
- `${pvc.name}`
    - Replaced with the name of the `PersistentVolumeClaim` object that triggered provisioning.
- `${pvc.annotations['<ANNOTATION_KEY>']}` (e.g. `${pvc.annotations['example.com/key']}`)
    - Replaced with the value of the specified annotation from the `PersistentVolumeClaim` object that triggered provisioning

3.1.2 Example node-stage-secret

We will need to pass additional value - named **luks-passphrase** to be used on NodeStageVolume API.

```
apiVersion: v1
kind: Secret
metadata:
    name: example-secret
    namespace: default
type: kubernetes.io/lb-csi
Data:
    jwt:  ZXlKaGJHY2lPaUp…dwo=
    luks-passphrase:    ZXlKewewaGdds2lPaUp…dwdao=
```

> **Note**
>
> This adds some complexity around secret management, since we will need to potentially create a secret with JWT for every different luks-passphrase and that might have implications for JWT rotation etc...

3.1.2 Additional Parameter Injection

Instructing the plugin to create an encrypted volume and which parameters to use will be defined in the StorageClass As well.

We will define the following new parameters in the StorageClass:

```
csi.lightbitslabs.com/luks-encrypted: "true"
csi.lightbitslabs.com/luks-cipher: "aes-xts-plain64"
csi.lightbitslabs.com/luks-key-size: "512"
```

- **csi.lightbitslabs.com/luks-encrypted**: indicate rather volumes should be encrypted. (default to “false”) default compression to faile
- **csi.lightbitslabs.com/luks-cipher**: cipher to use; must be supported by the kernel and luks. (should declare for sane default)
- **csi.lightbitslabs.com/luks-key-size**: key-size to use. (should declare for sane default)
- Additional keys may be defined as needed.

Since the parameters of the storage class are not propagated to the **NodeStageVolume** API where we need them, we need to find a different way to propagate them.

We will use the **csi.CreateVolumeResponse.Volume.VolumeContext** to pass in parameters coming from **csi.CreateVolumeRequest.Parameters**

This value is for SP use according to the docs and is opaque to the CO.

We will retrieve this information from **csi.NodeStageVolumeRequest.VolumeContext** to know if we need to use encryption for this volume and how.

3.2. Global Secret Injection Support

FI-TS requested to work with a single JWT, provided using Secret as a volume mount to the plugin.

3.3. Implementation Changes

The following APIs are the only ones that need to change:

CreateVolume (Controller)

- Parsing **csi.CreateVolumeRequest.Parameters** and passing LUKS related parameters on **csi.CreateVolumeResponse.Volume.VolumeContext**.

NodeStageVolume

- Detecting whether the device is already encrypted using the **luksDump** command (2.3.6 Detecting rather Device is LUKS formatted)
- call **luksFormat** ONLY for the first time we mount this volume (2.3.2 Formatting the volume with LUKS). Use the passphrase defined at 3.1. Secret Injection.
- Call **luksOpen** (2.3.3 Opening the volume with LUKS). Use the passphrase defined at 3.1. Secret Injection.
- **luksOpen** call will create a device: `` `/dev/mapper/luks-<nguid>` ``. We will mount this device.

NodePublishVolume

- Mount the correct path (staging or device-mapper) as **target_path**

This API will mount the volume from **staging_target_path** to **target_path**. By doing that it will need to know if the volume is encrypted, and if so, it will need to access the `/dev/mapper/luks-<nguid>` and not the **staging_target_path**

We can detect rather the device is encrypted or not using the cryptsetup status

NOTE: Volume corruption detection and handling might require additional detection operations

NodeUnPublishVolume

- No special handling cause we unmount the target_path which is the same for encrypted or unencrypted.

NodeUnStageVolume

- Detects whether this volume is encrypted or not. We don't have **VolumeContext** on this API so we will lookup the appropriate block device under device-mapper (we do have the nguid).
- If the device is encrypted, it calls the **luksClose** function to remove the LUKS mapping (2.3.5 Closing Encrypted Block Device).

NodeExpandVolume

Expanding an encrypted volume requires additional steps. We will need to modify today's implementation to support encrypted volumes.

For offline resize - meanined if we reboot/mount/unmount the encrypted format will get the new size of the real block device.

For online resize - not requiring mount/unmount we should do the following steps:

- Use the mounter object to detect the **devicePath** from the VolumePath like this:

```
devicePath, _, err := mount.GetDeviceNameFromMount(mounter, volumePath)


If encrypted devicePath will have a value /dev/mapper/luks-<nguid>
```

- Determine **devicePath** is prefixed with /dev/mapper/. Use

  cryptsetup status `luks-<nguid>`

  to query if it is a LUKS device.

- Invoke resize command on the LUKS device to extend it to the maximum size:

```
cryptsetup resize luks-<nguid>
```

- `Invoke regular resize fs on the blockDevice`

4. Working with KMS (WIP)

WIP - In The Following Days we will provide a solution that uses KMS.

NOTE: Since this is not planned yet, the implementation will not limit future enhancements to support KMS.

We should design a solution that provides functionality working with KMS.

4.1. Benefits of working with KMS

- If for some reason our keys are lost, all our data will be lost as well. KMS will help keep these keys.
- When working on a Key-Per-Volume basis it is very hard to maintain hundreds or even thousands of keys on the system.

4.2. KMS Examples

There are many implementations of KMS, here are some Examples:

- On-Prem:
  - https://www.vaultproject.io/docs/secrets/key-management
- Cloud:
  - https://cloud.google.com/security-key-management
  - https://aws.amazon.com/kms/

5. Deployment And Documentation Updates

1. Helm charts need to be updated with examples StorageClass and Encrypted volume flow.
2. Documentation needs to be extended with this new feature, configuration options, etc...

6. Validation and Testing

TBD

About Lightbits Labs™

Lightbits Labs' mission is to lead the cloud-native data center transformation by delivering scalable and efficient software defined storage that is easy to consume. Founded in 2016, Lightbits brings the agility of hyperscale storage to private clouds and edge clouds. The company pioneered NVMe/TCP so the solution is easy to deploy at scale, while delivering performance that is similar to local flash. Lightbits Labs is backed by strategic investors including Cisco Investments, Dell Technologies Capital, Intel Capital, and Micron, as well as top investors and VCs including Avigdor Willenz, Lip-Bu Tan, Marius Nacht, SquarePeg Capital, and WRVI Capital.

www.lightbitslabs.com info@lightbitslabs.com

US Office
1830 The Alameda,
San Jose, CA 95126, USA

Israel (Kfar Saba) Office
17 Atir Yeda Street,
Kfar Saba, Israel 4464313

Israel (Haifa) Office
3 Habankim Street,
Haifa, Israel 3326115