Encrypting Sensitive Information at Rest at the Edge

Problem Statement

EVE has to provide a security capability that would enable various scenarios of storing sensitive information on the built-in storage of the Edge Node where EVE is running, while providing reasonable protections from this information leaking outside of the running EVE instance. Note, that we're not proposing an end-to-end encryption solution here, but rather designing a capability that would mitigate some of the attack vectors based on physical possession of the Edge Node. The data itself, while protected in-flight by the transport level security mechanism such as TLS, is expected to be un-encrypted before it lands on the Edge Node.

One big driving factor for this is App Instance: App Instances deployed by EVE Customers receive and process business sensitive information from sensors and the Cloud. Data collected and processed by these App Instances are stored in their virtual storage, which is backed by the hardware storage on the EVE platform. It is important that even if the secondary storage drive is stolen, the data remains secure. For this reason, data should be in encrypted form when it is stored. Here we explore various tools available in Linux which support disk encryption, and compare their performance on the EVE platform.

Disk Encryption Options on Linux

Native EXT4 Encryption (e4crypt/fscrypt)

Starting from Linux kernel 4.1, EXT4 filesystem supports native encryption. This means that encryption functionality is built into the filesystem, and closely works with the internal implementation of EXT4 filesystem. e4crypt tool is used to manage EXT4 encryption. Recently fscrypt tool was introduced by Google as a wrapper around e4crypt with additional functionalities like better key management (PAM plugins) and support for changing the encryption keys periodically. While e4crypt is supported as a package in Alpine Linux, fscrypt is yet to be validated officially on Alpine Linux, and one needs to explicitly build and package it in EVE, which runs on Alpine Linux. *fscrypt* supports encryption only for a given directory under a partition. The Linux kernel needs to be configured with CONFIG_EXT4_FS_ENCRYPTION for using this feature, does not result in any increase in kernel memory footprint. However fscrypt userspace utility which manages this encryption feature, takes around 4MB.

New Cipher option in fscrypt: Adiantum

Adiantum is a new encryption method by Google, and is used on Android for improving performance of disk encryption on low-end ARM processors lacking CPU instructions for special encryption operations. Adiantum is not a new file system, but rather a new ChaCha12 based cipher as a replacement for CPU intensive AES cipher. Option to use Adiantum encryption has been added to dm-crypt/cryptsetup and fscrypt. Adiantum cipher was added to Linux kernel from version 5.0 onwards.

Current EVE kernel version is 4.19, and we need to upgrade to kernel 5.0 or later to experiment with Adiantum on EVE and measure its performance. This effort is underway, and we need to run fscrypt on DomU running out of EVE with 5.2.2 kernel and get the performance numbers. Once EVE moves to Linux kernel 5.0+, we can look at changing fscrypt cipher to Adiantum.

Overlay Encryption (eCryptfs)

Unlike Native encryption, eCryptfs does not assume any particular file system, but rather works on top of existing file system and provides a virtual mountpoint. Files that are written on this virtual mount point get encrypted and stored in the underlying filesystem. It supports encryption only for a given directory under a partition. ecryptfs-utilss is the official package supported by Alpine Linux, which has the commands to manage eCryptfs. Kernel needs to be configured with CONFIG_ECRYPTFS_ENCRYPTION for using this feature.

Disk or Partition level Encryption (dm-crypt/LUKS)

LUKS does encryption at the given disk partition level. It groups one or more partitions into a Linux LVM (more like a virtual disk), and supports encryption at the LVM level. LVM needs to be enabled for encryption, before further partitioning and mounting. User will be writing to the LVM, which will be encrypted and stored on the underlying physical disk partition. Cryptsetup is the widely used userspace tool to configure LUKS encryption, and comes by default on Alpine. It is by default enabled in the EVE kernel.

Performance Measurement

Given the choices available for doing file encryption, it was decided to prototype each of these on one of the EVE platforms, and measure how they fare against each other. Flexible IO Tester (FIO) is a widely used tool in Linux to measure disk IO performance. one set of tests were done with FIO running on Dom0 (i.e. Base OS), and another set of tests were done with FIO running inside a DomU(VM), with its image coming from an encrypted disk on Dom0. Following are the details of the experiment:

EVE Model used: *Advantech ARK 1124 (Dom0 tests) and Supermicro E100-9APP (DomU tests)*

Processor Support for AES instruction set: Enabled on both Advantech and Supermicro models above.

FIO test specification: *60% read, 40% write, 1GB file, with 4 worker threads and testing for 90 second duration*

Performance reports of fscrypt, eCryptfs and LUKS have been uploaded as attachments.

Performance Overhead Comparison (on Dom0)

	No Encryption	**fscrypt/e4crypt**	**eCryptfs**	**dm-crypt(LUKS)**
Read rate (MB/s)	23.7	20.4	19	19.9
Write rate (MB/s)	15.9	13.6	12.7	13.3
Read overhead %	0	13.9 %	19.80%	16.00%
Write overhead %	0	14.40%	20.10%	16.35%

Performance Overhead Comparison (on DomU)

(With Advantech, CPU power became the bottleneck to run DomU, so DomU tests were run on Supermicro instead)

	No Encryption	**fscrypt/e4crypt**	**eCryptfs**	**dm-crypt(LUKS)**
Read rate (MB/s)	15.4	15.2	12.3	13.8
Write rate (MB/s)	10.3	10.1	8.19	9.19
Read overhead %	0	1.20%	20.10%	10.30%
Write overhead %	0	1.90%	20.40%	10.70%

EVE will use Native EXT4 encryption

Based on the above performance comparison, fscrypt (preferably with Adiantum) will be used to implement data encryption on EVE.

For detailed design aspects of fscrypt, the reader is strongly advised to read this document, and get familiar with the nomenclature used, more importantly the policy and protector constructs. A brief introduction about fscrypt is given below, for understanding the rest of this feature proposal.

Fscrypt uses 2 main constructs: Protectors and Policies. To encrypt a directory one needs, at minimum

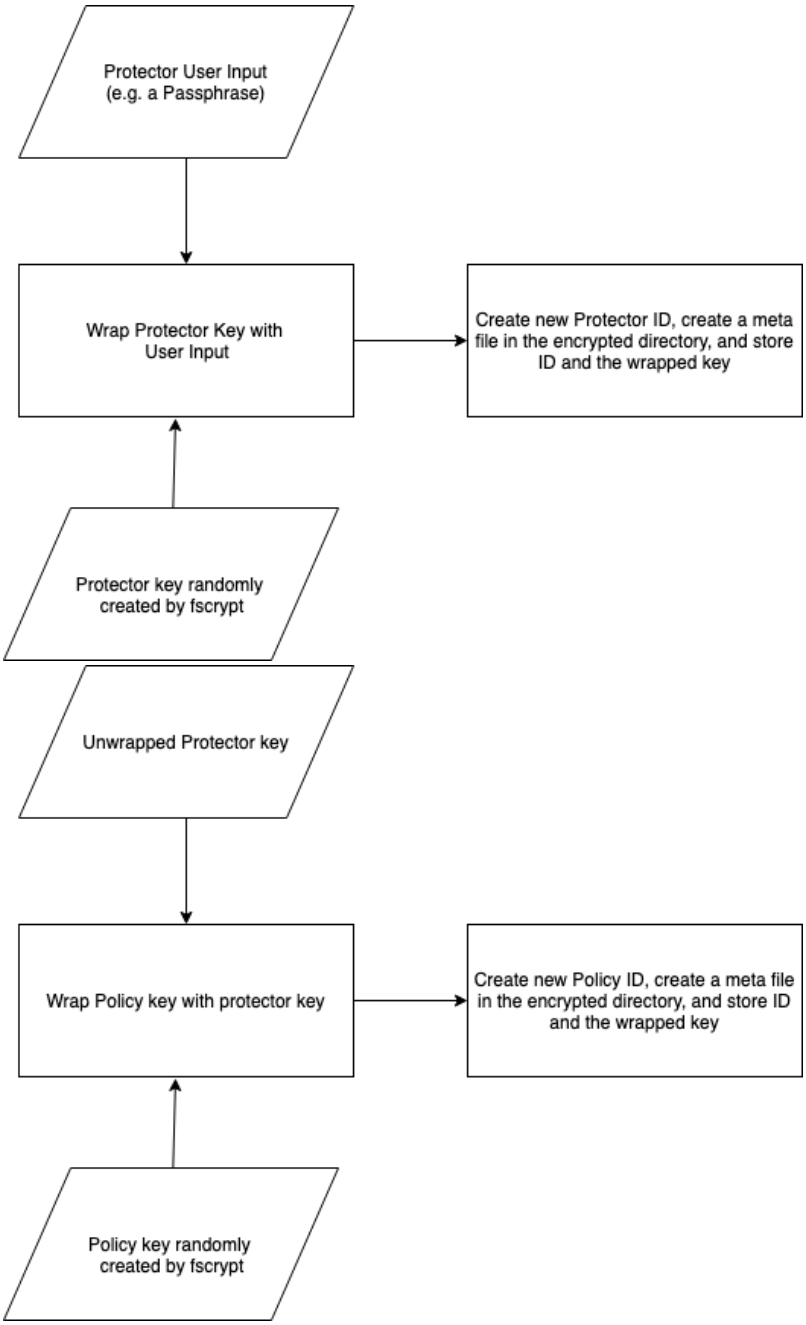
a) Create a Protector(mutable) - A Protector is created using a passphrase or PAM plugin(i.e. from user login credentials) or a key file containing the key.

b) Create a Policy for the directory(immutable) - This sets up the actual encryption key to be used in kernel (loaded in the kernel keyring(s), not visible in the userspace in plain-text) to encrypt the files in that directory.

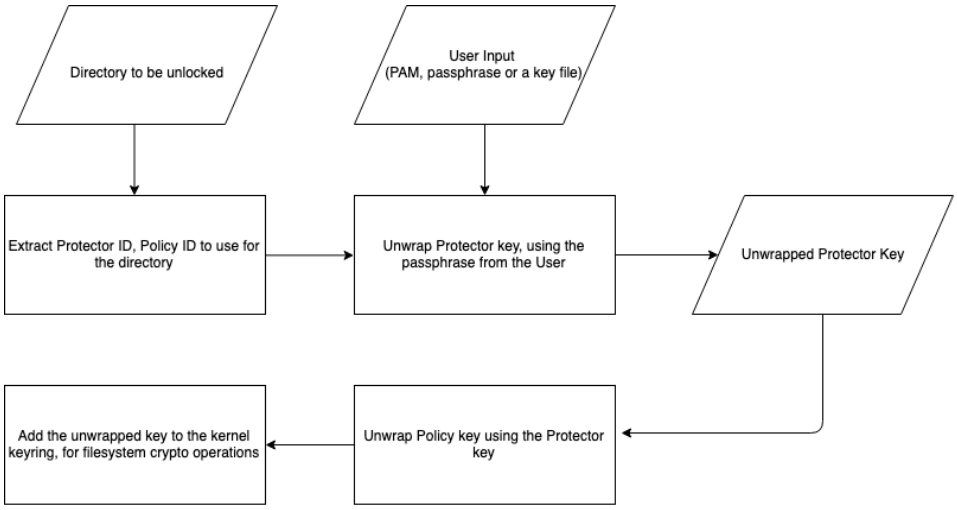
c) Associate a policy with a protector - Bind the protector with the policy. i.e. Protector protects the policy.

By having these 2 layers of protection, enables the user to change the protector, without re-encrypting the whole directory again. Following sections give a over-simplified view of fscrypt operations. There are many finer details which are not captured, one is advised to go through fscrypt architecture for the same.

Setting up Encryption through Protector and Policy Creation



Unlocking for Access through Policy ID and Kernel Keyring



Implementation Details on EVE

Directory layout

Currently most of the persistent and sensitive data is stored under /dev/sda9 partition, which is mounted at /persist.

E.g.

/persist/img - Stores the mutated DomU image disks

/persist/config - Stores the device configuration

/persist/checkpoint - Stores last known working device configuration

However, there are other directories which are not considered sensitive:

E.g:

/persist/IMGA/ - Logs from IMGA

/persist/IMGB/ - Logs from IMGB

Therefore the following approach is proposed:

Create a new directory /persist/vault and set it up for encryption. All sensitive information can be moved to a subdirectory under /persist/vault.

E.g.

/persist/img to move to /persist/vault/img

Some of /persist/config to move to /persist/vault/config *(only a selected subset of config which is treated as sensitive)*

Also, /dev/sda9 will move to EXT4 from EXT3, to take advantage of EXT4 native file system encryption.

Setting up Encryption

At the time of first use, the /dev/sda9 partition will be prepared for encryption using “fscrypt setup” instructions. Fscrypt uses /etc/fscrypt.conf to store its global configuration. A static /etc/fscrypt.conf will be packaged and placed to be visible, as /etc/ will be readonly.

Specifically, /dev/sda9 will be prepared for EXT4 native encryption, with the following 2 settings:

mkfs.ext4 -O encrypt /dev/sda9

and post mount,

fscrypt setup /persist

And finally, contents of vault directory will be enabled for encryption (fscrypt encrypt /persist/vault), and pass a randomly generated hex key as the passphrase to protect the protector used for /persist/vault. You can treat this passphrase as the key which will unwrap the protector key, which will in turn unwrap policy key. Policy key is the key used for the actual encryption.

Protecting the Master Passphrase

The goals of protecting the master passphrase are:

a) Access to hard drive or the EVE node should not lead to access to the master passphrase. (Physical Isolation)

b) The encryption keys are made available once the Identity of the EVE node is established and *possibly* when the node meets a certain software integrity criteria (Node Authentication and Attestation).

c) A capability to revoke passphrase from Controller, in case the EVE node is found to be compromised, which will render the contents of the drive permanently unrecoverable (Key Revocation)

Controller-Managed Key with TPM

On EVE nodes with TPM devices, TPM can be used for sealing the master passphrase against a set of PCR values, so the master passphrase will be unsealed only when the PCRs values indicate that there is no change in the software state. Once EVE supports measured boot(that measures all the components in the boot chain, and storing its measurements in the TPM PCRs), we can start using those PCRs to seal the master passphrase. Also, if we store passphrase in TPM, if the hard disk alone is physically compromised, the keys are still not available on the hard disk, so there will not be a way to decrypt the data on the disk, but if the entire device is physically compromised(which is more likely than just the drive being physically compromised), TPM will automatically unseal the encryption key.

Therefore, we can enhance this and arrive at a better model for EVE nodes with a TPM: Use a key from TPM and a key from Controller to arrive at the master encryption key. It is the best of both of the above approaches put together. With this, even if the entire EVE node is physically compromised, one can not access the complete encryption key unless the EVE node is authenticated and attested by the Controller, which can be advised to flag a given EVE node as black-listed, and hence revoke it's authentication and hence, the encryption key as well.

Physical Isolation: **Yes**

Node Authentication & Attestation: **Yes**

Key Revocation: **Yes**

Controller-Managed Key without TPM

On devices without TPM, we do not have a way to measure the software state or seal the encryption key outside the hard drive. The only other place for creating and storing the keys is the Controller. Therefore for EVE nodes without TPM, it is proposed that encryption keys will be stored in the Controller. After device registers and gets UUID from the Controller, it would fetch either new encryption key(if booting up for the first time) or get the stored encryption key from the Controller. After this EVE will either setup the directory for encryption (if booting up for the first time) or unlock the encrypted directory with the key fetched from the Controller.

Physical Isolation: **Yes**

Node Authentication & Attestation: **No**

Key Revocation: **Yes**

Master Key Rotation

Fscrypt supports changing the protectors password without re-encrypting all the files in the encrypted directory. If required, we can change the protector password, if we think that protector keys might have been compromised. In future we can consider rotating these protectors periodically (say every week) as a precautionary measure to enhance the security. The frequency can be either on-demand, or a configured time-interval which can be configured for each EVE node from the Controller. However care needs to be taken to not miss any rotation from Controller, as this will render the whole data locked forever.(i.e. Controller moved from key 2 to key 3, but EVE is yet to process the last config change from key 1 to key 2)

Recovery Procedure

In some cases, there may be genuine situations where the EVE node has developed some hardware issues(say it is not booting up for some reason) and hence has become offline. It may be desirable to recover the data from the encrypted partition to transition the data to another device for taking over. In such situations, Controller can provide a way to make the key for the problematic device available to the user for offline recovery of the data. The workflow for the same is outside the scope of this document.

Image Compatibility

New Installation

/dev/sda9 will be formatted with EXT4, and /persist/vault will be created and marked for encryption.

Upgrade of existing Installation to Image with Encryption

/dev/sda9 will be reformatted from EXT3 to EXT4, and existing contents of /dev/sda9 will be lost. In this scenario, the device will behave like it was booted after a USB installation. All the DomU images, device configuration will be downloaded again from the Controller.

However one change that will be done is to preserve /persist/config/tpm_in_use file, across EXT3 to EXT4 transition. This is done so that device software continues to see the TPM mode, and hence cloud connectivity is not lost during the transition.

Downgrade Implications

If the device is downgraded to an old image which does not support disk encryption, /dev/sda9 will be reformatted again with EXT3, and contents of /persist will need to be repopulated by downloading configuration and images from the controller. However, /persist/config/tpm_in_use file may be lost and hence cloud controller connectivity may not come up. This will need manual intervention to populate /persist/config/tpm_in_use file or one needs to delete the device.cert.pem file and reboot, which will set the tpm_in_use file automatically.

If this downgrade behaviour is not acceptable, then we need to explore options of doing disk encryption with EXT4 native encryption on an EXT3 filesystem. But performance of fscrypt on EXT3 filesystem is not as good as a native encryption on EXT4.

STRIDE Threat Modelling

Spoofing

Spoofing here refers to getting access to the encryption key by posing as a genuine device. By copying X.509 certificate and the private key of a genuine device, one can make a request to fetch the master encryption key from the Cloud Controller. Once encryption key is made available through this request, data stored on the original device can be decrypted at will. While it is not possible to spoof identity of devices with TPM (since the device certificate is rooted in the TPM), devices without a TPM have this vulnerability. We can mitigate the impact by periodically rotating the master encryption key, or giving an option to change master encryption key for a given edge device through a configuration change from the Controller.

Tampering

Tampering here refers to modifying the encrypted directory in order to affect the integrity of the file system. e.g. Reformatting the whole file system or changing the master encryption key. Or posing the man-in-the-middle attack and changing the master encryption key in the middle, and rendering the edge device and controller to go out of sync, and thus rendering the whole encrypted information unavailable after the next reboot cycle. By applying payload encryption, one can secure the master encryption by wrapping with device's public key, to avoid man in the middle attack.

Repudiation

Repudiation refers to act of denying that a particular event was initiated by the a particular user. In this context, by recording the events related to disk encryption - by logging the events of master key encryption change, reformatting of the partition, locking and unlocking events with details of time, user and mode both in the Controller and in the EVE software - one can maintain auditable log of events to prove the action and person who initiated the action. One may also want to have a way to check if the encryption key pushed by the controller was the key that was actually programmed on the EVE device. Having a one-way hash applied on the key and comparing the hash output between Cloud and the device can be one way to prove this, in case there is a dispute that EVE programmed a different key than one that was pushed by the Controller.

Information Disclosure

It is important that Controller or the EVE software does not disclose the encryption key inadvertently in any of the logs or user interface outputs. It is also important to clear the virtual memory caches immediately after the key change or lock events, to prevent unencrypted data from being exposed albeit for a brief time. Linux kernel provides a way of clearing vm caches, which should be used whenever the secure directory is locked or its keys are changed (*sysctl vm.drop_caches=3* to clear pagecache, dentires and inodes)

Denial of Service

A malicious user can hack into the system and change the encryption keys or just format the whole encrypted directory, thus rendering all the existing data unavailable. Or a man-in-the-middle attack can change the encryption key in flight between the Controller and EVE, and causing the encrypted directory to be unrecoverable after a reboot.

Elevation of Privileges

EVE does not support multiuser login, nor does it support SSH login by default. In some cases, it is possible to get access to Host from a Guest VM and thus escalating the privilege to that of the underlying hypervisor and thus the host OS, e.g. VENOM, CVE-2015-3456,. However preventing that type of attacks is outside the scope of this document/feature.

Attachments: