agherzan / **yubikey-full-disk-encryption**

Use YubiKey to unlock a LUKS partition

luks # encryption # yubikey # archlinux # yubico # linux # initramfs # luks-partition # unlock # disk-encryption

134 commits | **1** branch | **0** releases | **11** contributors | Apache-2.0

Branch: **master** | **New pull request** | **Find File** | **Clone or download**

Vincent43 Sanitize bash environment in ykfde scripts — Latest commit 7482ed9 23 days ago

src	Sanitize bash environment in ykfde scripts	22 days ago
CONTRIBUTING.md	Add Contribution guide	3 months ago
LICENSE	Initial version	2 years ago
Makefile	Makefile: add README to installation target	4 months ago
PKGBUILD	PKGBUILD: fix wrong license	2 months ago
README.md	README.md: fix AUR package name	2 months ago
testrun.sh	Sanitize bash environment in ykfde scripts	22 days ago

README.md

YubiKey Full Disk Encryption

This project leverages a YubiKey HMAC-SHA1 Challenge-Response mode for creating strong LUKS encrypted volume passphrases. It can be used in intramfs stage during boot process as well as on running system.

Be aware that this was only tested and intended for:

- Arch Linux and its derivatives
- YubiKey 4

There is similar project targeting Debian/Ubuntu based systems: yubikey-luks

Table of Contents

Design

The passphrase for unlocking *LUKS* encrypted volumes can be created in two ways:

Automatic mode with stored challenge (1FA)

In *Automatic mode* you create custom *challenge* with 0-64 byte length and store it in cleartext in */etc/ykfde.conf* and inside the initramfs image.

Example *challenge*: `123456abcdef`

The *YubiKey response* is a *HMAC-SHA1* 40 byte length string created from your provided challenge and 20 byte length secret key stored inside the token. It will be used as your *LUKS* encrypted volume passphrase.

Example *response* (ykfde passphrase): `bd438575f4e8df965c80363f8aa6fe1debbe9ea9`

In this mode possession of your *YubiKey* is enough to unlock a *LUKS* encrypted volume (1FA). It allows for the easy unlocking of encrypted volumes when *YubiKey* is present without need for user action.

Manual mode with secret challenge (2FA)

In *Secret mode* you will be asked to provide a custom *challenge* every time you want to unlock your *LUKS* encrypted volume as it will never be stored anywhere on system.

Example *challenge*: `123456abcdef`

It will be hashed using the *SHA256* algorithm to achieve the maximum byte length (64) for any given *challenge*. The hash will be used as the final *challenge* provided for *YubiKey*.

Hashing function:

```
printf 123456abcdef | sha256sum | awk '{print $1}'
```

Example hashed *challenge*: `8fa0acf6233b92d2d48a30a315cd213748d48f28eaa63d7590509392316b3016`

The *YubiKey response* is a *HMAC-SHA1* 40 byte length string created from your provided *challenge* and 20 byte length secret key stored inside the token. It will be concatenated with the *challenge* and used as your *LUKS* encrypted volume passphrase for a total length of 104 (64+40) bytes.

Example response: `bd438575f4e8df965c80363f8aa6fe1debbe9ea9`

Example ykfde passphrase:
`8d969eef6ecad3c29a3a629280e686cf0c3f5d5a86aff3ca12020c923adc6c92bd438575f4e8df965c80363f8aa6fe1debbe9ea9`

This strong passphrase cannot be broken by brute force. To recreate it one would need both your passphrase (something you know) and your *YubiKey* (something you have) which means it works like 2FA.

Keep in mind that the above doesn't protect you from physical tampering like *evil maid attack* and from *malware* running after you unlock and boot your system. Use security tools designed to prevent those attacks.

Install

From AUR

The easiest way is to install package from AUR using one of the AUR helpers.

Install with yay:

```
yay -Syu yubikey-full-disk-encryption-git
```

Install with trizen:

```
trizen -Syu yubikey-full-disk-encryption-git
```

From Github using 'makepkg'

```
wget https://raw.githubusercontent.com/agherzan/yubikey-full-disk-encryption/master/PKGBUILD
makepkg -srci
```

From Github using 'make'

```
git clone https://github.com/agherzan/yubikey-full-disk-encryption.git
cd yubikey-full-disk-encryption
sudo make install
```

When installing by using `make` you also need to install yubikey-personalization and expect packages.

Configure

Configure HMAC-SHA1 Challenge-Response slot in YubiKey

First of all you need to setup a configuration slot for *YubiKey HMAC-SHA1 Challenge-Response* mode using a command similar to:

```
ykpersonalize -v -2 -ochal-resp -ochal-hmac -ohmac-lt64 -oserial-api-visible -ochal-btn-trig
```

Above arguments mean:

- Verbose output (`-v`)

- Use slot 2 (`-2`)
- Set Challenge-Response mode (`-ochal-resp`)
- Generate HMAC-SHA1 challenge responses (`-ochal-hmac`)
- Calculate HMAC on less than 64 bytes input (`-ohmac-lt64`)
- Allow YubiKey serial number to be read using an API call (`-oserial-api-visible`)
- Require touching YubiKey before issue response (`-ochal-btn-trig`) *(optional)*

This command will enable *HMAC-SHA1 Challenge-Response* mode on a chosen slot and write random 20 byte length secret key to your YubiKey which will be used for creating ykfde passphrases.

Warning: choosing YubiKey slot already configured for *HMAC-SHA1 Challenge-Response* mode will overwrite secret key with the new one which means ykfde passphrases created with the old key will be unrecoverable.

You may instead enable *HMAC-SHA1 Challenge-Response* mode using graphical interface through yubikey-personalization-gui package. It allows for customization of the secret key, creation of secret key backup and writing the same secret key to multpile YubiKeys which allows for using them interchangeably for creating same ykfde passphrases.

Edit /etc/ykfde.conf file

Open the /etc/ykfde.conf file and adjust it for your needs. Alternatively to setting `YKFDE_DISK_UUID` and `YKFDE_LUKS_NAME` , you can use `cryptdevice` kernel parameter. The syntax is compatible with Arch's `encrypt` hook. After making your changes regenerate initramfs:

```
sudo mkinitcpio -P
```

Usage

You can list existing LUKS key slots with `cryptsetup luksDump /dev/<device>` .

Format new LUKS encrypted volume using ykfde passphrase

To format new *LUKS* encrypted volume, you can use ykfde-format script which is wrapper over `cryptsetup luksFormat` command:

```
ykfde-format --cipher aes-xts-plain64 --key-size 512 --hash sha512 /dev/<device>
```

Enroll ykfde passphrase to existing LUKS encrypted volume

To enroll new ykfde passphrase to existing *LUKS* encrypted volume you can use ykfde-enroll script, see `ykfde-enroll -h` for help:

```
ykfde-enroll -d /dev/<device> -s <keyslot_number>
```

Warning: having a weaker non-ykfde passphrase(s) on the same *LUKS* encrypted volume undermines the ykfde passphrase value as potential attacker will always try to break the weaker passphrase. Make sure the other non-ykfde passphrases are similarly strong or remove them.

Enroll new ykfde passphrase to existing YubiKey LUKS encrypted volume

To enroll new ykfde passphrase to existing YubiKey *LUKS* encrypted volume you can use ykfde-enroll script, see `ykfde-enroll -h` for help:

```
ykfde-enroll -d /dev/<device> -s <keyslot_number> -o
```

Unlock LUKS encrypted volume protected by ykfde passphrase

To unlock *LUKS* encrypted volume on a running system, you can use ykfde-open script, see `ykfde-open -h` for help.

As unprivileged user using udisksctl (recommended):

```
ykfde-open -d /dev/<device>
```

As root using cryptsetup (when udisks is not available):

```
ykfde-open -d /dev/<device> -n <volume_name>
```

To print only the ykfde passphrase to the console without unlocking any volumes:

```
ykfde-open -p
```

To test only a passphrase for a specific key slot:

```
ykfde-open -d /dev/<device> -s <keyslot_number> -t
```

Killing ykfde passphrase for existing LUKS encrypted volume

To kill a ykfde passphrase for existing *LUKS* encrypted volume you can use ykfde-enroll script, see `ykfde-enroll -h` for help:

```
ykfde-enroll -d /dev/<device> -s <keyslot_number> -k
```

Enable ykfde initramfs hook

Warning: It's recommended to have already working encrypted system setup with `encrypt` hook and non-ykfde passphrase before starting to use `ykfde` hook with ykfde passphrase to avoid potential misconfigurations.

Edit `/etc/mkinitcpio.conf` and add the `ykfde` hook before or instead of `encrypt` hook as provided in example. Adding `ykfde` hook before `encrypt` hook will allow for a safe fallback in case of ykfde misconfiguration. You can remove `encrypt` hook later when you confim that everything is working correctly. After making your changes regenerate initramfs:

```
sudo mkinitcpio -P
```

Reboot and test your configuration.

Enable ykfde suspend service (experimental)

You can enable the `ykfde-suspend` service which allows for automatically locking encrypted *LUKS* volumes and wiping keys from memory on suspend and unlocking them on resume by using `cryptsetup luksSuspend` and `cryptsetup luksResume` commands.

Warning: RAM storage stays unencrypted in that case.

Edit `/etc/mkinitcpio.conf` and add `shutdown` hook as the last in `HOOKS` array. After making your changes regenerate initramfs:

```
sudo mkinitcpio -P
```

Enable related systemd service:

```
systemctl enable ykfde-suspend.service
```

Reboot and test your configuration.

License