Yubikey based Full Disk Encryption (FDE) on NixOS

From NixOS Wiki
This page is a minimalistic guide for setting up LUKS-based full disk encryption with YubiKey pre-boot authentication (PBA) on a UEFI system using the BRTFS file system (although any file system can be used). The YubiKey PBA in NixOS currently features two-factor authentication using a (secret) user passphrase and a YubiKey in challenge-response mode. The described method also works without a user password, although this is not preferred. In the 19.03 release (and prior) this method will change the LUKS authentication key on each boot that passes the LVM mount stage by altering a salt value contained on the boot partition.

This guide was tested to work on NixOS 19.03 as of July 2019. Also see this repository (https://github.com/sgillespie/nixos-yubikey-luks) that provides a nix-shell expression as described in this guide. It provides the environment for setting up the yubikey.

Requirements

- A NixOS live system booted in UEFI mode on the target machine.
- A YubiKey Standard plugged into the target machine with a free configuration slot (that will be overwritten).

Setup

Install the packages required by the next steps to the live system and make two bash helper functions available.

Automatic Setup

Enter the nix-shell expression defined by this repository (https://github.com/sgillespie/nixos-yubikey-luks).

```
nix-shell https://github.com/sgillespie/nixos-yubikey-luks/archive/master.tar.gz
```

Manual Setup

Alternatively, you can manually set up the dependencies.

Packages:

- A C compiler, e.g. gcc
- The YubiKey Personalization command line tool
- OpenSSL

```
nix-env -i gcc-wrapper
nix-env -i yubikey-personalization
nix-env -i openssl
```

Helper functions:

- Convert a raw binary string to a hexadecimal string
- Convert a hexadecimal string to a raw binary string

Note that you can copy and paste these functions into the bash shell directly to define them.

```
rbtohex() {
    ( od -An -vtx1 | tr -d ' \n' )
}

hextorb() {
    ( tr '[:lower:]' '[:upper:]' | sed -e 's/\([0-9A-F]\{2\}\)/\\\\\\x\1/gI'| xargs printf )
}
```

We need to compile OpenSSL's key derivation function, which is the same one as run on start up. To compile, run the following command.

Note: *Because this will put the program in the current directory (rather than your PATH), replace pbkdf2-sha512 commands with ./pbkdf2-sha512.*

```
cc -O3 \
   -I$(nix-build "<nixpkgs>" --no-build-output -A openssl.dev)/include \
   -L$(nix-build "<nixpkgs>" --no-build-output -A openssl.out)/lib \
   $(nix eval "(with import <nixpkgs> {}; pkgs.path)")/nixos/modules/system/boot/pbkdf2-sha512.c \
   -o ./pbkdf2-sha512 -lcrypto
```

If the newlines in the above snippet are problematic for your terminal, you can use the snippet below. It is the same command but as one line.

```
cc -O3 -I$(nix-build "<nixpkgs>" --no-build-output -A openssl.dev)/include -L$(nix-build "<nixpkgs>"
```

Set up the Yubikey

Step 1: Configure the Yubikey

```
SLOT=2
ykpersonalize -"$SLOT" -ochal-resp -ochal-hmac
```

Step 2: Gather the initial salt for the PBA (set its length to what you find time-feasible on your machine).

```
SALT_LENGTH=16
salt="$(dd if=/dev/random bs=1 count=$SALT_LENGTH 2>/dev/null | rbtohex)"
```

Step 3: Get the user passphrase used as the second factor in the PBA.

If you plan on using a user password during the boot process (instead of an unassisted boot), enter a user password. Your choice here will change the command you run in step 8.

```
read -s k_user
```

Step 3.5: Make very sure, that $k_user contains the correct user passphrase, or you will not be able to access your system after shutting down the live system.

Step 4: Calculate the initial challenge and response to the YubiKey.

```
challenge="$(echo -n $salt | openssl dgst -binary -sha512 | rbtohex)"
response="$(ykchalresp -2 -x $challenge 2>/dev/null)"
```

Step 5: Derive the Luks slot key from the two factors.

Set the length of the Luks slot key and the cipher appropriately.

As an example, we will use AES-256, so we set the Luks device slot key length to 512 bit.

Set the iteration count used for PBKDF2 to a high value still time-feasible for your machine.

```
KEY_LENGTH=512
ITERATIONS=1000000
```

If you choose to authenticate with a user password, use the following line to generate the luks key.

```
k_luks="$(echo -n $k_user | pbkdf2-sha512 $(($KEY_LENGTH / 8)) $ITERATIONS $response | rbtohex)"
```

If you choose to authenticate without a user passphrase (not recommended), use this instead of the line above

```
k_luks="$(echo | pbkdf2-sha512 $(($KEY_LENGTH / 8)) $ITERATIONS $response | rbtohex)"
```

To test if the key is programmed correctly, you can challenge the yubikey and check that the response is the expected response previously generated (echo $response).

Partitioning

Create a GPT partition table and two partitions on the target disk.

- Partition 1: This will be the EFI system partition: 100MB-300MB
- Partition 2: This will be the Luks-encrypted partition, aka the "luks device": Rest of your disk

In the following we will use variables for identification, so set them to match your partition setup, e.g. like this:

```
EFI_PART=/dev/sda1
LUKS_PART=/dev/sda2
```

If you use an nvme drive your partition names will be something like /dev/nvme0n1p1 instead of /dev/sda1.

Setup the LUKS device

Step 6: Create the necessary filesystem on the efi system partition, which will store the current salt for the PBA, and mount it.

```
EFI_MNT=/root/boot
mkdir "$EFI_MNT"
mkfs.vfat -F 32 -n uefi "$EFI_PART"
mount "$EFI_PART" "$EFI_MNT"
```

Step 7: Decide where on the efi system partition to store the salt and prepare the directory layout accordingly.

```
STORAGE=/crypt-storage/default
mkdir -p "$(dirname $EFI_MNT$STORAGE)"
```

Step 8: Store the salt and iteration count to the EFI systems partition.

```
echo -ne "$salt\n$ITERATIONS" > $EFI_MNT$STORAGE
```

Step 9: Create the LUKS device.

- Set the cipher used by LUKS appropriately
- Set the hash used by LUKS appropriately

```
CIPHER=aes-xts-plain64
HASH=sha512
echo -n "$k_luks" | hextorb | cryptsetup luksFormat --cipher="$CIPHER" \
  --key-size="$KEY_LENGTH" --hash="$HASH" --key-file=- "$LUKS_PART"
```

LVM setup

The following is one of many methods for setting up the LVM partition. Another popular guide can be found here: https://qfpl.io/posts/installing-nixos/ (https://qfpl.io/posts/installing-nixos/)

Step 1: Setup the LUKS device as a physical volume.

The LUKS device first needs to be unlocked.

```
LUKSROOT=nixos-enc
echo -n "$k_luks" | hextorb | cryptsetup luksOpen $LUKS_PART $LUKSROOT --key-file=-
```

The physical volumje map can then be created.

```
pvcreate "/dev/mapper/$LUKSROOT"
```

Step 2: Setup a volume group on the LUKS device.

Set the name for the volume group appropriately

```
VGNAME=partitions
vgcreate "$VGNAME" "/dev/mapper/$LUKSROOT"
```

Step 3: Setup two logical volumes on the Luks device.

- Volume 1: This will be the swap partition: choose appropriate size, 2GB for example
- Volume 2: This will be the main btrfs volume, of which all filesystem partitions will be subvolumes: Rest of the free space

```
lvcreate -L 2G -n swap "$VGNAME"
FSROOT=fsroot
lvcreate -l 100%FREE -n "$FSROOT" "$VGNAME"

vgchange -ay
```

Step 4: Create the swap filesystem.

```
mkswap -L swap /dev/partitions/swap
```

Btrfs setup

These steps can mostly be followed the same for other filesystem types except calls to the brtfs command can be ignored.

Step 1: Create the main btrfs volume's filesystem.

```
mkfs.btrfs -L "$FSROOT" "/dev/partitions/$FSROOT"
```

Should the above fail, you might have encountered a bug that can be solved with doing the following, then attempting the above again:

```
mkdir /mnt-root
touch /mnt-root/nix-store.squashfs
```

Step 2: Mount the main btrfs volume.

```
mount "/dev/partitions/$FSROOT" /mnt
```

Step 3: Create the subvolumes, for example "root" and "home".

```
cd /mnt
btrfs subvolume create root
btrfs subvolume create home
```

Step 4: Create mountpoints on the root subvolume and finalise things for NixOS installation.

```
umount /mnt
mount -o subvol=root "/dev/partitions/$FSROOT" /mnt

mkdir /mnt/home
mount -o subvol=home "/dev/partitions/$FSROOT" /mnt/home

mkdir /mnt/boot
mount "$EFI_PART" /mnt/boot

swapon /dev/partitions/swap
```

NixOS installation

Configure and install NixOS as you normally would. Add the following to your configuration.nix, noting that more options are available here: https://search.nixos.org/options/?query=yubikey (https://search.nixos.org/options/?query=yubikey).

Replace anything that looks like a Bash variable with the value that it currently holds for in your shell and modify as needed.

```
# Minimal list of modules to use the EFI system partition and the YubiKey
boot.initrd.kernelModules = [ "vfat" "nls_cp437" "nls_iso8859-1" "usbhid" ];

# Enable support for the YubiKey PBA
boot.initrd.luks.yubikeySupport = true;

# Configuration to use your Luks device
boot.initrd.luks.devices = {
  "$LUKSROOT" = {
    device = "$LUKS_PART";
    preLVM = true; # You may want to set this to false if you need to start a network service first
    yubikey = {
      slot = $SLOT;
      twoFactor = true; # Set to false if you did not set up a user password.
      storage = {
        device = "$EFI_PART";
      };
    };
  };
};
```

Headless setup note: If you have set up your system to not use a user password and attempt to boot the system, you may find the system stalls with the following message:

"Gathering entropy for new salt (please enter random keys to generate entropy if this blocks for long)..."

If you see this message and no more dots appear after a while, you have run into a situation where the random number generator does not have enough entropy stored up. You can mitigate this by starting a network interface (assuming the device is on a network), which should fill the entropy pool and allow the computer to boot headless. Below is an example configuration that has been tested to work in a headless configuration.

```
boot = {
    # Used to make this device dhcp enabled during boot.
    kernelParams = [
      "ip=:::::eth0:dhcp" # Change to the appropriate IP kernel command.
    ];

    initrd = {

      network.enable = true;

      # This is the driver for a particular ethernet card. See `boot.network.enable` for more detai
      availableKernelModules = ["alx"];

      kernelModules = ["vfat" "nls_cp437" "nls_iso8859-1" "usbhid" "alx"];

      luks = {
        cryptoModules = [ "aes" "xts" "sha512" ];
        yubikeySupport = true;

        devices = [ {
          name = "$LUKSROOT";
          device = "$LUKS_PART";
          preLVM = false;
          yubikey = {
            slot = $SLOT;
            twoFactor = false;
            storage = {
              device = "$EFI_PART";
            };
          };
        } ];

      };
    };
  };
```

Finally, clean up and you should be ready to reboot into your new system.

This is a modified copy of the page from the original wiki. The original page can be found here: https://web.archive.org/web/20160911070220/https://nixos.org/wiki/Luks-based_FDE_with_Yubikey_PBA_and_btrfs_on_UEFI_NixOS (https://web.archive.org/web/20160911070220/https://nixos.org/wiki/Luks-based_FDE_with_Yubikey_PBA_and_btrfs_on_UEFI_NixOS)