Secure Boot with GRUB 2 and signed Linux images and initrds

back

Note A security vulnerability was fixed on 2016-06-02 which allowed an attacker to gain access to a GRUB rescue shell, thus bypassing Secure Boot. Thanks to Julian Brost for reporting this vulnerability.

There are many guides available how to setup Secure Boot with custom keys and load signed Linux kernels with built-in initrds. But I didn't find anything which allows me to securely boot kernels which use separate initrds (and thus don't require a kernel rebuild when the initrd updates) — the typical setup on e.g. Debian.

The idea is to create a signed GRUB EFI binary which contains the required modules and base config. Secure boot verifies this binary during boot. GRUB then loads the signed `grub.cfg` which contains the available kernels and then loads the selected (signed) kernels and initrds. GRUB's verification is based on GPG which is independent of Secure Boot.

The guide is based on a Debian system, but should be easily adaptable. It was tested with Debian's GRUB 2.02~beta2-36 (sid) and 2.02~beta2-22+deb8u1 (jessie) and efitools 1.7.0 running on GNU/Linux kernel 4.5.

Setup Secure Boot

Use any of the available guides for details, here's just the short version (the `efitools` README is also quite useful).

Enable `Setup Mode` in your UEFI firmware (delete all existing keys) to add your custom keys. Install `efitools` and `sbsigntool` on your system.

Create the keys:

```
openssl req -new -x509 -newkey rsa:2048 -subj "/CN=PK/"  -keyout PK.key  -out PK.crt  -days 7300 -nodes -sha256
openssl req -new -x509 -newkey rsa:2048 -subj "/CN=KEK/" -keyout KEK.key -out KEK.crt -days 7300 -nodes -sha256
openssl req -new -x509 -newkey rsa:2048 -subj "/CN=db/"  -keyout db.key  -out db.crt  -days 7300 -nodes -sha256
```

Prepare installation in EFI:

```
cert-to-efi-sig-list PK.crt PK.esl
sign-efi-sig-list -k PK.key -c PK.crt PK PK.esl PK.auth

cert-to-efi-sig-list KEK.crt KEK.esl
sign-efi-sig-list -k PK.key -c PK.crt KEK KEK.esl KEK.auth

cert-to-efi-sig-list db.crt db.esl
sign-efi-sig-list -k KEK.key -c KEK.crt db db.esl db.auth
```

Install keys into EFI (`PK` last as it will enable `Custom Mode` locking out further unsigned changes):

```
efi-updatevar -f db.auth db
efi-updatevar -f KEK.auth KEK
efi-updatevar -f PK.auth PK
```

The EFI variables may be immutable (`i`-flag in `lsattr` output) in recent kernels (e.g. 4.5.4). Use `chattr -i` to make them mutable again if you can't update the variables with the commands above:

```
chattr -i /sys/firmware/efi/efivars/{PK,KEK,db,dbx}-*
```

From now on only EFI binaries signed with any `db` key can be loaded. To sign a binary use:

```
sbsign --key /path/to/db.key --cert /path/to/db.crt /path/to/efi
```

Then use the `.signed` file to boot.

GRUB setup

GRUB 2 supports loading of GPG signed files only (e.g. config or kernels) through the `verify` module. The `grub-mkstandalone` command can be used to create a single GRUB binary. It can also embed an initial `grub.cfg` (however this file must be signed!). The `--pubkey` option adds a GPG public key which will be used for verification. However unlike the documentation states, the `verify` module must be manually included.

Signed grub image

The following minimal grub config, stored in `grub-initial.cfg` is used:

```
# Enforce that all loaded files must have a valid signature.
set check_signatures=enforce
export check_signatures

set superusers="root"
export superusers
password_pbkdf2 root grub.pbkdf2.sha512.10000.TODO

# NOTE: We export check_signatures/superusers so they are available in all
# further contexts to ensure the password check is always enforced.

# First partition on first disk, most likely EFI system partition. Set it here
# as fallback in case the search doesn't find the given UUID.
set root='hd0,gpt1'
search --no-floppy --fs-uuid --set=root TODO
# example, see below: search --no-floppy --fs-uuid --set=root 891F-FF86

configfile /grub.cfg

# Without this we provide the attacker with a rescue shell if he just presses
# <return> twice.
echo /EFI/grub/grub.cfg did not boot the system but returned to initial.cfg.
echo Rebooting the system in 10 seconds.
sleep 10
reboot
```

Note In the past the `sleep`/`reboot` part was missing (and `configfile` was `source`) which allowed an attacker to gain access to a rescue shell when `grub.cfg` could not be loaded and the attacker pressed return twice to skip the password dialog. As the attacker can modify the file system this was a major issue.

Replace the first `TODO` with the result of `grub-mkpasswd-pbkdf2` with your custom passphrase. The user/password is required to restrict access to the GRUB shell which allows running arbitrary commands. The GRUB documentation states that `check_signatures=enforce` will prevent any future loading of unsigned files so an attacker shouldn't be able to load any modified files, but better be safe.

Replace the second `TODO` with the UUID of your EFI system partition. With the following output of `lsblk -f`, replace it with `891F-FF86`.

```
NAME        FSTYPE      LABEL UUID           MOUNTPOINT
sda
├─sda1      vfat              891F-FF86      /boot/efi
...
```

The mentioned `grub.cfg` will contain the current list of kernels and is not stored in the EFI file directly so the Secure Boot `db` key doesn't have to be available when a new kernel is installed. Only the GPG key used by GRUB is necessary. The `grub.cfg` is of course also signed with this GPG key.

The following script will use a Secure Boot `db` key (generated above) to provide a single GRUB EFI binary which contains the `grub-initial.cfg`. Adapt the GPG key id and the paths as necessary.

```
#!/bin/sh

set -eu

GPG_KEY='TODO: add your GPG key id here'
TARGET_EFI='/boot/efi/EFI/boot/bootx64.efi'
SECUREBOOT_DB_KEY='../keys/db.key'
SECUREBOOT_DB_CRT='../keys/db.crt'

# GRUB doesn't allow loading new modules from disk when secure boot is in
# effect, therefore pre-load the required modules.
MODULES=
MODULES="$MODULES part_gpt fat ext2"           # partition and file systems for EFI
MODULES="$MODULES configfile"                  # source command
MODULES="$MODULES verify gcry_sha512 gcry_rsa" # signature verification
MODULES="$MODULES password_pbkdf2"             # hashed password
MODULES="$MODULES echo normal linux linuxefi"  # boot linux
MODULES="$MODULES all_video"                   # video output
MODULES="$MODULES search search_fs_uuid"       # search --fs-uuid
MODULES="$MODULES reboot sleep"                # sleep, reboot

rm -rf tmp
mkdir -p tmp

TMP_GPG_KEY='tmp/gpg.key'
TMP_GRUB_CFG='tmp/grub-initial.cfg'
TMP_GRUB_SIG="$TMP_GRUB_CFG.sig"
TMP_GRUB_EFI='tmp/grubx64.efi'

gpg --export "$GPG_KEY" >"$TMP_GPG_KEY"

cp grub-initial.cfg "$TMP_GRUB_CFG"
rm -f "$TMP_GRUB_SIG"
gpg --default-key "$GPG_KEY" --detach-sign "$TMP_GRUB_CFG"

grub-mkstandalone \
    --directory /usr/lib/grub/x86_64-efi \
    --format x86_64-efi \
    --modules "$MODULES" \
    --pubkey "$TMP_GPG_KEY" \
    --output "$TMP_GRUB_EFI" \
    "boot/grub/grub.cfg=$TMP_GRUB_CFG" \
    "boot/grub/grub.cfg.sig=$TMP_GRUB_SIG"

sbsign --key "$SECUREBOOT_DB_KEY" --cert "$SECUREBOOT_DB_CRT" \
    --output "$TMP_GRUB_EFI" "$TMP_GRUB_EFI"

echo "writing signed grub.efi to '$TARGET_EFI'"
cp "$TMP_GRUB_EFI" "$TARGET_EFI"

rm -r tmp
```

Note This script will overwrite your current default boot loader! Ensure you have a second way of booting (e.g. USB-stick) in case something goes wrong.

Install grub.cfg and signed kernels/initrds

The only part missing is the (signed) custom GRUB config and the signed kernels and initrds.

We'll be using three files to create the `grub.cfg`: `grub.cfg.head` is added to the beginning of the config, `grub.cfg.tail` to the end. Those can be used for custom settings/entries. For each existing kernels/initrds the third file `grub.cfg.menu` is used as template for a single menu entry.

In the simplest setup both `grub.cfg.head` and `grub.cfg.tail` are empty and `grub.cfg.menu` could look like this:

```
menuentry 'Debian GNU/Linux, with Linux VERSION' --unrestricted {
    echo   'Loading Linux VERSION ...'
    linux  /VMLINUZ root=TODO add your normal command line here
    echo   'Loading initial ramdisk ...'
    initrd /INITRD
}
```

The `--unrestricted` option allows booting without entering user/password, but doesn't allow modification of the boot entry.

The following script uses the kernels/initrds from `/boot` to create a signed `grub.cfg` with matching menu entries and signs and copies the config and all kernels/initrds to `/boot/efi` where they will be verified and loaded by our GRUB EFI binary created above. Adapt the GPG key and paths as necessary.

```
#!/bin/sh

set -eu

GPG_KEY='TODO: add your GPG key id here'
BOOT_DIRECTORY='/boot'      # source, no trailing /
EFI_DIRECTORY='/boot/efi'   # target, no trailing /
KERNEL_PREFIX='vmlinuz-'
INITRD_PREFIX='initrd.img-'

escape_for_sed() {
    printf '%s' "$1" | sed 's!/!\\/!g'
}

rm -rf tmp
mkdir tmp

TMP_GRUB_CFG=tmp/grub.cfg
TMP_KERNELS=tmp/kernels

: >"$TMP_KERNELS"
for x in "$BOOT_DIRECTORY/$KERNEL_PREFIX"*; do
    printf '%s\n' "$x" >>"$TMP_KERNELS"
done

# Newest kernel first.
cat grub.cfg.head >"$TMP_GRUB_CFG"
sort --reverse --version-sort "$TMP_KERNELS" | while read vmlinuz; do
    vmlinuz="${vmlinuz##$BOOT_DIRECTORY/}"
    version="${vmlinuz##$KERNEL_PREFIX}"
    initrd="${INITRD_PREFIX}${version}"

    vmlinuz="$(escape_for_sed "$vmlinuz")"
    version="$(escape_for_sed "$version")"
    initrd="$(escape_for_sed "$initrd")"

    sed -e "s/VERSION/$version/g" \
        -e "s/VMLINUZ/$vmlinuz/g" \
        -e "s/INITRD/$initrd/g" \
        <grub.cfg.menu \
        >>"$TMP_GRUB_CFG"
done
cat grub.cfg.tail >>"$TMP_GRUB_CFG"

for x in "$TMP_GRUB_CFG" "$BOOT_DIRECTORY/$KERNEL_PREFIX"* "$BOOT_DIRECTORY/$INITRD_PREFIX"*; do
    name="$(basename "$x")"

    echo "signing and copying '$x' to '$EFI_DIRECTORY'"
    cp "$x" "$EFI_DIRECTORY"
    rm -f "$EFI_DIRECTORY/$name.sig"
    gpg --default-key "$GPG_KEY" --detach-sign "$EFI_DIRECTORY/$name"
done

rm -r tmp
```

Testing

After running both scripts, `/boot` should look something like this:

```
/boot
├── ...
├── efi
│   ├── EFI
│   │   └── boot
│   │       └── bootx64.efi
│   ├── grub.cfg
│   ├── grub.cfg.sig
│   ├── initrd.img-4.5.0-2-amd64
│   ├── initrd.img-4.5.0-2-amd64.sig
│   ├── vmlinuz-4.5.0-2-amd64
│   └── vmlinuz-4.5.0-2-amd64.sig
├── ...
├── initrd.img-4.5.0-2-amd64
└── vmlinuz-4.5.0-2-amd64
```

The files directly in `/boot` are those from the kernel package and were signed and copied to `/boot/efi` (which is the disk's EFI partition).

`/boot/efi/grub.cfg` should contain the following:

```
menuentry 'Debian GNU/Linux, with Linux 4.5.0-2-amd64' --unrestricted {
    echo   'Loading Linux 4.5.0-2-amd64 ...'
    linux  /vmlinuz-4.5.0-2-amd64 root=...
    echo   'Loading initial ramdisk ...'
    initrd /initrd.img-4.5.0-2-amd64
}
```

Reboot and GRUB should load as usual, verifying the config and kernels/initrds and booting as usual.

Automatic update

Most systems provide hooks to run when installing new kernels. This can be used to automatically update the config and kernels in `/boot/efi`.

On Debian this is `/etc/kernel/postinst.d/` and `/etc/kernel/postrm.d/`. Just put a file which calls the second script from above (which creates the config and copies the kernels) in those directories and make the GPG key available.

Encrypted `/boot`

Encrypting `/boot` prevents potential attacks where the stored kernels are first modified by an attacker and then resigned by the user when e.g. creating a new initrd without validation.

With the described setup, encrypting `/boot` is easy. Just move it inside your LUKS partition, e.g. create a new logical volume and mount it as `/boot`. The unencrypted EFI system partition can then be mounted to `/boot/efi` as usual. As all data required to boot (GRUB, kernel, initrd) is stored in the EFI system partition, this works out of the box and keeps compatible with Debian’s usual way to install images to `/boot`.

back

Last updated 2018-05-20 17:09:22 CEST