Secure Boot

The Right Way

Simon Kadesh

2022-10-28

The Wrong Way

- Disable it
  - Vulnerable
  - Evil Maids
- Use a pre-signed loader
  - PreLoader and shim
  - initramfs signing
  - updates
  - encryption

The Right Way

- PKI
- Key Creation
- Kernel Signing
- Unified Kernel Image
- Automation

PKI

- Platform Key - OEM key
- Key Exchange Key - Signed by Platform Key, Signs DBs
- Signature Database - Allow list
- Forbidden Signatures Database - Deny list

Key Creation

Generate UUID:

```
$ uuidgen --random > GUID.txt
```

Platform Key:

```
$ openssl req -newkey rsa:4096 -nodes -keyout PK.key -new -x509 -sha256 \
      -days 3650 -subj "/CN=my Platform Key/" -out PK.crt
$ openssl x509 -outform DER -in PK.crt -out PK.cer
$ cert-to-efi-sig-list -g "$(< GUID.txt)" PK.crt PK.esl
$ sign-efi-sig-list -g "$(< GUID.txt)" -k PK.key -c PK.crt PK PK.esl PK.auth
```

Key Exchange Key:

```
$ openssl req -newkey rsa:4096 -nodes -keyout KEK.key -new -x509 -sha256 \
      -days 3650 -subj "/CN=my Key Exchange Key/" -out KEK.crt
$ openssl x509 -outform DER -in KEK.crt -out KEK.cer
$ cert-to-efi-sig-list -g "$(< GUID.txt)" KEK.crt KEK.esl
$ sign-efi-sig-list -g "$(< GUID.txt)" -k PK.key -c PK.crt KEK KEK.esl KEK.auth
```

Signature Database:

```
$ openssl req -newkey rsa:4096 -nodes -keyout db.key -new -x509 -sha256 \
      -days 3650 -subj "/CN=my Signature Database key/" -out db.crt
$ openssl x509 -outform DER -in db.crt -out db.cer
$ cert-to-efi-sig-list -g "$(< GUID.txt)" db.crt db.esl
$ sign-efi-sig-list -g "$(< GUID.txt)" -k KEK.key -c KEK.crt db db.esl db.auth
```

Kernel Signing

We use sbsigntools to sign EFI binaries For example:

```
# sbsign --key db.key --cert db.crt --output /boot/vmlinuz-linux \
        /boot/vmlinuz-linux
# sbsign --key db.key --cert db.crt --output <esp>/EFI/BOOT/BOOTx64.EFI \
        <esp>/EFI/BOOT/BOOTx64.EFI
```

These commands sign a linux kernel at /boot/vmlinuz-linux and a boot manager at <esp>/EFI/BOOT/BOOTx64.EFI with our allow database key, then replaces them with their signed versions At this we still run into the problem of having a boot manager between our firmware and our kernel, as well as that of unsigned initramfs

Unified Kernel Images

Instead of having a separate kernel and initramfs image, we can join them into one big file. How exactly this will work depends on your system, but an example can look like:

```
# mkinitcpio -p linux --uefi <esp>/EFI/Linux/linux.efi
```

We can set out kernel parameters in `/etc/kernel/cmdline` to be built into our initramfs. We then add the following to the `mkinitcpio` presets for the kernels we are using:

```
<preset>_efi_image="<esp>/EFI/Linux/linux.efi"
```

This will result in a bootable efi binary that we can sign like in the previous slide, but without the added vulnerability of unsigned initramfs and a boot manager

Automation

Some ideas for how to make this process easier (imagine manually signing your kernel every time there's an update)

- `sbkeysync` - Easily enroll your keys in the firmware once you generate them
- Package manager hooks
  - `/etc/pacman.d/hooks`
  - `/etc/apt/apt.conf.d`
- Scripts - automate the commands needed to sign your kernel
- `sbupdate` - AUR package with sane defaults to make kernel images once you have your keys