SecureBoot

UEFI Secure Boot • Key Management • Signing binaries for Secure Boot • Dealing with third-party kernels • Dealing with third-party drivers • FAQ

Contents

What is UEFI Secure Boot?

UEFI Secure boot is a verification mechanism for ensuring that code launched by firmware is trusted.

Proper, secure use of UEFI Secure Boot requires that each binary loaded at boot is validated against known keys, located in firmware, that denote trusted vendors and sources for the binaries, or trusted specific binaries that can be identified via cryptographic hashing.

Most x86 hardware comes from the factory pre-loaded with Microsoft keys. This means we can generally rely on the firmware on these systems to trust binaries that are signed by Microsoft, and the Linux community heavily relies on this assumption for Secure Boot to work. This is the same process used by Red Hat and SUSE, for instance.

Many ARM and other architectures also support UEFI Secure Boot, but may not be pre-loading keys in firmware. On these architectures, it may be necessary to re-sign boot images with a certificate that is loaded in firmware by the owner of the hardware.

Initial implementation plan: Implementation Plan.

Supported architectures

- **amd64**: A shim binary signed by Microsoft and grub binary signed by Canonical are provided in the Ubuntu main archive as *shim-signed* or *grub-efi-amd64-signed*.
- **arm64**: As of 20.04 ('focal'), a shim binary signed by Microsoft and grub binary signed by Canonical are provided in the Ubuntu main archive as *shim-signed* or *grub-efi-arm64-signed*. There is a GRUB bug under investigation that needs to be resolved before this works end to end.

Testing UEFI Secure Boot

If you're interested in testing Secure Boot on your system, consult the how-to here: UEFI/SecureBoot/Testing.

How UEFI Secure Boot works on Ubuntu

On Ubuntu, all pre-built binaries intended to be loaded as part of the boot process, with the exception of the initrd image, are signed by Canonical's UEFI certificate, which itself is implicitly trusted by being embedded in the shim loader, itself signed by Microsoft.

On architectures or systems where pre-loaded signing certificates from Microsoft are not available or loaded in firmware, users may replace the existing signatures on shim or grub and load them as they wish, verifying against their own certificates imported in the system's firmware.

As the system boots, firmware loads the shim binary as specified in firmware BootEntry variables. Ubuntu installs its own BootEntry at installation time and may update it any time the GRUB bootloader is updated. Since the shim binary is signed by Microsoft; it is validated and accepted by the firmware when verifying against certificates already present in firmware. Since the shim binary embeds a Canonical certificate as well as its own trust database, further elements of the boot environment can, in addition to being signed by one of the acceptable certificates pre-loaded in firmware, be signed by Canonical's UEFI key.

The next thing loaded by shim is the second-stage image. This can be one of two things: either GRUB, if the system is booting normally; or MokManager, if key management is required, as configured by firmware variables (usually changed when the system was previously running).

If booting normally; the GRUB binary (grub*.efi) is loaded and its validation is attempted against all previously-known trusted sources. The GRUB binary for Ubuntu is signed by the Canonical UEFI key, so it is successfully validated and the boot process continues.

If booting to proceed with key management tasks, the MokManager binary (mm*.efi) is loaded. This binary is explicitly trusted by shim by being signed by an ephemeral key that only exists while the shim binary is being built. This means only the MokManager binary built with a particular shim binary will be allowed to run and limits the possibility of compromise from the use of compromised tools. MokManager allows any user present at the system console to enroll keys, remove trusted keys, enroll binary hashes and toggle Secure Boot validation at the shim level, but most tasks require a previously set password to be entered to confirm that the user at console is indeed the person who requested changes. Such passwords only survive across a single run of shim / MokManager; and are cleared as soon as the process is completed or cancelled. Once key management is completed, the system is rebooted and does not simply continue with booting, since the key management changes may be required to successfully complete the boot.

Once the system continues booting to GRUB; the GRUB process loads any required configuration (usually loading configuration from the ESP (EFI System Partition), pointing to another configuration file on the root or boot partition), which will point it to the kernel image to load.

EFI applications up to this point having full access to the system firmware, including access to changing trusted firmware variables, the kernel to load must also be validated against the trust database. Official Ubuntu kernels being signed by the Canonical UEFI key, they are successfully validated, and control is handed over to the kernel. Initrd images are not validated.

In the case of unofficial kernels, or kernels built by users, additional steps need to be taken if users wish to load such kernels while retaining the full capabilities of UEFI Secure Boot. All kernels must be signed to be allowed to load by GRUB when UEFI Secure Boot is enabled, so the user will require to proceed with their own signing. Alternatively, users may wish to disable validation in shim while booted with Secure Boot enabled on an official kernel by using 'sudo mokutil --disable-validation', providing a password when prompted, and rebooting; or to disable Secure Boot in firmware altogether.

Up to this point, any failure to validate an image to load is met with a critical error which stops the boot process. The system will not continue booting, and may automatically reboot after a period of time given that other BootEntry variables may contain boot paths that are valid and trusted.

Once loaded, validated kernels will disable the firmware's Boot Services, thus dropping privileges and effectively switching to user mode; where access to trusted variables is limited to read-only. Given the broad permissions afforded to kernel modules, any module not built into the kernel will also need to be validated upon loading. Modules built and shipped by Canonical with the official kernels are signed by the Canonical UEFI key and as such, are trusted. Custom-built modules will require the user to take the necessary steps to sign the modules before they loading them is allowed by the kernel. This can be achieved by using the 'kmodsign' command [see {How to sign} section].

Unsigned modules are simply refused by the kernel. Any attempt to insert them with insmod or modprobe will fail with an error message.

Given that many users require third-party modules for their systems to work properly or for some devices to function; and that these third-party modules require building locally on the system to be fitted to the running kernel, Ubuntu provides tooling to automate and simplify the signing process.

How can I do non-automated signing of drivers?

Some projects may require the use of custom kernel drivers that are not set up in such a way as to work with DKMS. In these cases, people should make use of the tools included in the shim-signed package: the update-secureboot-policy script is available to generate a new MOK (if no DKMS-built modules have triggered generating one already).

Use the following command to enroll an existing key into shim:

```
sudo update-secureboot-policy --enroll-key
```

If no MOK exists, the script will exit with a message to that effect. If the key is already enrolled, the script will exit, doing nothing. If the key exists but it not shown to be enrolled, the user will be prompted for a password to use after reboot, so that the key can be enrolled.

One can generate a new MOK using the following command:

```
sudo update-secureboot-policy --new-key
```

And then enroll the newly-generated key into shim with the previously-mentioned command for that task.

Kernel modules can then be signed with the kmodsign command (see UEFI/SecureBoot/Signing) as part of their build process.

Security implications in Machine-Owner Key management

The MOK generated at installation time or on upgrade is machine-specific, and only allowed by the kernel or shim to sign kernel modules, by use of a specific KeyUsage OID (1.3.6.1.4.1.2312.16.1.2) denoting the limitations of the MOK.

Recent shim versions include logic to follow the limitations of module-signing-only keys. These keys will be allowed to be enrolled in the firmware in shim's trust database, but will be ignored when shim or GRUB validate images to load in firmware. Shim's verify() function will only successfully validate images signed by keys that do not include the "Module-signing only" (1.3.6.1.4.1.2312.16.1.2) KeyUsage OID. The Ubuntu kernels use the global trust database (which includes both shim's and the firmware's) and will accept any of the included keys as signing keys when loading kernel modules.

Given the limitations imposed on the automatically generated MOK and the fact that users with superuser access to the system and access to the system console to enter the password required when enrolling keys already have high-level access to the system; the generated MOK key is kept on the filesystem as regular files owned by root with read-only permissions. This is deemed sufficient to limit access to the MOK for signing by malicious users or scripts, especially given that no MOK exists on the system unless it requires third-party drivers. This limits the possibility of compromise from the misuse of a generated MOK key to signing a malicious kernel module. This is equivalent to compromise of the userland applications which would already be possible with superuser access to the system, and securing this is out of the scope of UEFI Secure Boot.

Previous systems may have had Secure Boot validation disabled in shim. As part of the upgrade process, these systems will be migrated to re-enabling Secure Boot validation in shim and enrolling a new MOK key when applicable.

MOK generation and signing process

The key generation and signing process is slightly different based on whether we are dealing with a brand new installation or an upgrade of system previously running Ubuntu; these two cases are clearly marked below.

In all cases, if the system is not booting in UEFI mode, no special kernel module signing steps or key generation will happen.

If Secure Boot is disabled, MOK generation and enrollment still happens, as the user may later enable Secure Boot. They system should work properly if that is the case.

The user installs Ubuntu on a new system

The user steps through the installer. Early on, when preparing to install and only if the system requires third-party modules to work, the user is prompted for a system password that is clearly marked as being required after the install is complete, and while the system is being installed, a new MOK is automatically generated without further user interaction.

Third-party drivers or kernel modules required by the system will be automatically built when the package is installed, and the build process includes a signing step. The signing step automatically uses the MOK generated earlier to sign the module, such that it can be immediately loaded once the system is rebooted and the MOK is included in the system's trust database.

Once the installation is complete and the system is restarted, at first boot the user is presented with the MokManager program (part of the installed shim loader), as a set of text-mode panels that all the user to enroll the generated MOK. The user selects "Enroll MOK", is shown a fingerprint of the certificate to enroll, and is prompted to confirm the enrollment. Once confirmed, the new MOK will be entered in firmware and the user will be asked to reboot the system.

When the system reboots, third-party drivers signed by the MOK just enrolled will be loaded as necessary.

The user upgrades an UEFI-enabled Ubuntu system to a new release where the system requires third-party drivers

On upgrade, the shim and shim-signed packages are upgraded. The shim-signed package's post-install tasks proceeds to generate a new MOK, and prompts the user for a password that is clearly mentioned as being required once the upgrade process is completed and the system rebooted.

During the upgrade, the kernel packages and third-party modules are upgraded. Third-party modules are rebuilt for the new kernels and their post-build process proceeds to automatically sign them with the MOK.

After upgrade, the user is recommended to reboot their system.

On reboot, the user is presented with the MokManager program (part of the installed shim loader), as a set of text-mode panels that all the user to enroll the generated MOK. The user selects "Enroll MOK", is shown a fingerprint of the certificate to enroll, and is prompted to confirm the enrollment. The user is also presented with a prompt to re-enable Secure Boot validation (in the case it was found to be disabled); and MokManager again requires confirmation from the user. Once all steps are confirmed, shim validation is re-enabled, the new MOK will be entered in firmware and the user will be asked to reboot the system.

When the system reboots, third-party drivers signed by the MOK just enrolled will be loaded as necessary.

In all cases, once the system is running with UEFI Secure Boot enabled and a recent version of shim; the installation of any new DKMS module (third-party driver) will proceed to sign the built module with the MOK. This will happen without user interaction if a valid MOK key exists on the system and appears to already be enrolled.

If no MOK exists or the existing MOK is not enrolled, a new key will automatically created just before signing and the user will be prompted to enroll the key by providing a password which will be required upon reboot.

UEFI/SecureBoot (last edited 2020-04-08 14:07:55 by dannf)