Securing the Linux Boot Process: From Start to Finish

Jakob Hagl[1], Oliver Mann[2] and Martin Pirker[3]
[1]*Software Security Group, SBA Research, Vienna, Austria*
[2]*Cyber Defense Center, Kapsch BusinessCom AG, Vienna, Austria*
[3]*Josef Ressel Center for Blockchain Technologies and Security Management, St. Pölten University of Applied Sciences, Austria*

Keywords: UEFI, Secure Boot, Linux, Boot Process, Full Disk Encryption, Privacy.

Abstract: The security of the operating system is a prominent feature in today's Linux distributions. A common security practice is to encrypt the hard drive, to protect the data at rest. The UEFI Forum released the secure boot specification, an optional boot process protocol that improves security during boot up on secure boot enabled hardware. A combination of secure boot with the Linux operating system, along with full disk encryption in an effort to implement maximum security is non-trivial. This paper explores the challenges of this undertaking and reports on a practical evaluation with five major Linux distributions, how far they support these security features by default and what can be improved manually.

1 INTRODUCTION

When looking back to the early days of computers, the startup processes of different vendors were neither uniform, nor compatible, or in any way secure. The BIOS (Basic Input/Output System) standardized the common low-level building block(s) on a personal computer. In an effort to further improve the boot loader functions and features of computers, the industry founded the UEFI (Unified Extensible Firmware Interface) forum (UEFI Forum, 2019). UEFI's mission is to improve the early code executing in every PC, to specify a more secure boot process, with faster boot times, improved performance and a more cost-effective time-to-market shipment of products. One of the many improvements and changes with UEFI was the so-called UEFI secure boot protocol. Overall, these developments, alongside technologies such as full disk encryption, provide an excellent basis for securing the startup process on modern computer hardware. However, secure boot is as of today mainly used by the Microsoft Windows line of operating systems, mostly due to Microsoft's dominating position in the market of operating systems.

Contribution. This papers reviews the options for deployment of UEFI secure boot in combination with full disk encryption on five major Linux distributions: Arch Linux, Debian, Fedora, openSUSE and Ubuntu. On the one hand this provides an up to date overview on what features are available on the reviewed distributions by default, and on the other hand this is also a brief guide on how to implement improvements to these features in practice.

2 BACKGROUND

2.1 UEFI

The UEFI specification is the successor of the earlier EFI (Extensible Firmware Interface) specification from Intel. It defines a standard environment for booting operating systems and for execution of early pre-boot applications. The reference implementation of the specification is TianoCore (TianoCore, 2017), an open source implementation based on the original "Tiano" EFI code base as donated by Intel in 2004.

With the emergence of UEFI secure boot a specific class of devices was identified by Richardson (Richardson, 2017): Devices in UEFI class "3+" have secure boot always enabled *and* boot only UEFI compatible boot loaders. UEFI Class 3 and higher do *not* offer support for a traditional "legacy BIOS" boot process via a compatibility support module (CSM), and Intel planned to deprecate CSM support by 2020.

Hagl, J., Mann, O. and Pirker, M.
Securing the Linux Boot Process: From Start to Finish.
DOI: 10.5220/0010313906040610
In *Proceedings of the 7th International Conference on Information Systems Security and Privacy (ICISSP 2021)*, pages 604-610
ISBN: 978-989-758-491-6

2.2 UEFI Secure Boot

The UEFI secure boot protocol protects the boot process from being manipulated to launch malicious code, before the operating system is loaded and fully functional. When the computer system is booting and secure boot is activated, the firmware checks every UEFI binary or application that is loaded. It ensures that these binaries have a valid signature, meaning the signature is checked against a local database of trusted certificates, or that the individual checksums of these binaries are stored in an "allowed" list. Furthermore, it checks if certificates or checksum values are not stored in the "deny" list. If these checks are successful, the firmware executes a binary. If not, the firmware refuses to load a binary, therefore halts the boot process and alerts the user.

The local database of trusted certificates or checksums are stored as UEFI secure variables in the firmware (UEFI Forum, 2020b). To change or reset these variables physical access is required, which strengthens the security aspect of secure boot. On the other hand, secure boot can be easily turned off when an attacker has direct access to a device's firmware setup menu, if it is not being protected by any kind of user authorization (Paul and Irvine, 2015).

UEFI Keys Overview. To manage and protect the required signatures for the verification of UEFI applications and images, the UEFI specification describes two distinct keys:

The first one is the Platform Key (PK), which is issued by the Original Equipment Manufacturer (OEM) of the hardware, when the hardware system is being built. The instance that controls the PK can install a new PK and update the Key Exchange Key (KEK).

The KEK is the second key and is used to protect the database from any unauthorized modification. This key is used to sign the database or can be used to sign executable binaries directly. Changes to the database are only possible when both parts of the key (public and private) are available, which means that the KEK establishes the trust between the boot loader and the firmware.

The KEK is issued by the operating system vendor (mainly Microsoft) and it is possible that a system can hold more than one KEK. OEMs use their own KEK for signing their own UEFI drivers and applications.

The database is split up into two parts, the signature database (db) and the forbidden signature database (dbx). The signature database holds a list of allowed certificates or allowed checksums. Signed UEFI binaries with a valid signature that corresponds to a certificate that is stored in the db, are trusted by the firmware.

The forbidden signature database contains hashes of specific UEFI binaries, certificates, or hashes of certificates that where revoked or compromised and therefore should not be trusted. The execution of a binary is refused when the hash is stored in the dbx, or the signature of the binary is stored in the dbx, or the key that was used to sign the binary is stored in the dbx (UEFI Forum, 2020b). An UEFI revocation list can be obtained from the UEFI Forum (UEFI Forum, 2020a). This list contains signatures of previously approved and signed UEFI applications used in booting systems with secure boot enabled, which were revoked due to security incidents.

To change or reset the four UEFI variables (see Table 1), physical access to the computer system is required. The UEFI specification requires that these keys should be changeable in the firmware setup by the end user (UEFI Forum, 2020b).

Table 1: UEFI-Variables needed by UEFI secure boot.

Name	Description
PK	Contains the PK_{pub}
KEK	Contains all KEK keys, signed by the PK_{priv}
db	Contains a list of trusted certificates, signed by the KEK_{priv}
dbx	Contains blacklisted certificates, signed by the KEK_{priv}

Microsoft mandates in their "Windows 8 ready" specification that the firmware of devices is compliant with the UEFI specification, version 2.3.1 Errata C. Furthermore, the secure boot protocol must be enabled when the device is leaving the OEM. A specific Microsoft signature must be stored in the db and a Microsoft KEK key must be stored in the KEK variable. This key is used by Microsoft to update the database on devices from multiple different OEM's.

The hardware specifications and policies for the newest Windows 10 version can be found in the Windows hardware developer documentation (Microsoft Corporation, 2018).

2.3 Full Disk Encryption

Disk encryption encrypts (or decrypts) data written to (or read from) a storage device. While an encryption of just user data is the simplest and least intrusive method—often this is only the user's home directory—it is an incomplete approach. In modern operating systems, background processes cache and store information also in non-encrypted areas, for example in a swap partition or the */var* or */tmp* folders. The solution is to encrypt both user *and* operating sys-

tem data, thereby blocking unauthorized third parties to easily access *any* data at rest. On the other hand, an encryption of all data means that the encrypted parts of the storage device must be unlocked at boot time, to make any access possible.

With full disk encryption (FDE) (Bossi and Visconti, 2015) the whole disk is encrypted. The term is also used for storage devices of UEFI capable systems where technically not all of the data on the storage device is encrypted, i.e. the EFI System Partition. The EFI System Partition stores the boot loader binaries and other UEFI application binaries and therefore must be accessible unencrypted for a system to boot.

Linux. On the Linux platform Linux Unified Key Setup (LUKS) standardizes the key management for encrypting whole partitions (Frühwirth, 2018). It is the proof-of-concept implementation of TKS1, a design for secure key processing (Frühwirth, 2004). The first version of LUKS is the reference implementation, which uses DM-crypt and the device-mapper subsystem of the Linux kernel to implement a transparent disk encryption subsystem, for on-the-fly encryption and decryption (Broz, 2020).

A typical partition layout for a Linux installation is as follows: First, an EFI system partition contains the boot loader files. Secondly, a boot partition which holds the Linux kernel and initramfs. Finally, one or more primary partitions, which store the rest of the operating system and more file systems. With common Linux distributions the installation process only sets up an encrypted primary partition, the boot partition is left unencrypted. For a FDE installation the boot partition needs to be encrypted as well, so that no third party can manipulate the files for booting.

The boot partition may be merged with the primary system partition into one single encrypted partition. This enhances usability and management, since one partition requires only one crypto key. The one drawback of this approach is that GRUB, which is commonly used as the boot loader with Linux, unfortunately only supports LUKS1 format and not the newer LUKS2. As of now no major GRUB version was published with LUKS2 support (Steinhardt, 2020).

3 SCENARIO & ENVIRONMENT

To evaluate the most used Linux distributions for their support of UEFI secure boot in combination with full disk encryption, a representative selection of operating systems alongside with a unified testing environment were set up. We selected the following distributions for our evaluation: Arch Linux, Debian 10.3, Fedora Workstation 31, openSUSE Leap 15.1 and Ubuntu 19.10, in their base variants without any additional modifications. Arch Linux is a "rolling release" distributions and therefore has no specific version number, we used a snapshot image.

Virtualization is the obvious choice to provide a unified testing environment, which brings the additional benefit of reproducible test results, even when conducting the practical tests on different hardware. QEMU (Quick EMUlator)(QEMU, 2020) with the hypervisor KVM (Kernel-based Virtual Machine)(KVM, 2016) was selected as the virtual environment for all tests, they provide excellent support for secure boot with OVMF (Open Virtual Machine Firmware)(Red Hat, 2015), a UEFI compatible firmware.

4 EVALUATION

We now report on our findings with each of the five Linux distributions. Arch Linux is first and described in detail as it requires all steps to be done manually. The following distributions are described more briefly, due to lack of space and as common steps and the boot chain repeats.

4.1 Arch Linux

General. As Arch Linux does not ship with a graphical installer nor a command-line based one, an installation of a test setup is done manually. While Arch does support booting in pure UEFI systems, the installation medium does *not* support secure boot out of the box. Therefore, secure boot must be deactivated in the firmware setup before the installation starts. Our desired configuration requires the creation of an unencrypted EFI partition and two LUKS partitions. Using GRUB as second stage boot loader requires the use of only LUKS version 1 for the boot partition. It contains the Linux kernel and initramfs files. The second encrypted partition is the primary root partition of the installation. After this initial setup the installation of the base system of Arch Linux proceeds according to the installation guide as detailed in the Arch Wiki (Arch Linux, 2020).

FDE. Arch supports booting from encrypted partitions, this requires certain modules to be added to the initramfs as mkinitcpio hooks. On Arch Linux the script mkinitcpio is readily available to (re-)generate the proper initramfs file. To bypass the kernel passphrase prompt for the boot *and* root partition, two distinct key files are necessary. These key files are

placed into the second LUKS key slot for the respective partition, so each partition can be unlocked with either a password (slot 1) or a key (slot 2). Both key files have to be compiled into the initramfs. On boot GRUB asks the user for the password to load the kernel and initramfs from the first encrypted partition. The initramsfs contains the two keys and the boot process continues without any further user input.

Secure Boot. We use the so-called *shim* boot loader as the first stage boot loader that validates and then chain-loads GRUB. Shim can be used on every UEFI compliant computer system, as it is signed by the Microsoft UEFI CA and therefore shim is trusted and executed by the UEFI firmware when secure boot is enabled.

The next step is to create a MOK (Machine Owner Key), which signs the files of GRUB and the kernel. A certificate signed by exactly that key is then stored into the shim controlled MOK list.

The GRUB boot loader also has support for the *shim_lock* module. This module forces GRUB to only load trusted modules and files, if they are not embedded inside the *core.img* of GRUB. Therefore, this requires a GPG-key for signing all files that are inside */boot*. The public part of the GPG-key and the necessary modules for verifying files are embedded into GRUB by the reinstallation process for the GRUB binary. The resulting new GRUB binary is signed with the MOK, and every file in the boot directory is signed with the private part of the GPG-key. This means that *even* the initramfs is validated by GRUB when loading the kernel. Additionally, the kernel image must now be signed with the private MOK key. The last step is to configure a new UEFI boot record that points to the shim binary.

After the installation is finished, we enable secure boot in the firmware setup and start the boot process. Now secure boot first validates the firmware, then follows the boot record to executes the shim binary, shim then validates and executes the GRUB binary, and the Linux kernel will only be loaded when its signature is valid. Therefore the chain of trust spans from the UEFI firmware through the boot process to the Linux kernel. The chain could even be extended further, by enabling the Linux kernel lockdown mode, where only signed kernel modules are permissible to be loaded.

4.2 Fedora

General. Fedora features UEFI boot support since version 11, which was released on June 2nd 2009. The installation process relies on a Live Desktop environment. The graphical installer of Fedora includes the option of an encrypted primary partition, therefore no additional configuration needs to be done regarding this feature. After the installation completes and the machine reboots, the boot partition of Fedora is *not* encrypted while the primary partition, which holds the root filesystem partition and the swap volume, *is* encrypted with LUKS2.

Secure Boot. The installation program creates a boot entry called *Fedora*, which references the shim binary on the EFI system partition. During the boot process shim validates and chain loads the GRUB2 binary *grubx64.efi*, which is signed by the Fedora Secure Boot CA key. After the user selects a kernel entry from the GRUB2 menu, GRUB validates the kernel file and loads both, the kernel and initramfs, files into memory. The initramfs file however is *not* validated by GRUB2. All kernels that are built and shipped by the Fedora Project are already pre-signed with the Fedora Secure Boot CA key. The kernel modules are signed by a temporary key that is generated during the build phase. It is not saved, a new key is used with each kernel build (Boyer et al., 2013). The kernel is started in lockdown mode.

FDE. To also encrypt the first partition hosting */boot*, the initial step is a backup of the contents of the boot partition (e.g. to the home directory of the root user). Then the partition is unmounted, reformatted as LUKS1 encrypted partition, with an Ext4 filesystem with the same UUID that was used by the original unencrypted partition. This avoids changes to */etc/fstab*. The */etc/crypttab* requires a new entry for the new boot partition. Finally, the original */boot* contents can be restored.

The GRUB configuration also needs to be updated accordingly, however a reinstallation of the GRUB2 binary is not necessary since the EFI build of GRUB2 already includes the needed modules, such as *verify*, *cryptodisk* and *luks* support (Doron et al., 2020).

Just like with other distributions, to avoid a second passphrase prompt by the kernel, the kernel requires a key in the initramfs to unlock both the boot and the root partition. Fedora uses the program *dracut* to generate a new initramfs image. The key file to be embedded in the initramfs can be placed via *install_items*. After a rebuild of the initramfs and a reboot, only GRUB2 asks for the boot partition LUKS passphrase. The kernel unlocks the partitions with the key file embedded in the initramfs image.

4.3 OpenSUSE Leap

General. OpenSUSE implemented the support for UEFI in openSUSE 12.2, which was released on September 5th 2012, and an experimental support for

secure boot in openSUSE 12.3, which was released on March 13th 2013. OpenSUSE, just like Fedora, utilizes a graphically guided installation process. Support for full disk encryption is available out-of-the-box: openSUSE does *not* utilize a separate unencrypted boot partition, therefore all steps necessary for encryption of the main partition are performed by the graphical installation process. After completion of the installation process and reboot of the (virtual) machine, first GRUB2 asks for the LUKS passphrase, and then the kernel asks a second time.

Secure Boot. OpenSUSE uses shim, the same way as Fedora. It chain-loads a GRUB2 binary that is signed by the openSUSE secure boot CA key, which is embedded into the trust database of shim. GRUB2 unlocks the primary partition and validates the kernel file, then loads the kernel and initramfs file into memory. The initramfs file is *not* validated. The validation of the kernel is successful since all kernels are signed with the openSUSE secure boot key. The chain of trust ends here, since the kernel does *not* enforce any further restrictions, like the kernel lockdown mode. The kernel loads any kernel module, without validation.

FDE. The use of only one encrypted partition has one major downside, the partition can only use LUKS1 format, as GRUB2 does not (yet) support unlocking of LUKS2 partitions. Therefore openSUSE cannot take advantage of the newer LUKS2 format.

As openSUSE already stores */boot* on the encrypted main partition, all that needs to be done is to create and embed a key file into the initramfs to omit a second passphrase prompt at boot time. This key file creation works the same way as on Fedora, as openSUSE also relys on dracut to build the initramfs.

4.4 Ubuntu

General. Ubuntu supports booting on UEFI based systems since version 11.10, released on October 13th 2011. Ubuntu utilizes a graphical installation program, just like Fedora, which includes the option of an encrypted primary partition, therefore no additional configuration needs to be done. After the installation finishes, the (virtual) machine reboots and the kernel prompts for the passphrase for the encrypted primary partition. The disk is partitioned the same way as on Fedora: an EFI system partition, an unencrypted boot partition, and an encrypted primary partition, using LUKS2.

Secure Boot. The UEFI firmware first loads shim, which is signed by Microsoft's UEFI CA. Shim validates and chain-loads the GRUB2 binary *grubx64.efi* that is stored on the same partition. GRUB2 is signed by the Ubuntu UEFI key that is embedded into the shim trust database. GRUB2 validates the kernel and loads the necessary files from the boot partition. The initramfs file is *not* validated. All kernels and kernel modules that are built and shipped by Canonical are signed with the Canonical UEFI key. The kernel is started in lockdown mode.

FDE. To encrypt */boot*, the same principle applies as with Fedora's boot partition. First backup the original content, create an encrypted LUKS1 partition, then restore the files and reconfigure GRUB2 to unlock the partition during startup. A key file embedded into the initramfs avoids a second passphrase prompt from the kernel during boot.

4.5 Debian

General. Debian supports booting from an UEFI based firmware since Debian Wheezy (7.0), which was released on May 4th 2013. With the included graphical installer one can configure an encrypted primary partition, just like on Fedora and Ubuntu. After the installation of the base system the (virtual) machine reboots. On boot the kernel asks for the LUKS passphrase for the encrypted primary partition. The disk is split up the same way as on Fedora and Ubuntu: An EFI system partition, an unencrypted boot partition, and an encrypted primary partition, using LUKS2.

Secure Boot. The Debian installation process creates a boot entry called *debian*, which points to *shim64.efi*. Shim validates and chain loads the GRUB2 binary *grubx64.efi*, which is signed with the Debian UEFI key. Both shim and GRUB2 are stored on the EFI system partition. GRUB2 validates the kernel, the initramfs file is *not* validated. The kernel and all its modules are signed by the Debian UEFI key. The lockdown mode of the kernel is active if secure boot is active.

FDE. Since Ubuntu is built on the Debian architecture, the process of setting up an encrypted boot partition is approximately the same as described in Section 4.4.

5 DISCUSSION

Encryption of all partitions on a computer system prevents third parties from tampering around with data stored on it. The secure boot protocol from the UEFI specification detects unauthorized changes to the boot chain, by validating every step that is involved in the process.

Table 2: Summary of the state of all features on the tested distributions. (✓) means that the feature works out-of-the-box, (✗) means that the feature does *not* work out-of-the-box.

Distribution	UEFI boot	Secure Boot	Full Disk Encryption	Chain of Trust to Kernel
Arch Linux	✓	✗	✓	✓
Debian	✓	✓	✗	✓
Fedora	✓	✓	✗	✓
openSUSE	✓	✓	✓	✗
Ubuntu	✓	✓	✗	✓

We have analyzed and tested five major Linux distributions: Arch Linux, Debian 10, Ubuntu 19.10, Fedora Workstation 31 and openSUSE Leap 15.1. For a compact overview of our results see Table 2. We are happy to report all distributions support booting from an UEFI firmware.

The implementation of secure boot is generally realized with shim, the secure boot compatible first stage boot loader from the Fedora Project. Shim chain-loads a second stage boot loader in the form of GRUB2, who is signed by the individual distributions certificate authority. Booting further, the kernel image is validated by the second stage boot loader.

Ubuntu, Debian and Fedora offer the so-called lockdown mode, a way for the chain of trust to reach up to the kernel space by limiting the access to modify the kernel itself. On openSUSE the lockdown mode is not enforced. The chain of trust ends with GRUB2 validating the kernel image.

On Arch Linux the user has to install the whole system manually. This makes an installation with secure boot and full disk encryption the most time-consuming. On the other hand, the implementation of Arch Linux also validates the initramfs, in contrast to the other tested distributions.

Per default none of the tested distributions offer the functionality of using a key file to unlock the encrypted root partition, though every tool/program that is necessary for this feature is already installed in the base installation of Ubuntu, Debian, Fedora and openSUSE. Only openSUSE supports an encrypted */boot* folder per default, with the disadvantage that only LUKS1 can be used on the encrypted root partition.

6 CONCLUSION

Unfortunately, as can be seen in Table 2 none of the evaluated Linux distributions provides support for all of the possible security features out-of-the-box. We assume that the majority of the user-base most likely does not care, and will not invest the effort to implement missing ones when using one of these distributions. It remains to be seen if either all distributions will implement all the features one day by themselves, or if users will show more interest and demand that they are implemented. As the Linux world is continuously improving, maybe at the time of reading these lines this has already happened?

A potential future work would be the analysis of TPM (Trusted Platform Module) integration into the boot chain. The TPM can be used to host the key to the encrypted primary partition and the TPM would release the key only if the components in the chain of the boot process have not been tampered with. The challenges of TPM-based keys and integration of them into the boot process of major Linux distributions will be revisited in a future paper.

ACKNOWLEDGEMENTS

The work presented in this paper was done at the Josef Ressel Center for Blockchain Technologies and Security Management (BLOCKCHAINS), St. Pölten University of Applied Sciences, Austria.

The financial support by the Christian Doppler Research Association, the Austrian Federal Ministry for Digital, and Economic Affairs and the National Foundation for Research, Technology and Development is gratefully acknowledged.

REFERENCES

Arch Linux (2020). Arch linux wiki - installation guide. https://wiki.archlinux.org/index.php/Installation_guide.

Bossi, S. and Visconti, A. (2015). What users should know about full disk encryption based on LUKS. In *Cryptology and Network Security*, pages 225–237. Springer.

Boyer, J., Fenzi, K., Jones, P., Bressers, J., and Weimer, F. (2013). Fedora 18 - uefi secure boot guide. https://docs.fedoraproject.org/en-US/Fedora/18/pdf/UEFI_Secure_Boot_Guide/Fedora-18-UEFI_Secure_Boot_Guide-en-US.pdf.

Broz, M. (2020). dm-crypt: Linux kernel device-mapper crypto target. https://gitlab.com/cryptsetup/cryptsetup/-/wikis/DMCrypt.

Doron, B., Jones, P., and Canillas, J. M. (2020). Include several modules in the efi build of grub2

for security use-cases. https://fedoraproject.org/wiki/Changes/Include_security_modules_in_efi_Grub2.

Frühwirth, C. (2004). Tks1-an anti-forensic, two level, and iterated key setup scheme.

Frühwirth, C. (2018). Luks1 on-disk format specification version 1.2.3.

KVM (2016). Kernel virtual machine. https://www.linux-kvm.org/page/Main_Page.

Microsoft Corporation (2018). Windows hardware compatibility program specifications and policies. https://docs.microsoft.com/en-us/windows-hardware/design/compatibility/whcp-specifications-policies.

Paul, G. and Irvine, J. (2015). Take control of your pc with uefi secure boot. *Linux Journal*, (257):58–72.

QEMU (2020). Qemu - a generic and open source machine emulator and virtualizer. https://www.qemu.org/.

Red Hat, I. (2015). Open virtual machine firmware (ovmf) status report. http://www.linux-kvm.org/downloads/lersek/ovmf-whitepaper-c770f8c.txt.

Richardson, B. (2017). Last mile barriers to removing legacy bios. https://uefi.org/learning_center/presentationsandvideos/.

Steinhardt, P. (2020). GRUB: disk: Implement support for LUKS2. https://git.savannah.gnu.org/cgit/grub.git/commit/?id=365e0cc3e7e44151c14dd29514c2f870b49f9755.

TianoCore (2017). Tianocore. https://https://www.tianocore.org/.

UEFI Forum (2019). Uefi faqs. https://uefi.org/faq.

UEFI Forum (2020a). Uefi revocation list file. https://uefi.org/revocationlistfile.

UEFI Forum (2020b). Unified extensible firmware interface specification version 2.8 (errata a). https://uefi.org/specifications.