Guidance for Managing UEFI Secure Boot

Executive summary

Recent vulnerabilities involving Secure Boot (e.g., PKFail [1], BlackLotus [2], BootHole [3], and similar unnamed [4]) have demonstrated the need to scrutinize the configuration of Secure Boot on enterprise devices. This document details instructions for system owners to query Secure Boot configuration, compare observed results to industry norms, and recognize and recover from misconfigurations. Organizations that neglect Secure Boot configuration may be at a greater risk of exposure to bootkits and other persistence techniques.

Checking the configuration of Secure Boot is also an important component of Supply Chain Risk Management (SCRM). Secure Boot is responsible for enforcing security policy at boot time based on a set of certificates and hashes placed within its data stores by system and operating system (OS) vendors. As the industry transitions away from 2011 signing certificates—that are nearing expiration—to new 2023 equivalents, there is even more need for organizations to scrutinize their Secure Boot configurations to ensure they are accurate and secure.

Introduction

Secure Boot was introduced to the Unified Extensible Firmware Interface (UEFI) industry standard in the mid-2000s. Secure Boot officially became part of the published specification with UEFI 2.0’s release in 2006. [5] The core concept behind Secure Boot is to add a security policy enforcement mechanism into the boot process of computing devices.

Prior to the inclusion of Secure Boot, any boot binaries produced in accordance with Basic Input/Output System (BIOS) or Extensible Firmware Interface (EFI) executable structures could be used at boot time. This openness allowed for system vendors, device vendors, firmware developers, computer enthusiasts, and more to contribute value-adding boot software. However, this openness also created the opportunity for malware developers to create bootkits (boot-time, root-privilege, persistent malware) designed to execute at the earliest stages of device boot—fully outside the purview of anti-malware solutions focused on the user software environment. Secure Boot is one of several solutions capable of limiting the software allowed to execute during the device boot process.

Secure Boot is ubiquitous in the modern computing market thanks to several leading software and hardware vendors collaborating to require the technology. Most mainstream operating system (OS) distributions feature signatures that are trusted by the preloaded Secure Boot settings on devices, so Secure Boot will allow the OS to launch and execute. Many of these OS distributions and hardware platforms also are capable of automatically updating the device’s Secure Boot values over time to react to changes in the boot security environment.

The default Secure Boot settings that come with most devices will prevent unsigned and unknown boot software from executing, while being fairly open to allow many mainstream OS distributions by default. NSA published UEFI Secure Boot Customization guidance and instructions for organizations to further limit which OS distributions and other boot software can run. [6] Customizing Secure Boot can be a challenge to configure and maintain but can block vulnerable or unwanted boot software.

Secure Boot functions as a boot-time option for Supply Chain Risk Management (SCRM). The default Microsoft ecosystem shipped with most devices requires binaries to pass code quality, safety, and compatibility tests. Binaries that are determined trustworthy receive signatures, while those that fail the tests do not. Any binary previously signed that is later found to have a vulnerability or embedded malware is revoked. Device owners may rely upon the decisions made by Microsoft regarding boot binary trustworthiness or they may customize Secure Boot to limit or expand the scope of what is considered trustworthy at boot time.

The modern Secure Boot ecosystem is mature, widely supported, normally enabled by default, and routinely scrutinized by cybersecurity experts. Secure Boot can function well with little oversight or attention from an organization most of the time, continuously protecting the organization's device from bootkits and other boot malware. However, Secure Boot, as a part of the larger endpoint security ecosystem, is a complex technology that can present challenges and require configuration at times. The objective of this document is to address some of these issues, including:

- Confusion relating to how Secure Boot interacts with other protective technologies,
- Lack of acceptance testing on newly procured devices, and
- Few publications covering how to detect and respond to Secure Boot issues.

The modern Secure Boot ecosystem is:

- Mature,
- Widely supported,
- Normally enabled by default, and
- Routinely scrutinized by cybersecurity experts.

Secure Boot process

Secure Boot uses certificates, hashes, and signatures to determine what binaries are permitted to execute at boot time. Vendors operate Certificate Authorities (CAs) that sign certificates for keys that vendors use to sign trusted boot binaries. Each CA has an associated public certificate that may be loaded into Secure Boot and used to validate the signatures. When a binary is not able to be signed by a CA, vendors (or system owners) may choose to produce and store a hash of the binary as an alternative.

Secure Boot variables

UEFI stores Secure Boot information in the following four variables:

1. **Exclusion/deny list database (DBX):** A DBX is a collection of certificates and hashes associated with denied, revoked, or otherwise untrusted boot binaries. Any

boot binary with a signature validated by a DBX certificate or any boot binary with a measurement hash that matches a hash within the DBX is considered untrusted.

2. **Allow list database (DB):** A DB is a collection of certificates and hashes used to evaluate binaries for trustworthiness at boot time. Any binary that has not matched a DBX record is evaluated against the DB certificate list and hash list. Matches are treated as trusted. Unknowns/non-matches are treated as untrusted.

3. **Key Exchange Key (KEK):** A KEK is one or more CA certificates that are trusted to validate change commands covering the DB and DBX values. KEKs are not used to determine trustworthiness of boot binaries. Commands to change DB and DBX records will be trusted if signed by a KEK whose certificate is placed in Secure Boot's KEK data store. DB and DBX changes not signed by a trusted KEK will be rejected.

4. **Platform Key (PK):** A PK is a certificate associated with the CA authorized to sign changes to the KEK values. Only one PK exists per system. The PK can also sign changes to the DB, DBX, and UEFI environment variables. KEK changes signed by other keys will be rejected. The PK is not used to determine trustworthiness of boot binaries.

A UEFI system boots in a series of phases:

1. The first is the **Security** phase, routinely abbreviated as **SEC**. It initializes hardware components and basic resources.
2. The second phase is the **Pre-Extensible Firmware Interface (EFI) Initialization** phase and is commonly abbreviated to **PEI**. UEFI Secure Boot enforcement mechanisms are initialized, and the DBX and DB begin to be used to audit boot binaries.
3. The third phase is the **Driver eXecution Environment**, known as **DXE**, when storage controllers, storage media, graphics components, networking, and other mid-level resources are initialized.
4. The fourth phase involves **Boot Device Select (BDS)** when the boot manager is initialized, bootable EFI partitions are identified, and one of the bootable partitions is selected.
5. The fifth phase runs the **bootloader** software from the bootable partition.
6. The sixth phase involves launching the operating system **kernel**.

UEFI Boot Process Phases

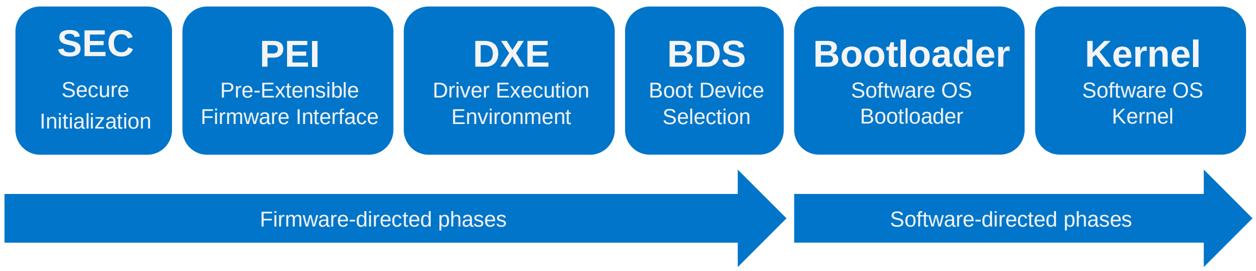

Figure 1: Systems utilize UEFI boot in several phases. UEFI Secure Boot begins auditing boot software during the PEI phase. The bootloader phase features a transition from firmware-controlled boot (tightly coupled to the device) to software-controlled boot (any UEFI compatible OS).

Secure Boot auditing and enforcement

The degree to which Secure Boot audits boot binaries starting with the PEI phase varies. In fast boot and minimal auditing scenarios, the system prioritizes boot speed over security. Frequently utilized (e.g., storage controller or integrated graphics module) or unchanged binaries may not be evaluated against Secure Boot. In a slower boot mode or full auditing mode, more boot binaries are tested against the Secure Boot DB and DBX.

A match against the DBX prevents execution of an untrusted boot binary, a match to the DB permits execution, and no match to the DBX or DB also prevents execution. Blocked binaries will not execute during or following boot, and any resources, interfaces, or structures set up by the blocked binary will not be created. The blocking does not necessarily stop the system from booting if the blocked binary is not essential to boot (e.g., a RAID controller blocked by the DBX does not prohibit boot to a different storage controller or external media).

Firmware in the PEI, DXE, and BDS phases carries out Secure Boot enforcement. Each executable is checked prior to execution. Secure Boot enforcement transitions to software control during the bootloader phase. Firmware checks the bootloader—which may also take the form of a boot shim or consist of multiple sub-phases—against the DBX and DB. The bootloader is then responsible for continuing the auditing forward. Bootloaders usually check kernels or secondary bootloaders against Secure Boot before transferring execution to them.

The most popular boot shim for Linux extends Secure Boot to add software-controlled Machine Owner Key (MOK) and MOK Exclusion (MOKX) databases similar to the DB and DBX. These are databases where Linux distribution vendors may place their own certificates without needing to engage with the firmware certificate system established by Microsoft and system vendors. Linux kernels and kernel modules are signed for use with MOK instead of the Microsoft DB values.

Some kernels continue enforcing Secure Boot in the operating system environment. Device drivers, services, and kernel modules may require Secure Boot signatures or hashes depending on system configuration—a scenario most common on systems with kernel mode code integrity (Microsoft) or MOK (Linux).

The goal of Secure Boot is to restrict the executables at boot time to those trusted by the system owner. Most system owners find the trustworthiness decisions made within the Microsoft UEFI ecosystem sufficient. As an alternative, Secure Boot customization allows system owners to take control of trustworthiness decisions rather than relying upon the industry standard.

Secure Boot is an active, local device enforcement mechanism. Secure Boot has no mechanism to attest its state to an external party beyond the configuration query commands described later in this publication. The PK, KEK, DB, and DBX values are recorded within the Trusted Platform Module's (TPM) Platform Configuration Register (PCR) 7. The TPM can be used to verify the configuration of Secure Boot via use of a Full Disk Encryption (FDE) solution that utilizes the value of PCR 7 or an attestation solution that includes PCR 7 in a TPM integrity quote. However, FDE and TPM attestation can also operate without relying upon or processing data describing Secure Boot's state. This may be a point of confusion where some system administrators may mistakenly think that if FDE is enabled or TPM attestation is used, then Secure Boot is automatically active as well, but this is not the case. System administrators should ensure that Secure Boot is enabled separately.

Recent Secure Boot vulnerabilities

PKFail

Devices are intended to ship with system vendor and Microsoft Secure Boot values preinstalled. However, sometimes system vendors ship devices with test certificates preinstalled from a mainboard or firmware manufacturer. The past instances when test certificates and keys were mistakenly disclosed publicly, combined with weak authentication values, have allowed malicious actors to bypass the enforcement capabilities of Secure Boot when test certificates are preinstalled on a device. PKFail involved the improper shipping of devices with preinstalled test certificates to device owners. [1]

BlackLotus

A Windows bootloader flaw (CVE-2022-21894, CVE-2023-24932) enabled malicious actors to prevent the enforcement of Secure Boot at the bootloader and kernel stages of UEFI boot. Affected machines would indicate that Secure Boot was both active and enforcing while having lost access to the DB and DBX values starting at the bootloader phase. The BlackLotus bootkit would then inject its own Secure Boot values to poison subsequent boots and swap in its own chain of boot executables as the trusted ones instead of the legitimate ones. BlackLotus proved difficult to mitigate immediately following its release since the number of bootloader hashes that needed to be added to the DBX exceeded available DBX memory on many devices. [2] Patches and new enforcement mechanisms in the bootloader eventually mitigated this threat. [3]

BootHole

A flaw in the GRand Unified Bootloader (GRUB) enabled malformed configuration files to grant arbitrary execution at boot time. The vulnerability gave malicious actors a mechanism to bypass Secure Boot and the MOK extensions set by Shim. Shim is a pre-boot shim designed to bridge the Microsoft Secure Boot ecosystem with Linux distribution Secure Boot ecosystems. Several generations of GRUB were affected, requiring an expansion of DBX records. [7] Some older devices did not have enough DBX memory to accommodate all of the revocations. Updated signed versions of GRUB were released to correct the problem. [8]

How to check Secure Boot

Device owners and administrators should not assume that the presence of a Trusted Platform Module (TPM), utilization of an FDE solution like BitLocker, or enablement of processor security technologies guarantees the use of Secure Boot. These technologies are capable of applying complementary security features that are not necessarily dependent on each other to function. Acceptance testing of new devices should be performed and validate that Secure Boot is both configured properly and enforcing. The following guidance is intended to aid administrators in organizations that are not already employing enterprise supply chain scanning solutions that alert on misconfigured firmware.

1. *Is Secure Boot configured and enforcing?*

First, check to see if Secure Boot is enabled and enforcing. Some devices—particularly custom-built devices—may ship with Secure Boot disabled for a variety of reasons. Use one of the following commands:

Windows PowerShell – Open an administrative PowerShell terminal and type the following:

```
Confirm-SecureBootUEFI
```

A result of True indicates that Secure Boot is enabled. A result of False indicates that Secure Boot is available but may be disabled or in permissive mode. Devices that do not implement Secure Boot may see a “not supported” message.

Linux terminal – Open a terminal and type the following:

```
sudo mokutil -sb-state
```

The utility should indicate if Secure Boot is enabled, disabled, or unsupported. Additional information about the Secure Boot mode may also be provided. The mode should be listed as standard, deployment, or user mode. Secure Boot is not providing enforcement when setup, audit, or permissive modes are indicated.

2. *Are Secure Boot certificates and hashes installed?*

Second, check the values stored within Secure Boot. Pay particular attention to any identifying information relating to the PK and KEKs. Data should be stored within the PK, KEK, and DB. An absence of records in the DBX does not necessarily indicate a problem.

Windows PowerShell – Open an administrative PowerShell terminal and type the following:

```
Get-SecureBootUEFI -Name PK -OutputFilePath PK.esl
Get-SecureBootUEFI -Name KEK -OutputFilePath KEK.esl
```

```
Get-SecureBootUEFI -Name DB -OutputFilePath DB.esl
Get-SecureBootUEFI -Name DBX -OutputFilePath DBX.esl
```

Linux terminal – Open a terminal and type the following:

```
sudo efi-readvar -v PK -o PK.old.esl
sudo efi-readvar -v KEK -o KEK.old.esl
sudo efi-readvar -v db -o db.old.esl
sudo efi-readvar -v dbx -o dbx.old.esl
```

The efitools open source project contains a utility sig-list-to-certs that can extract certificates from each of the Secure Boot data stores. However, the utility does not list out hashes. NSA has developed utilities to separate ESL files into certificates and hashes. The utilities are available in PowerShell, Python, and C and can be downloaded from the NSA GitHub Hardware and Firmware Security Guidance repository secureboot folder.

3. Is the Secure Boot configuration as expected?

An expected Secure Boot configuration has the following characteristics:

- **A PK set by the system vendor.** PKs do not need to be unique per device and may feature a PK common to all devices produced by a system vendor or common only within the same model.
- **A KEK certificate from the system vendor, a Microsoft 2011 KEK certificate, and a Microsoft 2023 KEK certificate. [9] T**he Microsoft KEK is designed to help manage the Microsoft Secure Boot ecosystem for both the Windows OS and third-party vendors. The system vendor KEK supports situations where firmware code has been changed and Secure Boot must be adjusted to recognize the changes.
- **A DB certificate for the Microsoft Windows Production CA (PCA) 2011, a DB certificate for the Microsoft UEFI CA 2011, a DB certificate for the Microsoft Windows UEFI CA 2023, a Microsoft UEFI CA 2023 certificate, and a Microsoft Option ROM UEFI CA 2023 certificate. [9]** There may also be a system vendor DB certificate to support certain kinds of updates (e.g., firmware update). Some software vendors may also place DB certificates—common for virtualization-providers and anti-malware suites. There may be some DB hashes of trusted system components.
- **A list of DBX hashes. [9]** A copy of the Microsoft Windows PCA certificate from 2010 may also appear in the DBX.

Note that some devices may use only 2023 certificates while other devices use both the 2011 and 2023 certificates. System vendor and mainboard vendor support timelines dictate whether or not boot firmware signed with 2011 keys will be upgraded to 2023 signatures. Older devices are more likely to need to retain the Microsoft UEFI CA 2011 certificate. The transition from 2011 certificates to 2023 certificates is in progress as of this guidance's publication date.

Some particularly sensitive devices may each carry a unique PK. Utilizing unique PKs has the impact that KEK changes must be signed for each individual device rather than groups of the same model. Unique PKs enhance security with the tradeoff of more complexity and time to make changes. Some systems utilizing Linux may opt to remove the Microsoft Windows PCA certificate. Removal of the Microsoft UEFI CA certificate is unusual since it is utilized to sign the Linux pre-bootloader Shim.

Normal Configuration for UEFI Secure Boot

PK Platform Key	**KEK** Key Exchange Key	**DB** Allow Database	**DBX** Exclusion Database
System Vendor PK Certificate	System Vendor KEK Certificate Microsoft KEK Certificate 2023	Microsoft Windows PCA 2011 Certificate Microsoft Windows PCA 2023 Certificate Microsoft UEFI CA 2011 Certificate Microsoft UEFI CA 2023 Certificate System Vendor DB Certificate Anti-malware Suite Vendor DB Certificate	Microsoft Windows PCA 2010 Certificate Several Hashes

Figure 2: A normal distribution of certificates and hashes on a user device. All certificates are from an identified vendor. Some systems may feature hashes in the DB, may lack a system vendor DB certificate, and may lack an anti-malware suite vendor DB certificate. Some systems may have a DB certificate from the hypervisor vendor or other hardware or software vendors

4. Is the Secure Boot configuration not set properly?

Some common indicators of an improper Secure Boot configuration include:

- Indications that Secure Boot is disabled, such as from using the commands noted above.

- The absence of values stored within the PK, KEK, or DB. No values stored within the DBX does not necessarily indicate a problem.
- The use of test, evaluation, demonstration, or similar non-production certificates. [1]
- The use of known compromised certificates such as those listed for the DBX in the Microsoft Secure Boot Objects GitHub repository. [9]
- Hashes placed in the wrong Secure Boot variable stores (e.g., DBX hashes placed in the DB). [9]
- Certificates placed in the wrong value store (e.g., KEK certificate placed in the DB).
- Indications that Secure Boot is in setup, audit, or permissive mode.
- Enablement of Compatibility Support Module (CSM) in the system firmware configuration (see BIOS setup or UEFI boot configuration).

Incorrect Configuration for UEFI Secure Boot

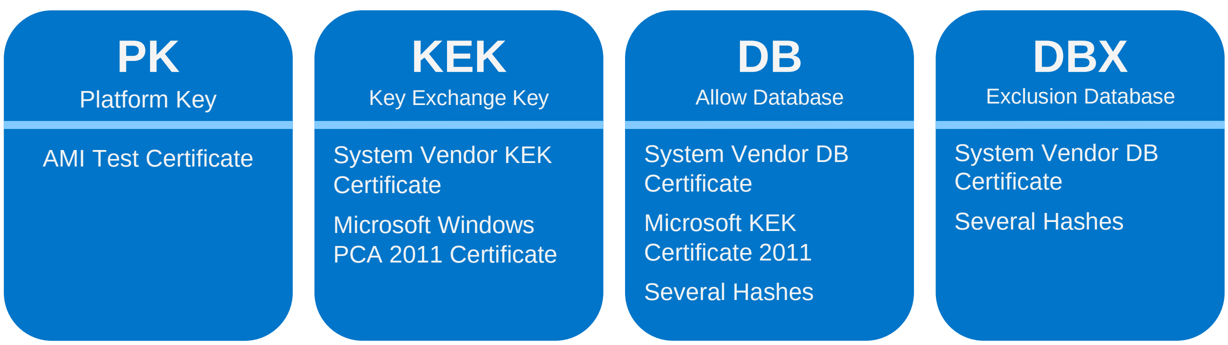

Figure 3: An incorrect configuration of UEFI Secure Boot may feature no values in any value store or something resembling the above. Multiple mistakes have been combined in this example (with both proper and improper settings). Note the use of an unsafe and compromised "test certificate" for the PK, the incorrect placement of a KEK certificate in the DB, the incorrect placement of a DB certificate in the KEK, and copies of the same certificate placed both in the DB and the DBX (the DBX placement would override the DB placement).

5. Recovery from an improper Secure Boot configuration

Most Secure Boot configuration issues can be resolved by restoring the standard or factory Secure Boot certificates from within the UEFI boot configuration interface. Set Secure Boot to enforcing/on/standard mode following the initialization. A hotkey may need to be pressed during the device's boot process to access the UEFI boot configuration interface.

Some Secure Boot configuration errors can be resolved via updates. A firmware update from the system vendor is the first place to check. The second option is to use the OS update mechanism. OS updates can deliver UEFI update capsules that can update firmware or Secure Boot values.

If Secure Boot initialization or firmware updates do not resolve a bad configuration, contact the system vendor for further guidance. System vendor and Microsoft certificates may be installed manually. [9] However, manually setting up Secure Boot may require hands-on visits to individual workstations—something resource-intensive in large infrastructures.

Conclusion

UEFI Secure Boot is a boot security technology introduced in 2006 designed to limit the execution of binaries at boot time to those allowed by policy. Secure Boot utilizes four certificate and hash data stores to identify which authorities are trusted to change and define Secure Boot policy. Most device owners find the Microsoft ecosystem preloaded on most devices to be a sufficient balance of compatibility and security. Secure Boot can be customized to further limit boot execution or extended to support Linux operating systems. Secure Boot helps mitigate threats that may exist in the hardware, firmware, and software supply chains of computing devices.

Modern user devices boot so quickly that the actions of Secure Boot may seem invisible. This does not diminish the vital role Secure Boot plays in constraining boot binaries to those which are necessary for the device to boot and deemed trustworthy by the device owner. Incorrect deployment of Secure Boot exposes device owners to bootkits resident in firmware or booting of unauthorized boot file systems outside the control of the software OS. Such threats can silently compromise the very foundations many anti-malware solutions are built upon.

References and resources

[1] Binarly. 2024. PKfail: Untrusted Platform Keys Undermine Secure Boot on UEFI Ecosystem. https://www.binarly.io/blog/pkfail-untrusted-platform-keys-undermine-secure-boot-on-uefi-ecosystem

[2] NSA. 2023. BlackLotus Mitigation Guide. https://media.defense.gov/2023/Jun/22/2003245723/-1/-1/0/CSI_BlackLotus_Mitigation_Guide.PDF

[3] Microsoft. 2023. Secure Boot Security Feature Bypass Vulnerability (BlackLotus). https://msrc.microsoft.com/update-guide/vulnerability/CVE-2023-24932

[4] Eclypsium. 2024. Microsoft Issues Patches for 24 New Secure Boot Vulnerabilities. https://eclypsium.com/blog/microsoft-issues-patches-for-24-new-secure-boot-vulnerabilities/

[5] UEFI Forum. 2024. Specifications: UEFI Specification. https://uefi.org/specifications

[6] NSA. 2020. UEFI Secure Boot Customization. https://media.defense.gov/2020/Sep/15/20022497594/-1/-1/0/CTR-UEFI-Secure-Boot-Customization-UOO168873-20.PDF

[7] NSA. 2020. GRUB Boot Vulnerability Mitigations. https://www.nsa.gov/portals/75/documents/resources/cybersecurity-professionals/CSA_Mitigate_the_GRUB2_BootHole_Vulnerability_20200730_nsa_gov%20Copy.pdf

[8] Eclypsium. 2020. There's a Hole in the Boot (BootHole). https://eclypsium.com/blog/theres-a-hole-in-the-boot/

[9] Microsoft. 2023. Secure Boot Objects. https://github.com/microsoft/secureboot_objects/

Disclaimer of endorsement

Purpose

This document was developed in furtherance of NSA's cybersecurity missions, including its responsibilities to identify and disseminate threats to National Security Systems, Department of Defense, and Defense Industrial Base information systems, and to develop and issue cybersecurity specifications and mitigations. This information may be shared broadly to reach all appropriate stakeholders.

Contact

Cybersecurity Report Feedback: CybersecurityReports@nsa.gov
Defense Industrial Base Inquiries and Cybersecurity Services: DIB_Defense@cyber.nsa.gov
Media Inquiries / Press Desk: NSA Media Relations: 443-634-0721, MediaRelations@nsa.gov