Base Boot Security Requirements 1.2

Document number:	ARM DEN0107
Release Quality:	REL
Confidentiality:	Non-Confidential
Date of Issue:	January 9, 2023

Contents

About this document

Release information

The change history table lists the changes that have been made to this document.

Date	Issue	Confidentiality	Change
Oct 6, 2020	1.0	Non-confidential	Initial release
June 10, 2021	1.1	Non-confidential	Updates • Updates to sync with BBR now that it has the base firmware update requirements • Fix error in requirement numbering • Moved the db/dbx variable attributes requirements to the Secure Boot section • For measured boot, made creating the TPM event log an explicit requirement. • Removed reference to the deprecated EFI_VARIABLE_AUTHENTICATED_WRITE_ACCESS attribute • Minor cleanup/clarifications
January 9, 2023	1.2	Non-confidential	Updates • Clean up • Add informative security checklist chapter

Base Boot Security Requirements

Arm Non-Confidential Document Licence ("Licence")

Arm Limited. Company 02557590 registered in England.
110 Fulbourn Road, Cambridge, England CB1 9NJ.

Arm document reference: LES-PRE-21585 version 4.0

References

This document refers to the following documents.

Ref	Document Number	Title
[1]	DEN0044F	Arm® Base Boot Requirements
[2]	UEFI Specification 2.8	Unified Extensible Firmware Interface Specification. Version 2.8
[3]	DEN 0094A	Arm® Base System Architecture Version 0.8
[4]		TCG PC Client Platform Firmware Profile Specification, Family “2.0”, Level 00 Revision 1.04, June 3, 2019
[5]		TCG EFI Protocol Specification, Family “2.0”, Version 1.0, Revision 00.13, March 30, 2016
[6]		TCG ACPI Specification, Family “1.2” and “2.0”, Version 1.2, August 18, 2017
[7]		TCG PC Client Platform Physical Presence Interface Specification, Family “1.2” and “2.0”, Version 1.30, July 28, 2015
[8]		TCG PC Client Platform Reset Attack Mitigation Specification, Family “2.0”, Version 1.10, January 21, 2019
[9]	PSA	Platform Security Architecture https://developer.arm.com/architectures/security-architectures/platform-security-architecture
[10]	PSA Certified	https://www.psacertified.org/
[11]	NIST SP 800-132	Recommendation for Password-Based Key Derivation Part 1: Storage Applications, December 2010.
[12]	NIST SP 800-57 Part 1 Rev. 5	Recommendation for Key Management, NIST Special Publication 800-57 Part1 Rev 5, May 2020.

Terms and abbreviations

This document uses the following terms and abbreviations.

Term	Meaning
Critical data	Critical data includes configuration settings and policies that need to be in a valid state for a device to maintain its security posture during boot and runtime. All other data is non-critical.

Mutable	Changeable with respect to the platform of a system.
Non-host platform	A peripheral or controller in a system that has no DMA protections and cannot be restricted from reading or writing Normal world memory.
Non-secure	Something belonging to the Non-secure privilege levels (Non-secure EL0, EL1, and EL2).
Normal world	The Non-secure privilege levels (Non-secure EL0, EL1, and EL2) and resources, for example memory, registers, and devices, that are not part of the Secure world.
Platform	The set of hardware and firmware mechanisms and services that an operating system and applications can rely on.
Secure world	The environment that is provided by the Secure privilege levels in the Arm v8-A architecture, S-EL0, S-EL1, S-EL2, EL3, and the resources, for example memory, registers, and devices, that are accessible exclusively from the Secure privilege levels.
TCG	Trusted Computing Group
TPM	Trusted Platform Module. A security module that is defined by TCG.

Conventions

Typographical conventions

The typographical conventions are:

italic

Introduces special terminology, and denotes citations.

Bold

Denotes signal names, and is used for terms in descriptive lists, where appropriate.

`Monospace`

Used for assembler syntax descriptions, pseudocode, and source code examples.

Also used in the main text for instruction mnemonics and for references to other items appearing in assembler syntax descriptions, pseudocode, and source code examples.

SMALL CAPITALS

Used for some common terms such as IMPLEMENTATION DEFINED.

Also used for a few terms that have specific technical meanings, and are included in the Glossary.

Red text

Indicates an open issue.

Blue text

Indicates a link, which can be:

- A cross-reference to another location within the document.
- A URL, for example http://infocenter.arm.com.

Numbers

Numbers are normally written in decimal. Binary numbers are preceded by `0b`, and hexadecimal numbers by `0x`.

In both cases, the prefix and the associated value are written in a monospace font, for example `0xFFFF0000`. To improve readability, long numbers can be written with an underscore separator between every four characters, for example `0xFFFF_0000_0000_0000`. Ignore any underscores when interpreting the value of a number.

Feedback

Arm welcomes feedback on its documentation.

Feedback on this book

If you have comments on the content of this book, send an email to <<feedback-address>>. Give:

- The title (<<title>>).
- The number and issue (<<Arm Document Unique ID>> <<version>> <<quality>> <<issue>>).
- The page numbers to which your comments apply.
- The rule identifiers to which your comments apply, if applicable.
- A concise explanation of your comments.

Arm also welcomes general suggestions for additions and improvements.

1 Introduction

This document specifies security interface requirements and guidance for systems that are compliant with the Arm® Base Boot Requirements (BBR) [1] specification. BBR specifies the requirements for boot and runtime services that system software, like operating systems and hypervisors, can rely on. Meeting these requirements enables a suitably built operating system image to run on all compliant systems. BBR is based on industry firmware standards like UEFI and ACPI. The focus of BBR is standards-based boot and runtime services, and it does not address security.

This document identifies the platform requirements for BBR-based systems that enable standard, suitably built operating systems to seamlessly use standard security interfaces. These interfaces include the following security related functionality:

- UEFI authenticated variables
- UEFI secure boot
- UEFI secure firmware update using Update Capsules
- TPMs and measured boot

Compliance with this specification provides assurance that the security features in scope are implemented according to standards. However, compliance does not provide assurance that a platform is secure. In the process of architecting a system, system-level threat modeling should be performed to evaluate threats, risks, and mitigations. The Platform Security Architecture (PSA) [9] and the PSA Certified framework [10] do provide a comprehensive approach to platform security that is based on defined set of security goals. PSA provides architecture and requirements specifications for building secure platforms. This specification complements PSA. PSA Certified provides a measure of the robustness of an implementation, through an assessment process that is performed by a security certification laboratory.

2 Security requirements

In the following sections of this document, the normative requirements are described in tables. These requirements are distinct from the supporting informative text. The informative text provides additional context to help clarify the rationale for each requirement.

Any system that is designed to conform to this specification must provide a complete implementation of the requirements that is specified in the following sections:

- Authenticated variables (section 2.1)
- Secure boot (section 2.2)
- Secure firmware update (section 2.3)

If the platform implements TPM-based measured boot, the implementation must comply with the requirements in the following section:

- TPMs and measured boot (section 2.4)

2.1 Authenticated variables

UEFI authenticated variables provide a means for a platform owner to control the setting of critical UEFI settings, like variables that affect UEFI Secure Boot. Changes to authenticated variables are verified using digital signatures. The changes must be signed by an appropriate private key.

Authenticated variables must be protected from unauthorized modification. Arm recommends that the implementation of the protection of authenticated variables be considered as part of the platform threat model, which is beyond the scope of this specification.

Systems must implement support for UEFI authenticated variables as specified in Table 1:

Table 1, UEFI Authenticated Variable Requirements

ID	Requirement
R010_BBSR	Authenticated variables must be supported and be compliant with the following sections of the UEFI Specification [2]: Globally Defined Variables (section 3.3) Variable Services (section 8.2)
R040_BBSR	A minimum of 128 KB of non-volatile storage must be available for NV UEFI variables. There is no maximum non-volatile storage limit.
R050_BBSR	The maximum supported variable size must be a least 64 KB.
R060_BBSR	The platform must support EFI variables with any valid combination of the following UEFI variable attributes set: EFI_VARIABLE_NON_VOLATILE EFI_VARIABLE_BOOTSERVICE_ACCESS EFI_VARIABLE_RUNTIME_ACCESS

	EFI_VARIABLE_APPEND_WRITE EFI_VARIABLE_TIME_BASED_AUTHENTICATED_WRITE_ACCESS

2.2 Secure boot

Secure boot is a process that cryptographically authenticates all firmware that runs on a system. Secure boot requires that all firmware is cryptographically signed. This enables the verification process to detect whether firmware components have been compromised or corrupted.

Secure boot begins in an immutable bootloader component, for example a boot ROM, that loads the first mutable firmware image. Before transferring control to the loaded image, the integrity and authenticity of the image is verified using digital signatures. The boot process continues with each image in the boot chain performing integrity and verification of the next image before that image is executed or used. This process forms a chain of trust that is anchored in the immutable bootloader and continues through all code that is executed up to the runtime environment, for example the OS.

As can be seen in Figure 1, one portion of the boot chain is UEFI secure boot, where all components loaded by UEFI-compliant firmware are cryptographically verified. UEFI secure boot includes standards for how secure boot keys are managed. UEFI Secure Boot is defined by the UEFI Specification [2].

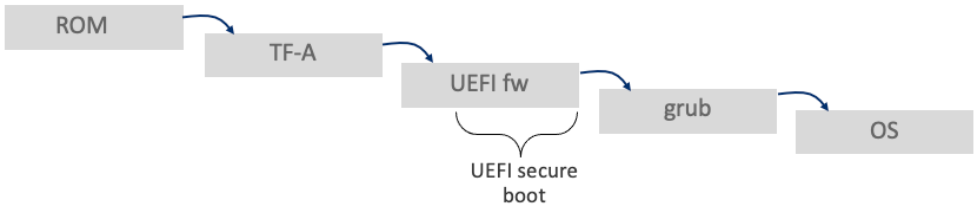

Figure 1, Secure Boot Chain Example

Systems must implement support for UEFI Secure Boot as specified in the Table 2.

Table 2, Secure Boot Requirements

ID	Requirement
R070_BBSR	System firmware must implement UEFI Secure Boot to prevent unauthorized EFI drivers, option ROMs, or programs from being executed during boot.
R080_BBSR	To support UEFI Secure Boot, the system firmware must be compliant with the following sections of the UEFI Specification: Runtime Services Rules and Restrictions (section 8.1) Variable Services (section 8.2) Secure Boot and Driver Signing (section 32)
R020_BBSR	To prevent rollback, the db signature database variable EFI_IMAGE_SECURITY_DATABASE must be created to include the EFI_VARIABLE_TIME_BASED_AUTHENTICATED_WRITE_ACCESS attribute.

R030_BBSR	To prevent rollback, the dbx signature database variable EFI_IMAGE_SECURITY_DATABASE1 must be created to include the EFI_VARIABLE_TIME_BASED_AUTHENTICATED_WRITE_ACCESS attribute to prevent rollback.
R090_BBSR	All UEFI images, for example drivers, applications, boot loaders, that are not contained in the system firmware image must have their signature verified in accordance with Secure Boot UEFI Image Validation in the UEFI Specification section 32.5.
R100_BBSR	System firmware must implement the Secure Boot variable as documented in Globally Defined Variables in the UEFI Specification section 3.3.
R110_BBSR	If authentication of a component fails, that component must not continue to load or execute.
R120_BBSR	It must not be possible for a user to bypass UEFI Secure Boot failures. A physically present user override is not permitted for images that fail signature verification.

2.3 Secure firmware update

A secure firmware update process ensures that only authorized changes are permitted to the firmware in a system. This process ensures that firmware components maintain their integrity.

The Update Capsule architecture is defined by the UEFI Specification. This architecture provides a flexible mechanism to deliver and apply updates for system firmware components, for example Trusted Firmware-A or UEFI, or firmware for I/O devices in the system.

The Base Boot Requirements (BBR) [1] specification requires that in-band system firmware updates be implemented using UEFI Update Capsules. The following requirements must be implemented:

Table 3, Secure Firmware Update Requirements

ID	Requirement
R130_BBSR	In-band firmware updates must be implemented in accordance with the requirements in BBR: • Firmware must implement UEFI update capsules (UEFI specification section 8.5.3) • Firmware must implement the Firmware Management Protocol Data Capsule Structure format (UEFI specification section 23.3) • Firmware must implement an ESRT that describes firmware updated in-band (UEFI specification section 23.4)
R140_BBSR	Capsule payloads for updating system firmware must be digitally signed.
R150_BBSR	Before updates to system firmware are applied, images must be verified using digital signatures.

Note: Prior to the call to UpdateCapsule(), operating systems need to clean the cache by VA to Point of Coherency using the DC CVAC instruction on each ScatterGatherList element that is passed. This is needed only when UpdateCapsule() is called after ExitBootServices(). This requirement will be clarified in a future version of the UEFI specification.

Note: Capsules might be delivered through a file within the EFI system partition, as described in Delivery of Capsules via file on Mass Storage device in the UEFI Specification section 8.5.5.

2.4 TPMs and measured boot

This specification does not mandate use of a TPM. However, if measured boot based on a TPM is implemented, the requirements in this section must be followed.

Measured boot shares some common characteristics with secure boot. During the boot flow hashes are computed of all firmware components before use. However, instead of verifying the components using digital signatures, the hashes (or measurements) are securely stored in a TPM.

A TPM is a security module that provides a number of foundational building blocks for platform security, including:

- Platform Configuration Registers (PCRs) that securely store boot measurements and form the basis for a system to be able to perform secure attestation. PCRs provide one mechanism to implement security policies by sealing TPM objects to PCR values.
- Endorsement key that provides a unique, unclonable identity that is bound to the hardware
- Key storage and management
- Secure cryptography in which keys are never brought into the clear
- Key generation
- True random number generator

A TPM implementation is typically a discrete chip but may also be implemented as a component in a protected environment such as an on-die secure enclave or as a service in the Secure world. System-level threat modeling is required to evaluate potential threats and mitigations to any TPM implementation.

Note: The Arm Base System Architecture specification [3] specifies that, if a system implements a TPM, it must be compliant with version 2.0 of the TCG specifications.

Table 4, TPM and Measured Boot Requirements

ID	Requirement
R170_BBSR	Mutable firmware components and critical data must be measured into PCR[0] through PCR[7] during boot, following the PCR usage guidelines in the TCG PC Client Platform Firmware Profile Specification Revision 1.04 [4].
R180_BBSR	Mutable, Secure world firmware components must be measured into PCR[0].
R190_BBSR	Signed critical data must be measured into PCR[0].
R200_BBSR	All measurements that are made into TPM PCRs must be made with a SHA-256 or stronger hashing algorithm.
R210_BBSR	All measurements that are made into TPM PCRs must be logged in an event log compliant with the definition in the TCG PC Client Platform Firmware Profile Specification Revision 1.04 [4] specification.
R220_BBSR	For systems that implement system description using ACPI, a TPM 2.0 device must be advertised through ACPI tables, as specified in the TCG ACPI Specification [6]. This enables the TPM to be discovered by an operating system or hypervisor.
R230_BBSR	UEFI firmware must implement the EFI_TCG2_PROTOCOL as defined in the TCG EFI Protocol Specification Family 2.0 [5].

R240_BBSR	If physical presence authorization for a TPM is implemented by firmware in a platform, the implementation must follow the requirements in TCG PC Client Platform Physical Presence Interface Specification [7]. The TCG specification describes two methods to provide physical presence authorization: the command method and the hardware method. Either method is acceptable to comply with this specification.

Note: It is acceptable for the first mutable firmware component to measure itself if that component has been cryptographically verified by the immutable bootloader.

As required by the TCG PC Client Platform Firmware Profile Specification, firmware components that are measured into PCR[0] must be logged in the event log using the event type EV_POST_CODE. The following table contains recommended strings for the event data that are used for Secure world firmware components. In the following table, %d represents a platform appropriate integer value, and %s represents a platform appropriate string for the component:

EV_POST_CODE event data	Component description
SYS_CTRL_%d	For firmware for any kind of auxiliary controller in the SoC
BL_%d	For any bootloader component on the application processor. For example BL2 in Trusted Firmware-A would be “BL_2”.
SECURE_RT_EL0_%s	Secure EL0 runtime component
SECURE_RT_EL1_%s	Secure EL1 runtime component
SECURE_RT_EL2	Secure EL2 runtime component
SECURE_RT_EL3	EL3 runtime component. For example BL31 in Trusted Firmware-A nomenclature.

2.5 Platform reset attacks

A platform reset attack occurs when an attacker with physical presence causes a system to be unexpectedly rebooted without a clean shutdown of the operating system. The attacker then makes the system boot on an alternate boot device, like a USB drive or DVD, into an OS that is under the control of the attacker. Actions like resetting the system, power cycling the system, or causing a kernel crash can cause the reboot. A key to the attack succeeding is that the volatile system memory can retain its contents across the attacker-forced reboot. The retention of memory contents can happen even when cycling the system power. After booting into the attacker-controlled OS, the attacker can then scan memory to identify secrets like disk encryption keys.

A mitigation against this attack is defined in TCG PC Client Platform Reset Attack Mitigation Specification v1.10 [8]. The TCG specification defines two UEFI variables that can be set by an operating system to mitigate the attack. The UEFI MemoryOverwriteRequestControl variable tells the firmware to clear memory prior to booting an operating system. The variable MemoryOverwriteRequestControlLock is a protection flag which prevents MemoryOverwriteRequestControl from being modified.

Note: When the UEFI MemoryOverwriteRequestControl variable, ACPI _DSM method, and PSCI memory protection API co-exist, Operating Systems may call any of these interfaces to mitigate the reset attack.

3 Platform security checklist

3.1 Overview

Section 2, *Security requirements,* of this specification describes the requirements to conform to standard security interfaces for a system based on the Base Boot Requirements (BBR) standard [1]. These security interfaces include: UEFI secure boot, secure firmware update, and TPM-based measured boot. However, compliance to these interface standards does not provide assurance that a platform is secure. The underlying platform must have additional security properties to ensure the integrity of the system's firmware and runtime software. This section provides guidance in the form of a checklist to assist in assessing the security properties of a system that may impact secure boot, secure firmware update, and measured boot.

This checklist can be used alone or in conjunction with a security certification program. One program oriented towards embedded and IoT systems is the Platform Security Architecture (PSA) [9] and the PSA Certified framework [10]. PSA provides a comprehensive approach to platform security that is based on defined set of security goals. PSA provides architecture and requirements specifications for building secure platforms.

For PSA, this checklist can assist with completing the PSA Level 1 Device Assessment Questionnaire when a system with BBR-compliant firmware is being PSA certified. The checklist provides specific details about how the requirements in the PSA questionnaire are met. For example, a PSA Level 1 requirement is that a system use secure storage to protect sensitive data. This checklist clarifies that for a BBR-based system, the UEFI secure boot variables are considered sensitive data and must be in tamper-evident storage.

In addition, the checklist provides additional hardening requirements that are necessary for a secure system, but go beyond the PSA Level 1 requirements. For checklist items that map to a PSA Level 1 requirement, a cross-reference to the corresponding PSA requirement is provided.

The checklist is informative and is not required for SystemReady Security Interface Extension certification.

3.2 Security scope

The security goal of this checklist is limited to ensuring the integrity of firmware components, critical data, and other components loaded during the boot flow (such as bootloaders). System-level threat modeling is necessary to identify the set of threats a system may face in the wider context of its usage model.

The following threats are in scope for the checklist:

- Attacker tampers with components loaded and executed during boot
- Attacker tampers with firmware configuration data that affects secure boot, such as UEFI variables
- Attacker attempts to update a firmware component with an authentic but out-of-date image that may contain vulnerabilities
- Attacker bypasses image verification failures during secure boot
- DMA attacks against secure boot
- Attacker uses a signed EFI utility (for example UEFI shell) to subvert the security posture of the system
- Attacker abuses a weak firmware password implementation to get access to firmware configuration

The following are not security goals of this checklist:

- Attacks against the availability of a system
- Glitches of the SoC power supply or clocks during secure boot in order to bypass verification checks
- Laboratory attacks in which devices are unpackaged and probed

- Physical attacks against a TPM, including physical man-in-the-middle attacks

3.3 Resources

The following resources may be useful to users of this checklist:

- Platform Security Boot Guide, v1.1, July 2020, https://developer.arm.com/documentation/den0072/0101/
- Platform Security Requirements, v1.0, September 2020, https://developer.arm.com/documentation/den0106/latest/
- PSA Certified™ Level 1 Questionnaire, https://www.psacertified.org/development-resources/certification-resources/
- U-Booting Securely, Dmitry Janushkevich, F-Secure Hardware Security Team, May 2020

3.4 Guideline conventions

In the guidance sections that follow each normative guideline is identified with an ID and the guidance description. Some guidelines have additional informative information such as examples or cross reference information to the corresponding PSA requirement. For PSA cross references, guidance is provided for how to complete the corresponding checklist item in the PSA Certified checklist.

The informative information is provided in a separate row in the guidance table and is italicized. See the example in Figure 2.

ID	Guideline
G100_BBSR	Systems must implement a secure boot flow that begins in a root of trust for verification.
	For PSA Level 1 C1.2, describe the properties of the root of trust for verification.

Figure 2, Guideline example

3.5 Secure boot

Secure boot is the process by which the integrity of all mutable firmware components in a system is verified before the components are used.

Secure boot is rooted in a *root of trust for verification*. A root of trust for verification is an immutable location, such as a boot ROM, which cryptographically verifies the first mutable firmware in the system using digital signatures before the mutable firmware is executed. Beginning with the root of trust for verification, each component verifies the next component in the boot chain. The firmware boot chain extends to include verifying the OS bootloader. See the example in Figure 3.

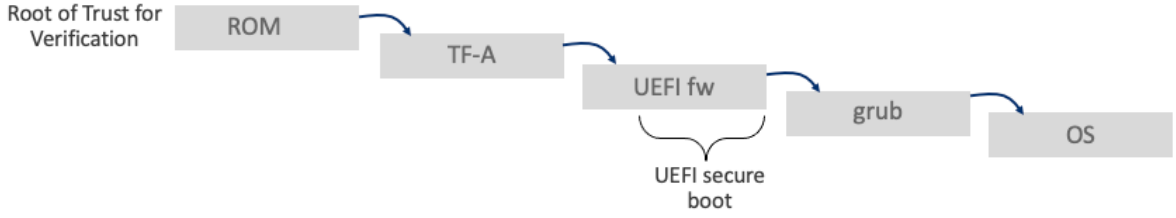

Figure 3, Example Secure Boot Flow

UEFI secure boot (see requirements in section 2.2) is a portion of the secure boot chain, as can be seen in Figure 2. The effectiveness of UEFI secure boot depends on the integrity of preceding components in the boot chain, continuing back to the root of trust for verification.

3.5.1 Image integrity

The secure boot process verifies image integrity using digital signatures. Images verified during secure boot may include signed critical data as well as code. The table below describes the guidelines for secure boot.

Table 5, Secure Boot Guidelines

ID	Guideline
G100_BBSR	Systems must implement a secure boot flow that begins in a root of trust for verification.
	For PSA Level 1 C1.2, describe the properties of the root of trust for verification.
G110_BBSR	The public keys (or hash of the public keys) used by the root of trust for verification must be immutable or tamper resistant.
	For PSA Level 1 C1.4, describe how the public key used by the root of trust for verification is protected.
G120_BBSR	Each stage in the secure boot chain must verify the next stage image using digital signatures before the next stage firmware image is executed.
	The firmware-based secure boot chain extends to the Non-secure bootloader. At a system-level the chain of verification may extend further and include the OS. *Images loaded using network-based protocols (for example PXE, HTTP, ISCSI) must be verified in the same way as images from local storage devices.* *For PSA Level 1 C1.2 and D1.1, describe the secure boot chain and how each stage is verified. Note that for PSA the secure boot chain does extend to verifying the OS.*
G130_BBSR	Secure boot must use hashes and asymmetric keys with at minimum 128-bit security strength. See NIST SP 800-57 [12] for a definition of 128-bit security strength and a list of compliant cryptographic algorithms.
	For PSA Level 1 C2.4 and S2.3, describe the cryptographic algorithms and key sizes used for secure boot image verification.
G140_BBSR	If network-based protocols are used to load images, it is strongly recommended that a protocol be used that can authenticate network connections. Cryptography (asymmetric keys, hashes) used must have at least 128-bit security strength. For example, for HTTP Boot, HTTPS is strongly recommended.

G150_BBSR	Secure boot should have rollback detection during image verification to detect authentic but vulnerable images.
	For PSA Level 1 C2.2, S1.2, and D1.2, describe how rollback detection is implemented.

3.5.2 Critical data integrity for secure boot

In many secure boot architectures there is data, including configuration options and keys, that must be protected from tampering to achieve the security goals of secure boot. For UEFI-based firmware, this includes authenticated variables that enable secure boot and contain verification keys. For U-boot firmware this includes the U-boot environment.

The table below specifies guidelines for protecting critical data that may affect secure boot.

Table 6, Secure Boot Critical Data Integrity

ID	Guideline
G200_BBSR	UEFI authenticated variables must be maintained in tamper-evident secure storage.
	For example, UEFI secure boot variables must be kept in tamper-evident secure storage, including: *SecureBoot* *PK* *KEK* *db* *dbx* *dbt* *dbr* *Tamper-evident means that it must be possible to detect unauthorized modifications made to the variables in non-volatile storage. The specific tamper-evident or tamper-resistant properties required for a given system are threat model dependent.* *For PSA Level 1 S2.2 and D4.5, describe how the storage for authenticated variables is protected.*
G210_BBSR	UEFI variable storage should be protected against rollback to valid but out of date variable data.
G220_BBSR	Firmware configuration parameters that control the firmware boot flow must be protected. It must be possible to configure a system to protect against unauthorized modifications made to firmware configuration parameters that affect secure boot.
	For UEFI firmware, how are unauthorized users are prevented from disabling secure boot or configuring the firmware to be in Setup Mode? *Are the UEFI firmware configuration menus accessible without a password?* *For U-boot based systems describe how the U-boot environment is protected. Is CONFIG_ENV_IS_NOWHERE set?* For PSA Level 1 S2.2, d*escribe how firmware configuration parameters are protected.*

3.5.3 Secure boot hardening

In addition to image verification and the protection of sensitive data, additional hardening is needed so secure boot cannot be subverted by an attacker. For example, if image verification fails how does the system behave?

The table below specifies guidelines for hardening secure boot.

Table 7, Secure Boot Hardening

ID	Guideline
G300_BBSR	If image signature verification fails during secure boot, a recovery process may be initiated, or the boot process must halt in a secure state. A failure in signature verification must not leave the system firmware in a state where it is vulnerable to attack.
	For U-boot based systems, an improperly configured bootcmd variable can result in secure boot failures to fail open, giving an attacker open access to the U-boot command line.
G310_BBSR	It must not be possible for a user to interrupt the secure boot process or bypass secure boot failures. A physically present user override is not permitted for images that fail signature verification.
	For U-boot based systems, the bootdelay variable should be configured so that the boot process cannot be interrupted.
G320_BBSR	All devices in a system that are DMA capable should be behind an SMMU. This includes both Secure and Non-secure devices. An SMMU provides a means to mitigate DMA attacks from a malicious device. A device may be given unrestricted DMA access if it can be established to be trustworthy through an authentication process such as SPDM. The SMMU should have a default “deny” policy after reset. If this is not possible, firmware must configure the SMMU with a default policy of aborting transactions as early as possible during the boot process, preferably in the root of trust for verification. Note: A system with non-host platforms can comply with this guideline since non-host platforms are considered trustworthy.

3.6 Secure firmware update

A secure firmware update process ensures that only authorized changes are permitted to all Secure and Non-secure firmware images in a system. This process ensures that firmware components maintain their integrity.

The table below describes guidelines for both out-of-band (for example from a BMC) or in-band (for example from an OS) firmware updates.

Table 8, Secure Firmware Update

ID	Guideline
G400_BBSR	Firmware update images must be signed with a digital signature that ensures the integrity and authorization of the update, and the update must be verified using digital signatures prior to the update being written to non-volatile storage.
	For PSA Level 1 C2.1, S1.1, and D1.2, *explain how firmware update images are verified.*

G410_BBSR	Firmware update must use hashes and asymmetric keys with at least 128-bit security strength. See NIST SP 800-57 [12] for a definition of 128-bit security strength and a list of compliant cryptographic algorithms.
	For PSA Level 1 C2.4, describe the cryptographic algorithms and key sizes used for signing.
G420_BBSR	The firmware update process must prevent unauthorized rollback of firmware images to older insecure firmware versions. A mechanism may be provided to support authorized rollback for recovery reasons.
	For PSA Level 1 C2.2, S1.2, and D1.2, describe rollback protection during firmware updates.

3.7 Measured boot

Measured boot shares some common characteristics with secure boot. During the boot flow hashes are computed of all firmware components before use. However, instead of verifying the firmware components using digital signatures, the hashes (or measurements) of the components are securely stored in a TPM device. In addition, measurements may be computed of critical data and security relevant system state. The measurements can be later used for attestation (see PSA Level 1 requirement S4.1) or for implementing TPM-based security policies.

The table below specifies the guidelines for measured boot.

Table 9, Measured Boot Guidelines

ID	Guideline
G500_BBSR	Measured boot must be rooted in a root of trust for measurement (RTM). The RTM makes the initial integrity measurement into the TPM. The RTM may be a single, immutable component such as a boot ROM. Alternatively, the RTM may consist of multiple components in the boot chain where measurements are held in memory before being placed in the TPM. In this case each component must verify the next using digital signatures and the RTM boot chain must be rooted in a root of trust for verification. If a component in the RTM is verified it may measure itself.
G510_BBSR	Each stage in the measured boot chain must measure the next stage firmware before the next stage image is executed. This includes Non-secure firmware images. Other security-critical data and state may be measured.
G520_BBSR	Starting with the root of trust for measurement, each stage of boot should extend the measurement made into the TPM. If this is not possible because of system architecture constraints, the measurement may be held in protected memory until the digest value can be extended into the TPM.
	For systems that implement a firmware TPM, it is not possible to extend measurements until the firmware TPM is operational. In this case it is permissible to hold measurements in secure memory until the measurements can be extended.

G530_BBSR	Firmware configuration parameters that control measured boot must be protected. It must not be possible for an unauthorized user to modify firmware configuration parameters that affect measured boot.
	An example of a parameter that may affect measured boot is a firmware option that enables and disables the TPM or measured boot.

3.8 Security hardening

The table below specifies additional hardening guidance.

Table 10, Hardening Guidelines

ID	Guideline
G600_BBSR	Functionality that is not needed for the intended use of the system software shall not be installed, or must be disabled if non-installation is not practical.
	For UEFI firmware, components that are not required to boot the platform must not be signed by a production secure boot certificate. This includes components such as the UEFI shell and utilities for manufacturing, test, and debug. *For U-boot firmware, commands not needed for the operation of the system should be disabled.* *For PSA Level 1 S4.2 and D3.3, describe how this requirement is met.*
G610_BBSR	If firmware makes use of passwords they should conform with security best practices, in particular, password storage, password length and complexity, and the number of failed authentication attempts. See NIST 800-132, Recommendation for Password-Based Key Derivation Part 1: Storage Applications [11].
	For PSA Level 1 S5.1, S5.2, S5.3, D4.1, and D4.2, describe how this requirement is met.
G620_BBSR	Where default passwords are used, they must be unique per device and must not be easily determined by automated means or obtained from publicly available information.
	For PSA Level 1 S5.2 and D4.2, describe how this requirement is met.