tianocore

EDK II Secure Coding Guide

TABLE OF CONTENTS

EDK II SECURE CODING GUIDE

Technical Briefing

08/08/2019 06:27:00

Revision 02.0

Contributed by

Jiewen Yao, Intel Corporation

Vincent J. Zimmer, Intel Corporation

Special Acknowledgements

This document checklist is collected based upon the security experience and previous security issue report. We would like to thank Sugumar Govindarajan, John Mathew, Kirk Brannock, and Karunakara Kotary of Intel Corporation, who provided the thought on hardening the platform.

Acknowledgements

Revision History

Revision	Revision History	Date
01.0	Initial release.	June 2019
02.0	Add "Threat model for EDK II" as the appendix section	Aug 2019

EXECUTIVE SUMMARY

Introduction

The purpose of this document is to help build robust firmware using EDK II. This supplements other documents related to secure design of EDK II code.

Audience

This document is intended for firmware security developers, security reviewers, and firmware security validation engineers.

SECURE CODING GUIDELINES: GENERAL

Guidelines general

General

#GENERAL.1: The code in trusted region MUST check any date from an untrusted region, such as a Portable and Executable (PE) image, capsule image, System Management Mode (SMM) communication buffer, Memory Mapped Input/Output (MMIO) Base Address Register (BAR), etc.

#GENERAL.2: When a code processes the untrusted data, it MUST avoid buffer overflows. Please do not access (write or read) the buffer beyond the size field.

#GENERAL.3: When code processes the untrusted data, it MUST avoid integer overflow. Please use addition instead of subtraction, use division instead of multiplication.

#GENERAL.4: When code processes the untrusted data, it MUST avoid untrusted data overlap with the trusted region.

#GENERAL.5: When code processes the untrusted data, it MUST check the untrusted data in all possible paths.

#GENERAL.INPUT.1: The code MUST check for valid input and reject everything else

#GENERAL.INPUT.2: The code MUST perform sanity checks and bounds checks, such as check type, length, range, format.

#GENERAL.INPUT.3: The code MUST use canonical representation. Always use fully qualified pathnames for files that get opened

#GENERAL.INPUT.4: The code MUST beware of character encoding and watch out for escape characters if using a shell script

#GENERAL.INPUT.5: The code MUST validate as much and as deep as possible to prevent unintentional errors if the code is changed; balance against coding time/performance.

#GENERAL.INPUT.6: The code MUST be careful of boundary conditions (e.g., off by one error, array indices) and conditionals (e.g., reverse logic)

#GENERAL.BUFFER.1: The code MUST check buffer sizes, copies, and indices (esp. sizeof)

#GENERAL.BUFFER.2: The code MUST check for appropriate buffer size. Maximums should be defined globally where possible to avoid assumptions in lower code layers.

#GENERAL.BUFFER.3: The code MUST check for NULL Pointers dereferenced in the code.

#GENERAL.BUFFER.4: The code MUST check NULL for NIL-terminate strings.

#GENERAL.ARITH.1: The code MUST check based on data type limitations (e.g., integer underflows and overflows).

#GENERAL.ARITH.2: The code MUST properly cast numeric variables involved in string manipulation.

#GENERAL.ARITH.3: The code SHOULD use SafeInt library to handle integer of external input.

#GENERAL.FAIL.1: Once the check fails, the code MUST fail secure – fail closed.

#GENERAL.FAIL.2: Once the check fails, the code MUST not provide hints to hackers (e.g., by disclosing information on failure). Don't help an attacker with 'guiding' error messages.

ASSERT

#ASSERT.1: ASSERT MUST be used for something that NEVER occurs. If something MIGHT occur, use ERROR HANDLING, please. Check function return values in a consumed API. (from Code Complete)

#ASSERT.Variable.1: GetVariable with Non-Volatile (NV)+Runtime (RT) without Authentication (AU)/ReadOnly(RO) attribute MUST NOT ASSERT(), or at least have error handling code followed by.

#ASSERT.Variable.2: SetVariable with NV attribute MUST NOT ASSERT(), or at least have error handling code followed by.

#ASSERT.Variable.3: GetVariable with AU/RO attribute MAY ASSERT(), if driver assumes variable must exist.

#ASSERT.Variable.4: SetVariable without NV attribute MAY ASSERT(), before EndOfDxe.

#ASSERT.Resource.1: Memory Allocation MUST NOT ASSERT() after EndOfDxe.

#ASSERT.Resource.2: Memory Allocation MAY ASSERT() before EndOfDxe, if the allocation failure prevents the system from booting. E.g. SEC/PEI phase, the error means configuration error or hardware error. If we need return, we might return to the CPU reset vector, which is meaningless.

#ASSERT.Resource.3: MMIO/IO Allocation for external devices MUST NOT use ASSERT().

#ASSERT.Resource.4: MMIO/IO Allocation for onboard devices MAY use ASSERT() before EndOfDxe.

#ASSERT.SMM.1: SMI handler MUST NOT use ASSERT for external input, after EndOfDxe.

#ASSERT.SMM.2: SMM driver MAY use ASSERT in the entrypoint to construct the environment.

#ASSERT.NETWORK.1: Network driver MUST NOT use ASSERT for the packet from the remote.

#ASSERT.SHELL.1: Shell MUST NOT use ASSERT for the resource request, or user input.

Deprecated API

#DEPRECATEDAPI.1: The code MUST not use any deprecated API.

Table 1.1 EDK II Deprecated APIs

Category	Deprecated API	Replacement
BaseLib	StrCpy	StrCpyS
	StrnCpy	StrnCpys
	StrCat	StrCatS
	StrnCat	StrnCats
	AsciiStrCpy	AsciiStrCpyS
	AsciiStrnCpy	AsciiStrnCpys
	AsciiStrCat	AsciiStrCatS
	AsciiStrnCat	AsciiStrnCats
	UnicodeStrToAsciiStr	UnicodeStrToAsciiStrS
	AsciiStrToUnicodeStr	AsciiStrToUnicodeStrS
PcdLib	[Lib]PcdSet[Ex]8	[Lib]PcdSet[Ex]8S
	[Lib]PcdSet[Ex]16	[Lib]PcdSet[Ex]16S
	[Lib]PcdSet[Ex]32	[Lib]PcdSet[Ex]32S
	[Lib]PcdSet[Ex]64	[Lib]PcdSet[Ex]64S
	[Lib]PcdSet[Ex]Ptr	[Lib]PcdSet[Ex]PtrS
	[Lib]PcdSet[Ex]Bool	[Lib]PcdSet[Ex]BoolS
PrintLib	UnicodeValueToString	UnicodeValueToStringS
	AsciiValueToString	AsciiValueToStringS
UefiLib	GetVariable	GetVariable2
	GetEfiGlobalVariable	GetEfiGlobalVariable2

Race Condition

#RACECONDITION.1: The code MUST be careful of Time-of-Check/Time-of-Use (TOC/TOU) attack for the data crossing a trusted region, such as flash region access, SMM communication buffer access. The right way is to copy the data from an untrusted region to a trusted region and only access the data in the trusted region.

#RACECONDITION.2: The code MUST be careful of race conditions for the Bootstrap Processor (BSP) and Application Processors (AP). The BSP and AP may run different code in different trusted regions. Identify and Keep security critical sections short and simple.

Policy

#POLICY.BLAKLIST.1: If a prohibited list is used, a system error on getting prohibited list data MUST cause a prohibited list match and verification failure.

#POLICY.BLAKLIST.2: If a prohibited list is used, the prohibited list match MUST always cause a verification failure, no matter if the allowed list matches or not.

#POLICY.WHITELIST.1: If an allowed list is used, any error on getting allowed list data MUST cause verification failure.

Environment

#ENVIRONMENT.RUNTIME.1: The runtime module MUST be built with 4K alignment so that a Runtime image can be protected by the OS. (SecurityEnhancement)

#ENVIRONMENT.UEFI.1: The boot module MAY be built with 4K alignment so that it can be protected by firmware.

#ENVIRONMENT.NX.1: The Code region SHOULD be set to ReadOnly (RO), and Data region SHOULD be set to NonExecutable (NX). (SecurityEnhancement)

#ENVIRONMENT.NX.2: The unallocated memory SHOULD be non-present or at least Non-Executable.

#ENVIRONMENT.STACK.1: Stack SHOULD be set to be NX. (SecurityEnhancement)

#ENVIRONMENT.STACK.2: Stack Guard SHOULD be enabled to catch stack overflow.

#ENVIRONMENT.HEAP.1: Heap SHOULD be set to be NX for data. [SecurityEnhancement]

#ENVIRONMENT.HEAP.2: A platform MAY use heap guard before release to check potential heap overflow.

#ENVIRONMENT.ASLR.1: A platform MAY enable Address Space Layout Randomization (ASLR). (SecurityEnhancement)

#ENVIRONMENT.ASLR.2: If ASLR is used, the platform MUST not expose any randomized information, such as a function address in a module, global data address, or the CPU architecture state address.

#ENVIRONMENT.ASLR.3: If ASLR is used, the platform MUST choose appropriate entropy. Too small of entropy may make ASLR not useful. Too much entropy may impact memory allocation in a resource-constrained environment.

#ENVIRONMENT.CONTROLFLOW.1: Control Flow Guard MAY be enabled to prevent Return Oriented Program (ROP), Call Oriented Program (COP)/Jump Oriented Program (JOP) attack. (CET EDKII)

Crypto

#CRYPTO.1: A platform SHOULD NOT use any deprecated crypto-algorithm.

#CRYPTO.2: A platform SHOULD NOT implement its personally owned crypto algorithms or protocols.

#CRYPTO.3: A platform MUST follow cryptographic standards exactly.

#CRYPTO.HASH.1: A platform SHOULD use Secure Hash Algorithm (SHA) 256 equivalent or stronger.

#CRYPTO.HASH.2: A platform SHOULD NOT use SHA1 or Message Digest (MD) 4, MD5.

#CRYPTO.SYM.1: A platform SHOULD use Advanced Encryption Standard (AES) equivalent or stronger.

#CRYPTO.SYM.2: The key MUST NOT be saved in flash as plain text.

#CRYPTO.ASYM.1: A platform SHOULD use Rivest-Shamir-Adleman algorithm (RSA) or Elliptic curve cryptography (ECC) equivalent or stronger.

#CRYPTO.ASYM.2: The private key MUST NOT be saved in flash as plain text.

#CRYPTO.RANDOM.1: A platform SHOULD use an approved random number generator.

Password

#PASSWORD.1: The password plaintext MUST NOT be saved to a variable. Alternative: 1) save SALT+HASH to a variable, 2) save to Hardware directly, 3) save to System Management RAM (SMRAM) for S3.

#PASSWORD.2: The password update MUST be in a secure environment, such as SMM, or before EndOfDxe.

#PASSWORD.3: The password in firmware MUST meet common password criteria (strength, update, algorithm, retry time, old password check, password lost, etc)

#PASSWORD.4: The password MAY be used to for authentication, (Entering setup page, setup variable access, WIFI PSK), for decryption (TLS private certificate)

#PASSWORD.5: The password in memory MUST be cleared after use. NOTE: The secrete MAY be in a global data region, stack or heap.

#PASSWORD.6: The password MUST NOT be hardcoded in the code.

#PASSWORD.7: If the code needs to compare the plain text password, the code MUST always compare all characters of the string, instead of breaking on the first mismatch.

#PASSWORD.8: Salt MUST be added to resist rainbow table attack.

#PASSWORD.9: Hash generation MUST add enough iteration to make sure the hash calculation is slow.

Secret

#SECRET.1: The secret MUST NOT be saved in a variable or disk directly as plain text. The secret includes but is not limited to the user password, hard drive password, Trusted Computing Group (TCG) OPAL password, or Trusted Platform Module (TPM) platform author value, network password, or private certificate password.

#SECRET.2: The secret in memory MUST be cleared after use, including usernames, passwords, keys, and other sensitive security information. NOTE: The secret MAY be in a global data region, stack or heap. The buffer content should be zeroed before released.

#SECRET.3: The secret MUST NOT be hardcoded in the code.

#SECRET.4: If the code needs to compare the secret, the code MUST always compare the entire data before completion.

#SECRET.5: The length of the secret MUST be large enough to resist a brute force attack.

Network

#NETWORK.1: The network driver MUST always validate the packet from the network. Don’t trust any length, size, offset field.

#NETWORK.2: Each layer if the network driver MUST validate its own header.

#NETWORK.TLS.1: The public cert MUST be stored in the UEFI authenticated variable or read-only region flash region.

#NETWORK.TLS.2: The private cert plain text MUST NOT be stored to readable flash region.

#NETWORK.WIFI.1: The WIFI password plain text MUST NOT be stored to readable flash region.

Hardware

#HARDWARE.1: The untrusted input from hardware MUST be verified, such as Dual-Inline-Memory-Modules (DIMM) Serial Presence Detect (SPD) data, Universal Serial Bus (USB) descriptor, Bluetooth Low Energy (BLE) advisory information, etc.

DMA

#DMA.1: The device Direct Memory Access (DMA) MUST be disabled by default, either via Peripheral Component Interconnect (PCI) Bus Master Enable (BME) or Input/Output Memory Management Unit (IOMMU) engine. (IOMMU EDK II)

#DMA.2: The device DMA MUST be enabled if and only if it is requested to perform a transaction. After completing the transaction, the device DMA MUST be disabled.

#DMA.3: If an IOMMU engine is used for DMA protection, the IOMMU MUST cover all physical DRAM regions.

#DMA.4: The DMA capable region MUST have no overlap with any other existing code region or data region.

#DMA.5: The IOMMU engine SHOULD be configured for fine granularity control. The DMA capable region SHOULD be per device in the DXE phase.

#DMA.6: The DMA in a debug device in debug image MAY be enabled at all times.

Other

#SIDECHANNEL.1: The high privilege code (SMM) MUST use SpeculationBarrier() after validation of untrusted data but before use to mitigate Bounds Check Bypass. (SideChannel)

#SIDECHANNEL.2: The high privilege code (SMM) MUST use StuffRsb before RSM to mitigate Branch Target Injection. (SideChannel)

#MDS.1: The high privilege code (SMM) MUST rendezvous all logical processors both on entry to and exit from SMM to ensure that a sibling logical processor does not reload data into microarchitectural structures after the automatic flush. (MDS)

SECURE CODING GUIDELINES: BOOT FIRMWARE

Flash

#FLASH.1: The platform MUST lock flash part no later than EndOfDxe.

#FLASH.2: The flash lock MUST happen in all boot mode (normal, S3, S4, capsule update, recovery, etc).

Flash Update

#FLASH.UPDATE.1: Firmware Flash Update MUST check the integrity of a new Firmware image. (CapsuleRecovery)

#FLASH.UPDATE.2: Firmware Flash Update MUST check the version of a new Firmware image.

#FLASH.UPDATE.3: The verification process MUST happen in the trusted execution environment.

#FLASH.UPDATE.4: The new Firmware MUST exist in a trusted region when the verification performs.

#FLASH.UPDATE.5: The verification and update MUST happen in the same trusted execution environment and with no interruption.

#FLASH.UPDATE.6: System Firmware Update requests MUST be handled before EndOfDxe.

#FLASH.UPDATE.7: Device Firmware Update requests MAY be handled after EndOfDxe.

Recovery

#FLASH.RECOVERY.1: Firmware recovery MUST check the integrity of a Firmware recovery image if it is from an external source. (CapsuleRecovery)

#FLASH.RECOVERY.2: Firmware recovery MUST check the version of a Firmware recovery image if it is from an external source.

Variable

#VARIABLE.1: A platform MUST lock the variable before EndOfDxe if it is critical, such as memory configuration, TPM Physical Presence (PP) flag. NOTE: This locked variable can still be updated in SMM. (Variable)

#VARIABLE.2: A platform MUST use the same lock policy in a normal boot and S4.

#VARIABLE.3: A platform MAY define the variable property (attribute, max size, min size). This variable property can be a list defined by the platform.

#VARIABLE.4: The return status for variable access MUST be verified.

#VARIABLE.SET.1: A platform MUST use error handling on variable set if the variable NOT locked, such as OUT_OF_RESOURCE, or attribute mismatch.

#VARIABLE.SET.2: A platform MUST provide a way to handle variable out of resource. Such as clean unknown variable.

#VARIABLE.GET.1: A platform MUST use error handling variable get if the variable is NOT locked. Such as NOT_FOUND, or unexpected value.

#VARIABLE.GET.2: A platform MUST not assume to get the correct variable content/size if the variable is NOT locked.

#VARIABLE.ATTRIB.1: A platform MUST NOT set variable RT attribute unless it is needed.

#VARIABLE.ATTRIB.2: A platform MUST NOT set variable NV attribute unless it is needed.

#VARIABLE.CHECK.1: A platform MUST enable Human Interface Infrastructure (HII) variable check.

#VARIABLE.CHECK.2: A platform MUST enable Platform Configuration Database (PCD) variable check.

#VARIABLE.CHECK.3: A platform MUST enable UEFI variable check.

#VARIABLE.QUOTA.1: A platform MUST define a reasonable variable quota.

#VARIABLE.CONFIDENTIALITY.1: Current variable driver does not provide confidentiality support. If the variable confidentiality is needed, the caller MUST encrypt it and decrypt it.

#VARIABLE.CONFIDENTIALITY.2: The encryption and decryption MUST be in a secure execution environment, such as SMM.

S3

#S3.1: S3 BootScript MUST be saved in a secure place (lockbox) (S3Resume)

#S3.2: S3 BootScript image dispatch and parameter MUST be saved in a secure place (lockbox).

#S3.3: S3 CPU data MUST be saved in a secure place (SMM).

#S3.4: S3 configuration data (memory, chipset) MUST be saved in a secure place (ReadOnly variable, SMM).

Secure Boot

#SECUREBOOT.1: Platform MUST NOT provide a way to disable secure boot without authentication.

#SECUREBOOT.2: Platform MUST verify all images in the secure boot path. The bypass of any image verification is NOT ALLOWED.

#SECUREBOOT.3: If a firmware update is supported, the new firmware image must be signed.

#SECUREBOOT.4: If a firmware recovery is supported, the recovery firmware image must be signed. The recovery firmware image is from an external source.

#SECUREBOOT.5: The verification MUST happen in all boot path (normal, S3, S4, capsule update, recovery, etc).

#SECUREBOOT.UEFI.1: UEFI secure boot MUST be used to verify the 3rd part image, such as PCI Option ROM, or OS loader.

#SECUREBOOT.UEFI.2: If UEFI secure boot is used, any 3rd part image MUST be verified.

#SECUREBOOT.UEFI.3: If UEFI secure boot is used, a platform MUST implement the PlatformSecureLib to provide a secure platform-specific method to detect a physically present user.

#SECUREBOOT.Key.1: If signing verification is required, the public key or hash MUST be stored in hardware, or boot block or UEFI authenticated variable.

TCG

#TCG.1: If TCG trusted boot is enabled, a platform MUST do measurement following the TCG specification. (TPM2 EDK II)

#TCG.2: All parent Firmware Volumes (FV) MUST be reported and measured. The child FV MAY NOT be measured.

#TCG.3: If an image is not in a FV, it MUST be measured. The image inside of a FV MAY NOT be measured.

#TCG.4: Any FV MUST be measured, including external FV from firmware update capsule, or recovery.

#TCG.5: The platform MUST link Tcg2MeasurementLib to be the last NULL instance for SecurityStub.

#TCG.SMBIOS.1: The platform MUST make sure the measured SMBIOS record is the same cross boot. (If the record might be different, it MUST NOT be measured)

#TCG.MOR.1: Memory Override (MOR) request MUST be handled by the platform/Memory Reference Code (MRC).

#TCG.MOR.2: Storage MOR request MUST be handled by platform BDS before EndOfDxe.

#TCG.MOR.3: If the MOR variable is not got, the platform MUST treat it as MOR requested.

#TCG.MOR.4: Secure MOR MUST be used for the platform. (SecureMOR)

#TCG.PP.1: TCG TPM Physical Presence (PP) request MUST be handled before EndOfDxe.

#TCG.PP.2: TCG storage PP request MUST be handled before EndOfDxe.

#TCG.PP.3: TCG TPM PP flag MUST be locked in normal boot and S4.

#TCG.PP.4: TCG storage PP flag MUST be locked in normal boot and S4.

#TCG.TPM.1. A Platform MUST generate random TPM2 PlatformAuth and discard it in normal boot.

#TCG.TPM.2: A Platform MUST generate random TPM2 PlatformAuth and discard it in S3 if device error in S3.

#TCG.TPM.3: A platform MAY disable TPM if the TPM device returns an error.

#TCG.TPM.4: A platform MUST handle TPM device error in both normal boot and S3.

#TCG.TPM.5: If S3 Startup(STATE) fails, the platform MUST send Startup(Clear) and extend error code to PCRs.

Option ROM

#OROM.1: Any 3rd party option ROM MUST NOT be dispatched before EndOfDxe. The 3rd party option ROM means a PCI option ROM on the 3rd party PCI card. If a PCI option ROM is integrated inside of the firmware, it is NOT considered as a 3rd party option ROM and it

MAY be dispatched before EndOfDxe.

#OROM.2: Platform BDS MUST handle the deferred PCI option ROM after EndOfDxe.

Console

#CONSOLE.1: A trusted console SHOULD be available before EndOfDxe, based upon the need. A trusted console means: 1) Integrated device such as a PS2/USB keyboard/mouse without any option ROM, 2) A chipset-integrated video card which is welded to the board and whose driver is inside of the boot firmware instead of in a PCI option ROM, 3) A 3rd party video card which is welded to the board and whose driver is inside of the boot firmware instead of in a PCI Option ROM.

#CONSOLE.2: The trusted console MAY support features, such as TCG Physical Presence, hard drive Password or TCG OPAL password.

#CONSOLE.3: If a remote console is used, additional authentication MAY be used, such as user password.

Storage

#STORAGE.1: A platform MUST connect trusted storage device before EndOfDxe, based upon the need. A trusted storage means:

1. Integrated device, such as a Universal Serial Bus (USB), Advanced Technology Attachment (ATA), AT Attachment Packet Interface (ATAPI), Non-Volatile Memory express (NVMe), Universal Flash Storage (UFS), Embedded MultiMedia Card (eMMC), Secure Digital (SD) Card, without any option ROM,
2. A chipset-integrated storage card, such as Redundant Arrays of Independent Drives (RAID), which is welded to the board and whose driver is inside of the boot firmware instead of in a PCI Option ROM,
3. A 3rd party storage card, such as Small Computer System Interface (SCSI), Fiber Channel (FC), which is welded to the board and whose driver is inside of the boot firmware instead of in a PCI Option ROM.

#STORAGE.2: A platform MUST send TPer Reset to storage device if there is a MOR request.

#STORAGE.3: A platform MUST let user input hard drive password or TCG OPAL password to unlock the hard drive if the hard drive is locked.

#STORAGE.4: A platform MUST disable the DMA on the storage if it is not a boot device. (Disconnect Device)

Silicon

#SILICON.1: The lockable silicon register MUST be locked before EndOfDxe, include SMRAM, Flash, Top Swap, Remap Bar, etc

#SILICON.2: The lockable silicon register MUST be locked in all booth path including S3, S4, Capsule, Recovery, etc.

SECURE CODING GUIDELINES: INTEL PLATFORMS

- SMM
- Intel® Boot Guard
- Intel® BIOS Guard

SMM

#SMM.0: Applications and functions should use the least privilege that will get the job done. If DXE or Runtime can finish the work, don't use SMM.

#SMM.1: SMM module MUST lock SMM, at SmmReadyToLock.

- **#SMM.1.1**: Boot firmware MUST set SMRAM Range Register (SMRR) for Top Segment (TSEG) correctly at SmmReadyToLock for all processors.
- **#SMM.1.2**: Boot firmware MUST use TSEG only and MUST NOT use A/B Segment (ABSEG) as SMRAM.
- **#SMM.1.3**: Boot firmware MUST remove SMRAMC_D_OPEN at SmmReadyToLock.
- **#SMM.1.4**: Boot firmware MUST set SMRAMC_D_LOCK at SmmReadyToLock.
- **#SMM.1.5**: Boot firmware MUST close all unnecessary services at SmmReadyToLock, such as the capability to register a new SMM driver into SMRAM.
- **#SMM.1.6**: All the above locks MUST be performed in the S3 resume path before the control is transferred to OS waking vector.

#SMM.2: SMM module MUST NOT call any code outside of SMRAM, after SmmReadyToLock. (SMMProtection)

- **#SMM.2.1**: Boot firmware MUST make sure there is an SMM code call outside of SMRAM after SmmReadyToLock.
- **#SMM.2.2**: UEFI/PI Boot firmware MUST make sure there is NOT boot services or UEFI protocols are called from SMM code after SmmReadyToLock.
- **#SMM.2.3**: UEFI/PI Boot firmware MUST make sure there is NOT runtime services are called from SMM code after SmmReadyToLock.
- **#SMM.2.4**: UEFI/PI Boot firmware MUST make sure there are NOT dynamic PCDs that are accessed from SMM code after SmmReadyToLock.
- **#SMM.2.5**: UEFI/PI Boot firmware MUST choose the library or MACRO correctly to make sure these libraries or MACROs do not call outside of SMRAM after SmmReadyToLock.
- **#SMM.2.6**: All the above restrictions MUST be performed in the S3 resume path.
- **#SMM.2.7**: Boot firmware MUST enable ExecutionDisable (XD) feature provided by Intel CPU.
- **#SMM.2.8**: If the hardware supports SMM_Code_Access_Chk Model Specific Register (MSR), boot firmware MUST enable SMM_CODE_ACCESS for all processors.

#SMM.3: SMM module MUST check any communication memory between SMM and non-SMM, to make sure it does not impact the integrity, confidentiality or availability of SMM. (SmmComm)

- **#SMM.3.1**: Boot firmware SMM module MUST check any data from a non-SMM component, to make sure the data buffer is NOT overlapped with SMRAM.
- **#SMM.3.2**: If the data buffer contains a pointer, the Boot firmware SMM module MUST check the pointer to make sure the buffer pointed is NOT overlapped with SMRAM.
- **#SMM.3.3**: If the data buffer is from MMIO BAR, the Boot firmware SMM module MUST check the pointer to make sure the buffer pointed is NOT overlapped with SMRAM.
- **#SMM.3.4**: Boot firmware SMM module MUST calculate the data buffer carefully to avoid integer overflow.
- **#SMM.3.5**: Boot firmware SMM module MUST copy the communication buffer to SMRAM before the check, to resist TOC/TOU or DMA attacks.

#SMM.4: SMM module MUST check any communication memory between SMM and non-SMM, to make sure it does not impact the integrity, confidentiality or availability of VMM. (SmmComm)

- **#SMM.4.1**: Boot firmware SMM module MUST check any data from a non-SMM component, to make sure the data buffer is a fixed communication buffer range. “Fixed” here means a buffer used for Boot firmware SMM only, such as ReservedMemory, ACPINvs, or UefiRuntimeServicesData.
- **#SMM.4.2**: If the data buffer contains a pointer, the Boot firmware SMM module MUST check the pointer to make sure the buffer pointed is in a fixed communication buffer range.
- **#SMM.4.3**: If the data buffer is from MMIO BAR, the Boot firmware SMM module MUST check the pointer to make sure the buffer pointed is in MMIO range.
- **#SMM.4.4**: Boot firmware SMM module MUST calculate the data buffer carefully to avoid integer overflow.
- **#SMM.4.5**: Boot firmware SMM module MUST set the DRAM region to be not-present other than SMRAM and the fixed communication buffer range.

#SMM.5: SMM module MUST check MMIO access, to make sure it does not impact any bits which can only be accessed in SMM, or which only need to be accessed in VMM. Such as Intel Virtualization Technology for Direct I/O (VTd) BAR or Serial Peripheral Interface (SPI) BAR.

- **#SMM.5.1**: If the data buffer is from MMIO BAR, Boot firmware SMM module MAY check if the BAR is the same as the original value in preboot, and deny the access if the BAR is changed.

#SMM.6: SMM module MUST be built with 4K alignment so that the SMM image can be protected with section attributes by SMM. (MemoryProtection)

- **#SMM.6.1**: SMM module MUST be built with 4K alignment.
- **#SMM.6.2**: The code section of the SMM image MUST be set as read-only in the page

table.

- **#SMM.6.3**: The data section of the SMM image MUST be set as non-executable in the page table.
- **#SMM.6.4**: The SMM entrypoint code MUST be read-only.
- **#SMM.6.5**: The platform MUST configure SMM with a static page table.
- **#SMM.6.6**: The page table itself MUST be Read-Only.
- **#SMM.6.7**: The Global Descriptor Table (GDT), Interrupt Description Table (IDT) MUST be Read-Only. (The only exception is IA32 GDT with stack guard enabled because stack switch need task switch)
- **#SMM.6.8**: All other SMRAM data must be non-executable. (Stack, Heap)

#SMM.7: A platform MUST remove any unnecessary System Management Interrupt (SMI) handlers. (Profile)

- **#SMM.7.1**: A platform MAY enable SMI handler profile feature to check all SMI handlers.

#SMM.8: An SMM module SHOULD use a read-only page to save the critical data structure. As such this critical data structure is locked after SmmReadyToLock.

#SMM.10: An SMM module MUST use a read-only page for any S3 data, that needs to be referred before SMM rebase in the S3 resume phase.

#SMM.11: If Control Flow Enhancement Technology (CET) is supported, the SMM MUST enable CET to prevent ROP attack. (CET EDK II)

Intel® Boot Guard

#BootGuard.1: A full secure/verified boot flow MUST include Intel® Boot Guard, OEM Boot Block (OBB) Verification.

#BootGuard.2: Intel® Boot Guard SHOULD be used to verify the Initial Boot Block (IBB).

#BootGuard.3: If Intel® Boot Guard is used, all IBB portion MUST be signed.

#BootGuard.4: If Intel® Boot Guard is used, the verification MUST happen in all boot path, including normal, S3, S4, capsule update, recovery.

#BootGuard.5: After the memory is initialized, the code in the IBB flash region MUST not be invoked. Only the memory copy MAY be invoked, including PSI service, PPI, and a callback function.

#BootGuard.6: After the memory is initialized, the data in the IBB flash region MUST not be referred. Only the memory copy MAY be referred, including HOB, global data in PPI, system state, GDT, IDT, Firmware Information Table (FIT), Boot Policy Manifest (BPM), Key Manifest (KM), etc.

#BootGuard.7: Once Intel® Boot Guard is enabled, there MUST be no way to disable it.

#ObbVereification.1: The IBB MUST be used to verify the OEM Boot Block (OBB).

#ObbVereification.2: Any OBB Firmware Volume images MUST be signed.

#ObbVereification.3: The verification MUST occur for the OBB code in memory.

Intel® Bios Guard

#BiosGuard.1: Intel® Bios Guard SHOULD be used for a firmware update.

#BiosGuard.2: The platform MUST make sure all the processes are in SMM when performing the Intel® Bios Guard update.

#BiosGuard.3: Once Intel® Bios Guard is enabled, there MUST be no way to disable it.

APPENDIX:THREAT MODEL FOR EDK II

This chapter provides the basic assumptions for the threat model of EDK II firmware. The threat model discussed here is a general guide and serves as the baseline of the EDK II firmware. For each specific feature in EDK II firmware, there might be additional feature-based threat models in addition to the general threat model.

In UEFI Threat Model, we discussed the asset, threat and mitigation. Here we will revisit these items and based upon STRIDE).

Threat	Desired Property
Spoofing	Authentication
Tampering	Integrity
Repudiation	Non-Repudiation
Information Disclosure	Confidentiality
Denial of Service	Availability
Elevation of Privilege	Authorization

In EDK II firmware, the denial of service can be temporary in the current boot, or permanent in which case the system never boot again. The latter is more serious and it is named as permanent denial of service (PDoS).

We will consider the below adversary for the EDK II firmware:

Adversary	Capability
Network Attacker	The attacker may connect to the system by network in order to eavesdrop, intercept, masquerade, or modify the network packet.
Unprivileged Software Attacker	The attacker may run ring-3 software in an OS application layer. The attacker may perform a software based side channel attack (such as using cache timing).
System Software Attacker	The attacker may run ring-0 software in the OS kernel or hypervisor, or run 3rd party firmware code in firmware boot phase. The attacker may perform the software based side channel attack (such as using cache timing, performance counters, branch information, or power status).
Simple Hardware Attacker	The attacker may touch the platform hardware (such as power button or jumper) and attach/remove a simple malicious device (such as hardware debugger, PCI Leach to the external port, PCIE card to the PCIE slot, memory DIMM, NIC cable, hard drive, keyboard, USB device, Bluetooth device). The attacker may hijack the simple system bus (such as the SPI bus or I2C bus).
Skilled Hardware Attacker	The attacker may hijack the complex system bus (such as memory bus, or PCI express bus). The attacker may perform the hardware based side channel attack, such as power analysis, thermal analysis, or electromagnetic analysis. The attacker may perform a glitch attack.

We will consider the below mitigations for the EDKII firmware:

Mitigation	Objective
Protection	The mitigation is to prevent such an attack for damaging the system.
Detection	The mitigation is to detect if the system is under attack.
Recovery	The mitigation is to recover the system if it is under attack.

- Asset: Flash Content
- Asset: Boot Flow
- Asset: S3 Resume
- Asset: Management Mode
- Asset: Build Tool

Asset: Flash Content

NIST SP 800-147 and SP 800-147B provide system firmware protection guidelines, including the detailed information on system firmware protection and update. NIST SP 800-193 provides platform firmware resiliency guidelines. It extends protection to 3 principles: protection, detection, and recovery. It also enlarges the scope from system firmware (BIOS) to all the firmware on the platform.

The flash content here includes both firmware code (such as PEI, DXE, SMM etc) and firmware data (such as UEFI variables, Microcode, etc).

Threat	Example
Spoofing	N/A
Tampering	If the firmware is not protected or locked, the attacker might modify the firmware directly. If the firmware update process is not authenticated, the attacker might send a malicious firmware update image for update.
Repudiation	N/A
Information Disclosure	If the system software stores the secret in the firmware, the attacker may read the firmware content and get the secret.
Denial of Service	If the attacker can modify the firmware content (code or data) and cause the firmware crash, the system might no longer boot. It becomes a permanent denial of service.
Elevation of Privilege	If the attacker can modify the firmware content (code or data) and store a Trojan in firmware, the Trojan may hide itself and gain the higher privilege.

Adversary	Example
Network Attacker	If the network is enabled before SMM lock and flash lock, the attacker may send mal-formed network packets.
Unprivileged Software Attacker	The attacker may trigger a firmware update, or write the UEFI variable.
System Software Attacker	The attacker may access a silicon register to unlock the flash access register. The attacker may create a race condition to break the flash write protection or flash update authentication.
Simple Hardware Attacker	The attacker may press the power button during flash update or recovery, or set a jumper to modify the system boot mode from normal boot to recovery or even manufacturing mode. The attacker may attach PCI Leach to perform DMA attack during flash update or recovery. The attacker may hijack the SPI bus to read or write to the chip data.
Skilled Hardware Attacker	N/A

Mitigation	Example
Protection	For the code region, the flash write protection must always be applied. During the flash update, the new firmware image must be authenticated and the version must be checked to prevent a rollback attack. In order to mitigate Time-of-check/Time-of-use (TOC/TOU) attacks, the new image must be copied to a secure environment before the check. The DMA protection must be enabled during flash update. For the data region, the UEFI authenticated variable write must happen in an isolated execution environment. The authenticated variable data must be authenticated and the rollback protection must be applied. Just as in code region protection, in order to mitigate TOC/TOU attack, new variable content must be copied to a secure environment before the check and DMA protection must be applied to this environment. In addition, the secret must not be saved to the firmware code or data region.
Detection	The detection happens in the next boot. For the code region, the industry may have different solutions to make sure the initial boot code is unmodified, such as Project Cerberus, Intel® Boot Guard, etc. For the data region, the UEFI variable driver needs to detect if the variable region is modified without using UEFI variable services.
Recovery	If something wrong is detected, the entity which detects the failure needs to start the recovery process, and the recovery data must be in a known good and secure configuration and be delivered from a trusted and always available source.

Asset: Boot Flow

The main system firmware work is to initialize the silicon and then transfer control to an operating system. Because the firmware is almost the first component running on the system, another responsibility of the system firmware is to maintain the secure boot chain (defined in UEFI specification) and the trusted boot chain (defined by TCG).

Here the secure boot chain means that the first entity needs to verify if the second entity is good before running it, not run the second entity if the verification fails. The trusted boot chain means that the first entity needs to measure the second entity before running it and then always run the second entity. The attestation may happen later. The system firmware needs to maintain both boot flows carefully. The verification and measurement must not be bypassed.

In addition, the system firmware may need to authenticate the end user, to determine if the user is authorized to perform some action. For example, the user may be asked to input a hard driver password to continue the boot. Or the user may be asked to input an administrator password to enter a setup page. Those actions must not be bypassed as well.

Threat	Example
Spoofing	If the firmware needs to authenticate the user, the attacker may spoof the identity, or bypass the authentication check.
Tampering	The attacker may want to modify the secure boot logic or trusted boot logic (either code or configuration data) to bypass the verification or measurement.
Repudiation	N/A
Information Disclosure	The user identity and device password are secret information. The attacker may want to steal them.
Denial of Service	The attacker may modify the secure boot configuration data to cause a system crash during verification.
Elevation of Privilege	If the attacker bypasses the user authentication, he or she may enter firmware setup page to modify the configuration.If the attacker bypasses the secure boot verification, he or she may run the unauthorized 3rd part code in the ring-0 environment.

Adversary	Example
Network Attacker	The attacker may send malformed network packets to inject code into the system. The attacker may send a bad UEFI image to bypass or break the secure boot logic.
Unprivileged Software Attacker	The attacker may write a malformed UEFI authenticated variable to break the secure boot configuration.
System Software Attacker	The attacker may send a command to the isolated execution environment in order to modify the secure boot configuration. The attacker may enable a side channel to get secrets from memory.
Simple Hardware Attacker	The attacker may attach PCI Leach to perform DMA attack to read the secret from memory, or write the code region to bypass the verification.
Skilled Hardware Attacker	The attacker may hijack the memory bus to read secrets from memory, or write the code region to bypass the verification.

Mitigation	Example
Protection	Do check for untrusted external input before use (such as network packet, option ROM, OS loader, and UEFI authenticated variable). Do not run any untrusted 3rd part code before verification. If the secret is generated, it must be cleared after use (such as temporary input from HII). If the secret needs to be stored, the choice includes: to save secret to hardware directly (such as OPAL password), to save hash plus salt to a UEFI variable (such as user password), to save the secret in an isolated environment (such as TPM MOR2). Side channel prevention must be applied in this case. DMA protection must be enabled. Memory encryption must be used if the memory bus attack is in scope.
Detection	N/A
Recovery	N/A

Asset: S3 Resume

S3 resume is a special boot flow. It is defined by ACPI specification. During S3 resume, the system restores the configuration from a normal boot and jumps to OS waking vector.

All protection applied to the normal boot must also be applied in S3 resume.

Threat	Example
Spoofing	N/A
Tampering	The attacker may try to modify the S3 configuration, also known as S3 boot script.
Repudiation	N/A
Information Disclosure	If the s3 configuration includes a secret (such as HDD password), the attacker may want to steal the secret.
Denial of Service	The attacker may destroy the S3 configuration to prevent the system from booting.
Elevation of Privilege	The attacker may disable the protections stored in the S3 configuration such as register lock.

Adversary	Example
Network Attacker	N/A
Unprivileged Software Attacker	The attacker may write a malformed UEFI variable to break the S3 configuration.
System Software Attacker	The attacker may send a command to the isolated execution environment to modify the S3 configuration. If there is a secret saved in the isolated environment, the attacker may send a commend to get the secret, or use a side channel to steal the secret.
Simple Hardware Attacker	N/A
Skilled Hardware Attacker	N/A

Mitigation	Example
Protection	The S3 configuration data must be saved to a secure place. For example, embedded into read only code region, a read only variable, an isolated execution environment, or a lock box. If the S3 configuration data is secret, then it must be saved in an isolated execution environment or a lock box to prevent unauthorized reads.
Detection	N/A
Recovery	N/A

Asset: Management Mode

Management mode is a special system execution environment. X86 systems have system management mode (SMM), and ARM has ARM TrustZone. The firmware code in management mode is considered as a secure world and having high privilege.

Threat	Example
Spoofing	N/A
Tampering	The attacker may update the management mode memory to inject code or data.
Repudiation	N/A
Information Disclosure	The management mode may contain a secret (such as password, TPM MOR2 entropy), or its own information (code and data structure location). This information may be exposed to normal world.
Denial of Service	The management mode only has limited resource (such as memory). The attacker may send command to management mode code to make it run out of resource.
Elevation of Privilege	The attacker may gain unauthorized execution rights in management mode. For example, if the management code calls the normal world code, the attacker may replace the original code with malicious code to gain the privilege. The attacker may construct a confused-deputy attack for management mode. For example, the OS kernel may send a command to management mode to let it modify the hypervisor memory or management mode memory.

Adversary	Example
Network Attacker	N/A
Unprivileged Software Attacker	N/A
System Software Attacker	The attacker may take advantage of an implementation flaw in the management mode code to read or modify the management mode content, or content of a higher privilege environment, such as a hypervisor. The attacker may use a side channel to steal a secret in the management mode memory.
Simple Hardware Attacker	N/A
Skilled Hardware Attacker	N/A

Mitigation	Example
Protection	The management mode code must lock the management mode after it is constructed, no later than 3rd part code running. The management mode code must not call out to the normal world code. The system must remove unnecessary management mode handlers. The required management mode handler must check the untrusted external input, including the communication buffer, the pointer inside of the communication buffer, the general purpose register served as communication buffer pointer, the hardware base address register. The checked content must be copied into management mode memory to prevent TOC/TOU. The management mode handler must prevent unauthorized access to itself and high privileged content such as hypervisor or OS kernel memory. The management mode handler must prevent side channel attacks for the secret. The management mode handler must not allocate more resources to serve the request. If additional sources are allocated, they must be freed before the handler return to the normal world.
Detection	N/A
Recovery	N/A

Asset: Build Tool

In 1983, Ken Thompson received the Turing Award with Dennis Ritchie. There he delivered a speech - Reflections on Trusting Trust, and demonstrated how to inject a Trojan Horse into the compiler. Afterward the compiler generated a buggy binary. It is not impossible.

This is not a traditional attack to the final system, but it represents an attack to the tool chain in the build environment.

The mitigation is: only trust the tool chain from a trusted source with the source code, and protect the tool chain in the build environment.

REFERENCES

Books and Papers

[McConnell] *Code Complete: A Practical Handbook of Software Construction, Second Edition*, Steve McConnell, Microsoft, 2004, ISBN: 978-0735619678

[Maguire] *Writing Solid Code: Microsoft's Techniques for Developing Bug-Free C Programs*, Steve Maguire, Microsoft, 1993, ISBN: 978-1556155512

[HowardLeBlanc] *Writing Secure Code: Practical Strategies and Proven Techniques for Building Secure Applications in a Networked World, Second Edition*, Michael Howard, David LeBlanc, Microsoft, 2004, ISBN: 978-0735617223

[Howard] *24 Deadly Sins of Software Security: Programming Flaws and How to Fix Them*, Michael Howard, David LeBlanc, John Viega, McGraw-Hill, 2009, ISBN: 978-0071626750

[Graff] *Secure Coding: Principles&Practices*, M.G. Graff and K.R. van Wyk, O'Reilly, 2002, ISBN: 978-0596002428

[Ransome] *Core Software Security: Security at the Source*, James Ransome and Anmol Misra, CRC Press, 2014, ISBN: 978-1466560956. Particularly, chapters 5 and 9.

[Viega] *Secure Programming Cookbook for C and C++: Recipes for Cryptography, Authentication, Input Validation & More*. John Viega, Matt Messier, O'Reilly Media, 2003, ISBN: 978-0596003944

[ViegaMcGraw] *Building Secure Software: How to Avoid Security Problems the Right Way*, John Viega, Gary McGraw, Addison-Wesley Professional, 2001, ISBN: 978-0201721522

[Teer] *Solaris Systems Programming*, Chapter 9, Secure C Programming, Rich Teer, Prentice Hall, 2007, ISBN: 978-0768682236

[MITRE] System Engineering Guide, MITRE, Page 192, Security Code Review

Web

[Android] "Android Secure Coding Standard", https://wiki.sei.cmu.edu/confluence/display/android/Android+Secure+Coding+Standard

[Apple] “Secure Coding Guide”, https://developer.apple.com/library/mac/documentation/Security/Conceptual/SecureCodingGuide/Introduction.html

[Banned Function] Microsoft Security Development Lifecycle (SDL) Banned Function Calls, https://msdn.microsoft.com/en-us/library/bb288454.aspx

[Jordan] “Ten dos and don’ts for secure coding”, Michael Jordan, https://searchsecurity.techtarget.com/tip/Ten-dos-and-donts-for-secure-coding

[MDS] Deep Dive: Intel Analysis of Microarchitectural Data Sampling https://software.intel.com/security-software-guidance/insights/deep-dive-intel-analysis-microarchitectural-data-sampling

[Microsoft] “Security in Software Localization”, Mohamed Elgazzar, https://docs.microsoft.com/en-us/globalization/design/security-guidelines

[MicrosoftSDL] “What are the Microsoft SDL practices?”, https://www.microsoft.com/en-us/securityengineering/sdl/practices

[Msdn] “Guidelines for Writing Secure Code”, https://docs.microsoft.com/en-us/previous-versions/visualstudio/visual-studio-2010/ms182020(v=vs.100)

[Michael] “Defend Your Code with Top Ten Security Tips Every Developer Must Know”, Howard, Michael and Brown, Keith, https://blogs.msdn.microsoft.com/laurasa/2012/07/25/defend-your-code-with-top-ten-security-tips-every-developer-must-know/

[Mozilla] “Secure Development Guidelines”, https://developer.mozilla.org/en-US/docs/Mozilla/Security/Secure_Development_Guidelines

[Linux] “Secure Programming for Linux and Unix HOWTO, Background, Sources of Design and Implementation Guidelines”, http://www.linux-tutorial.info/modules.php?name=Howto&pagename=Secure-Programs-HOWTO/sources-of-guidelines.html

[OWASP] OWASP Secure Coding Practices, https://www.owasp.org/index.php/OWASP*Secure_Coding_Practices*-_Quick_Reference_Guide

[RedHat] “Secure Coding”, https://developers.redhat.com/topics/secure-coding/

[SEI] “SEI CERT C Coding Standard”, https://wiki.sei.cmu.edu/confluence/display/c/SEI+CERT+C+Coding+Standard

[SideChannel] Host Firmware Speculative Execution Side Channel Mitigation, https://software.intel.com/security-software-guidance/insights/host-firmware-speculative-execution-side-channel-mitigation

[SideChannel2] Deep Dive: Analyzing Potential Bounds Check Bypass Vulnerabilities, https://software.intel.com/security-software-guidance/insights/deep-dive-analyzing-potential-bounds-check-bypass-vulnerabilities

[SideChannel3] Security Best Practices for Side Channel Resistance, https://software.intel.com/security-software-guidance/insights/security-best-practices-side-channel-resistance

[SideChannel4] Guidelines for Mitigating Timing Side Channels Against Cryptographic Implementations, https://software.intel.com/security-software-guidance/insights/guidelines-mitigating-timing-side-channels-against-cryptographic-implementations

[Wheeler] “Secure Programming for Linux and Unix HOWTO -- Creating Secure Software”, David Wheeler, http://www.dwheeler.com/secure-programs/

[Witteman] “Secure Application Programming in the presence of Side Channel Attack”, Marc Witteman, https://www.riscure.com/uploads/2018/11/201708_Riscure_Whitepaper_Side_Channel_Patterns.pdf

Firmware Specific

[CapsuleRecovery] Yao, Zimmer, A Tour Beyond BIOS- Capsule Update and Recovery in EDK II, https://github.com/tianocore-docs/Docs/raw/master/White_Papers/A_Tour_Beyond_BIOS_Capsule_Update_and_Recovery_in_EDK_II.pdf

[CET] Control Flow Enforcement Technology, https://software.intel.com/sites/default/files/managed/4d/2a/control-flow-enforcement-technology-preview.pdf

[CET EDK II] CET in SMM https://github.com/tianocore/tianocore.github.io/wiki/CET-in-SMM

[HSTI] Hardware Security Testability Specification https://msdn.microsoft.com/en-us/library/windows/hardware/mt712332.aspx

[IOMMU EDKII] Yao, Zimmer, A Tour Beyond BIOS Using IOMMU for DMA Protection,

https://firmware.intel.com/sites/default/files/Intel_WhitePaper_Using_IOMMU_for_DMA_Protection_in_UEFI.pdf

[MemoryMap] Yao, Zimmer, A Tour Beyond BIOS Memory Map And Practices in UEFI BIOS, https://github.com/tianocoredocs/Docs/raw/master/White_Papers/A_Tour_Beyond_BIOS_Memory_Map_And_Practices_in_UEFI_BIOS_V2.pdf

[MemoryProtection] Yao, Zimmer, A Tour Beyond BIOS- Memory Protection in UEFI BIOS, https://www.gitbook.com/book/edk2-docs/a-tour-beyond-bios-memory-protection-in-uefi-bios/details

[MOR] TCG Platform Reset Attack Mitigation Specification, https://www.trustedcomputinggroup.org/wp-content/uploads/Platform-Reset-Attack-Mitigation-Specification.pdf

[Profile] Yao, Zimmer, Zeng, Fan, A Tour Beyond BIOS Implementing Profiling in UEFI, https://github.com/tianocore-docs/Docs/raw/master/White_Papers/A_Tour_Beyond_BIOS_Implementing_Profiling_in_EDK_II.pdf

[S3Resume] Jiewen Yao, Vincent Zimmer, A Tour Beyond BIOS Implementing S3 Resume with EDK II, https://github.com/tianocore-docs/Docs/raw/master/White_Papers/A_Tour_Beyond_BIOS_Implementing_S3_resume_with_EDKII_V2.pdf

[SecurityEnhancement] Yao, Zimmer, A Tour Beyond BIOS Securiy Enhancement to Mitigate Buffer Overflow in UEFI, https://www.gitbook.com/book/edk2-docs/a-tour-beyond-bios-mitigate-buffer-overflow-in-ue/details

[SecurityDesign] Yao, Zimmer, A Tour Beyond BIOS Security Design Guide in EDK II, https://github.com/tianocore-docs/Docs/raw/master/White_Papers/A_Tour_Beyond_BIOS_Security_Design_Guide_in_EDK_II.pdf

[SecureMOR] Secure MOR implementation, https://docs.microsoft.com/en-us/windows-hardware/drivers/bringup/device-guard-requirements

[SmmComm] Yao, Zimmer, Zeng, A tour beyond BIOS secure SMM communciation, https://github.com/tianocore-docs/Docs/raw/master/White_Papers/A_Tour_Beyond_BIOS_Secure_SMM_Communication.pdf

[SMMProtection] Yao, SMM Protection in EDKII. http://www.uefi.org/sites/default/files/resources/Jiewen%20Yao%20-%20SMM%20Protection%20in%20%20EDKII_Intel.pdf

[TCG OPAL] Storage Work Group Storage Security Subsystem Class: Opal, Version 2.01 Final, Revision 1.00, https://trustedcomputinggroup.org/wp-content/uploads/TCG_Storage-Opal_SSC_v2.01_rev1.00.pdf

[TCG SIIS] TCG Storage Interface Interactions Specification, Version 1.06, Revision 1.08, https://www.trustedcomputinggroup.org/wp-content/uploads/TCG_SWG_SIIS_Version_1_06_Revision_1_08_public-review.pdf

[TPM2] Trusted Platform Module Library Specification, Family “2.0”, Level 00, Revision 01.38 – September 2016, https://trustedcomputinggroup.org/tpm-library-specification/

[TPM2 PFP] PC Client Specific Platform Firmware Profile Specification Family “2.0”, Level 00 Revision 1.03 Version 51, https://trustedcomputinggroup.org/wp-content/uploads/PC-ClientSpecific_Platform_Profile_for_TPM_2p0_Systems_v51.pdf

[TPM2 EDK II] Yao, Zimmer, A Tour Beyond BIOS with the UEFI TPM2 Support in EDKII https://firmware.intel.com/sites/default/files/resources/A_Tour_Beyond_BIOS_Implementing_TPM2_Support_in_EDKII.pdf

[WSMT] Windows SMM Security Table, https://msdn.microsoft.com/en-us/library/windows/hardware/dn495660(v=vs.85).aspx#wsmt.aspx)

http://download.microsoft.com/download/1/8/A/18A21244-EB67-4538-BAA2-1A54E0E490B6/WSMT.docx

[Variable] Yao, Zimmer, Zeng, A Tour Beyond BIOS Implementing UEFI Authenticated Variables in SMM with EDKII – Verion 2, https://github.com/tianocore-docs/Docs/raw/master/White_Papers/A_Tour_Beyond_BIOS_Implementing_UEFI_Authenticated_Variables_in_SMM_with_EDKII_V2.pdf