

UEFI & EDK II Training

UEFI Aware Operating System

tianocore.org



LESSON OBJECTIVE



Explain How the OS and UEFI Work together



Explain the UEFI Requirements for UEFI aware OS



Explain How Secure Boot Fits with UEFI



What about coreboot?



UEFI AWARE OS REQUIREMENTS

Common Requirements

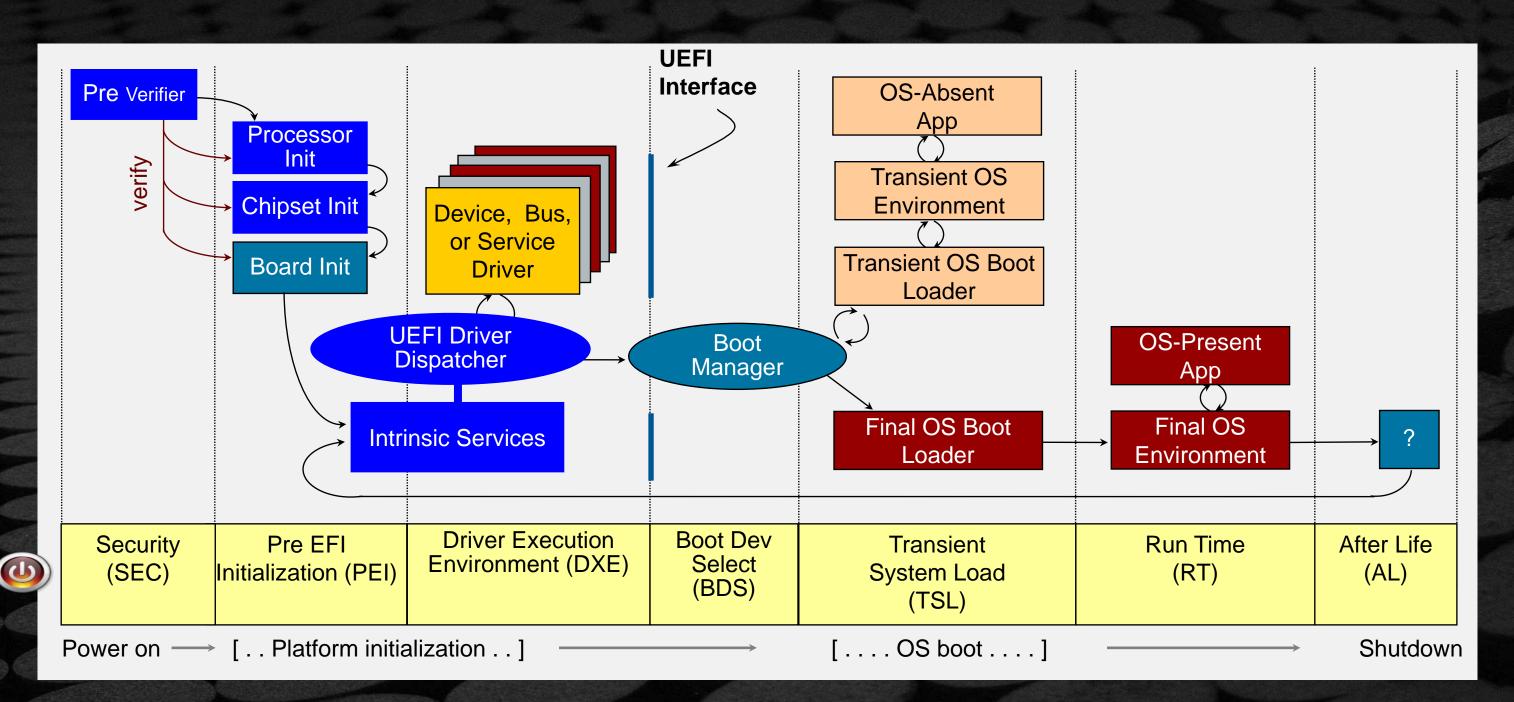


UEFI OPERATING SYSTEMS

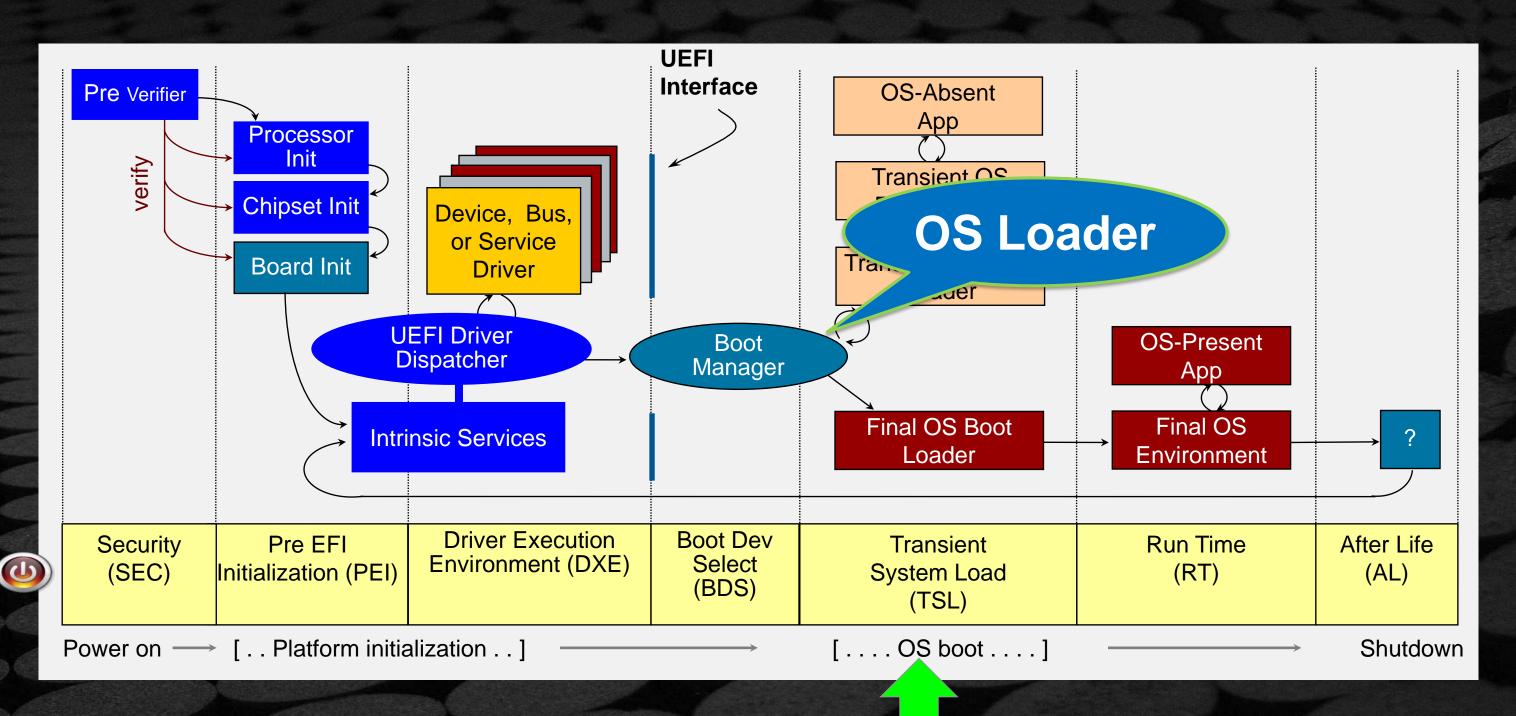




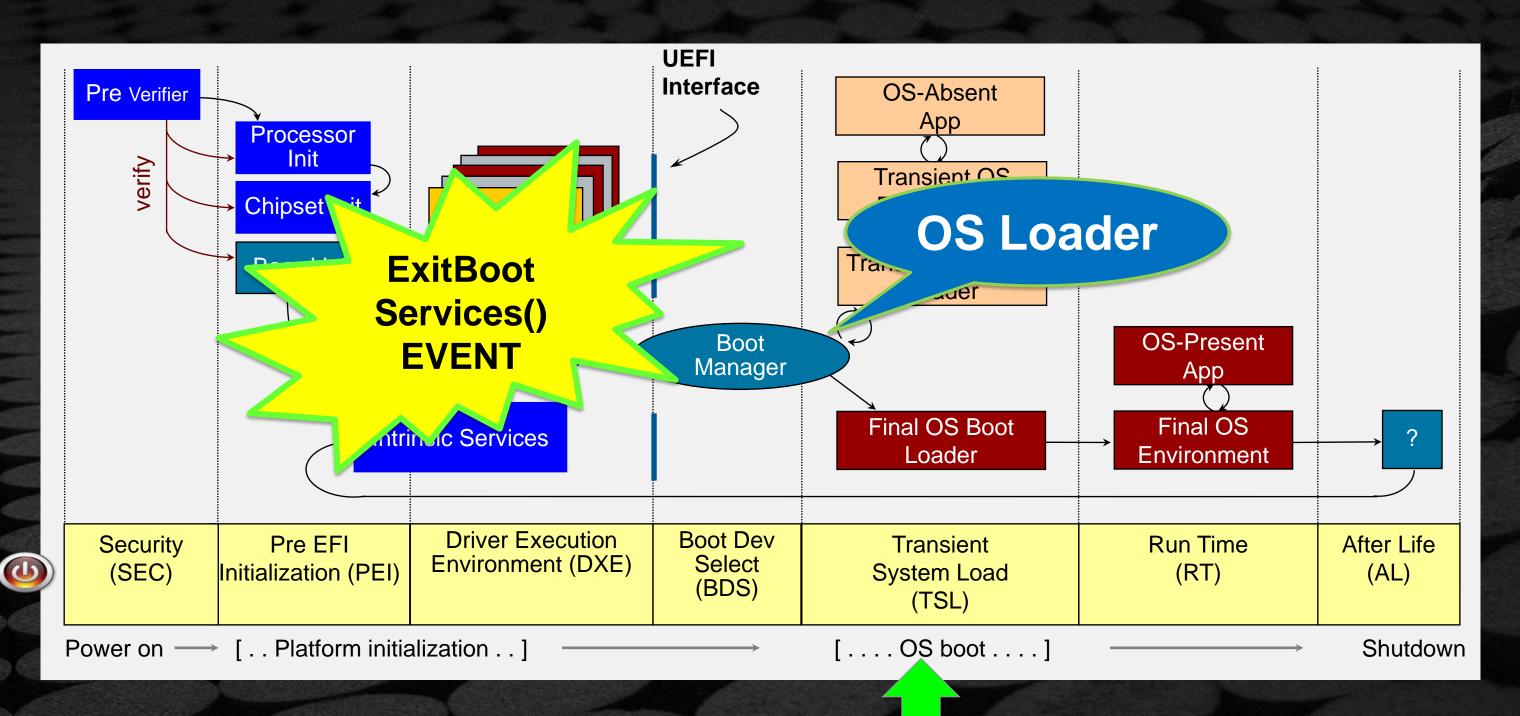




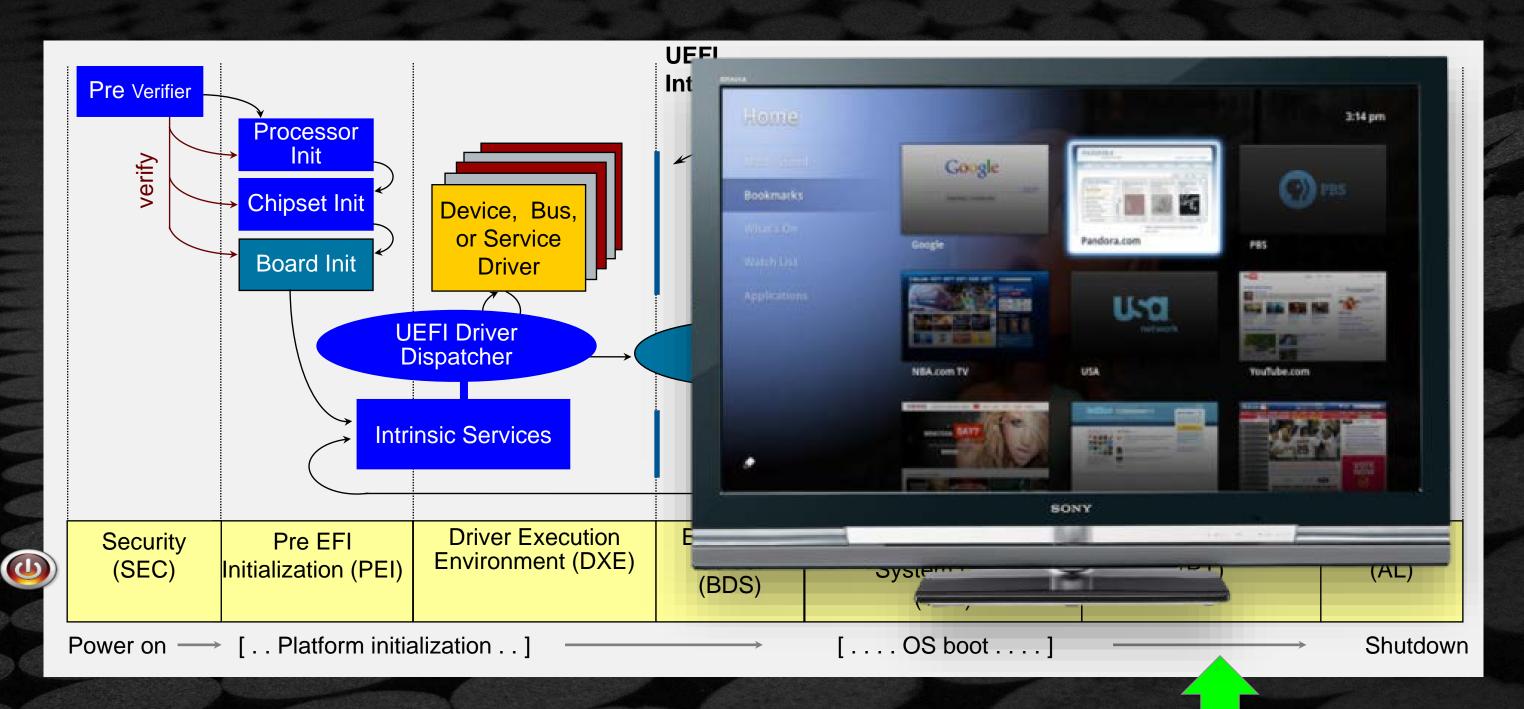














UEFI OS REQUIREMENTS

UEFI Drivers:
Boot devices/console

UEFI OS installer

UEFI OS Loader

Disk Partition/Formats

Firmware Requirements

Set Boot Path to Boot to UEFI OS



UEFI System Classes (based on firmware interfaces)

UEFI Class 0

- Boots Legacy int 19 ONLY
- Legacy BIOS Only (16 bit)
- No UEFI or UEFI PI Interfaces

UEFI Class 2

- Boots Legacy int 19 or UEFI
- Uses UEFI / PI Interfaces
- Legacy BIOS runtime Interfaces w/ CSM

UEFI Class 1

- Boots Legacy int 19 ONLY
- Uses UEFI / PI Interfaces
- Only legacy BIOS runtime Interfaces

Limited Benefits

- ✓ OEMs / ODMs Internal
- ✓ Double code development
- ✓ Compromised security MBR exposure



UEFI System Classes (based on firmware interfaces)

UEFI Class 0

- Boots Legacy int 19 ONLY
- Legacy BIOS Only (16 bit)
- No UEFI or UEFI PI Interfaces

UEFI Class 1

- Boots Legacy int 19 ONLY
- Uses UEFI / PI Interfaces
- Only legacy BIOS runtime Interfaces

UEFI Class 2

- Boots Legacy int 19 or UEFI
- Uses UEFI / PI Interfaces
- Legacy BIOS runtime Interfaces w/ CSM

UEFI Class 3

- Boots **ONLY** UEFI
- Uses UEFI / PI Interfaces
- Runtime exposes only UEFI interfaces



UEFI System Classes (based on firmware interfaces)

Full Benefits

- ✓ UEFI Innovation
- ✓ Smaller code size/ Validation
- ✓ Extensibility

Only Class after 2020

Enabling Secure Boot creates another Class

UEFI Class 3 +

- Boots ONLY UEFI
- Uses UEFI / PI Interfaces
- Runtime exposes only UEFI interfaces

UEFI Secure Boot "ON"



Required UEFI Drivers: OS Install & Boot

Boot Device

Console Output

Console Input

NVRAM Driver

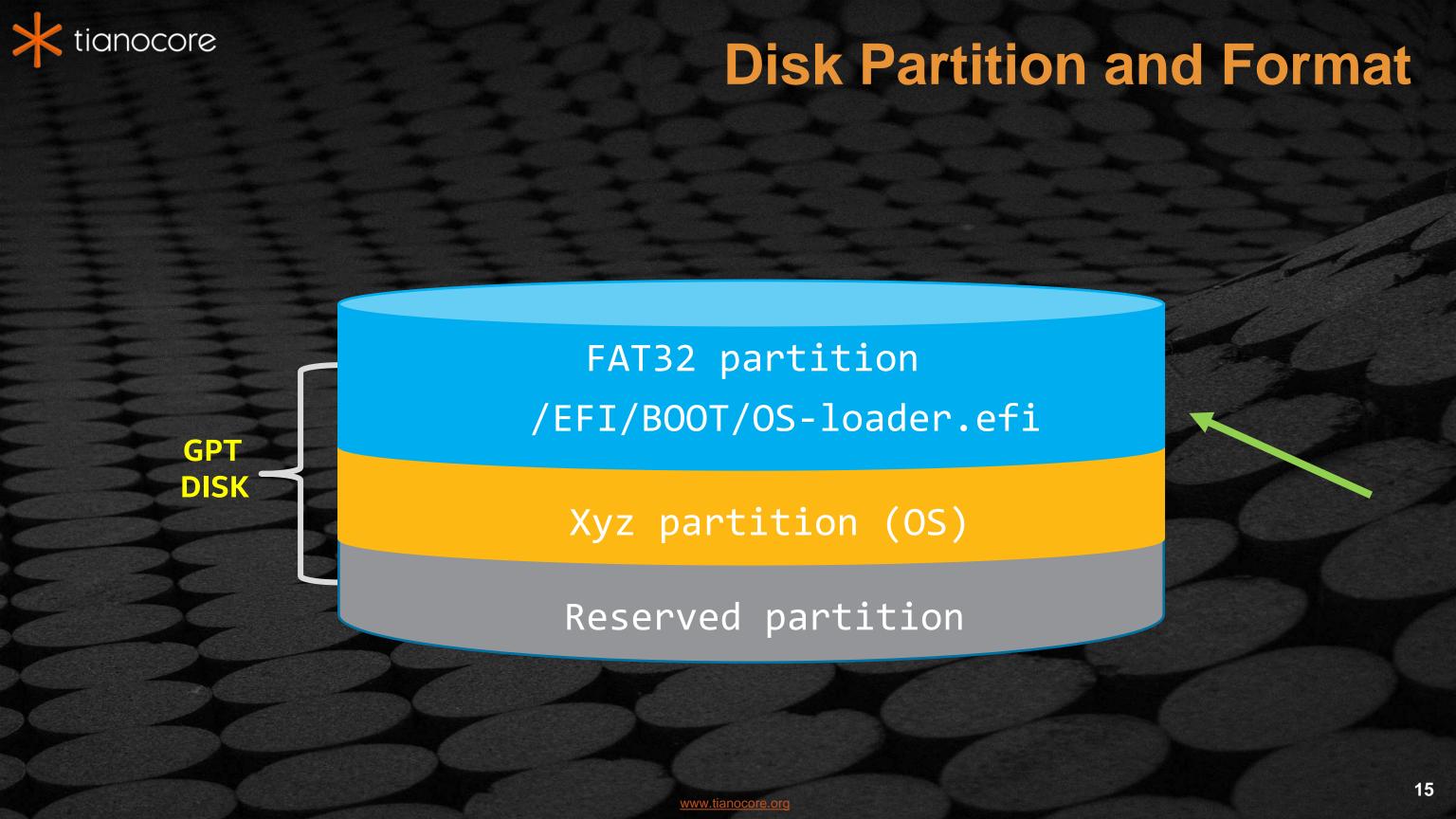


UEFI OS LOADER

- OS install process includes UEFI loader
 - /efi/boot/bootx64.efi /efi/redhat/grub.efi
- Call UEFI boot & runtime services to start OS
- Exit UEFI Boot Services
- Transfer control to native OS

UEFI OS INSTALLER

- Discover UEFI storage devices
- Setup storage device: GPT w/ FAT32 boot partition
- Create boot variable BootXXXX





INTERFACE INSIDE OS RUNTIME

UEFI Runtime Services



Runtime Services Available to the UEFI Aware OS

Active Consoles
Input Console
Output Console

UEFI Boot Services Table ask Priority Level Services

Event and Timer Services
Protocol Handler Services
Image Services

Driver Support Services

DXE Services Table

Global Coherency Domain Services
Dispatcher Services

Handle Database

Protocol Interface

Boot Service Data Structures

EFI System Table **UEFI Runtime Services Table**

Variable Services
Real Time Clock Services

Reset Services

Status Code Services¹

Virtual Memory Services

Version Information
UEFI Specification Version
Firmware Vendor
Firmware Revision

System Configuration Table
DXE Services Table

HOB List
ACPI Table

SMBIOS Table

•••

SAL System Table

Runtime Data Structures





Accessing RT services from Windows API

- GetFirmwareEnvironmentVariable: MSDN Link
- SetFirmwareEnvironmentVariable: MSDN Link
- Example: (determine if UEFI or Legacy BIOS)

18





ACCESSING RT SERVICES FROM LINUX OS

Firmware Test Suite, it includes a Linux kernel driver to help with it's interactions with UEFI. Note that this is a Linux-centric test suite, solution won't work for other OSes.

- http://kernel.ubuntu.com/git/hwe/fwts.git
- https://bugs.launchpad.net/ubuntu/+source/linux/+bug/1633506
- https://patchwork.kernel.org/patch/9323781/
- http://www.basicinputoutput.com/2016/03/introduction-to-firmware-test-suitefwts.html



SECURITY WITH UEFIHow does UEFI ensure the Operating System is trusted?



BOOT SECURITY TECHNOLOGIES

Hardware Root of Trust Boot Guard, Intel® TXT

Measured Boot

Using TPM¹ to store hash values

Verified Boot

Boot Guard +

UEFI Secure Boot

¹TPM – Trusted Platform Module

Resources: https://firmwaresecurity.com/2015/07/29/survey-of-boot-security-technologies/



HARDWARE ROOT OF TRUST

Boot Guard

CPU verifies signature
Verification occurs before system
FW starts
Hash of public key is fused in CPU

Intel® TXT

Uses a Trusted Platform Module (TPM) & cryptographic Provides Measurements

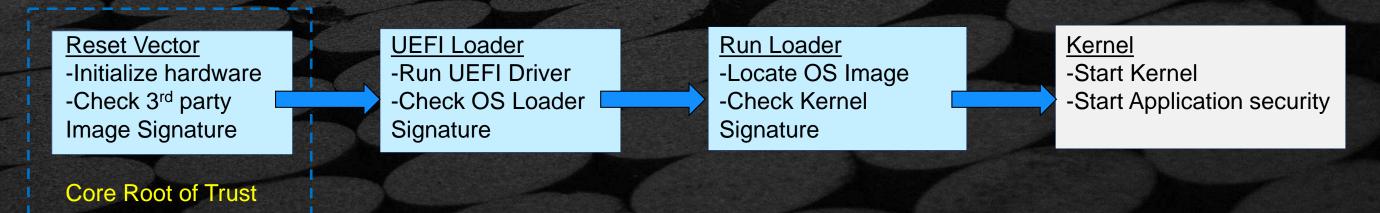
Verification

Measurements



Software ID checking during every step of the boot flow:

1. UEFI System FW (updated via secure process)





Software ID checking during every step of the boot flow:

- 1. UEFI System FW (updated via secure process)
- 2. Add-In Cards (signed UEFI Option ROMs)

Reset Vector UEFI Loader Run Loader Kernel -Run UEFI Driver -Start Kernel -Initialize hardware -Locate OS Image -Check 3rd party -Check OS Loader -Check Kernel Start Application security **Image Signature** Signature Signature Core Root of Trust



Software ID checking during every step of the boot flow:

- 1. UEFI System FW (updated via secure process)
- 2. Add-In Cards (signed UEFI Option ROMs)

FW check 3rd party Signature before invoke

Reset Vector

- -Initialize hardware
- -Check 3rd party Image Signature

Core Root of Trust

Verified

UEFI Loader

- -Run UEFI Driver
- -Check OS Loader Signature

Run Loader

- -Locate OS Image
- -Check Kernel Signature

<u>Kernel</u>

- -Start Kernel
- -Start Application security



Software ID checking during every step of the boot flow:

- 1. UEFI System FW (updated via secure process)
- 2. Add-In Cards (signed UEFI Option ROMs)

FW check 3rd party Signature before invoke

Verified

FW checks Loader

Signature before hand off

UEFI Loader

-Run UEFI Driver

-Check OS Loader Signature

Run Loader

-Locate OS Image

-Check Kernel Signature

Kernel

-Start Kernel

-Start Application security

Reset Vector

-Initialize hardware -Check 3rd party Image Signature

Core Root of Trust



Software ID checking during every step of the boot flow:

- 1. UEFI System FW (updated via secure process)
- 2. Add-In Cards (signed UEFI Option ROMs)
- 3. OS Boot Loader (checks for "secure mode" at boot)

FW check 3rd party Signature before invoke FW checks Loader Signature before hand off

Reset Vector

- -Initialize hardware
- -Check 3rd party Image Signature

Core Root of Trust

Verified

UEFI Loader

- -Run UEFI Driver
- -Check OS Loader Signature

Verifie

Run Loader

- -Locate OS Image
- -Check Kernel Signature

<u>Kernel</u>

- -Start Kernel
- -Start Application security



Software ID checking during every step of the boot flow:

- 1. UEFI System FW (updated via secure process)
- 2. Add-In Cards (signed UEFI Option ROMs)
- 3. OS Boot Loader (checks for "secure mode" at boot)

FW check 3rd party Signature before invoke

Verified

UEFI Loader

-Run UEFI Driver

FW checks Loader

Signature before hand off

-Check OS Loader Signature Loader checks Kernel Signature before handoff

Verified

Run Loader

-Locate OS Image

-Check Kernel Signature Verified

<u>Kernel</u>

-Start Kernel

-Start Application security

Reset Vector

-Initialize hardware

-Check 3rd party Image Signature

Core Root of Trust



AUTHENTICATED VARIABLES

PK **KEK** DB **DBX**

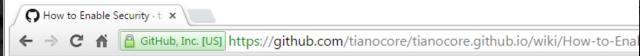
SetupMode

SecureBoot



Security Package Project Page Wiki Link

- Wiki Link: <u>How-to-Enable-Security</u>
- PDF: How to Sign UEFI ImagesV1.31
- Build command line switch SECURE_BOOT_ENABLE = TRUE
- Install the OpenssILib CryptoPkg:
 OpenSSL-Howto.txt



HOW TO ENABLE SECURE BOOT SERVICE

Download PDF with examples

How to Sign UEFI Drivers & Applications .pdf NOTE SVNs: For Nt32Pkg requires - For OVMF Requires -r13160 beyond UDK2010 version

How to add Secure Boot to DSC and FDF

Based on original variable driver in MdeModulePkg, variable driver in SecurityPkg provides authenticated variable service in UEFI 2.3.1 spec. Runtime crypto library, OpenSSL* library and variable driver are required to enable this feature.

- 1. Ensure OpensslLib* library instance is defined in [LibraryClasses] section of the platform DSC file:
- IntrinsicLib|CryptoPkg/Library/IntrinsicLib/IntrinsicLib.inf
- OpensslLib|CryptoPkg/Library/OpensslLib/OpensslLib.inf
- 2. Ensure BaseCryptLib library instances are defined in the platform DSC file:
 - For PEI driver: BaseCryptLib|CryptoPkg/Library/BaseCryptLib/PeiCryptLib.inf



Windows Secure Boot Key Creation and Management Guidance

- Windows <u>Secure Boot Key Creation &</u>
 <u>Management Guide</u>
- Creation and management of the Secure Boot keys and certificates in a manufacturing environment.
- Addresses questions related to creation, storage and retrieval of Platform Keys (PKs), secure firmware update keys, and third party Key Exchange Keys (KEKs).



Last Updated: 10/14/2016

Vishal Manan, Architect, OEM Consulting, vmanan@microsoft.com

Arie van der Hoeven, Architect, OEM Consulting, ariev@microsoft.com

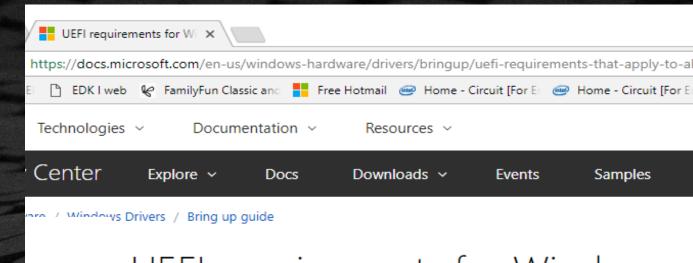
This document helps guide OEMs and ODMs in creation and management of the Sec certificates in a manufacturing environment. It addresses questions related to creation of Platform Keys (PKs), secure firmware update keys, and third party Key Exchange Ke

31



Many Platforms are Requiring UEFI Secure Boot Enabled

- Secure Boot now mandated for specific platforms
- See "Security requirements" on UEFI requirements for Windows editions on SoC Platforms



UEFI requirements for Windows editions on SoC platforms

団 04/20/2017 • © 14 minutes to read • Contributors ■ 🚭

This topic describes UEFI requirements that apply to Windows 10 for desktop editions (Home, Pro, Enterprise, and Education) and Windows 10 Mobile. For additional requirements that apply only to Windows 10 Mobile, see <u>UEFI</u> requirements for Windows 10 Mobile.

Summary of requirements

The following table lists all current requirements for UEFI compliance as defined in the UEFI specification (Section 2.6 of the UEFI 2.3.1 specification). In

nents for



COREBOOT

How does coreboot work with UEFI



Intel® FSP "Produced" to "Consuming" Intel® Architecture Firmware

ME

Green "H" w/ EDK2 Intel® FSP OS Glue Code (PEI Core / Arch PPIs) **UEFI** Specification Single Si Init Binary CPU SA **Platform Drivers** Scope UEFI/PI Intel® FSP Hardware/Silicon

OS

Payload

coreboot ramstage

Intel® FSP

coreboot romstage

Hardware/Silicon

Intel FSP is independent of the bootloader solutions

34



Consumers: EDK II firmware and coreboot

Functionality	coreboot	UEFI / PI
The reset vector and pre cache-as-ram setup	bootblock	Security Phase (SEC)
Cache as Ram setup, early silicon initialization, memory setup. Covered largely by Intel® Firmware Support Package	romstage	Pre-EFI Initialization (PEI) Create HOBs
Normal device setup and mainboard configuration. Publish SMBIOS/ACPI Tables	ramstage	Early Driver Execution Environment (DXE)
Memory map hand-off	CBMEM	UEFI Memory Map
The OS or application bootloader	payload	DXE BDS and UEFI Drivers

coreboot



SUMMARY



Explain How the OS and UEFI Work together



Explain the UEFI Requirements for UEFI aware OS



Explain How Secure Boot Fits with UEFI



What about coreboot?



Questions?





Return to Main Training Page



Return to Training Table of contents for next presentation link





Redistribution and use in source (original document form) and 'compiled' forms (converted to PDF, epub, HTML and other formats) with or without modification, are permitted provided that the following conditions are met:

Redistributions of source code (original document form) must retain the above copyright notice, this list of conditions and the following disclaimer as the first lines of this file unmodified.

Redistributions in compiled form (transformed to other DTDs, converted to PDF, epub, HTML and other formats) must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

THIS DOCUMENTATION IS PROVIDED BY TIANOCORE PROJECT "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL TIANOCORE PROJECT BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS DOCUMENTATION, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Copyright (c) 2020, Intel Corporation. All rights reserved.

40

Backup



UEFI Secure Boot

- Deficiency: Boot path malware targets
- UEFI and Secure Boot harden the boot process
- Firmware/software in the boot process must be signed by a trusted Certificate Authority (CA)
- Firmware image is hardware-protected
- 3rd party drivers signed using CA-holding trusted keys
- Trusted signing key's database factory-initialized and OS-updated



WHY??? SECURE BOOT WITH UEFI

Without

- Possible corrupted or destroyed data
- BootKit virus MBR Rootkits
- Network boot attacks e.g.
 PXESPOILT
- Code Injection Attacks



- Data integrity
- Trusted boot to OS
- Trusted drivers
- Trusted Applications



UEFI Secure Boot Flow PEI FV **T.** Enrol **Authenticated Variable** 2C. Signed OpRom.efi Image Load PK Certificate And + SignInfo measure KEK Into TPM 2A. Signed Image Certificate db Discover Certificate dbx OsLoader.efi Variable Certificate 2B. Signature + SignInfo DXE FV Verification' **Image Verify**