Title:	UEFI infrastructure for recovery
Revision:	-002
Date:	11/17/2016
Authors:	Vincent Zimmer

Summary:

This document describes the usage of UEFI for platform recovery, protection, detection, platform maintenance, and update.

Background:

History of in-band firmware/BIOS networking

What is PXE?

Platform firmware has had integrated network boot capabilities since the Pre-Boot Execution Environment (PXE) was defined in the mid 1990's as part of the Wired For Management (WFM) efforts. PXE can be thought of as 'in-band' networking since it runs on the main host CPU's, not an out-of-band chipset or platform microprocessor, or a 'non-host processor' (NHP). PXE entails two elements – (1) client side API's, including Base-Code and UNDI, for purposes of orchestrating the download of a network boot program (NBP) and (2) a wire protocol for interacting with the boot server. The PXE boot process is 'client initiated' in that the in-band firmware initiates a DHCP discovery process to start the networking interactions with a boot server.

Where was PXE defined?

PXE was originally part of the PC/AT BIOS and a specification jointly owned by a small consortium of companies. With the advent of the Extensible Firmware Interface (EFI) in the late 1990's, the 'base code' and 'UNDI' interfaces from BIOS PXE were mapped into EFI interfaces. This continued through the EFI 1.10 specification in 2001. The EFI1.10 specification was an Intel-owned document. In order to support broader industry adoption, EFI 1.10 was contributed to the Unified Extensible Firmware Interface (UEFI) forum in 2005, along with some post EFI1.10 networking API's. The latter included a modular IPV4 network stack that broke out IP, UDP, TCP, DHCP, ARP and other elements into separate API's, as opposed to the EFI1.10 reference implementation of the monolithic PXE stack. With this modular network stack in UEFI2.0 in 2006, the foundation was laid to create additional networking services on the UEFI platform. These services have included a refactored PXE Client that leverages the modular network stack and a ISCSI initiator, both in the open source.

Post-PXE?
The question was posed to the industry group in 2007 about how to evolve PXE. At the time, there were many extant scenarios built upon the IPV4 PXE wire protocol, including support in all of the Linux distributions and the Windows Deployment Services (WDS) feature in Microsoft Windows. The most important feature request entailed addition of IPV6 support. As such, the UEFI Forum worked w/ the

IETF and generated RFC 5970 that includes the option tags for IPV6 network boot. This RFC, along with a network interaction flow, form 'netboot6', or a variant of PXE that interoperates across IPV6.

The modular network stack and the IPV6 and IPV4 variants of PXE [PXE-SPEC] can be found in the EFI Developer Kit 2 (EDKII) project on Github in the Network Package (NetworkPkg) [EDK2].

Netboot6, along w/ UEFI Secure Boot [ITJ-SB], provided UEFI-only features not available on a PC/AT BIOS. These features, along w/ fast-boot, were integrated into Microsoft Windows8 and helped motivate the decision to mandate UEFI 2.3.1c specification conformance for this operating system in 2012. So the EFI effort that commenced in 1998 culminated in the 2012 launch of Windows8 that required UEFI.

This was the tipping point for the standards adoption and many of the preceding tactics, such as feature creation and open source, helped to motivate this decision.

Post-PXE:

Fast forward to 2016.

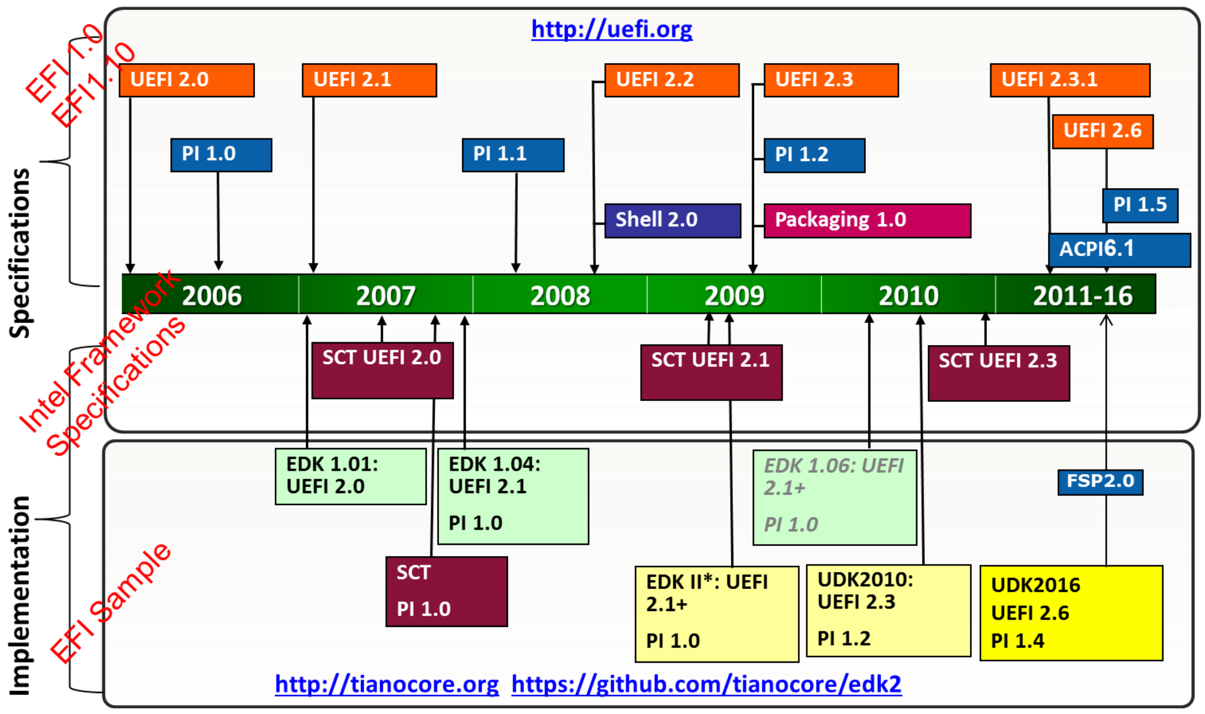
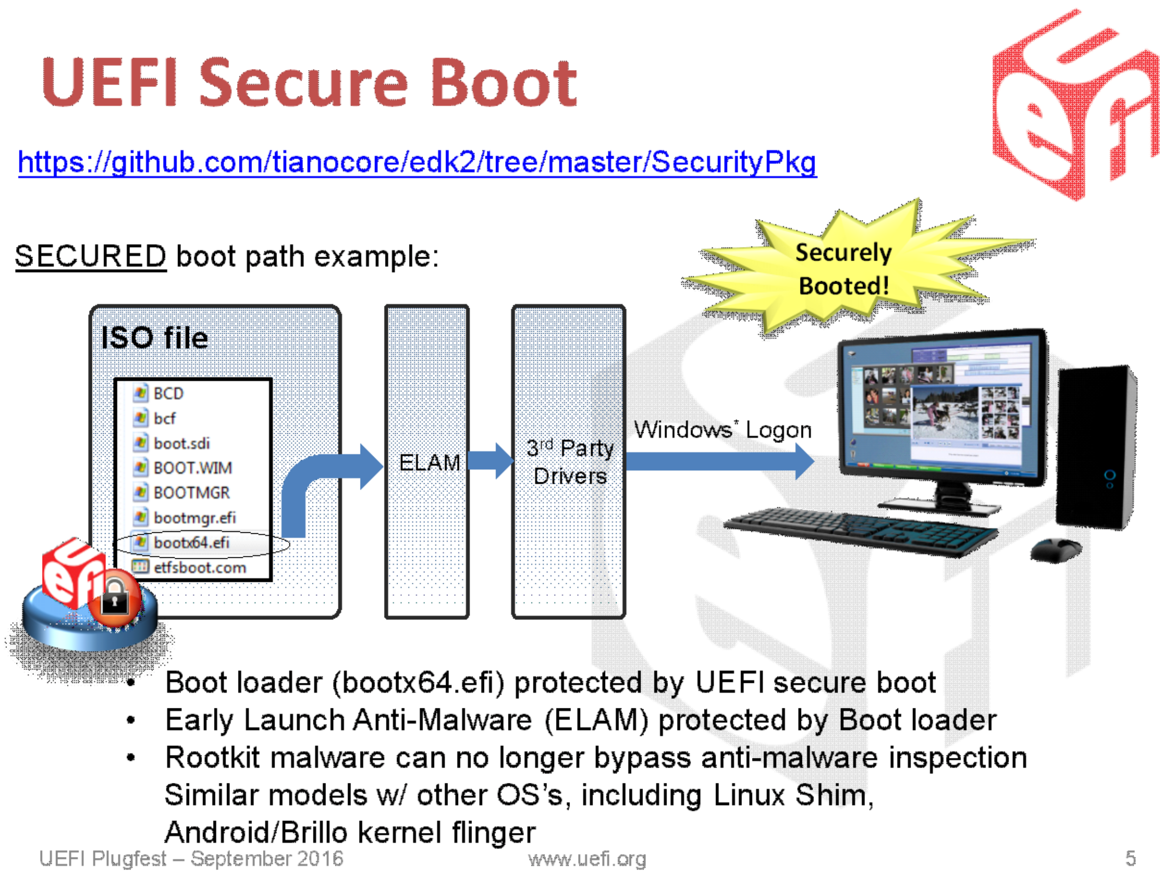


Figure 1 Specification and roadmap [UEFI-PF]

Since the UEFI 2.3.1 specification the UEFI Specification defined several new features, including OS recovery, HTTP, HTTP boot, TLS, WIFI, and the ESRT. The model described above shows how the specifications typically have a reference implementation. The project is called the “EFI Developer Kit II” and the UEFI Developer Kit (UDK) are validated branches of EDKII used for product development.

To begin, the capability builds upon today’s UEFI Secure Boot for ensuring the integrity of the network boot program, viz.,



UEFI Secure Boot

https://github.com/tianocore/edk2/tree/master/SecurityPkg

SECURED boot path example:

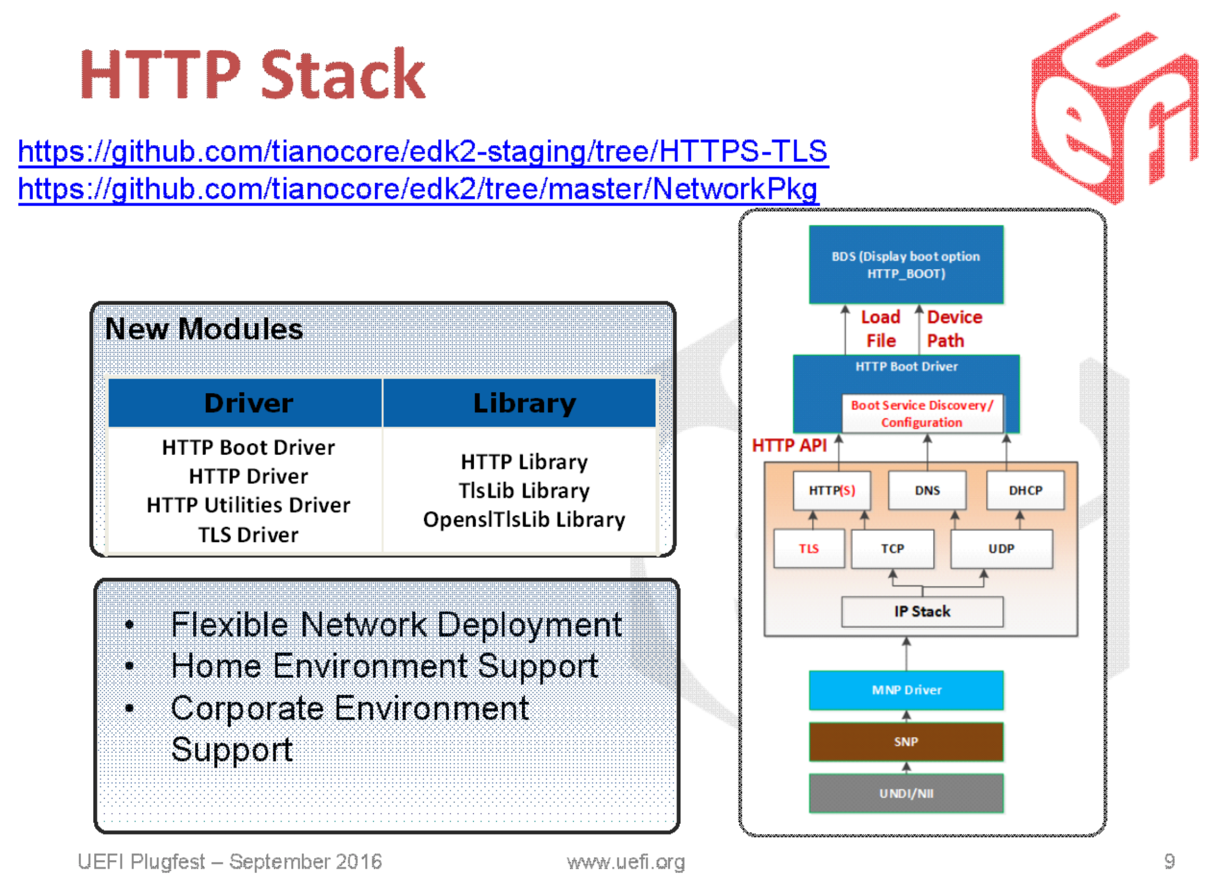
- Boot loader (bootx64.efi) protected by UEFI secure boot
- Early Launch Anti-Malware (ELAM) protected by Boot loader
- Rootkit malware can no longer bypass anti-malware inspection

Similar models w/ other OS’s, including Linux Shim, Android/Brillo kernel flinger

UEFI Plugfest – September 2016 www.uefi.org 5

Figure 2 Secure Boot [UEFI-PF]

This capability can be used for local or network boot, including the new HTTP boot.



HTTP Stack

https://github.com/tianocore/edk2-staging/tree/HTTPS-TLS
https://github.com/tianocore/edk2/tree/master/NetworkPkg

New Modules

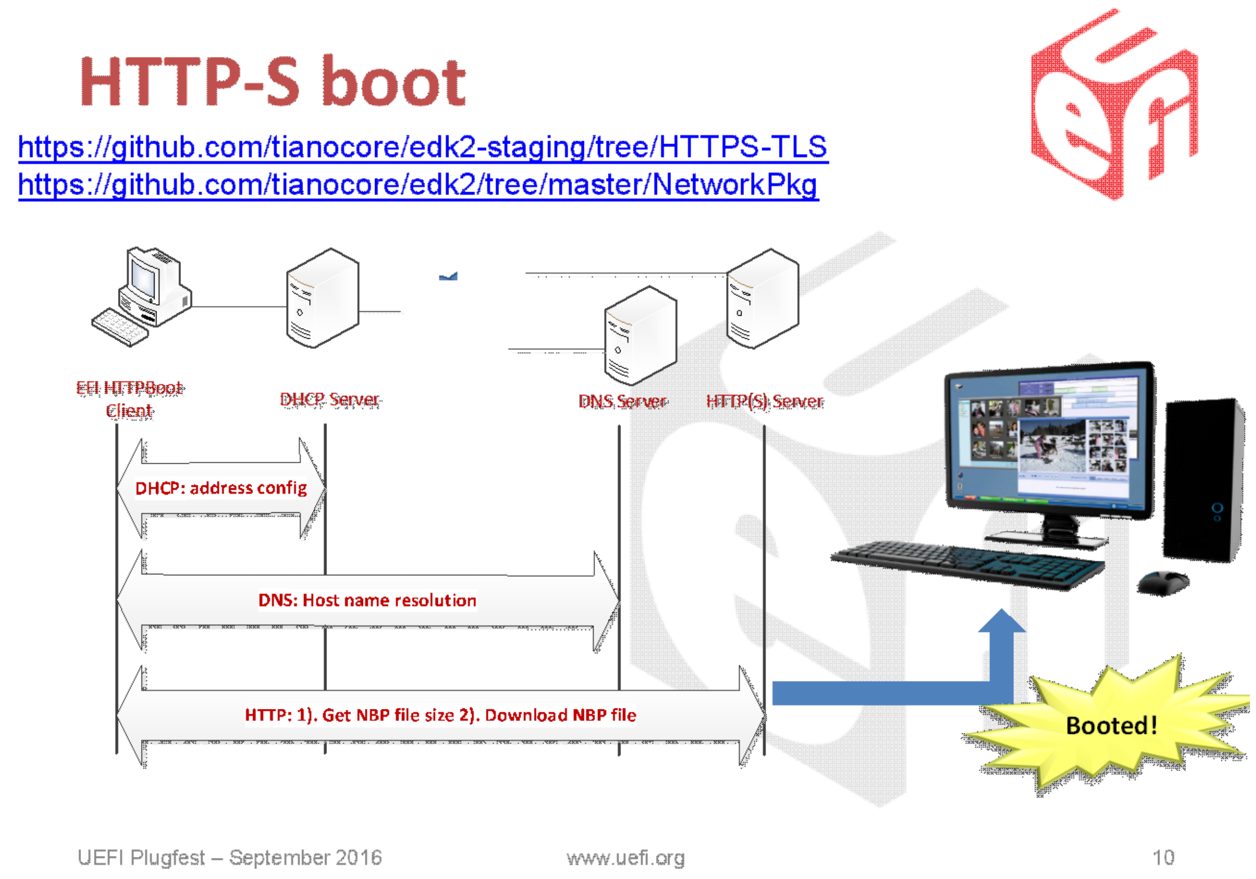
Driver	Library
HTTP Boot Driver HTTP Driver HTTP Utilities Driver TLS Driver	HTTP Library TlsLib Library OpenslTlsLib Library

- Flexible Network Deployment
- Home Environment Support
- Corporate Environment Support

UEFI Plugfest – September 2016 www.uefi.org 9

Figure 3 HTTP Stack [UEFI-PF]

Beyond HTTP boot, the UEFI specification has the ability to support TLS for HTTP-S, viz.,



HTTP-S boot

https://github.com/tianocore/edk2-staging/tree/HTTPS-TLS
https://github.com/tianocore/edk2/tree/master/NetworkPkg

UEFI Plugfest – September 2016 www.uefi.org 10

Figure 4 HTTP-S Boot [UEFI-PF]

The ability to perform a launch of a network application across the internet provides a substantial improvement in capabilities, but there is another technology trend to be observed. Specifically, the networking capability of a client has largely moved from wired to wireless. As such, relegating the pre-OS networking capability to a wired connection limits the applicability of the usage. To that end, the UEFI Forum has introduced Wi-Fi capability in UEFI 2.5 and refined the interfaces in the UEFI2.6. The updates in 2.6 were to decouple the details of the specific radio technology so that a generic wireless connection manager (WCM) could be paired with a hardware-specific Universal Network Device Interface (UNDI) driver, with a generic and/or hardware-enlightened Supplicant Driver to provide richer servers like EAP-TLS, TTLS, etc. The topology is shown below.

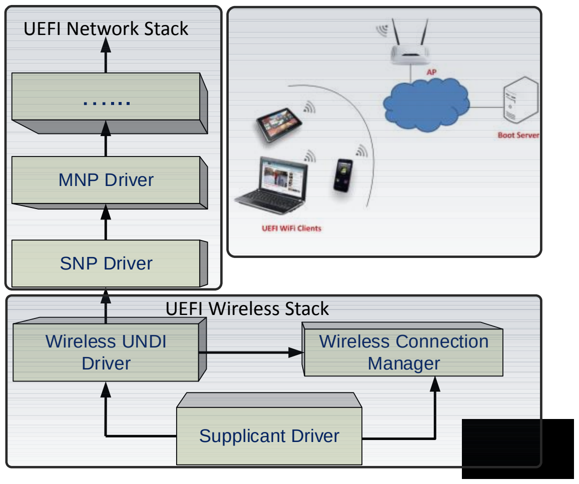
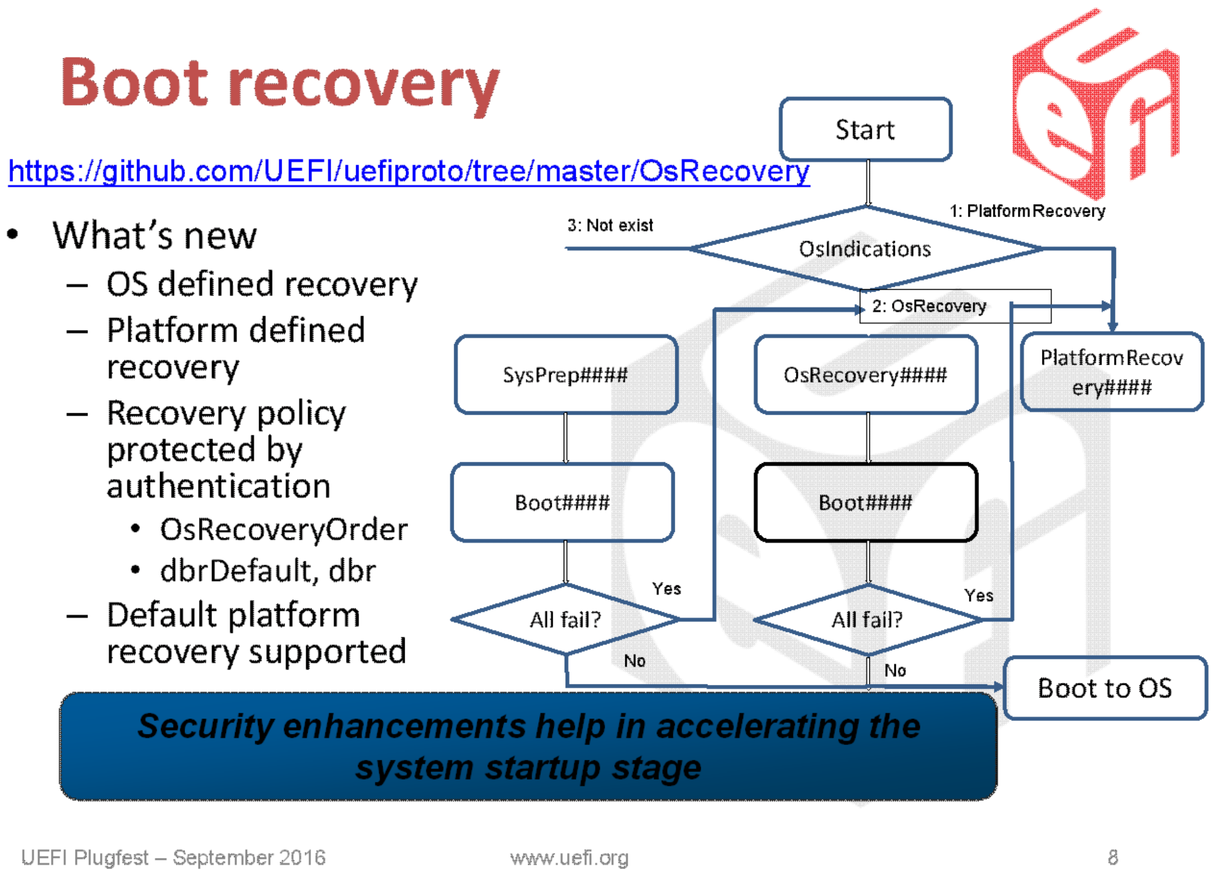


Figure 5 UEFI Wifi Stack

To conserve flash space, the Supplicant can re-use the same TLS services that the HTTP driver employs via the EFI_TLS_PROTOCOL. In the figure above the UNDI API is the same as for wired so that the upper layer drivers, including SNP, MNP, IP, TCP, DHCP,… can be re-used.

Finally, the UEFI 2.6 specification introduces platform recovery. This includes the flow below.


Start
OsIndications
1: PlatformRecovery
3: Not exist
2: OsRecovery
SysPrep####
OsRecovery####
PlatformRecovery####
Boot####
Boot####
All fail?
Yes
No
All fail?
Yes
No
Boot to OS

Boot recovery

https://github.com/UEFI/uefiproto/tree/master/OsRecovery

- What's new
  - OS defined recovery
  - Platform defined recovery
  - Recovery policy protected by authentication
    - OsRecoveryOrder
    - dbrDefault, dbr
  - Default platform recovery supported

Security enhancements help in accelerating the system startup stage

UEFI Plugfest – September 2016 www.uefi.org 8

Figure 6 OS Recovery [UEFI-PF]

OS recovery defines new OsRecovery#### boot options that can point to a recovery agent, or a .efi file, on the local disk or network. The OsRecovery#### boot options are stored via authenticated variables ([UEFI] 2.6 chapter 7 and [EDK2-VAR]) that must be signed, as opposed to the circa 1999 EFI Boot options Boot#### that are in world-readable/writable variables.

The overall flow of networking and the recovery boot can be found in below

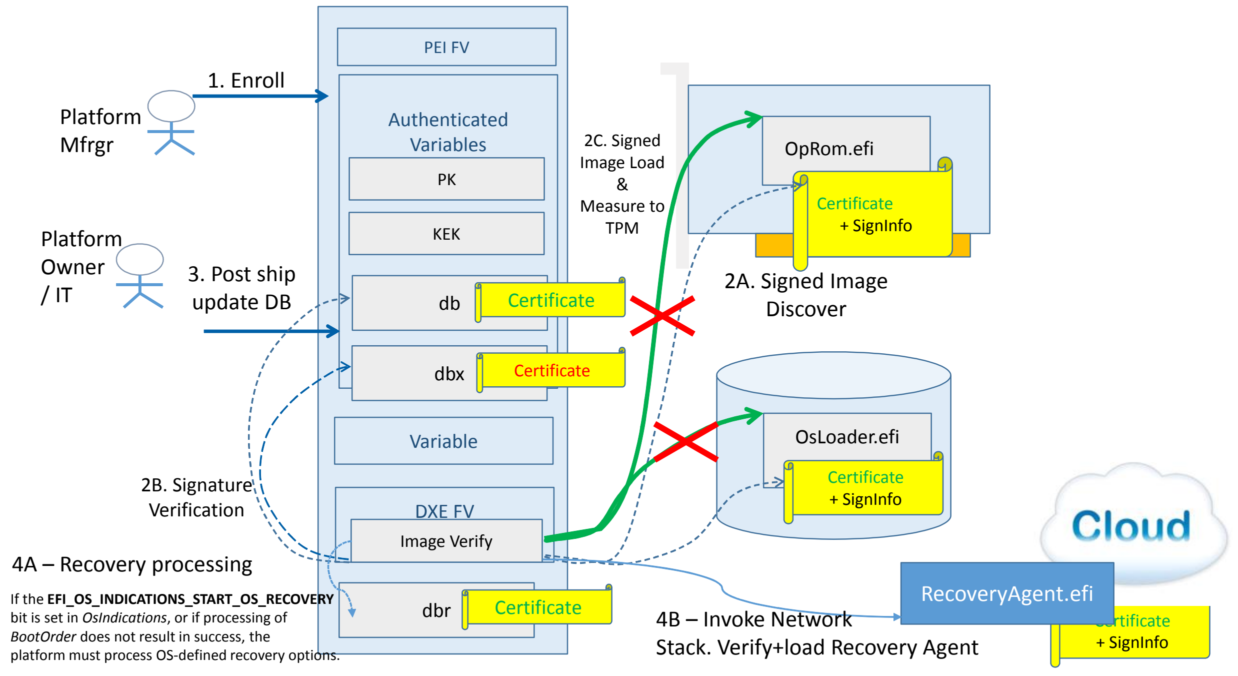

PEI FV
1. Enroll
Platform Mfrgr
Authenticated Variables
PK
KEK
Platform Owner / IT
3. Post ship update DB
db
Certificate
dbx
Certificate
Variable
2B. Signature Verification
DXE FV
Image Verify
4A – Recovery processing
If the EFI_OS_INDICATIONS_START_OS_RECOVERY bit is set in OsIndications, or if processing of BootOrder does not result in success, the platform must process OS-defined recovery options.
dbr
Certificate
2C. Signed Image Load & Measure to TPM
OpRom.efi
Certificate + SignInfo
2A. Signed Image Discover
OsLoader.efi
Certificate + SignInfo
Cloud
RecoveryAgent.efi
Certificate + SignInfo
4B – Invoke Network Stack. Verify+load Recovery Agent

Figure 7 Overall flow

The UEFI specification defines recovery triggers, including **EFI_OS_INDICATIONS_START_PLATFORM_RECOVERY** and **EFI_OS_INDICATIONS_START_OS_RECOVERY**.

The typical life of a platform today includes Step 1 where UEFI secure boot and recovery options are provisioned in the factory. The system typically ships with UEFI secure boot enforced and the state of UEFI secure boot variables measured into PCR[7]. When a system restarts, the UEFI PI firmware will execute from the system board flash firmware volume (FV) and in Step 2 cryptographically challenge any PCI adapter cards. If they pass signature check, the option ROM will execute. Afterward, the UEFI firmware will process the Boot#### options and cryptographically verify the OS loader, pre-OS application, or loadable UEFI driver from the mutable disk wherein UEFI loaders are in designated paths on the EFI System Partition (ESP) in Step 3. The ESP is a FAT formatted partition for a Guided Partition Table (GPT) formatted disk. If the Boot#### options themselves have been deleted, corrupted, or the .efi OS loaders have been deleted, corrupted, then the system will process the hardened OsRecovery#### options in Step 4. These are stored in an authenticated variable OsRecoveryOrder with the corresponding RecoveryAgent.efi from local disk, flash, or network needing to be UEFI Secure Boot cryptographically verified to the dbr (and not in the black list of dbrx), or 'database of recovery options.' This database is the analog of the db/dbx for UEFI secure boot white / black-listing, respectively.

The reason that the network recovery is interesting is that the GPT is mirrored to the end of the disk, but the ESP is world readable/writable (with OS protection at runtime). The typical disk topology is as follows.

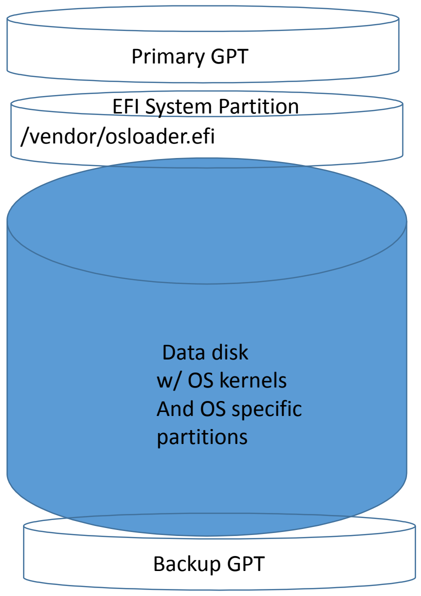


Figure 8 Typical disk topology

Given the mutability of the ESP and potential perturbation of Boot####, a recovery option located on a web server with an OsRecovery boot options.

The present triggers for recover are the new OS indication bits.

There has historically been the ability to detect and go into recovery mode as one of the many PI boot modes **BOOT_IN_RECOVERY_MODE**, which is typically done by an early PEI Module (PEIM). This is detected in PEI and passed into DXE. This can be used to inform the platform recovery flow **EFI_OS_INDICATIONS_START_PLATFORM_RECOVERY**, too.

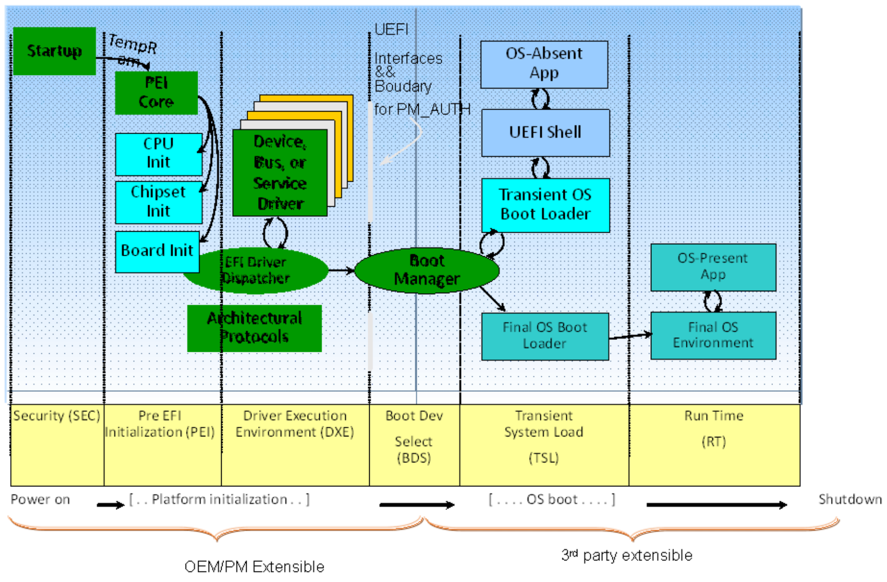


Figure 9 Overall boot flow

UEFI network services – getting into the platform:

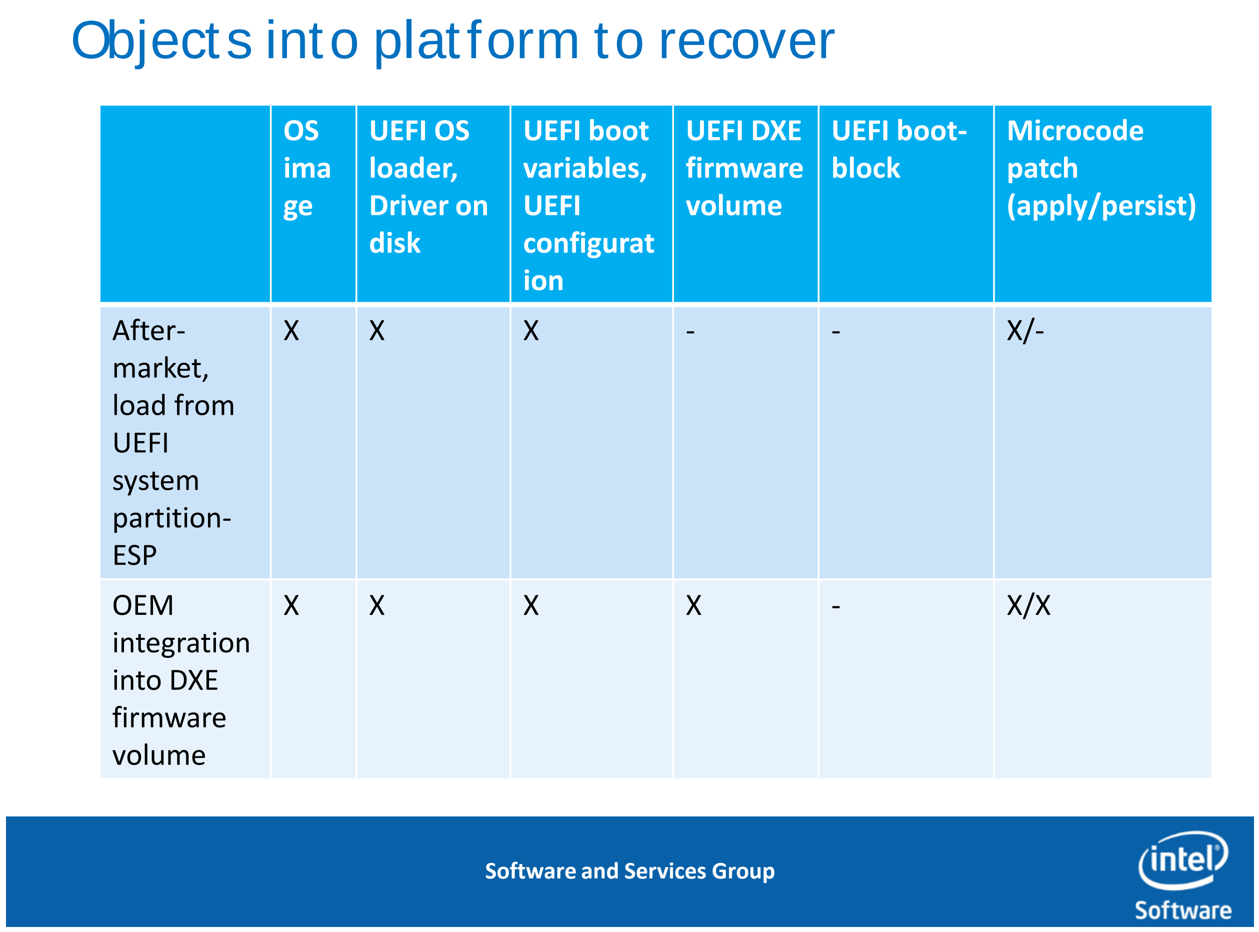
UEFI is just a variant of host firmware, or a "BIOS" (Basic Input Output System). UEFI is distinct from a PC/AT BIOS, the preceding de facto standard codified starting in the 1982 PC and attendant OS usage of Microsoft DOS. Or other technologies like U-Boot which are instances of a boot loader used by embedded Linux and real-time operating systems. UEFI has a standard which defines the basic interfaces for purposes of interoperability, meaning that if you create value-add capabilities above and beyond the basic specification, they should snap into a conformant underlying implementation of UEFI on an OEM system board. The limitations of UEFI from a technical standpoint include: you need healthy NOR or NAND flash from which to load the UEFI codes, thus a missing or wholly corrupt NOR SPI flash would not have any UEFI. To ameliorate firmware storage container integrity, and technologies like Boot Guard can ensure that the OEM's authentic UEFI instance is on the platform, just as UEFI Secure boot can ensure 3rd party extensions, such as option ROM's and boot loaders, are authentic.

The UEFI network stack itself and the additional value drivers can impact the flash footprint, too. To address that concern, most OEM's already compress the latter portion of their UEFI firmware. The OEM's are the authority to control the contents of their flash, though, so UEFI network stack addition added into this store needs to be done complicit with those parties. Alternately, the UEFI network stack drivers can be deployed on the EFI System Partition (ESP), a region of the disk designated for firmware usage. The ESP deployment can also allow for a post-ship provisioning of these capabilities on an aftermarket platform, too.

This network recovery usage adds a few additional drivers on top of the built-in UEFI network stack in order to download an ISO-formatted image from a web server.   The appeal of using a web server versus a PXE server is provisioning.   PXE servers entail having a custom DHCP service and TFTP service. This is non-routable and works only a LAN.   Also, TFTP-timeouts bedevil deploying this type of scenario on broad networks. HTTP, on the other hand, is routable and most OS's have provision to act as a web server.

Recovery usage:

One usage of UEFI is to recover and update platform state. To that end, some items that can be recovered are described below:



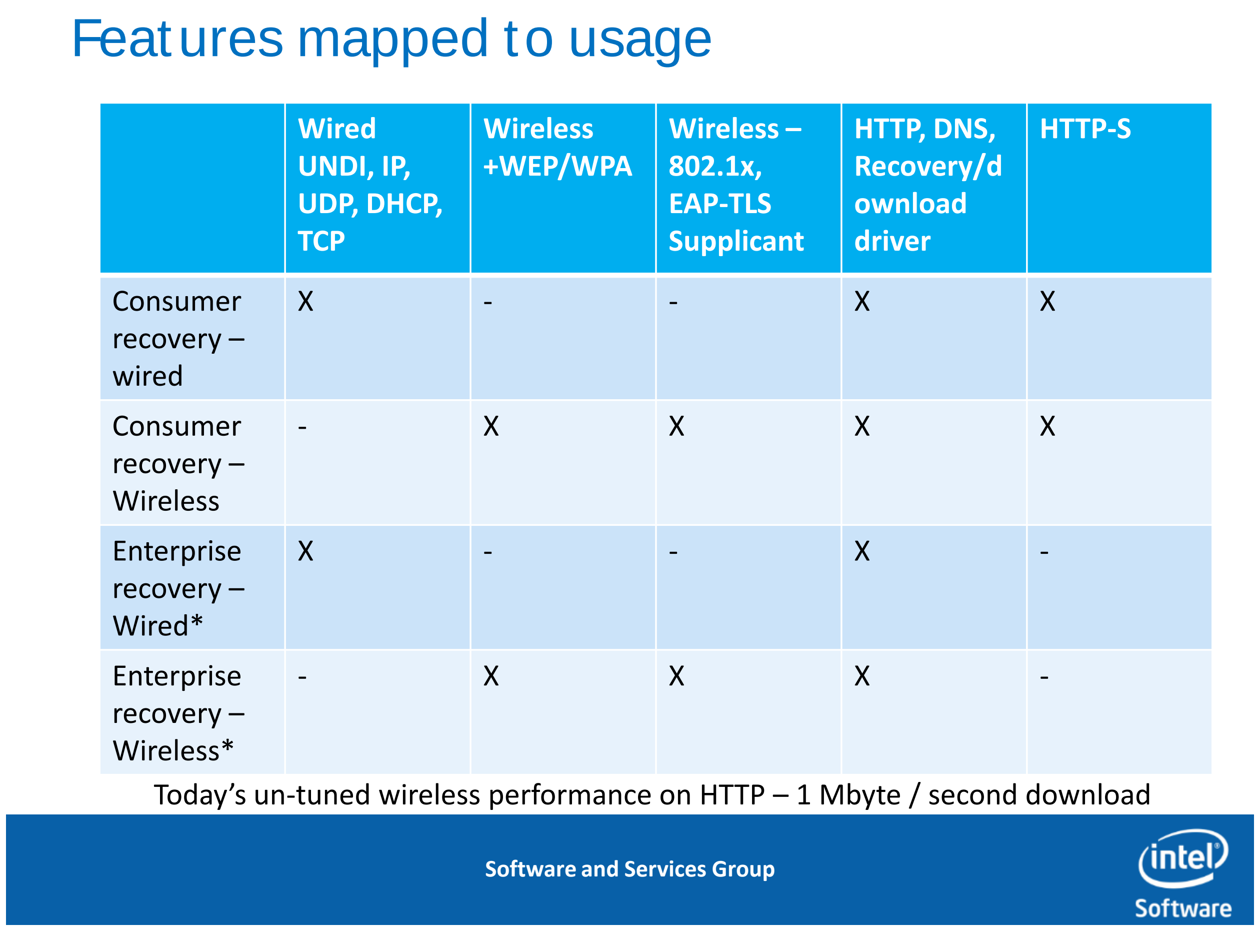
Objects into platform to recover

	OS image	UEFI OS loader, Driver on disk	UEFI boot variables, UEFI configuration	UEFI DXE firmware volume	UEFI boot-block	Microcode patch (apply/persist)
After-market, load from UEFI system partition-ESP	X	X	X	-	-	X/-
OEM integration into DXE firmware volume	X	X	X	X	-	X/X

Software and Services Group

Figure 10 Objects to recover

In addition, there is the ability to support different network transports depending upon consumer versus enterprise deploy.



Features mapped to usage

	Wired UNDI, IP, UDP, DHCP, TCP	Wireless +WEP/WPA	Wireless – 802.1x, EAP-TLS Supplicant	HTTP, DNS, Recovery/download driver	HTTP-S
Consumer recovery – wired	X	-	-	X	X
Consumer recovery – Wireless	-	X	X	X	X
Enterprise recovery – Wired*	X	-	-	X	-
Enterprise recovery – Wireless*	-	X	X	X	-

Today's un-tuned wireless performance on HTTP – 1 Mbyte / second download

Software and Services Group

Figure 11 Consumer versus Enterprise

As UEFI is platform neutral, these elements should work on a UEFI conformant Linux, Android, or Windows platform.   In fact, as long as UEFI firmware is present the platform can orchestrate a UEFI recovery flow.

The UEFI recovery flow can complement the Windows boot process. Namely, if the platform cannot boot the main OS or there is some failure discovered, the platform can initiate some of the UEFI recovery usages. This flow is shown below.

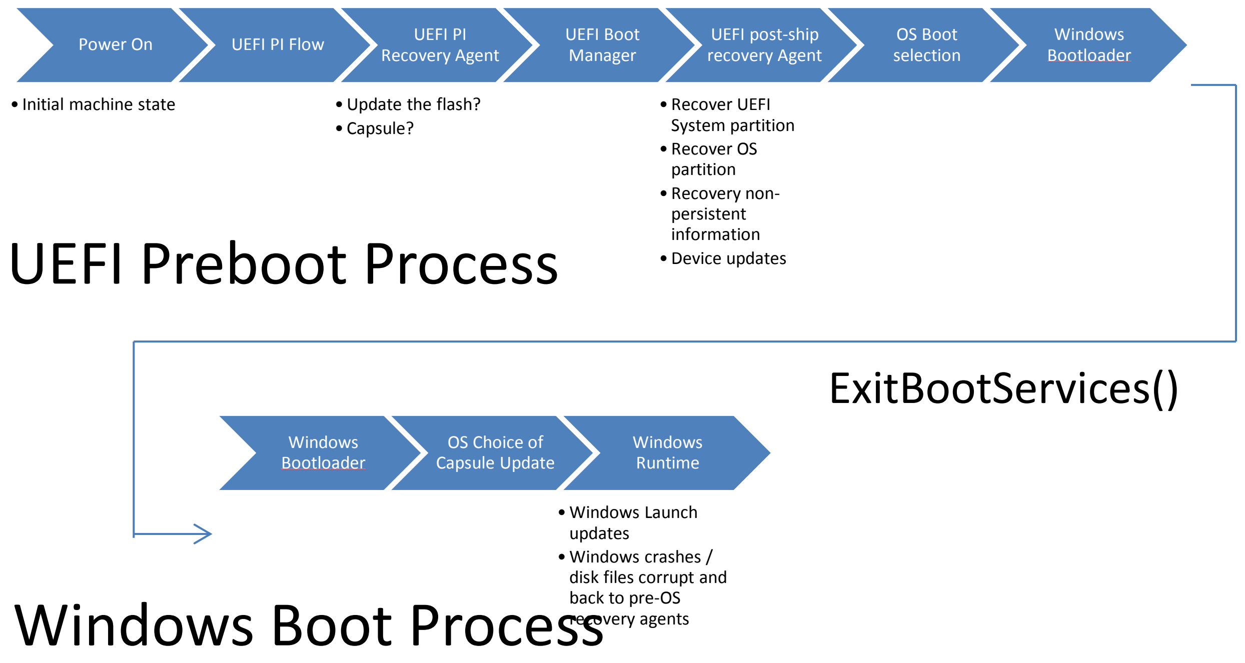
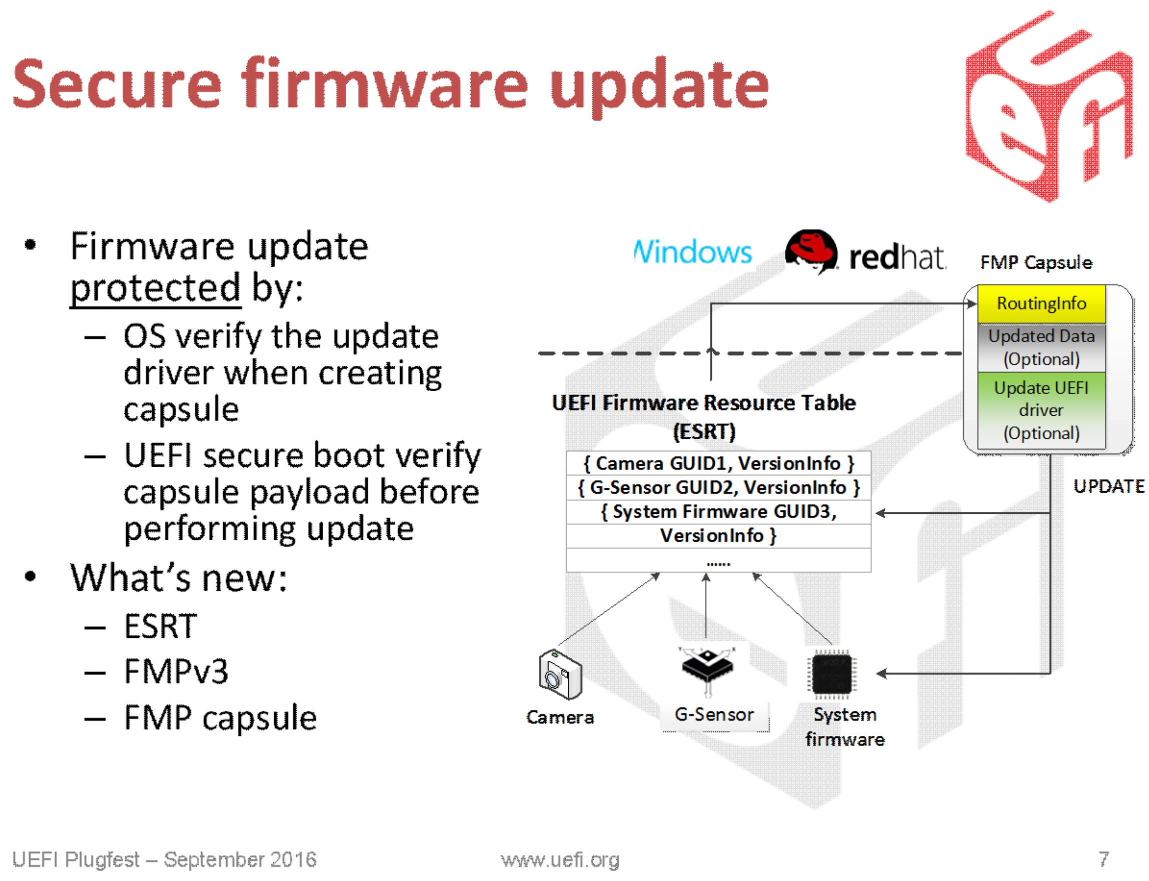


Figure 12 Recovery flow

Latest updates for capsule update include the following:



Secure firmware update

- Firmware update protected by:
  - OS verify the update driver when creating capsule
  - UEFI secure boot verify capsule payload before performing update
- What's new:
  - ESRT
  - FMPv3
  - FMP capsule

Windows
redhat
FMP Capsule
RoutingInfo
Updated Data (Optional)
Update UEFI driver (Optional)
UPDATE
UEFI Firmware Resource Table (ESRT)
{ Camera GUID1, VersionInfo }
{ G-Sensor GUID2, VersionInfo }
{ System Firmware GUID3, VersionInfo }
......
Camera
G-Sensor
System firmware

UEFI Plugfest – September 2016 www.uefi.org 7

Figure 13 Capsule update flow [UEFI-PF]

Opportunities for the future:

The present OS indicator triggers are world read-writable like other non-authenticated UEFI variables. Perhaps a future trigger could use the ownership model of [SECURE-MOR] to have one distinguished entity in the runtime (hypervisor or static OS) that can signal the recovery. Otherwise the UEFI API's are quite open [UEFI-RT].

Another potential expansion of the standards would be having ACPI flows [UEFI-ACPI] such that integrated devices could report that they are in a 'need to recover' state. Similarly PCI devices [PCI] could be expanded in the future such that their error reporting can include the ability to request the platform perform recovery.

Conclusion:

The UEFI standards and open source provide a foundation upon software recovery can be deployed. And this recovery work composes with the long lineage of EFI (circa 1999) into the

UEFI specifications of the last decade, along with the other NIST specifications. In fact, all of the platform features and standards can be seen in a composite figure.

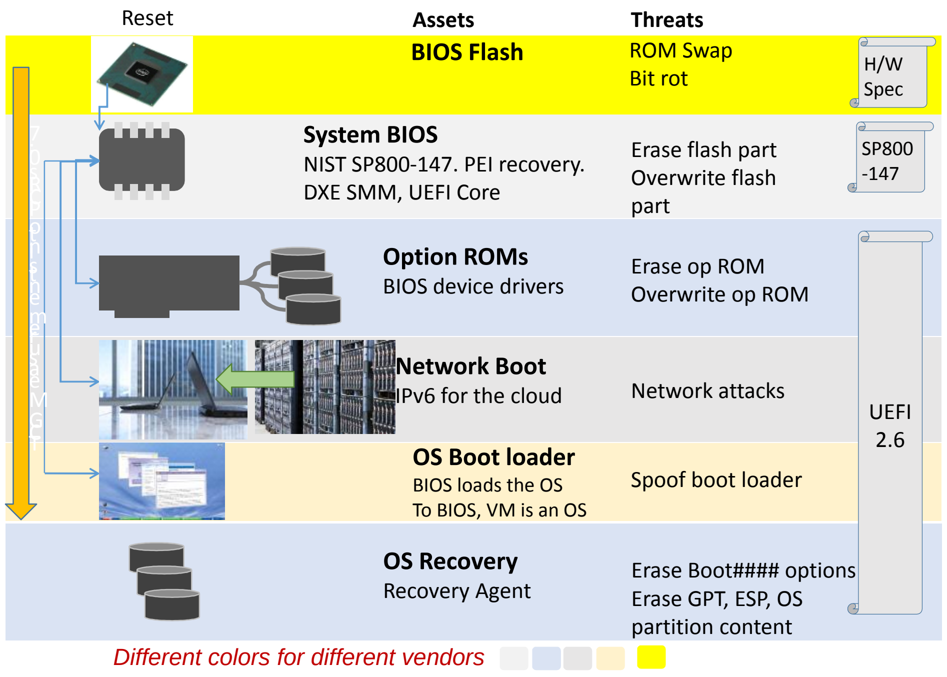


Figure 14 Stack of trusted components

Sometimes the question is asked where is the reference to 147 and / or a protection profile for UEFI. UEFI is a pure interface specification that uses complementary specifications like the TCG Platform Specification, OS logo (Android CDD or Windows logo), and/or the NIST 147 to provide more prescriptive details on ‘how’ to build the host firmware.

References:

[EDK2] EFI Developer Kit II https://github.com/tianocore/edk2

[EDK2-HTTP-BOOT] Getting started w/ EDKII HTTP boot https://github.com/tianocore-docs/Docs/raw/master/White_Papers/EDKIIHttpBootGettingStartedGuide_0_8.pdf

[EDK2-Network] EFKII networking drivers https://github.com/tianocore/edk2/tree/master/NetworkPkg

[EDK2-TLS] Getting started w/ EDKII and TLS https://github.com/tianocore-docs/Docs/raw/master/White_Papers/EDKIIHttps_TLS_BootGettingStartedGuide_07.pdf

[EDK2-VAR] Authenticated Variable Support in EDKII https://github.com/tianocore-docs/Docs/raw/master/White_Papers/A_Tour_Beyond_BIOS_Implementing_UEFI_Authenticated_Variables_in_SMM_with_EDKII_V2.pdf

[ITJ-SB] Magnus Nystrom, Martin Nicholes, Vincent Zimmer, "UEFI Networking and Pre-OS Security," in *Intel Technology Journal - UEFI Today: Boostrapping the Continuum*, Volume 15, Issue 1, pp. 80-101, October 2011, ISBN 978-1-934053-43-0, ISSN 1535-864X http://www.intel.com/content/dam/www/public/us/en/documents/research/2011-vol15-iss-1-intel-technology-journal.pdf

[PCI] Peripheral Component Interconnect https://pcisig.com/specifications

[PXE-SPEC] Pre-Boot Execution Environment 2.1
http://download.intel.com/design/archives/wfm/downloads/pxespec.pdf

[RFC] T. Huth (IBM Germany), J. Freimann (IBM Germany), V. Zimmer (Intel), D. Thaler (Microsoft), "DHCPv6 Options for Network Boot," Internet RFCs, ISSN 2070-1721, RFC 5970, September 2010, http://www.rfc-editor.org/rfc/rfc5970.txt

[SECURE-MOR] Secure MOR, Microsoft, 7/30/2016, https://msdn.microsoft.com/en-us/windows/hardware/drivers/bringup/device-guard-requirements

[TCG-WP] Zimmer, Shiva Dasari, Sean Brogan, "Trusted Platforms: UEFI, PI, and TCG-based firmware," Intel/IBM whitepaper, September 2009
https://people.eecs.berkeley.edu/~kubitron/courses/cs194-24-S14/hand-outs/SF09_EFIS001_UEFI_PI_TCG_White_Paper.pdf

[UEFI] Unified Extensible Firmware Interface http://www.uefi.org UEFI Specification, Revision 2.6, http://www.uefi.org/sites/default/files/resources/UEFI%20Spec%202_6.pdf
UEFI Platform Initialization (PI) Specification, revision 1.5
http://www.uefi.org/sites/default/files/resources/PI%201.5.zip

[UEFI-ACPI] ACPI 6.1 specification
http://www.uefi.org/sites/default/files/resources/ACPI_6_1.pdf

http://uefi.org/specifications

[UEFI-PF] Zimmer, "UEFI Network and Security Update," September 2016,
http://www.uefi.org/sites/default/files/resources/UEFI_Plugfest_VZimmer_Fall_2016.pdf

[UEFI-RT] Zimmer, "Accessing UEFI from the operating system runtime," December 2012,
http://vzimmer.blogspot.com/2012/12/accessing-uefi-form-operating-system.html

[UEFI-SB] Rosenbaum, Zimmer, "A Tour Beyond BIOS into UEFI Secure Boot," Intel Corporation, July 2012, https://github.com/tianocore-docs/Docs/raw/master/White_Papers/A_Tour_Beyond_BIOS_into_UEFI_Secure_Boot_White_Paper.pdf