Clarifying the Ten Most Common Misconceptions About UEFI

Authors:

Michael Krau
Technical Marketing Engineer, Intel Corporation
michael.p.krau@intel.com

Dong Wei
Fellow, HP
dong.wei@hp.com

Overview

The Unified Extensible Firmware Interface (UEFI) Forum is a world-class nonprofit industry standards body that works in partnership to enable the evolution of platform technologies. The Forum champions firmware innovation through industry collaboration and the advocacy of a standardized interface that simplifies and secures platform initialization and firmware bootstrap operations. Both developed and supported by representatives from more than 250 industry-leading technology companies, UEFI specifications promote business and technological efficiency, improve performance and security, facilitate interoperability between devices, platforms and systems and comply with next-generation technologies.

CONTENTS

This paper outlines and clarifies the most prevalent misconceptions about UEFI technology, while providing key developer and end-user insights surrounding implementations, compatibility, signing keys and other key aspects of UEFI technology. Below is a table of contents, outlining the most common misunderstandings about UEFI.

INTRODUCTION

Often, the source of a misconception is traced to the application of "known facts," with misunderstood or over-generalized conclusions. We exist in a world with an abundance of information, facts and theories, increasing our chances of generalizing data points based on a singular event or condition, across an entire range of possibilities. Knowing this, we should carefully inspect each misconception (and associated axiom) and incorporate the discoveries of new facts to determine where the truth lies.

Understanding the most prevalent misconceptions about UEFI necessitates an initial exploration of its benefits. In many cases, the origin of misconceptions about UEFI can be traced to an incomplete understanding of the nature and context of its benefits.

BENEFITS OF UEFI TECHNOLOGY

Outlined below are ten notable benefits UEFI technology offers OEMs, enterprise consumers and end-users. (Graham-Smith, 2013)**,** (James, 2011)

1. **Enables better disk and network support:** UEFI works well with modern hard disks, bringing full support for GUID Partition Table (GPT) at the firmware layer. It also offers network support for IPv6 during the boot phase.
2. **Ability to be emulated:** A number of virtualization platforms can emulate UEFI firmware, allowing UEFI-dependent operating systems (OS) to be loaded within them.
3. **Multiple OS support:** UEFI is designed to support the installation of multiple, different OSes.
4. **Serves as an abstraction layer between the firmware and the OS:** This speeds up deployment of new platforms and OS features.
5. **Serves as a boot device driver target:** Device drivers can target the UEFI environment instead of the unique hardware, allowing for reuse across platforms.
6. **Provides a robust environment:** UEFI provides a robust environment for pre-OS diagnostic tools.
7. **New modules are easily added to the graphical interface:** This includes device drivers for motherboard components and external peripherals.

8. **Device functions are not tied to a particular platform**: UEFI is based in the software environment, meaning that it is not tied to specific architectures or platforms.
9. **Closer degree of integration between the OS and pre-boot environment**: This efficiently transfers data and capabilities to the operating system.
10. **Malware prevention with UEFI Secure Boot mechanism:** The optional UEFI Secure Boot mechanism protects users from malware and other forms of tampering from the point of boot initiation.

The below sections examine the ten most common misconceptions about UEFI technology, addressing topics ranging from security, to boot time, to open source implementations.

Misconception 1: UEFI is slow to boot

The genesis of this misconception is rather straightforward. The UEFI boot process on most current, commercially-available platforms takes measurable time from the power-on state to the loading of the OS. One can verify the boot process timing with a stopwatch: Start the watch while pressing the system "power on" button and stop the watch when the OS splash screen appears. When performing this test on most of today's available UEFI platforms and other mobile platforms with alternative booting mechanisms, the results seem to demonstrate that UEFI is slower.

Upon a closer examination of the facts, we find the potential for some generalizations and assumptions that could lead to the development of an incorrect premise. The following content compares the results of various platforms. The time measurement comparison is made between current, available UEFI booting platforms, versus other mobile platforms with alternative booting mechanisms. This comparison can be misleading, as current platforms with UEFI boot typically target PC- and server-class products, while the platforms for other booting mechanisms (e.g. coreboot and Uboot) apply to mobile devices and appliances. The very nature of the platform classes creates differences that can influence the platform boot processing time. These differences are as follows:

- ***PC- and server-class platforms*** – These are multi-role platforms with large memory and resource capabilities, as well as a robust set of resource options. A PC platform with the UEFI

Secure Boot implementation comes in several manufactured configurations and allows end users to customize the system with after-market components. Specifically, a single UEFI Secure Boot implementation could boot several different configurations, including those not supported in the original platform development. Therefore, the booting mechanism is expected to monitor ad-hoc changes and resolve them, without noticeably disrupting the booting process.

- ***Mobile device and appliance platforms*** – These limit the scope of the device to its general function and restrict the set of configurations. Most devices and appliances work to establish memory and a storage "footprint" that is often smaller than the basic footprint of a PC-class device. Namely, the mobile device class has a reduced resource layout and set of options, as well as fewer boot code execution paths (each code execution path takes time to load). In the case of mobile device platforms that have fewer code paths, the overall boot times are quicker than PC-class platforms.

In fact, many firmware industry members regard UEFI Secure Boot as inherently "faster" than traditional BIOS. The quicker pre-boot times are an integral by-product of UEFI's design and organization. UEFI's modularity promotes system optimization; therefore, only necessary components are included in the boot start-up mode. Traditional Legacy BIOS did not allow for such optimization.

While platforms with configuration capabilities allow the user to create a series of boot time settings, this flexibility can negatively influence the platform boot time. For instance, if a platform is configured to await user input (or if it has a waiting window for user input) at boot time, then the delay will prolong the overall time measurement of the platform boot process. While this is a simple and obvious example, it is important to note that providing user configuration capabilities can increase boot timing.

Another consideration is that some platforms simply take longer to boot. This is not because of the boot code execution paths or customizable configuration settings, but rather to allow the initialization of hardware containing the OS screen splash image. In one case, it could take the driver several seconds to initialize high-speed, spinning media drives. Some booting environments (like UEFI) try to optimize the boot process by starting the device initialization and using the spin-up time to perform concurrent tasks.

Nonetheless, occasions exist in which all other tasks are completed prior to the device being fully initialized. In these situations, the boot time increases, given that the initialization process takes longer. This is a classic example in which the measurement of the boot time can be accurate, but may not reflect the actual time spent on boot code execution prior to the OS loading.

When encountering the perception that UEFI is slow to boot, it is important to note that the PC industry deployed UEFI as a means of "fast booting" a system. For instance, in Windows 8, the PC boot time has significantly improved due to fast booting with UEFI. Also, because of UEFI, PC boot times no longer create a bottleneck for further technological improvements; rotational media is now a primary cause. By comparing UEFI boot and other mechanisms with identical hardware and equivalent boot settings, we can realize the full potential of UEFI boot. On a fully-optimized UEFI implementation, in which the hardware initialization of external devices is not an issue, the time from "power on" to the end of the firmware boot and the hand-off to the OS has been clocked at a half-second.

Boot time is not a single, variable formula. Rather, it is based upon a complete series of environmental factors. While a mobile device will boot faster than a server, it is not because the mobile device is without UEFI. Rather, it is because the mobile device is ultimately designed to boot faster.

MISCONCEPTION 2: UEFI IS TOO LARGE TO BOOT

This misconception closely relates to the first one, pertaining to UEFI being "slow to boot." In fact, people often merge the two concepts into a single declaration: "UEFI is too big and too slow to boot." The combination of the two statements is a way to hedge the assertion that if UEFI is too large, then it is also too slow. Conversely, if UEFI is too slow, then it must also be too large. This logic has flaws. It is possible to create a slow program to run in a small space, just as it is possible to create a large program to run quickly. From a firmware developer's perspective, while a larger program could take longer to execute, the size of the final firmware image and its impact on code execution time is generally not a concern. Rather, the size of a flash device that stores the boot images is the determining factor. The larger flash device typically correlates with increased price and part size.

Consider a comparison of the UEFI boot image size, versus that of another boot mechanism. Unfortunately, by trying to compare the amount of space used for UEFI boot code to the size of code used by other mechanisms, we face a similar comparison issue, as discussed in the time measurement case.

When comparing the size of boot mechanisms, it is important to note that UEFI implementations provide both bootstrap capabilities, as well as a firmware standard. The use of a standard does mean a certain sacrifice of space for conformance and associated flexibility. The tradeoff is small, considering that anyone wishing to reuse their code across several designs, or to boot multiple OS environments from a single implementation, will need to start with a standard, to ensure consistency.

The UEFI image is typically targeted for a multi-role platform, with several configuration options and expansions. Interestingly, those options can increase the size of the firmware image without impacting the execution time, as the support for options must be present (occupying space) even if it is non-existent on a specific platform (negating the need for code execution).

Therefore, to compare the UEFI conformant boot to other boot mechanisms, one should consider the minimum requirements of UEFI conformance as a starting point. In the UEFI specification, section 2.6.1 defines the minimum functional set of code necessary to meet UEFI conformance. This information, along with the API data, is provided in Figure 1, below.

Figure 1: API Data and Minimum Functional Set of Code Necessary to Meet UEFI Conformance

Protocol	Data Field Pointers	Total APIs	Optional APIs**
EFI_SYSTEM_TABLE	13 *	0	0
EFI_BOOT_SERVICES	0	39	4
EFI_RUNTIME_SERVICES	0	15	11
EFI_LOADED_IMAGE_PROTOCOL	1	0	0
EFI_LOADED_IMAGE_DEVICE_PATH_PROTOCOL	1	0	0

EFI_DEVICE_PATH_PROTOCOL	1	0	0
EFI_DECOMPRESS_PROTOCOL Protocol	0	2	0
EFI_DEVICE_PATH_UTILITIES_PROTOCOL	0	8	7
Total	16	64	22

* Including Start of Linked Lists

** Optional defined as capable of returning: Unsupported, NULL, or Error due to platform limitation

In practice, there have been fully-compliant UEFI boot implementations (including Platform Initialization components) smaller than 300KB. In fact, an optimized, fully-compliant UEFI boot image (per Figure 1) free of non-essential drivers has been produced for Intel architecture systems in 128KB of flash. Of course, this is a trade-off situation, in which reduced size also limits the capability and/or platform flexibility. If support for some option or extended capability is required at boot time, then code to support the option or extension must be incorporated into the boot image. Even if the added code is never executed (meaning, the option or extension is not engaged, activated or used), the code must still reside in the flash, in case it is needed for future support. Therefore, the question of the code size at boot time depends on which capabilities are anticipated in the field (both mandatory and optional), in addition to which capabilities the boot firmware supports.

The strength of UEFI, with regard to design decisions, results from the modular and extensible features of the UEFI architecture. The UEFI architecture can easily be adjusted, or "right-sized," without a code size or development time penalty, to meet the needs of the specific, intended application. Right-sizing is simply a matter of picking the right menu options, as opposed to adapting to a pre-existing code base. The modular design makes it easier to add/remove components and optimize the booting environment. This is not the case with the non-modular, monolithic boot architectures. UEFI's modularity allows the addition or removal of components, allowing software components to interoperate without additional code.

MISCONCEPTION 3: THE UEFI SPECIFICATION IS TOO EXTENSIVE

This misconception tends to be one of the most misleading, as the facts are complementary to the perception. The UEFI specification is more than 2,000 pages (version 2.4 is 2,484 pages in length). In light of arguments made against specifications of equal or even lesser lengths, one may wonder, "Why so big?" The extensibility of the UEFI specification is an attempt to quantify and qualify the boot experience, by adding new information and innovation as lessons are learned and new solutions discovered. The argument is not that the specification is big, but that it is "too" extensive, where size is a function of thoroughness.

While the UEFI specification may be large, this does not mean it is "too extensive." The only way to determine this is to analyze the content of the UEFI specification and compare it with the goals of the specification (as stated in the introduction of the UEFI 2.4 specification):

"The specification is applicable to a full range of hardware platforms from mobile systems to servers. The specification provides a core set of services along with a selection of protocol interfaces. The selection of protocol interfaces can evolve over time to be optimized for various platform market segments. At the same time, the specification allows maximum extensibility and customization abilities for OEMs to allow differentiation." (The UEFI Forum, 2013)

The UEFI specification includes both mandatory and optional protocols. It is intended to be extremely thorough, with the goal of providing a standard boot mechanism for a broad array of computing devices. The aim of the UEFI specification is to not only provide a standard booth method for these devices, but to also grant the product developers and manufacturers the ability to customize products.

To make this goal more comprehensive, the UEFI specification is not limited to processor architecture, platform classification or a target OS. Essentially, the UEFI specification was crafted without predisposition toward any implementation-specific detail of the product booting via UEFI. This non-biased position has led to UEFI boot support in both Intel and ARM architectures. In fact, ARM recommends UEFI as the preferred boot loader for 64-bit processors, based on the ARMv8 AArch64 architecture. The architecture optimizes applications—from smartphones to servers—by introducing a new set of features, including a larger register file, enhanced addressing range and cryptography instructions.

What appears to be “too extensive” is actually a representation of the UEFI’s thoroughness in providing answers to the issues surrounding system boot. In reality, a developer will not need to implement every interface or every element of the specification. Instead, the intent is for the developer to use the specification as a basis for creating a system boot solution, tailored to specific product needs and capabilities. In other words, the developer can determine the scope of the product and select the necessary interfaces for that product. No single product will ever implement all of the interfaces defined in the UEFI specification.

The original 2.0 UEFI specification, published in January 2006, was 1,437 pages long. Since then, the specification has grown to 2,484 pages. The additional pages include support for other processor architectures, streamlined Human Interface Interactions, new technology expansions and innovations, and lessons learned.

MISCONCEPTION 4: THE UEFI PLATFORM IS JUST AS INSECURE AS THE LEGACY BIOS

The facts surrounding this misconception demonstrate that *when the advanced UEFI security features are not implemented*, UEFI systems have many of the same security issues as previous systems that lacked such security features. While this is true, it limits the context and definitions used as a way to guarantee an outcome, regardless of mitigation or reason.

A system that has not implemented a feature cannot be expected to demonstrate that feature. Therefore, a system without advanced security features will lack the same security features as another system that was never designed for advanced security. This is true of any system, regardless of boot technology. However, in the case of UEFI, there are advanced UEFI security features. These security features are defined in the UEFI specification and can be implemented as a design option under UEFI to make a system more secure. This is not true for most of the other boot technologies. Since UEFI solutions have many capabilities (such as security and legacy support), there is a tendency for some vendors to combine options.

UEFI boot has been used across many systems and for many years. In fact, some systems have been using UEFI in conjunction with a Legacy BIOS support feature (Compatibility Support Module, or CSM) for more than a decade. In those cases, the end user might not be aware that the system is using UEFI, as the CSM makes the boot process appear to be a standard Legacy BIOS boot. In these systems, UEFI support is not intended to be more than a mechanism, which allows the CSM to provide the final boot support. While these systems can be considered "UEFI systems" (officially a Class 1: UEFI with CSM only), they are specifically designed to limit the exposure of UEFI capabilities in the boot process (UEFI Forum Industry Communications Work Group, 2011).

The requirements and capabilities of UEFI—in both specifications and implementations—have been (and still are) in a steady state of evolution and growth. When the PC industry established needs, in terms of replacing Legacy BIOS, the UEFI solution was created. However, as the needs of the industry grew (both in the PC space and for other platform classes), the UEFI specification also grew and evolved to meet these needs (see Misconception 3: The UEFI Specification is too Extensive).

The UEFI specification was generated to define a boot environment that could provide a solid user experience, in terms of performance, flexibility and security. It was intended to evolve over time and to adapt to the new demands of the industry's technology, manufacturers and consumers. Conversely, Legacy BIOS boot mechanisms have not adapted to the industry evolution that paved the way for a "modern firmware" definition to support next-generation mobile devices.

While the standard nature of the UEFI interfaces can create opportunity for malicious attacks, the mechanisms included in the UEFI specification provide methods to detect and defend against such attacks. Current UEFI specifications and implementations incorporate security features, such as key signing and signature checking. With these features available and enabled, the security of the UEFI booting platform is strengthened.

MISCONCEPTION 5: UEFI CANNOT SUPPORT FULL OPEN SOURCE IMPLEMENTATIONS

This misconception results from the comparison between UEFI boot, coreboot and other open source firmware solutions. Some in the industry believe solutions like coreboot are the better choice, given that they are open source. The same people also tend to believe that UEFI cannot support a full open source implementation. In fact, the UEFI specification includes support for open source implementations; however, vendors may not opt to provide open source implementations for their products, due to their intellectual property (IP) protections. Aside from IP protections, UEFI vendor implementations do not always support open source firmware, due to the absence of open source silicon drivers. This is not new, as even in the coreboot community, silicon drivers are often provided as binary blocks of code.

To be clear, open source UEFI implementations exist and are available in today's market. In 2004, the UEFI open source community website at www.tianocore.org was launched under a BSD license. TianoCore is one example of an open source project that supports UEFI firmware. The following are additional examples of open source UEFI applications.

- ***QEMU (Quick EMUlator)*** – A free and open source hosted hypervisor that conducts hardware virtualization. OVMF is a TianoCore project to enable UEFI support for IA-32 (x86) and X64 (x86-64) guests on Virtual Machines.

- ***Beagle Board*** – In 2009, the ARM-binding for UEFI was published as part of the UEFI 2.3 specification. With this publication, Apple and HP contributed to the UEFI reference implementations, including one for Beagle Board (http://beagleboard.org/), enabling silicon vendors to supply UEFI drivers for their hardware. Beagle Boards are inexpensive, single-board computers based on Texas Instruments processors sporting an ARM Cortex-A series core. Initially, these boards were developed for Linux distributions to improve support for ARM devices. However, after receiving notable support by Linux distributions, the scope of work altered to focus on simplifying physical computing on advanced GUI-enabled and/or networked-enabled devices. As a result, there has been increased support for numerous computing environments from Ubuntu, QNX, Windows Embedded, Android, among other web tools and programming styles. (Wei, 2013)

- ***ARM Ltd. Reference Platforms*** – As ARM's preferred boot loader, UEFI provides an OS-independent booting solution to support multiple OSes such as Windows and Linux. There are several advantages for ARM pre-boot firmware to use standardized UEFI. UEFI not only enables code sharing among ARM products or among x86 products, it also enables code sharing across processor architectures. In turn, code sharing helps reduce development costs for OEMS/ODMs. Further, the modularity of UEFI technology allows silicon vendors to deliver drivers for their own hardware, which gives OEM/ODMs flexibility to use modules from different suppliers. (Wei, 2013)

- ***MinnowBoard*** – Circuitco Electronics produced the MinnowBoard platform in 2013. This platform, using the Intel® E640 Atom® processor, was delivered as an open hardware platform to the developer community. In the following months of product availability, hardware design files and other design documents were published to the open community as a part of this open hardware project. By December of 2013, the UEFI source build environment for the MinnowBoard was also released to the community. This build environment includes many platform and silicon-specific modules in binary format, while the majority of sources for building the production and debug versions of the firmware boot image were made publically available.

MISCONCEPTION 6: UEFI HAS A RESTRICTIVE PROCESSOR ARCHITECTURE

The basis for this misconception is the belief that UEFI supports only 64-bit processors. One can rectify this misconception easily, given that support for 32-bit processors has always been included in the UEFI specification. The UEFI specification clearly documents support for 32-bit applications in all 32-bit architectures (for which there are processor bindings). The same statement is true for all 64-bit applications and architectures for which there are processor bindings.

The UEFI specification has no architectural limits and allows flexibility in design implementations. The UEFI specification supports any processing architecture that has defined interface connections. Currently, the UEFI specification contains interface definitions for IA-32, IA-64, X64 (on IA architecture), ARM 32-bit, and ARM 64-bit architectures. The UEFI Forum will add other processor architectures, as the proponents of those processors express an interest or need for UEFI boot support in additional architectures.

Contrary to some widely held beliefs, UEFI implementations for IA-32 architecture do exist on the market today. Examples include the MinnowBoard and the HP ElitePad. Although some OSes may not validate 32-bit UEFI boot support, it is important to note this is an implementation decision and not an oversight in the UEFI specification.

The UEFI Forum never envisioned supporting 32-bit and 64-bit architectures on the same device. Consequently, the UEFI specification does not offer provisions for dual-mode boot. Ultimately, it is up to the implementers to decide which processor(s) to support. Any implementation that attempts to provide 32-bit and 64-bit UEFI boot support on the same system would require a proprietary solution, and therefore would not be appropriate for an industry wide specification. From the UEFI Forum's perspective, this proprietary matter should ensure industry interoperability to support technological growth.

Currently, many industry members have concerns about switching the upcoming Ubuntu 14.04 release to a UEFI-only image, as some older systems cannot support multiple images in the El Torito bootable CD-ROM format. To keep up with the growing number of 32-bit UEFI systems (mainly Class 3: UEFI-only systems, without CSM), the new release may include an additional, non-UEFI image. The goals are to allow the 32-bit image to have UEFI support, as well as to accommodate older systems with a non-UEFI image. In other words, the proposed initiative would be to provide a 32-bit UEFI distribution as the default version and an alternative 32-bit legacy version for end-users with older systems. It is important to note that modern 32-bit systems with full UEFI support do not fall into the same category as older, legacy 32-bit systems.

As the market trends toward 64-bit processors, it is undeniable that there will be far fewer 32-bit UEFI systems and software offerings readily available. This imbalance resulted from company choices about product offerings, rather than implementation mandates set by the UEFI Forum. On most systems, IA processors will ultimately migrate to the ubiquitous 64-bit. Until then, it will always be possible to support IA-32 on UEFI systems.

MISCONCEPTION 7: UEFI FORUM CONTROLS VENDOR IMPLEMENTATIONS

This is based on the belief that an industry forum cannot promote an open standard architecture, along with vendor-specific implementations of the standards that result in proprietary solutions. This sounds logical when applied to the basic question, “How can something be ‘open’ and ‘proprietary’ at the same time?” Yet, the fallacy of this concept is derived from the presumption that both “proprietary” and “open” traits must be holistic in the implementation.

The UEFI specification is both platform- and architecture-independent. It was developed to promote cross-functionality and broad adoption across multiple hardware platforms and operating systems. Therefore, it is “open” and establishes interfaces and infrastructures to use in the boot process of nearly any platform of virtually any operating environment.

The UEFI specification provides pre-boot software services through dynamically locatable and executable protocols. These are discovered by standard search applications using standard Globally Unique Identifiers (GUIDs). In previous boot solutions (like Legacy BIOS), such pre-boot services were provided either by jumping into predetermined, fixed entry points in memory, or by calling software interrupts with parameters stored in the processor’s registers. In the case of UEFI, the ability to call pre-boot services is ported across platforms and architectures. In the case of Legacy BIOS, the solution is nearly guaranteed to be bound by the architecture, if not the platform.

While the UEFI specification is open, it does not specify how the protocols are implemented. In fact, the UEFI specification only requires a subset of system implementation protocols and services to be labeled UEFI-compliant. In theory, the vendor can choose to be non-UEFI compliant and still use the UEFI specification to define the pre-boot space. Vendors who opt for UEFI compliance can support the compliant protocols, as long as they: 1) accept the specified inputs, 2) return the specified outputs and 3) perform the tasks as defined in the specification. A vendor may also choose to add optional UEFI protocols or custom protocols to the implementations. The modular design provides the ability to transfer protocols between platforms and allows proprietary implementation components across product lines.

There appears to be growing convergence and sharing among UEFI implementations. This decision is left to the vendors as a method of improving their products by reusing code and sharing their debug experiences, to ultimately reduce development time and increase consistency. To summarize, the vendor may customize the firmware to meet customer requirements, by tailoring the device implementations to fulfill the optimal set of features. The UEFI specification provides open protocols and mechanisms, while the vendor provides proprietary or open implementations and products based on their preferences.

MISCONCEPTION 8: UEFI IS NOT COMPATIBLE WITH MOBILE DEVICES

This misconception is very myopic in nature. The only beliefs supporting this conclusion are: 1) There are many non-mobile devices utilizing UEFI technology for pre-boot, and 2) There are many mobile devices that do not use UEFI for boot.

The first contradictory fact can be found in the UEFI specification. As mentioned before, the UEFI specification is not implementation-specific. Moreover, the UEFI specification has design elements intended for mobile platforms and non-PC devices. As the specification evolved, these elements became the migratory path for mobility, keeping the UEFI specification implementation-independent, yet compatible with mobile applications and developments.

When evaluating the uses of UEFI, it is clear that many PC- and server-class platforms use UEFI as the preferred booting solution. However, UEFI has long been used as a boot mechanism and operating environment for other device classes, including printers, scanners, network routers, network switches and storage devices. While these may not qualify as “mobile” devices, they demonstrate that UEFI is implementation-independent. The release of Intel®’s Quark SoC X1000 open source code shows that not only is UEFI suited to address the mobile market, but also for the IoT (Internet of Things) market.

Further investigation reveals that multiple mobile implementations use UEFI as their boot solution. Many tablets and smartphones now boot with UEFI technology. This fact demonstrates that UEFI is compatible with mobile devices. In fact, the extensibility and flexibility provided by UEFI aligns with mobile and embedded devices design requirements. UEFI bridges architectures supporting ARM and IA technologies, as well as provides interface layers that allow developers to reuse code, not only on similar platforms, but also across platform classes and architectures.

One example is the C-code provided on the UEFI open source community website, www.tianocore.org, which is used to support the boot process for many product classes across ARM and IA architectures. UEFI is an excellent fit for mobile devices, as it is compatible with a multitude of computing applications and strategies.

MISCONCEPTION 9: UEFI FORUM RESTRICTS WHICH CERTIFICATE AUTHORITIES CAN PROVIDE SIGNING KEYS

The UEFI specification does not define who can serve as a Certificate Authority (CA) for signing keys. In fact, in a vertically integrated environment in which platform hardware, firmware and the operating system are self-owned, one can set his or her own CA to provide signing keys for the firmware and bootloader. Nonetheless, this practice is not scalable in a horizontal environment, in which firmware modules are from third parties. If each entity were to own the CA, all the keys from third parties would be stored in limited non-volatile database. This is not practical.

To provide some background, the UEFI Forum conducted extensive research about setting up a CA and internal portal to provide a signing service for a “UEFI CA Key.” Unfortunately, the Forum did not have the infrastructure to sustain this service. Externally contracting the CA signing services was also considered, but as a non-profit organization, the Forum did not have the financial means to fund this endeavor. The Forum invited various entities, including the Linux Foundation, to provide additional signature authorities, yet no serious inquiries from groups or agencies with resources and groundwork to support this service have been received.

Are you interested in being a UEFI key provider?

If you would like to learn more about becoming a key provider for the UEFI Forum, please contact: admin@uefi.org.

Without any other potential leads or offers, the UEFI community extended an invitation to Microsoft to consider volunteering its existing infrastructure and signing services. Microsoft accepted the invitation to provide an interim signing service, which also aided the Windows 8 release. Some believe it was

Microsoft's plan all along to be the sole CA for UEFI, preventing other CAs from stepping up. Contrarily, the UEFI Forum has been actively seeking additional key providers.

Additionally, it is important to note that Microsoft could have restricted others from using the CA signing services, but instead realized the industry need and opened up the service to the rest of the ecosystem. The signing service is currently offered through VeriSign for an annual $99 fee. This allows third-party entities to sign UEFI binaries with a Microsoft UEFI CA Key.

It is also worth noting that Microsoft does not require that the Microsoft UEFI CA Key be included on integrated systems. In fact, the key is not needed for a system that is designed to be closed (no add-in I/O devices) and only supports Windows 8/8.1. However, systems that support add-in I/O devices and other OSes require multiple sources of UEFI drivers and boot loaders. As a result, the embedded Microsoft UEFI CA Key is desirable, due to the limited amount of non-volatile space normally seen on these systems.

Currently, Microsoft continues to serve as the only key provider for UEFI, but the Forum is in constant pursuit of securing at least one more certificate authority (Jackson, 2013). Ultimately, the Forum looks forward to the industry community developing a collaborative, alternative signing service, as this would be ideal for all parties—particularly those in the open source community.

If you are reading this and are interested in learning more about becoming a possible key provider for the UEFI Forum, please contact: admin@uefi.org.

MISCONCEPTION 10: UEFI CA KEY AND THE FIRMWARE KEY ARE THE SAME THING

This misconception results from misinformation and confusion about the signing keys used to support the UEFI Secure Boot features and those used to support the secure-system firmware update, as articulated in the NIST 800-147 document. The following information differentiates between the two.

- ***UEFI Secure Boot*** – UEFI Secure Boot is a technology used to perform signature authentication for the third-party code that a system launches during pre-boot, therefore eliminating major security holes during handoff from the UEFI firmware to the operating system. The third-party code includes the UEFI drivers and applications (e.g., UEFI Option ROMs and OS bootloaders). To minimize the number of signing keys stored in the nonvolatile storage, it is desirable to use the UEFI CA key to sign the UEFI Option ROMs, and possibly OS bootloaders as well. In addition to signing Option ROM images, the CA can also sign Option ROM-secure update drivers. In other words, the "UEFI CA key" signs the third-party UEFI drivers and applications, but not the system firmware update image.

- ***Secure System Firmware Update*** – NIST 800-147 articulates the need for a secure-system firmware update mechanism. One way to achieve this is to digitally sign the system firmware update image. However, since the system firmware update image does not use third-party code launched during pre-boot, there is no need to sign it with the UEFI CA key. In fact, it is more desirable to sign the system firmware update image with a separate key owned by the Original Equipment Manufacturer (OEM). This way, potential UEFI CA revocations would not impact the security of the system firmware update.

The UEFI CA key is intended for third-party UEFI application/driver images, while core firmware updates can be under a separate, private OEM key. Some users claim that UEFI attempts to restrict implementations and IP capabilities. In reality, OEMs own their implementations and can use their own signing keys for firmware updates, as well as add additional security.

Conclusion

This paper discusses the most common misconceptions about UEFI technology, addressing misunderstandings about topics ranging from device compatibility, to boot speed, to signing keys and more. While no technology is without flaws, UEFI addresses both present and emerging security issues, streamlines the user experience and supports extensibility and standardization. Some firmware industry professionals consider the composition of the UEFI board and membership as evidence that the Forum serves only the interests of select companies. It is believed that certain companies will devote resources to this industry standardization only if there is a direct benefit or return on investment. In reality, not all benefits are exclusively beneficial to a particular company. Standards organizations exist to benefit all members, as well as the industry at-large.

The UEFI specification was developed at a time when computer technology was improving and expanding at a rate faster than the original boot mechanisms could support. New devices, larger media devices, new communication standards and more advanced processors demanded changes in the boot process. Thus, the UEFI Forum was created to provide a standard and extensible UEFI boot mechanism to support new technology in the boot space.

When looking at the UEFI Forum membership list, one can see that the industry is well-represented. The Forum's membership consists of approximately 250 member companies, most of which are corporations. There are fifteen individual members, most of whom are academic leaders and professors without a corporate affiliation. Observers and members of the UEFI Forum have noticed a unique culture for collaboration. Even members who are direct market competitors work together for the good of the industry through UEFI (and now also ACPI) specification advancements. The UEFI Forum has several working groups that help develop specifications and hosts industry events. Each working group has a chair, who is elected by a unanimous vote of the board (even though the UEFI Forum bylaws do not require a unanimous vote).

Several of the perceptions addressed in this paper speak to the belief held by some that the Forum makes a concerted effort to create additional restrictions to the advancement of some, and to the detriment of others. In fact, the core initiatives of the Forum are to demystify some very common pieces that improve the design envelope and the internal ecosystem, to promote interoperability and platform robustness and to increase available choice. When a diverse group of industry constituents works together to remove roadblocks to innovation and interoperability—adding features like Secure Boot to improve malware resistance—they serve the greater good. By allowing companies to focus their investments on the platform hardware, users gain access to more interesting and creative product designs.

For more information about the UEFI Forum, visit uefi.org.

References

- Graham-Smith, D. (2013, May 3). *UEFI BIOS explained*. Retrieved from PC Pro: http://www.pcpro.co.uk/features/381565/uefi-bios-explained
- Jackson, J. (2013, February 28). *UEFI Presidnet: We need more key providers*. Retrieved from NetworkWorld: http://www.networkworld.com/research/2013/022813-uefi-president-we-need-more-267245.html?page=3
- James, J. (2011, October 19). *10 things you should know about UEFI*. Retrieved from TechRepublic: http://www.techrepublic.com/blog/10-things/10-things-you-should-know-about-uefi/
- Microsoft. (2013, April 4). *Windows UEFI Firmware Update Platform*. Retrieved from Windows: http://download.microsoft.com/download/5/F/5/5F5D16CD-2530-4289-8019-94C6A20BED3C/windows-uefi-firmware-update-platform.docx
- Minnowboard.org. (n.d.). *Design Goals*. Retrieved from minnowboard.org: http://www.minnowboard.org/design-goals/
- The UEFI Forum. (2013). UEFI Specification, Version 2.4, Chapter 1.0. 59. Retrieved from http://uefi.org/specs/access
- UEFI Forum Industry Communications Work Group. (2011, March 9). *Evaluating UEFI using Commercially Available Platforms and Solutions.* Retrieved from uefi.org: http://www.uefi.org/sites/default/files/resources/UEFI_EvaluationPlatforms_2012_03.pdf
- Wei, D. (2013, September 26). *UEFI – A New Opportunity for Preboot Firmware on ARM-based Systems*. Retrieved from ARM Connected Community Blog: http://community.arm.com/groups/processors/blog/2013/09/26/uefi-a-new-opportunity-for-preboot-firmware-on-arm-based-systems
- Wei, D. (2013, September 26). *UEFI – A New Opportunity for Preboot Firmware on ARM-based Systems, Part 2*. Retrieved from ARM Connected Community Blog: http://community.arm.com/groups/processors/blog/2013/09/26/uefi-a-new-opportunity-for-preboot-firmware-on-arm-based-systems-part-2