Decoding UEFI Firmware

Unraveling the Intricacies of System Firmware, its Ecosystem and Supply Chain

Authors:

Richard Wilkins, Ph.D.
Principal Technology Liaison
Phoenix Technologies
Dick_Wilkins@phoenix.com

Brian Mullen
Director of Product Security
AMI
briam@ami.com

Tim Lewis
CTO
Insyde Software
tim.lewis@insyde.com

Dong Wei, MBA
Lead Standards Architect and Fellow
Arm
Dong.Wei@arm.com

William Keown
Development Engineer
Lenovo
wkeown@lenovo.com

Vincent Zimmer
Senior Principal Engineer, Firmware
Intel
vincent.zimmer@intel.com

August 2023

Understanding System Firmware and its Relationship to UEFI

In recent years, the industry discussions and press reports related to “Platform Firmware”, “System Firmware”, or “Host Firmware” and its flaws and vulnerabilities refer to the firmware as “UEFI”. That is an easy leap to make as system firmware complying with the UEFI standard has largely replaced BIOS (Basic I/O System) in x86 personal computers and servers. UEFI, as an interface standard, is also widely adopted on Arm-based servers, DPU/IPUs, edge and IoT devices even though these systems never used BIOS. In addition, the UEFI interface standard is also supported on RISC-V and Loongson-based systems. Therefore, system firmware complying with the UEFI standard is used on a significant range of devices with computing and networking capabilities. Replacing one four letter acronym with another is an easy thing to do. The problem is that the system firmware that initializes hardware and loads software in these systems is very complex, as will be described below. It is a misnomer to apply the UEFI label to all system firmware implementations and doing so drives confusion and misunderstandings with end users.

This is not to say that implementation flaws and design vulnerabilities are not a big deal. Compromised firmware can have a serious effect on the operation and security of systems, and it may be possible to gain malware persistence that resists system resets and software reinstallation. This paper will not attempt to excuse or minimize these effects. It attempts to educate readers about the complexities involved and show that there isn’t a silver bullet of “following best practices”, or similar, that will provide a simple solution to “just fix it”.

Introduction

What is “UEFI”? The Unified Extensible Firmware Interface standard is an international standard describing a consistent way for firmware on a computing platform to interact with operating systems that are loaded into memory by it. It has various security features including Secure Boot, Secure Update, and others. The standard is maintained by the UEFI Forum, a nonprofit industry standards body, with an open membership. UEFI, as the name describes, defines the interface, not the implementation. There are many different system firmware implementations out there that are complying with this UEFI standard interface.

A Very Brief History – BIOS was created in the late 1970s to provide load and basic I/O services to operating systems on the original x86 PCs. It became a de facto standard for these x86 PCs and servers. In the late 1990s, new and more complex processor types were becoming available where continued use of BIOS, that is tied to the x86 architecture, just did not make sense. The Extensible Firmware Interface design was created to replace the way BIOS and OSes communicate on systems with Intel CPUs. Eventually, the UEFI Forum was formed in 2005 to make the standard interface more open, broaden its usage, and allow the system firmware community to participate in maintenance and further development of the standard.

The Specifications

The UEFI Forum actively maintains several interrelated specifications.

a. PI Specification (Platform Initialization)

   Originally catering exclusively to Intel CPU architectures, the PI Spec has recently expanded to embrace Arm and anticipated RISC-V architectures. This specification delineates the process of initializing a platform, spanning from power-on to operating system load. PI provides one possible implementation of UEFI with the UEFI DXE components providing the 'kernel' or core services found in the UEFI specification.

   Note: For Arm A-profile based systems, *Trusted Firmware* is used for the initial platform initialization phase. The next stage of the platform initialization may, or may not, be PI Specification compliant depending on the implementations, even if the system firmware is UEFI compliant. Trusted Firmware also provides additional interfaces to the operating system that are Arm-specific and not governed by the PI, UEFI or ACPI specifications.

b. UEFI Specification (Unified Extensible Firmware Interface)

   The UEFI specification outlines an interface for firmware to interact with the operating system and the boot drivers, incorporating elements such as Secure Boot and Secure Update to bolster security. Managed by the UEFI Forum, this specification defines the interface only, avoiding prescription of specific implementations. **Note:** While large in scope, much of the UEFI specification is optional and vendors may choose what portions of it are needed for their systems.

c. ACPI Specification (Advanced Configuration and Power Interface)

   The ACPI specification defines power management, enumeration of devices on buses without standard discovery mechanism, and configuration interfaces. It encapsulates various aspects of system management, including devices, power control, facilitating hardware abstraction for the operating system.

   A popular alternative to ACPI, especially on embedded devices, is Device Tree, providing description of the hardware, but not a control abstraction to the operating system.

Implementations

As the UEFI Specifications do not specify an implementation, only an interface, several implementations have been developed.

1. Open Source

- **Tianocore** is a community-driven open-source project that builds upon the EDK II codebase. It offers a code repository for UEFI firmware development, including support for new hardware standards, features, and enhancements. Tianocore has gained popularity for its versatility and robustness.

- **UBoot** is a popular open-source bootloader commonly used in embedded systems and devices. It provides the initial boot and setup functionality for a system, loading and executing the operating system or other software components.

  Starting with its 2021.04 release, UEFI interface is added to the UBoot project, enabling UBoot to support OS boot loaders such as grub, and to support UEFI-based Secure Boot and Secure Firmware Update mechanisms.

- **LinuxBoot** is another popular open-source bootloader, currently used in server systems deployed by some of the cloud service providers. LinuxBoot uses Linux as the system firmware implementation, enabling Linux drivers for the boot devices to be used in the boot environment. LinuxBoot may or may not use Tianocore EDK II code to assist the passing of the ACPI information as well as some UEFI runtime services to the operating system. LinuxBoot currently does not officially support the UEFI boot services to the operating system as it kexec's to Linux directly. However, the LinuxBoot community has been interested and is working on adding a UEFI payload to support the UEFI boot services such that the full UEFI interface can be supported by LinuxBoot as well to allow the other generic operating systems to work as well. In fact, when this is properly done, the operating system should not care or know whether the underline implementation is Tianocore EDK II or LinuxBoot.

- **coreboot** is an open-source firmware project for many system architectures. It provides similar capabilities to UEFI PI PEI with its romstage and UEFI PI DXE with its ramstage. In some cases, it may also provide UEFI defined interfaces via its payload mechanism, using things like the UEFI Payload Package of EDKII.

2. Proprietary implementations that can be licensed from Independent Firmware Vendors (IFVs). These implementations may be based on one or more of the above open-source projects.
    a. Phoenix SecureCore
    b. AMI Aptio V
    c. Insyde's InsydeH2O

3. Proprietary system firmware implementations, that may, or may not, be fully conformant to the UEFI specification.
    a. Apple EFI - Apple's EFI firmware on MacBooks with Intel Silicon, while unique to its ecosystem, shares commonalities with standard UEFI implementations.

b. Historically, there were many other implementations. For example, Intel had UEFI layered on top of BIOS. HP had UEFI layered on top of two different proprietary codebases.
c. There are others not captured here.

This illustrates that, while UEFI is a standard interface definition, there can be many different implementations. It is not one size fit all. The system firmware that initializes hardware and loads software in these systems is indeed very complex. It is not just one thing named “UEFI”. In fact, there is no “one thing” that is UEFI.

Beyond the Surface: Other Components in the Image

As described above, there are many implementations of system firmware that have been developed from many source roots. In addition to that, all these firmware systems typically pull code from other sources, open-source and proprietary, some in source code format and some as pre-compiled binary “blobs”.

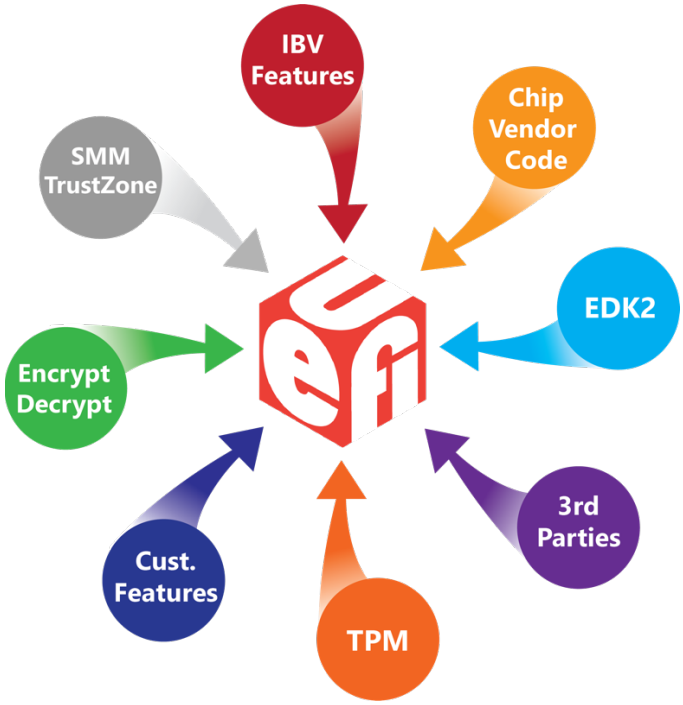


Figure 1 – Some of the many components & elements of UEFI based firmware.

1. **Chip Vendor Code**
   Chip vendors (Silicon Providers or SiPs) contribute essential low-level code to enable hardware communication and functionality.

2. **Open-Source Implementations**
   Code implementing typical UEFI requirements and features.

3. **3rd Party Binary Blobs or Source Modules**
   Firmware may include unchanged binary blobs or source modules from third-party vendors, potentially introducing vulnerabilities.

4. **Trusted Platform Module (TPM) Support**
   Trusted Platform Module (TPM) support, whether from dedicated TPM vendors or firmware based TPM (fTPM) code, contributes cryptographic and security features. Interfaces to the TPM are maintained by a related standards body called the Trusted Computing Group (TCG). The latter publishes the UEFI APIs to abstract the TPM.

5. **Customer Features developed by OEMs**
   Firmware allows for customization and introduction of unique features, although the security implications need meticulous consideration.

6. **Encryption/Decryption and Hashing Support**
   Encryption capabilities integrated into the firmware bolster data security and integrity. These frequently come from open-source projects like OpenSSL.

7. **Special Processor Mode Support (SMM/TrustZone)**
   Special modes like System Management Mode (SMM) or ARM TrustZone introduce isolated execution environments, enhancing security.

8. **Binary Blobs or Source Modules**
   Firmware may include unchanged binary blobs or source modules from third-party vendors, potentially introducing vulnerabilities.

9. **IFV/IBV Added Value Customizations**
   Features added by Independent Firmware Vendors (IFVs/IBVs).

10. **Device Firmware**
    There may be other Device Firmware, or the plurality of embedded firmware that executes on I/O devices, SOC microcontrollers, embedded controllers, and baseboard management controllers (BMCs) on a modern client system. A single system firmware image may include many instances of ‘device firmware’.

Supply Chain Dynamics

When a platform enters production, the UEFI compliant firmware that powers on that device is the collaborative product of numerous industry partners. Each partner’s contribution is passe

on to the next until the production platform's final UEFI compliant firmware image is finished. The UEFI specification makes this work, allowing each partner to add the pieces that are closely related to their specialty. There are many variations, with some partners taking on multiple roles.

Figure 2 – The general Firmware Supply Chain

Life for a UEFI compliant firmware image generally starts as a curated combination of open-source components (like Tianocore/EDK2), most often includes reference code from a silicon vendor (SiP), and depending on the development model, may include framework and/or feature modules from an IFV/IBV (independent Firmware/BIOS Vendor) that are targeted at specific reference pieces of hardware. This reference version, following the UEFI standards, allows a single OS image like those found in Windows or Linux to run on the amazing diversity of CPUs, silicon and platforms manufactured each year that run on UEFI-based firmware.

If involved in the supply chain, an ODM (Original Design Manufacturer) usually takes the upstream FW from an IBV/IFV or OEM (Original Equipment Manufacturer) development team and customizes it to meet the requirements of the specific platform and the requirements of the OEM.

The flexibility of the UEFI standard has helped this dynamic ecosystem evolve by supporting this model, but it also contributes to the difficulty of responding to security vulnerabilities. Each partner in the supply chain spends time evaluating the vulnerability, finding a mitigation, applying, and testing that mitigation to all the platforms they are working with, and then distributing a fixed version to the next partner.

> **This takes time**. Time is the malware author's friend. There is a balance between giving all partners a reasonable time to create fixes and reducing the time systems are without a patch. To make sure that one company's security fix does not become another company's zero-day attack, the UEFI Security Response Team (USRT) works with CERT/CC (Computer Emergency Response Team/ Coordination Center) to manage the disclosure process and make sure all partners in the supply chain work together to implement coordinated vulnerability disclosures.
>
> **This takes information**. The last link in the supply chain is the platform owner. This might be an individual end-user, or it might be a company's IT department. They are the least-informed partner, yet they are the partner that ultimately agrees to a security-related firmware update. To provide more visibility into the UEFI components, UEFI Forum's SBOM Sub-Team (USBT) is drafting recommendations about how partners disclose a SBOM (Software Bill of Materials) with details of each component, its version, its source code, and a unique identifier. This SBOM is delivered to the next link in the

supply chain, helping each to identify the contents of their UEFI compliant firmware and recognize when a publicly disclosed vulnerability might affect them. This aligns well with the ongoing Vulnerability Exchange (VEX) goals of the U.S. and other national governments.

The UEFI forum also reaches out to all partners in the supply chain with security best practices targeted to the unique security challenges of UEFI firmware, including threat modeling, coding standards, and training.

The UEFI community, working together, can help secure the diverse and creative ecosystem of UEFI firmware, one link in the supply chain at a time.

Vulnerability Management

In an ever-evolving landscape of technology, ensuring the security of firmware systems has become paramount. The Unified Extensible Firmware Interface (UEFI) ecosystem is a complex web of manufacturers, integrators, and developers, all collectively responsible for producing secure products. This section of this paper delves into the intricate realm of vulnerability management within the UEFI supply chain, examining the concepts of proactive and reactive vulnerability management, the responsibilities of each link in the chain, and the collaboration required to maintain the security of these critical components.

Defining Vulnerability Management: Vulnerability management is a multifaceted practice aimed at mitigating risks posed by vulnerabilities in products. This practice involves two distinct phases: proactive and reactive vulnerability management.

1. **Proactive Vulnerability Management:** The proactive phase focuses on identifying vulnerabilities before they manifest into real threats. The efficacy of proactive security can be measured using d using key performance indicator (KPI) metrics:

    - **Number of Vulnerabilities Found in Released Products:** This KPI serves as a litmus test for the robustness of proactive security practices. The lower the count of vulnerabilities discovered post-release, the stronger the proactive security measures in place.

1. **Reactive Vulnerability Management:** The reactive phase involves responding to vulnerabilities that have been detected, with an emphasis on rapid and comprehensive resolution. This phase is crucial in preventing the exploitation of vulnerabilities that manage to surface despite proactive efforts. The efficacy of an organization's reactive security or PSIRT.

    - **Speed of Vulnerability Remediation:** This KPI gauges how rapidly vulnerabilities are addressed once identified. Swift remediation is indicative of an agile and effective vulnerability management process.

The example described here is for firmware targeting typical computing devices, PCs, and Servers. UEFI compliant firmware is also used in embedded and IoT devices, appliances,

Electronic Vehicles, Autonomous Vehicles, and many other types of systems. The supply chains may be somewhat different in their details, but the complexities described here still apply.

Understanding the UEFI supply chain, as described in the previous section, is essential to grasp the complexities of vulnerability management. The supply chain comprises several crucial links, each playing a unique role:

1. **Silicon Provider (SiP)**
2. **Independent Firmware Vendor (IFV):**
3. **Original Design Manufacturers (ODMs) and Original Equipment Manufacturers (OEMs)**

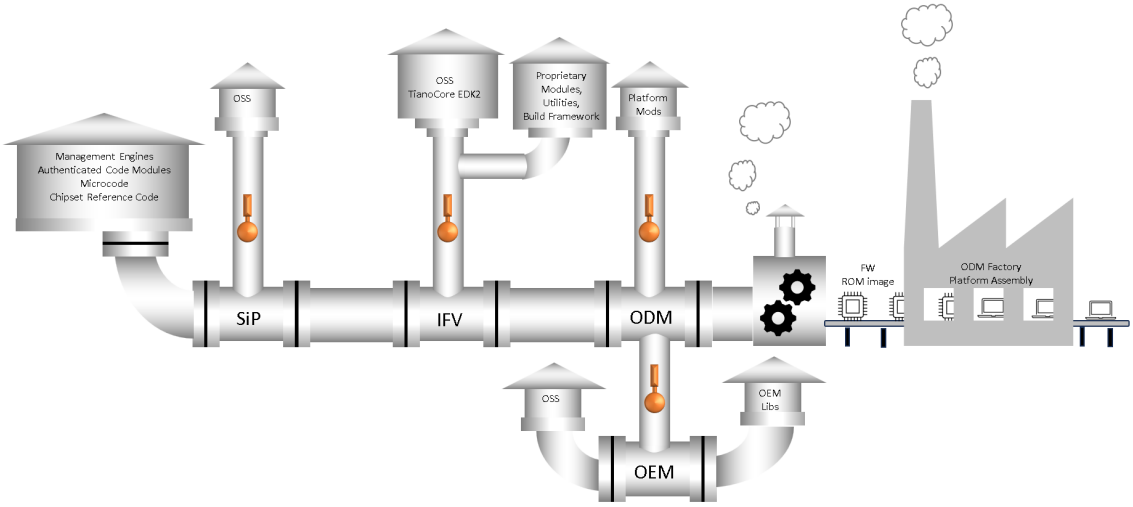


Figure 3 - Illustration of a UEFI Supply Chain

In this supply chain, each link is responsible for performing vulnerability management functions. For pro-active security, this could mean supporting various SSDLC processes, such as those covered in NIST SP 800-218, to minimize the number of vulnerabilities passed downstream to the next vendor. For reactive security, each vendor should have a team responsible for receiving and advising on security sightings. This team is typically called a PSIRT or Product Security Incident Response Team.

Each vendor's PSIRT in the supply chain is responsible for advising on security vulnerabilities which exist in their product. For example, the SiP should report all vulnerabilities in their hardware and reference code firmware including vulnerabilities against the open source they leveraged as well as the proprietary code they authored. If the next vendor in the supply chain is an IFV, the downstream vendor often relies on them to advise on all the SiP's vulnerabilities that affect the IFV product, plus all the vulnerabilities introduced by the IFV. The downstream vendor must also comprehend IFV introduced vulnerabilities that could come from OSS that they integrated or additional proprietary code that they added. When part of the supply chain, the IFV is often responsible for integrating any remediations by the SiP and passing those down the supply chain. This pattern of advising and remediating repeats for each link in the supply chain as it continues downstream.

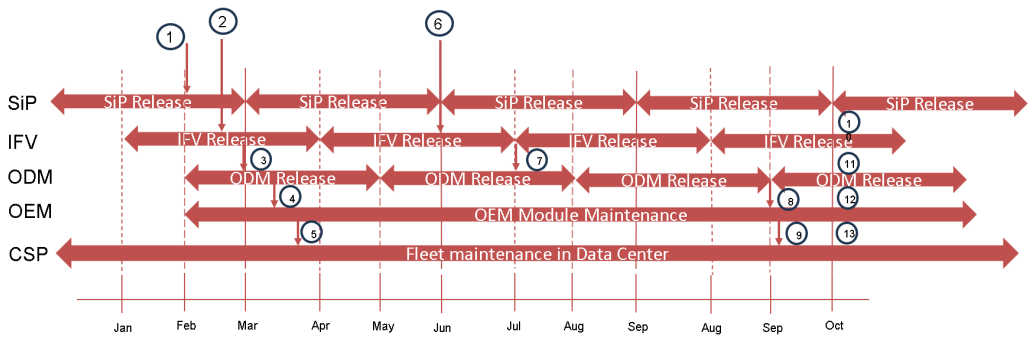
The finders of vulnerabilities in UEFI products typically report their sightings to the PSIRTs of manufacturers of the product the vulnerability was found in. This is typically the ODM or OEM. Once the sighting is received by the OEM, they must begin an investigation to determine which vendor in the supply chain is responsible for the remediation. Customarily, if the finder wants to go public with the vulnerability details, a 90-day public embargo is placed on the vulnerability details. This means that no details related to vulnerability are to be discussed outside of the finder and supply chain vendor involved in the investigation. It is important to note that all vendors affected by the vulnerability need to have their products remediated and released prior to the public embargo expiration or a zero-day vulnerability would result for vendors in the supply chain that do not have access to a fix for their exposed product. The embargo date can be negotiated with the finder if it is known ahead of time that some vendors cannot remediate their products prior to public disclosure.

During the public embargo period, each vendor in the supply integrates the fix, tests, and releases the fix project downstream. Then, the vendor repeats the process, integrates the fix, tests, and releases the fix. This pattern repeats until the whole supply chain is remediated. It should be clarified that each supply chain vendor has advised on fixes for vulnerabilities to their customers prior to a public advisory being released. These communications among supply chain partners are carried out under NDAs. This is so that those that might exploit the vulnerabilities are not made aware of them until such a time that the supply chain has already fixed the issues, and when a public disclosure is made.

In the illustration below, the SiP release cadence is every quarter (which is typical, but it can be less frequent than this). IFVs typically mirror the SiP release cadence and issue releases once a quarter (at most). ODMs typically follow the same pattern. It should be noted that there is a concept of EOL or end of life products in the UEFI firmware supply chain. These are products that have been determined to be too old or simply don't warrant the cost to maintain. To give an example, a scenario exists where the IFV could still be maintaining a product and issuing remediations, but the ODM has EOLed their derived product. In this case, remediations would never propagate past the IFV, leaving the OEMs, CSPs and retail users in a "vulnerable" position.

It was mentioned that there is a customary 90-day public embargo for vulnerabilities. Now, since readers have a basic understanding of the UEFI supply chain, let's look at the feasibility of the 90-day embargo for this supply chain.

In this illustration, the SiP receives the vulnerability sighting on February 1st.



1) It immediately begins an investigation into which product is determined to be affected. The SiP issues an advisory to the IFV within a week (which is a reasonable response time) of the sighting.
2) The IFV performs an investigation and determines its product is also affected so an IFV advisory is issued to the ODM.
3) The ODM follows suit by issuing an advisory to the OEM after it determines that it has inherited the same vulnerability.
4) And finally, the OEM issues an advisory to all its CSP customers that the issue exists, and a fix is being worked on.
5) The SiP addresses the issue in the first release the patch is available in.
6) The SiP may have had the patch earlier but typically SiPs only provide remediations in releases which are made available on a quarterly basis. After the SiP issues the patch, the IFV integrates the fix. Typically, this integration takes a minimum of 3 weeks. In this illustration, it can be observed that the IFV is able to integrate the changes just in time to have the fix available in the release issued at point 7.
7) Unfortunately, the remediation was made available to the ODM at a bad time as there is not enough time to include the remediation in the next release so it must be targeted for the subsequent release which is almost four months away and at that time it is made available to the OEM (to put up on their website so customers can download it).
8) After the OEM makes the UEFI FW release available, the CSP will download the image and update all the servers in the data center.
9) The next step is public disclosure. Given the organic sequence of events detailed in the illustration above, October 1st, ~300 days after the initial sighting, seems like an appropriate time to go public. In this case, a 300-day embargo period seems reasonable for the UEFI supply chain as opposed to the arbitrary 90-day embargo period.
10) Steps (10), (11), (12), and (13) represent all supply chain partners issuing public advisories after the 300-day embargo period expires.

This example demonstrates how complex and time-consuming vulnerability management is in the firmware ecosystem. It shows that arbitrary public vulnerability information embargo periods do not work well in this complex supply chain.

Open-source vulnerability management

It should be noted that in the above flow, there are open-source inputs into the supply chain elements, including “OSS/Tianocore EDK2.” Since the UEFI Forum does not endorse any specific UEFI implementation, the Tianocore community maintains its own vulnerability process via its infosec team. Open-source issues are managed through this team and it, in turn, can coordinate updates to the EDKII upstream, especially aligned with the EDKII ‘stable tags’ upon which many of the SiP, IFV, OEM, and others base their downstream deliverables. This process is complementary to the USRT efforts to handle UEFI specification issues and coordinate the various SiP, OEM, ODM, and IFV PSIRT organizations. Other open-source and proprietary inputs to the supply chain will have their own vulnerability handling processes.

Conclusion

Platform firmware's complexity transcends the mere label “UEFI”. It encompasses a plethora of standards, implementations, and supply chain dynamics. While vulnerabilities in firmware are a pressing concern, a nuanced understanding of the intricate relationships and dependencies within this ecosystem is imperative. A comprehensive approach to firmware security necessitates collaboration among industry stakeholders, continued refinement of best practices, and an unceasing commitment to adapt and evolve alongside the ever-changing landscape of platform firmware.

The UEFI Forum, its security response team (USRT), security sub-team (USST), SBOM Sub-team (USBT), specification working groups, and Industry Communications Working Group (ICWG) are dedicated to making UEFI compliant firmware as secure as possible. They are also dedicated to aiding members of the supply chain to respond to vulnerabilities in a timely manner and communicating with stakeholders to improve to state of the art in firmware.

About UEFI Forum

The UEFI Forum, a nonprofit industry standards body, 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 350 industry-leading technology companies, UEFI Forum specifications promote business and technological efficiency, improve performance and security, facilitate interoperability between devices, platforms, and systems, and comply with next-generation technologies.

The Forum’s spheres of input and influence are large: Membership represents major voices from all players in the industry—open source to proprietary technology, hardware to software, mobile to stationary devices. The Forum collaborates with other standards groups that are essential to computing.

For More Information

Please visit www.uefi.org for more information about UEFI, including current specifications and membership options.

# # #