iJRASET
International Journal For Research in Applied Science and Engineering Technology

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Volume: 9 **Issue:** V **Month of publication:** May 2021

DOI: https://doi.org/10.22214/ijraset.2021.34358

www.ijraset.com

Call: 08813907089 | E-mail ID: ijraset@gmail.com

International Journal for Research in Applied Science & Engineering Technology (IJRASET)
ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.429
Volume 9 Issue V May 2021- Available at www.ijraset.com

Implementation of Harmonized Capsule Update

Gururaj Pandhari[1], M. Anantha Sunil[2]
[1]*M. Tech (Electronics),* [2]*Assistant Professor, BMS College of Engineering, Bangalore*

Abstract: ***This paper deals with updating the firmware using a capsule. Firmware is a specific class of computer software that provides the low-level control for a device's specific hardware. It is held in non-volatile memory devices such as ROM, EPROM, EEPROM and Flash memory. Changing the firmware of a device was rarely or never done during its lifetime in the past but is nowadays a common procedure; some firmware memory devices are permanently installed and cannot be changed after manufacture. Common reasons for updating firmware include fixing bugs or adding features to the device. Capsule Updates are how UEFI-based firmware updates itself. For security considerations, the flash device is locked during the system boot. The operating system sends a firmware image (capsule) to the system and lets the system update the flash device before the flash device is locked in next boot. The existing firmware update system is based on Romley design, which has got many disadvantages like SMI dependency, no option to preserve NVRAM, etc. The proposed solution of update (Harmonized) is governed by Firmware Management Protocol (FMP). Few headers are added onto the capsule and different boot modes are set for the normal path and update path. After successful validation process, the image is passed to the device and firmware is updated.***
Keywords: Firmware, flash, capsule, UEFI, SMI, FMP

I. INTRODUCTION

Firmware or microcode is basically a set of instructions needed for certain devices to perform the tasks that they were made for. In simpler words, it is the programming that is responsible to run the machine. Unlike software, the firmware is not created to perform specific functions on the hardware; rather it is used to carry out the core functions of the hardware. Firmware can either provide a standardized operating environment for more complex device software (allowing more hardware-independence), or, for less complex devices, act as the device's complete operating system, performing all control, monitoring and data manipulation functions. Typical examples of devices containing firmware are embedded systems, consumer appliances, computers, computer peripherals, and others. Almost all electronic devices beyond the simplest contain some firmware.

Basic Input/output system (BIOS) is the dominant standard which defines a firmware interface. The main responsibility of BIOS is to set up the hardware, load and start an operating system (OS). When the computer boots, BIOS initializes and identifies system devices including the video cards, keyboard, mouse, hard disk and other hardware. It then locates software held on a boot device, for example, a hard disk or CD/DVD – and loads and executes that software giving it control of the computer. This process is also called as 'Booting' or 'Boot strapping'. Legacy BIOS is based upon Intel's original 16-bit architecture, commonly referred to as 8086 architecture and as technology advanced, Intel extended that 8086 architecture from 16 to 32 bit.

UEFI was developed as a replacement for Legacy BIOS to streamline the booting process and act as the interface between operating system and its platform firmware. It not only replaces most BIOS functions, but also offers a rich extensible pre-OS environment with advanced boot and runtime services. UEFI architecture allows users to execute applications on a command line interface. It has intrinsic networking capabilities and is designed to support multi-processors systems. The UEFI is responsible for setting up or producing the services and protocols that can be consumed by other software or firmware components.

Flashing involves the overwriting of existing firmware or data, contained in EEPROM or flash memory module present in an electronic device, with new data. This can be done to upgrade a device or to change the provider of a service associated with the function of the device, such as changing from one mobile phone service provider to another or installing a new operating system. If firmware is upgradable, it is often done via a program from the provider, and will often allow the old firmware to be saved before upgrading so it can be reverted to if the process fails, or if the newer version performs worse. As an alternative to vendor tools, open source alternatives have been developed such as flash ROM. Before updating the firmware, we need to make sure that the update is for the exact device model that you own. If we apply an update that is intended for a similar-but-different model, the device would be at a serious risk of becoming non-operational. In such cases, the old microcode will be overwritten with the new programs that are incompatible with the device model, so installing such update will brick the device.

Capsule Update is a more secure way to update the firmware using a firmware image where the capsule is a file that contains a firmware update image. A security benefit is that the capsule update method only works if the existing image on the target system has the same signature as used with to build the original firmware image currently on the target system. When the Capsule update method is enabled, The EDK II Build will generate a firmware image in the form of a capsule or file (.cap). The EDK II Build process will also create a UEFI Shell application called "CapsuleApp.efi" that will use the capsule file. When the Capsule update method is enabled, the verification method can also be selected for how a firmware update image is verified, or authenticated, before it is used. Image authentication is typically required for shipping products to make sure only valid firmware update images are applied to a specific product. When the UEFI Shell application, CapsuleApp.efi, is invoked with a .cap file image, the target system will reboot twice; once to authenticate the firmware update image & update the flash device and second time to boot using the updated flash device.

II. IMPLEMENTATION

Capsule Updates are how UEFI-based firmware updates itself. For security considerations, the flash device is locked during the system boot. Signed capsules help assure that the correct update is being applied to the platform. Using signed images with UEFI Capsule allows an OS-agnostic process to provide verified firmware updates, utilizing root-of-trust established by the firmware. As such, it might not be possible to update the flash device directly in an operating system (as shown in Fig. 1).

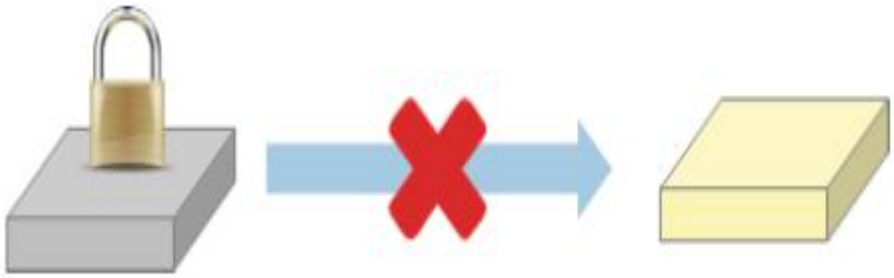


Fig. 1 Direct Flash Update

As an alternative, one possible way is that the operating system sends a firmware image to the system and lets the system update the flash device before the flash device is locked in next boot (as shown in Fig. 2). This firmware image is called a 'capsule'. The UEFI specification defines a set of capsule services to let operating system pass the information to the firmware. The UEFI specification also defines the capsule image format for the EFI_FIRMWARE_MANAGEMENT_PROTOCOL (FMP).

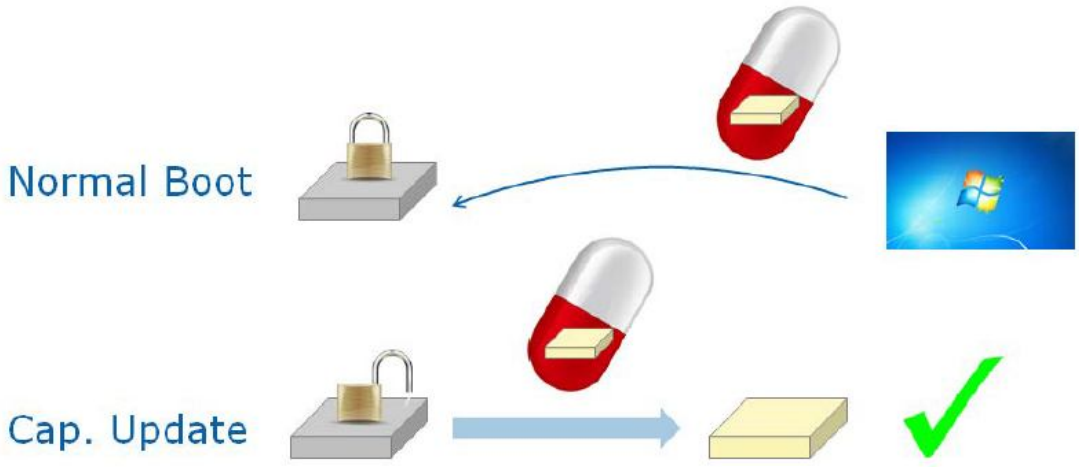


Fig. 2 UEFI Capsule Based Firmware Update

EDKII provides the sample drivers to performance the signed system firmware update via the capsule interface. It follows the NIST standard, Microsoft design and the UEFI specification.

ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.429
Volume 9 Issue V May 2021- Available at www.ijraset.com

A. UEFI specification

The UEFI specification defines UEFI capsule services, EFI_FIRMWARE_MANAGEMENT_PROTOCOL, FMP capsule format and EFI System Resource Table (ESRT) to support system firmware and device firmware update. The platform may consume a system FMP and a device FMP protocol to get the firmware image information and report the ESRT to a UEFI OS.

An OS agent may call the UEFI service Update Capsule () to pass the capsule image from OS to the firmware. Based upon the capsule flags, the firmware may process the capsule image immediately, or the firmware may reset the system and process the capsule image on the next boot.

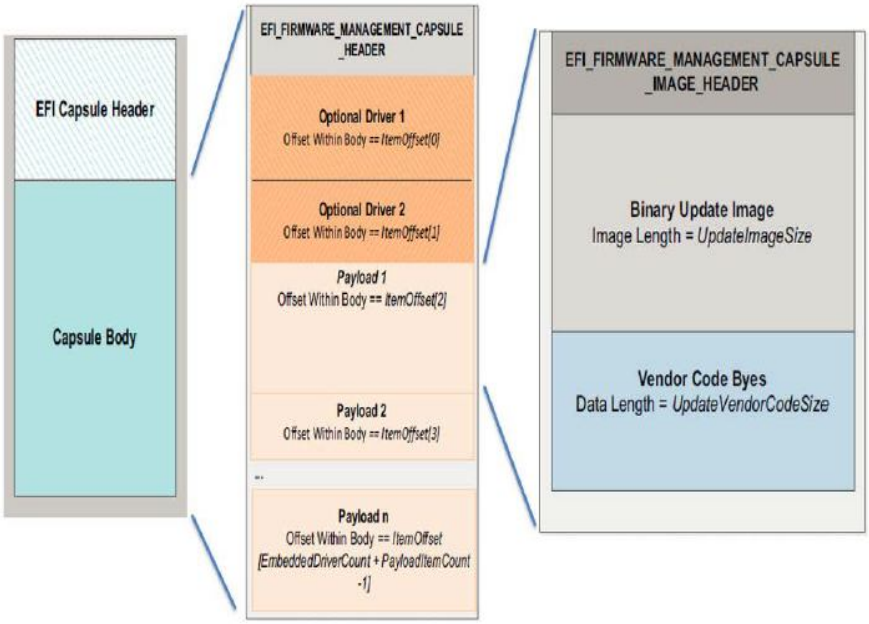


Fig. 3 UEFI FMP Capsule Format

The FMP capsule image format is shown in figure 3. It contains the update FMP drivers and the FMP payloads. The FMP payload contains the binary update image and optional vendor code. The platform may consume a FMP protocol to update the firmware image.

B. Capsule format and update process

UEFI capsules that provide firmware update payloads and composed of the following sections:

1) EFI_CAPSULE_HEADER- UEFI capsules that contain firmware update payloads must have the 'CapsuleGuid' field of the EFI_CAPSULE_HEADER

2) EFI_FIRMWARE_MANAGEMENT_CAPSULE_HEADER- The 'UpdateImageTypeId' field of the EFI_FIRMWARE_MANAGEMENT_CAPSULE_HEADER is set to a GUID value that uniquely identifies the updatable firmware component.

3) EFI_FIRMWARE_IMAGE_AUTHENTICATION- UEFI capsules that contain firmware update payloads must contain this data structure with 'AuthInfo.CertType' field set to EFI_CERT_TYPE_PKCS7_GUID to specify that 'AuthInfo.CertData' is a PKCS7 certificate. The platform must perform a PCKS7 authentication operation to verify that the firmware image in the UEFI capsule hasn't been modified or corrupted.

4) FMP_PAYLOAD_HEADER- Payload is the firmware image that is intended to be written to a firmware storage device. The first version of this data structure provides the 'FwVersion' and 'LowestSupportedVersion' values for the firmware payload.

Fig. 4b shows the different phases of firmware execution. If the boot mode is set to BOOT_ON_FLASH_UPDATE, it follows update path, else the normal path is followed.

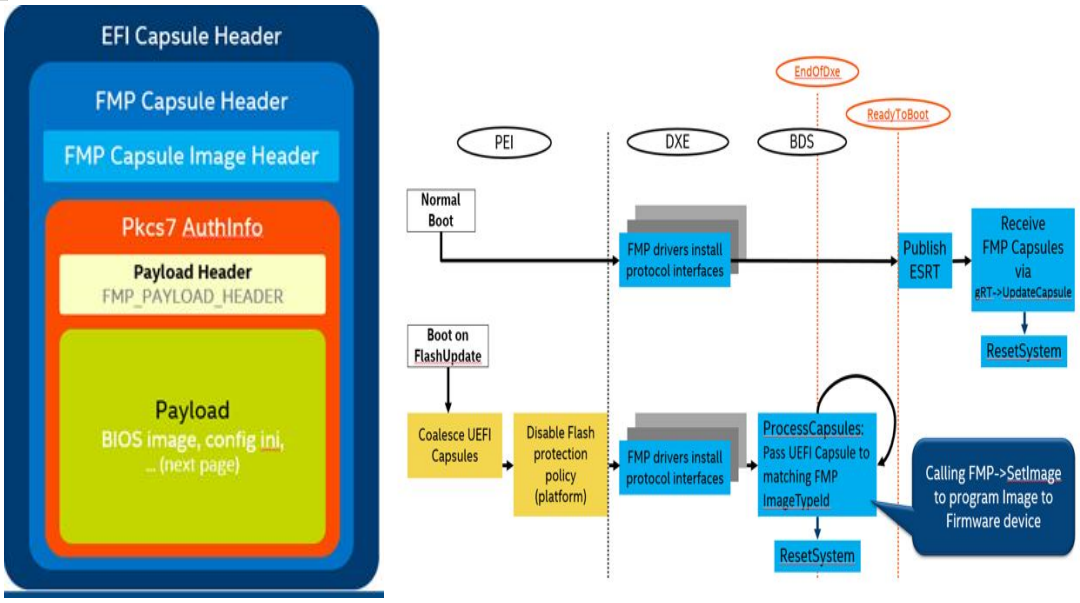


Fig. 4 a) Capsule format b) Update process

C. Capsule Update General Flow

The capsule update general flow is as shown in figure 5.

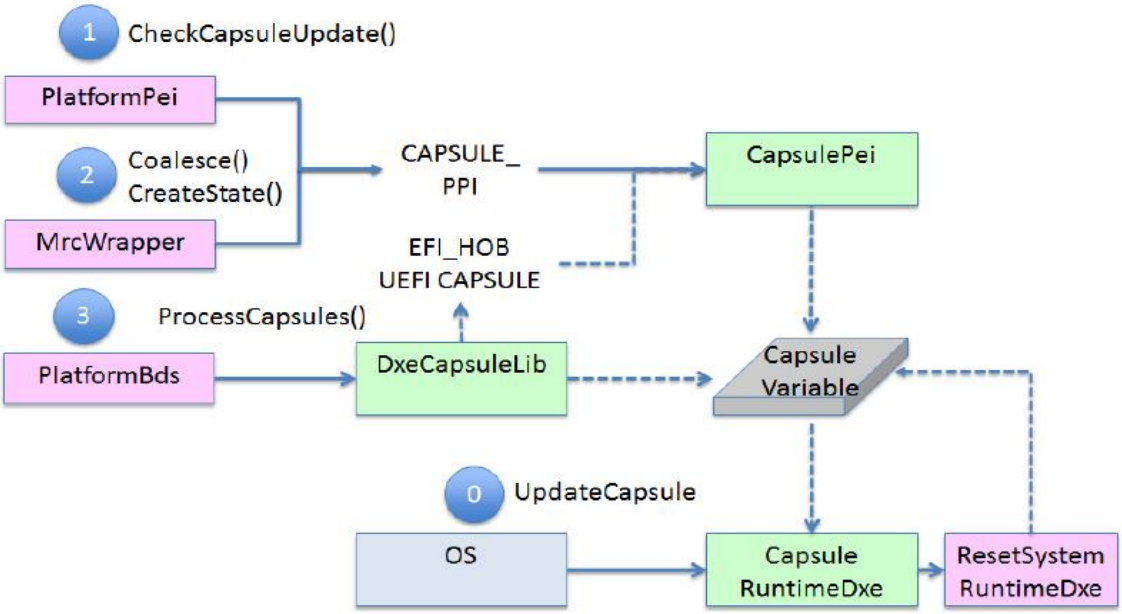


Fig. 5 Capsule Update General Flow

1) In a UEFI OS, an OS module calls Capsule runtime service UpdateCapsule() to pass the capsule image to the firmware with the reset capability. After that, the system resets. On the next boot, a platform PEI calls CheckCapsuleUpdate() to determine if a capsule needs to be processed. The CapulePei module will read the L"CapsuleUpdateData" variable to determine if there are capsules in the memory. If the condition is true, this PEIM module calls SetBootMode (BOOT_ON_FLASH_UPDATE**).**

2) After the Memory Reference Code (MRC) module initializes system memory, the platform PEI calls Coalesce() to determine the current location of the various capsule fragments and coalesce them into a contiguous region of system memory. Then the platform PEI module calls InstallPeiMemory(), to register the found memory configuration with the PEI Foundation. The contiguous region for capsule must be excluded in the PEI memory. Finally, the platform PEI module calls CreateState() to copy the capsules into PEI memory and to create a EFI_HOB_UEFI_CAPSULE. At this point, the following PEI/DXE/SMM modules can know the location of the coalesced UEFI capsule memory pages.

3) After the platform enters the BDS phase, the BDS detects boot mode. If the current boot mode is BOOT_ON_FLASH_UPDATE, the BDS calls CapsuleLib:ProcessCapsules() to process capsule images. BDS should call ProcessCapsules() twice. The first call must be before EndOfDxe event, because the system flash part is locked after EndOfDxe event. The system capsule must be processed in first call. If a device capsule FMP protocol is produced and the device capsule FMP has zero EmbeddedDriverCount, the device capsule is also processed.

4) ProcessCapsules() parses CAPSULE_HOB and processes the satisfied capsule images one by one. If a Windows UX capsule exists, it is processed first. If the capsule image is a UEFI defined FMP capsule, the ProcessFmpCapsuleImage() is called. ProcessFmpCapsuleImage() locates all FMP protocols, calls Fmp-> GetImageInfo() to get the firmware image information and compares with the FMP capsule. Only if the ImageTypeId, ImageIndex and HardwareInstance in FMP protocol and FMP capsule matched, then ProcessFmpCapsuleImage() calls Fmp->SetImage() to perform the firmware update. If the update returns success, ProcessFmpCapsuleImage() records the information in the capsule status variable so that the same image will not be processed twice. After EndOfDxe event and after ConnectAll(), the BDS calls ProcessCapsules() again. At that time, all device capsule FMP protocols are exposed.

5) If a capsule is processed in the first call, it is not processed in the second call. Alternately, if a capsule is not processed in the first call, it is processed in the second call. After all the capsules are processed, the system may reset if the reset is required by one of the capsules processed in the first call or second call. Per the NIST guideline, the flash update operation must happen in a secure environment. It could be in firmware boot phase before EndOfDxe, or in SMM. The EDKII Capsule update solution chooses to update the system firmware before EndOfDxe because the flash region remains open until the EndOfDxe. This latter usage is sometimes called 'temporal isolation' since no 3rd party code should execute prior to the EndOfDxe. Only system board vendor PI modules and drives should execute.

III.RESULTS

The build process generated a capsule update image (.cap file) along with the UEFI application CapsuleApp.efi. The .cap file and CapsuleApp.efi are copied to USB thumb drive. Then, system is booted to UEFI Shell and CapsuleApp.efi is used with .cap signed capsule file. OS run-time update application stores capsule in DRAM or HDD on ESP. Once the system is rebooted, FW finds & parses update capsule and writes new BIOS FV(s) to SPI flash memory. Fig. 6 shows the command to trigger the capsule using the application.

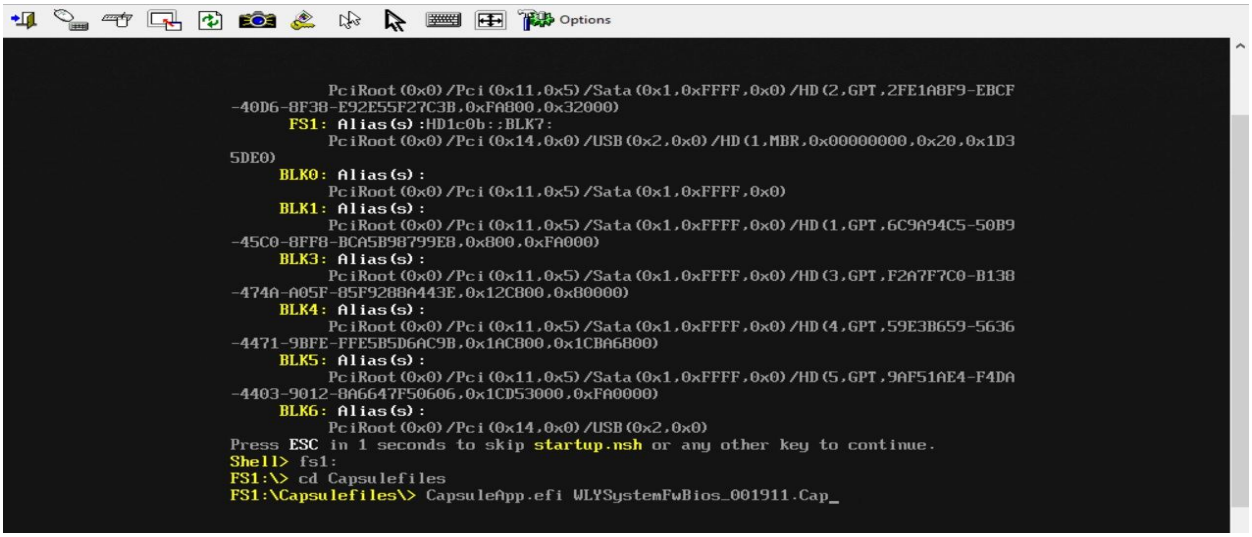


Fig. 6 Triggering of capsule

IV.CONCLUSION

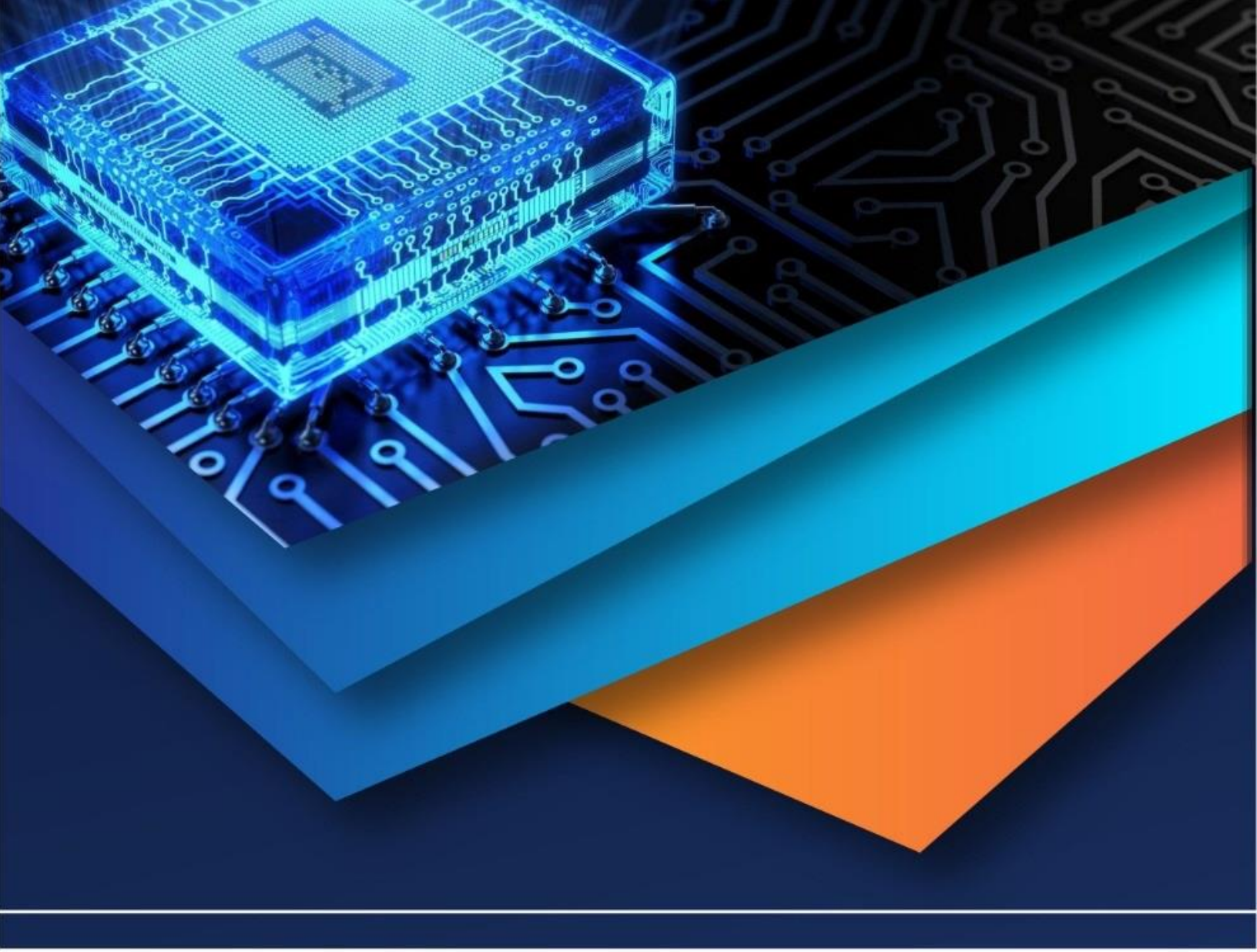
This solution only works if the existing image on the target system has the same signature as used with to build the original firmware image currently on the target system, thereby making the process very secure and less vulnerable to attacks. The UEFI Firmware Management Protocol provides an abstraction for device to provide firmware management support. With continuous advancements in technology, adding of new features becomes a common task. Hence, the proposed solution works best for updating the firmware as it is very secure and reliable.

V. ACKNOWLEDGMENT

This work was carried out during the Internship at Intel Tech India Pvt. Ltd., Bangalore. The author[1] would like to thank BMS College of Engineering and Intel Tech India Pvt. Ltd. for the Internship opportunity and the enthusiastic support.

REFERENCES

[1] Tianocore Community, "PI-Boot-Flow",[Online]. Available: https://github.com/tianocore/tianocore.github.io/wiki/PI-Boot-Flow.
[2] Vincent Zimmer and Jiewan Yao, “A Tour Beyond Capsule Update and Recovery in EDKII”, Dec 2016
[3] Tianocore Community, “Capsule_Update__Pres_gp”. [Online]. Available: https://github.com/tianocore- Presentation_Capsule_update
[4] Wikipedia, “Firmware”. [Online]. Available: https://en.wikipedia.org/wiki/Firmware
[5] eInfochips, “Understanding firmware update”. [Online]. Available: https://www.einfochips.com/blog/understanding-firmware-updates-the-whats-whys-and-hows/amp



10.22214/IJRASET

45.98

IMPACT FACTOR: 7.129

IMPACT FACTOR: 7.429

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY