2023-03-27

Created: 2023-05-01 22:18:21

Questions

- Basic Concepts `(230327.OP-TEE.SP.yaml)`
  - `[09:19]` https://github.com/OP-TEE/optee_os/pull/3973 `(OP-TEE.SP.0327.md)`
  - `[09:21]` What is StMM? `(OP-TEE.SP.0327.md)`
  - `[09:22]` What is FFA? `(OP-TEE.SP.0327.md)`
  - `[09:38]` Why are SPD and SPM mutually exclusive? `(OP-TEE.SP.0327.md)`
  - `[09:47]` What is BL32_AP_MM.fd? `(OP-TEE.SP.0327.md)`
  - `[10:13]` Does OP-TEE require UEFI? `(OP-TEE.SP.0327.md)`

Chats

- OP-TEE.SP.0327.md
  - `[09:18]` https://github.com/OP-TEE/optee_os/pull/3973 `(OP-TEE.SP.0327.md)`
  - `[09:20]` What is SPD? `(OP-TEE.SP.0327.md)`
  - `[09:20]` What is SPM? `(OP-TEE.SP.0327.md)`
  - `[09:46]` What is SP? `(OP-TEE.SP.0327.md)`
  - `[17:43]` What are EL0, EL1, EL2, S-EL0, S-EL1, S-EL2, S-EL3? `(OP-TEE.SP.0327.md)`
  - `[17:47]` Is UEFI required in ARM? `(OP-TEE.SP.0327.md)`

Q & A

Basic Concepts `(230327.OP-TEE.SP.yaml)`

Question: `[09:19]` https://github.com/OP-TEE/optee_os/pull/3973 `(OP-TEE.SP.0327.md)`

Answer:

This comment is discussing the implementation of secure variable storage for UEFI variables in OP-TEE, which is critical for enabling and protecting UEFI Secure Boot. The implementation involves adding a user mode secure partition context and support for loading Standalone MM of EDK2 into it. There are several changes made to the code, addressing comments and suggestions from a previous version, and some additional improvements.

The main points of the comment:

1. Secure variable storage for EFI variables is critical for enabling and protecting UEFI Secure Boot.
2. The implementation uses a Standalone MM of EDK2, which has several advantages, including reusing well-debugged applications, inheriting Fault Tolerant Writes (FTW) functionality, and being a better choice considering the FFA changes of the future Arm architectures.
3. A separate syscall handling is added to serve as a different kind of ABI and syscall IDs.
4. CFG_STMM_PATH is assigned the path of BL32_AP_MM.fd.
5. A repository is provided for testing, which contains necessary code to compile and test the implementation on QEMU.
6. A list of changes made since the previous version (RFC) and version 1 is provided.

Overall, the comment presents the implementation of secure variable storage for EFI variables and addresses previous comments and suggestions, providing a comprehensive overview of the changes made.

Question: `[09:21]` What is StMM? `(OP-TEE.SP.0327.md)`

Answer:

StMM stands for Standalone MM, which refers to the Standalone Memory Manager in the EDK2 (UEFI Development Kit 2) ecosystem. EDK2 is a development environment for creating UEFI (Unified Extensible Firmware Interface) firmware, and the Standalone MM is a component within this environment.

Standalone MM is designed to provide a minimal execution environment that can run independent of the traditional UEFI firmware. It offers memory management, security, and communication services to the applications running within it. This makes it a suitable option for implementing secure services, such as secure variable storage, without the need to rewrite applications as Trusted Applications (TAs) in a Trusted Execution Environment (TEE) like OP-TEE.

In the context of OP-TEE and secure variable storage, StMM can be used in cooperation with OP-TEE RPMB APIs to store UEFI variables in a secure storage, leveraging its built-in fault-tolerant write functionality to protect variables against corruption during writing.

Question: `[09:22]` What is FFA? `(OP-TEE.SP.0327.md)`

Answer:

FFA stands for the Firmware Framework for Arm, which is a specification introduced by Arm to standardize the interfaces and communication mechanisms between different firmware components running on Arm-based systems. The FFA specification is designed to enable a more secure, robust, and efficient firmware ecosystem on Arm platforms.

FFA provides a standardized interface for communication and resource sharing among different components, such as:

1. The Normal World (NW) and the Secure World (SW)
2. Different Secure Partitions (SPs) within the Secure World
3. The Secure Partition Manager (SPM) and the Secure Payload Dispatcher (SPD)

By standardizing these interfaces, FFA aims to simplify the development of secure firmware solutions and promote interoperability between different firmware components, regardless of their source or implementation. The FFA specification has been increasingly adopted in the Arm ecosystem, with components like OP-TEE and EDK2 working to integrate support for FFA-based communication and management.

Question: `[09:38]` Why are SPD and SPM mutually exclusive? `(OP-TEE.SP.0327.md)`

Answer:

SPD (Secure Payload Dispatcher) and SPM (Secure Partition Manager) are considered mutually exclusive because they represent two different approaches to managing secure environments within ARM-based systems. These approaches are not designed to be used simultaneously, as they have different roles and responsibilities in the system.

SPD is a component within ARM Trusted Firmware (ATF) that manages the lifecycle of a single Secure Payload, such as OP-TEE, and handles interactions between the Normal World (NW) and Secure World (SW). It primarily focuses on managing a single secure environment and ensuring isolation between the two worlds.

On the other hand, SPM is a more recent innovation in ARM's security architecture that manages multiple Secure Partitions (SPs) within the Secure World. It is responsible for the lifecycle management, communication, and context management of these partitions. SPM enables more flexible and granular security solutions, allowing for a standardized approach to managing multiple isolated execution environments within the Secure World.

In summary, SPD and SPM are mutually exclusive because they represent different stages in the evolution of ARM's security architecture, and they cannot be used together due to their differing roles and responsibilities in managing secure environments. While SPD focuses on managing a single secure payload, SPM is designed to manage multiple Secure Partitions, offering a more sophisticated and flexible approach to security in ARM-based systems.

Question: `[09:47]` What is BL32_AP_MM.fd? `(OP-TEE.SP.0327.md)`

Answer:

BL32_AP_MM.fd is a firmware binary file related to the Standalone Memory Manager (StMM) in the EDK2 (UEFI Development Kit 2) ecosystem. It represents the compiled StMM binary that can be loaded and executed in a secure environment, such as OP-TEE or a Secure Partition.

In the context of the implementation described in the earlier comment, CFG_STMM_PATH is assigned the path of BL32_AP_MM.fd. This configuration allows the OP-TEE environment to load the StMM binary into a secure partition context and execute it.

The Standalone Memory Manager (StMM) provides a minimal execution environment for running security-sensitive applications independent of the traditional UEFI firmware. It offers memory management, security, and communication services to the applications running within it. In the case of secure variable storage, StMM is utilized in cooperation with OP-TEE RPMB APIs to store UEFI variables in secure storage, while leveraging its built-in fault-tolerant write functionality to protect variables against corruption during writing.

Question: `[10:13]` Does OP-TEE require UEFI? `(OP-TEE.SP.0327.md)`

Answer:

No, OP-TEE (Open Portable Trusted Execution Environment) does not require UEFI (Unified Extensible Firmware Interface) to function. OP-TEE is an open-source Trusted Execution Environment (TEE) implementation that provides a secure environment for running Trusted Applications (TAs) separate from the normal operating system. It is designed to work with various types of firmware and bootloaders, not limited to UEFI.

UEFI is a modern firmware interface designed to replace the traditional BIOS (Basic Input/Output System) in personal computers and other devices. While OP-TEE can work with UEFI-based systems, it can also function with other firmware and bootloader solutions, such as U-Boot or Coreboot.

In some cases, OP-TEE and UEFI might be used together to provide a more comprehensive security solution. For example, when implementing secure variable storage for UEFI variables, OP-TEE can work alongside EDK2's Standalone Memory Manager (StMM) to store these variables in a secure storage. However, this is a specific use case, and OP-TEE itself does not inherently require UEFI to operate.

OP-TEE.SP.0327.md

Question: `[09:18]` https://github.com/OP-TEE/optee_os/pull/3973 `(OP-TEE.SP.0327.md)`

Answer:

I am an AI language model and I cannot access external websites or browse the internet. However, I can try to help you with OP-TEE (Open Portable Trusted Execution Environment) or answer questions related to the optee_os repository if you provide more information.

Question: `[09:20]` What is SPD? `(OP-TEE.SP.0327.md)`

Answer:

SPD stands for Secure Payload Dispatcher. It is a component in the ARM Trusted Firmware (ATF) framework that manages the interaction between the Normal World (NW) and the Secure World (SW) in ARM-based systems. The SPD is responsible for handling secure payloads, such as OP-TEE, and managing their life cycle, including initialization, runtime services, and context management during world switches. In a system running ARM TrustZone technology, the SPD plays a crucial role in maintaining the isolation and security of the secure payloads.

Question: `[09:20]` What is SPM? `(OP-TEE.SP.0327.md)`

Answer:

SPM stands for Secure Partition Manager. It is a component within the ARM architecture that manages Secure Partitions

(SPs) in a system. Secure Partitions are isolated execution environments that run in the Secure World, providing services to the Normal World. SPM is responsible for the life cycle management of these Secure Partitions, including initialization, communication, and context management during world switches.

The introduction of SPM is part of the ongoing evolution of ARM's security architecture, particularly with the advent of ARMv8.4-A and the ARMv9-A architectures. SPM aims to provide a more flexible and standardized approach to managing Secure Partitions, allowing for the development of more sophisticated security solutions on ARM-based systems.

Question: `[09:46]` What is SP? `(OP-TEE.SP.0327.md)`

Answer:

SP stands for Secure Partition, a concept in the ARM architecture that refers to an isolated execution environment running within the Secure World. Secure Partitions are designed to provide security-sensitive services or applications to the Normal World while maintaining isolation from other software components, including those running in the Secure World.

Each Secure Partition has its own memory, resources, and runtime environment, separated from other Secure Partitions and the Normal World. The Secure Partition Manager (SPM) is responsible for managing the lifecycle of these Secure Partitions, including their initialization, communication, and context management during world switches.

The introduction of Secure Partitions and the SPM represents an evolution in ARM's security architecture, enabling a more flexible and standardized approach to managing security solutions on ARM-based systems. This allows for the development of more granular security solutions, enhancing the overall security posture of devices running on ARM platforms.

Question: `[17:43]` What are EL0, EL1, EL2, S-EL0, S-EL1, S-EL2, S-EL3? `(OP-TEE.SP.0327.md)`

Answer:

In ARM architecture, EL0, EL1, EL2, and EL3 are Exception Levels, which represent different privilege levels for software execution. These Exception Levels help in isolating different software components running on an ARM-based system and ensure that more privileged levels can control access and resources for less privileged levels. The Exception Levels are:

1. EL0: User/Application level - The least privileged level where user applications and most of the software components run.
2. EL1: Operating system (OS) kernel level - The privilege level at which operating system kernels, such as Linux or Android, execute.
3. EL2: Hypervisor level - The privilege level for virtualization and hypervisor software, enabling the management of multiple virtual machines or OS instances.
4. EL3: Secure monitor level - The most privileged level, responsible for the secure monitor firmware, such as ARM Trusted Firmware (ATF), managing system security and TrustZone technology.

ARM architecture also supports Secure World (SW) execution, which provides an isolated environment for running security-sensitive applications, separate from the Normal World (NW). The Secure World has its own set of Exception Levels, denoted as S-EL0, S-EL1, S-EL2, and S-EL3. These levels correspond to the same privilege hierarchy as the non-secure Exception Levels but are used exclusively within the Secure World.

1. S-EL0: Secure User/Application level - The least privileged level in the Secure World, where secure user applications execute.
2. S-EL1: Secure OS level - The privilege level at which a secure operating system or Trusted Execution Environment (TEE), such as OP-TEE, runs.
3. S-EL2: Secure Hypervisor level (available only in ARMv8.4-A and later) - The privilege level for secure virtualization, managing secure virtual machines or secure OS instances.
4. S-EL3: Secure monitor level - The most privileged level in the Secure World, responsible for secure monitor firmware, such as ARM Trusted Firmware (ATF), managing system security and TrustZone technology.

These different Exception Levels allow for fine-grained control over system resources, access permissions, and isolation, ensuring that sensitive operations and data remain protected in a multi-layered security model.

Question: `[17:47]` Is UEFI required in ARM? `(OP-TEE.SP.0327.md)`

Answer:

No, UEFI (Unified Extensible Firmware Interface) is not required in ARM-based systems. UEFI is a modern firmware interface designed to replace the traditional BIOS (Basic Input/Output System) used in personal computers and other devices. While UEFI is supported on ARM platforms and can provide a standardized way to initialize and manage hardware components, it is not the only firmware solution available for ARM-based systems.

ARM-based devices can use a variety of firmware and bootloader solutions depending on the specific hardware, use case, and manufacturer preferences. Some common alternatives to UEFI include:

1. U-Boot: A widely used open-source bootloader for embedded systems that supports ARM and many other architectures. It is often used in devices like single-board computers, routers, and other embedded systems.
2. Coreboot: An open-source firmware project focused on providing a minimal and fast boot experience. It is often used in devices like Chromebooks and can be configured to work on a variety of ARM platforms.
3. Barebox: Another open-source bootloader and firmware solution, primarily targeting embedded systems and designed for speed and simplicity.

In summary, UEFI is not required for ARM-based systems. While it is an option and can offer certain advantages, there are alternative firmware and bootloader solutions available that can be used depending on the specific requirements of the device or system.
