2023-04-10

Created: 2023-05-01 22:17:45

Questions

- Qualcomm `(230407.OP-TEE.History.yaml)`
  - `[10:33]` What is relationship between QTEE and SPU? `(OP-TEE.History.0407.md)`
  - `[10:36]` When was source code of QTEE opened? `(OP-TEE.History.0407.md)`
  - `[11:03]` Does SPU rely on ARMv8? `(OP-TEE.History.0407.md)`
- Privilege Levels `(230329.TrustZone.yaml)`
  - `[14:42]` What is exception level? `(TrustZone.0329.md)`
  - `[14:43]` How does a software component know which EL it owns? `(TrustZone.0329.md)`
  - `[14:45]` Can a software component change the EL? `(TrustZone.0329.md)`
  - `[14:47]` Are ELs of TrustZone? `(TrustZone.0329.md)`
- TrustZone `(230410.Arm.yaml)`
  - `[14:50]` Which types of ARM CPU have TrustZone? `(TrustZone.0329.md)`
  - `[14:51]` Does ARM Cortex-R series have TrustZone? `(TrustZone.0329.md)`
- Boot `(230329.TrustZone.yaml)`
  - `[15:20]` What is FF-A? `(TrustZone.0329.md)`
  - `[15:21]` Does TF-A rely on FF-A? `(TrustZone.0329.md)`
  - `[15:24]` Does TF-A rely on TrustZone? `(TrustZone.0329.md)`
- FF-A, TF-A, and TrustZone `(230409.FF-A.yaml)`
  - `[15:28]` What is TF-A? `(FF-A.0409.md)`
  - `[15:29]` Please draw the relationship among TrustZone, TF-A, and FF-A. `(FF-A.0409.md)`
  - `[15:36]` What is relationship between FF-A and TF-A? `(FF-A.0409.md)`
  - `[15:37]` Could I say that TF-A refers FF-A? `(FF-A.0409.md)`
- Basic Concept `(230402.DRM.yaml)`
  - `[21:08]` What is license key? `(DRM.0402.md)`
  - `[21:10]` What if the license key is stolen? `(DRM.0402.md)`
  - `[21:11]` What is content key? `(DRM.0402.md)`
  - `[21:13]` What if the content key is stolen? `(DRM.0402.md)`
  - `[21:35]` What is identification attacker? `(DRM.0402.md)`
  - `[21:36]` What is exploitation attacker? `(DRM.0402.md)`
  - `[21:38]` Please summarize both in a table. `(DRM.0402.md)`
- TEE Identification `(230328.TEE.yaml)`
  - `[21:20]` What is TEE identification? `(TEE.0328.md)`
  - `[21:22]` Is TEE identification obversable? `(TEE.0328.md)`
  - `[21:26]` Can I use DID as TEE identification? `(TEE.0328.md)`
- Revocation of TA `(230328.TEE.yaml)`
  - `[22:07]` What is revocation for Trusted Application? `(TEE.0328.md)`
- DRM `(230327.OP-TEE.yaml)`
  - `[22:31]` Below text comes from “BKK16-201 - PlayReady OPTEE Integration with Secure Video Path” `(OP-TEE.0327.md)`
  - `[22:33]` What is secure data path? `(OP-TEE.0327.md)`
  - `[22:35]` So the video is played in secure world. Right? `(OP-TEE.0327.md)`
  - `[22:37]` How do we protect the video that displays in normal world? `(OP-TEE.0327.md)`
  - `[22:41]` What is “EME with OP-TEE on Hikey”? `(OP-TEE.0327.md)`
  - `[22:43]` What is AES OCDMI? `(OP-TEE.0327.md)`
  - `[23:01]` What is OCDM? `(OP-TEE.0327.md)`

Q & A

Qualcomm `(230407.OP–TEE.History.yaml)`

Question: `[10:33]` What is relationship between QTEE and SPU? `(OP–TEE.History.0407.md)`

Answer:

The Qualcomm Trusted Execution Environment (QTEE) and the Secure Processing Unit (SPU) are both security features implemented within Qualcomm's Snapdragon SoCs (System on a Chip) to provide hardware-based security for devices using these chips.

The QTEE leverages ARM TrustZone technology to create an isolated secure environment within the main processor. This environment ensures that sensitive data is protected and that security-related tasks are executed separately from the normal operating system, thereby reducing the attack surface for potential vulnerabilities.

The SPU, introduced in 2017, is a separate hardware component within the Snapdragon SoCs designed to handle secure tasks and protect sensitive data. It functions as a TEE and also uses ARM TrustZone technology. The SPU provides an additional layer of hardware-based security by isolating security functions from the main processor, further enhancing the security of the device.

In summary, both QTEE and SPU are security features implemented in Qualcomm's Snapdragon SoCs to provide hardware-based security. While they serve similar purposes, the SPU offers an additional layer of security by functioning as a separate, dedicated hardware component for handling security-related tasks, whereas QTEE creates a secure environment within the main processor itself.

Question: `[10:36]` When was source code of QTEE opened? `(OP–TEE.History.0407.md)`

Answer:

As of my last knowledge update in September 2021, the source code of Qualcomm's Trusted Execution Environment (QTEE) has not been made publicly available or open-source. QTEE is a proprietary technology developed by Qualcomm, and the company has not released its source code for public access or contribution.

Please note that this information may be outdated or incomplete, as I do not have access to real-time updates or knowledge beyond September 2021.

Question: `[11:03]` Does SPU rely on ARMv8? `(OP–TEE.History.0407.md)`

Answer:

Qualcomm's Secure Processing Unit (SPU) is a hardware component introduced in 2017 within their Snapdragon SoCs. While the SPU functions as a Trusted Execution Environment (TEE) and leverages ARM TrustZone technology, it is not specifically dependent on ARMv8 architecture.

However, it is worth noting that many of Qualcomm's Snapdragon SoCs, including those that feature the SPU, do utilize the ARMv8 architecture. The ARMv8 architecture provides improved performance, energy efficiency, and security features, including support for TrustZone.

In summary, while the SPU does not exclusively rely on ARMv8, it is often used in conjunction with ARMv8-based Snapdragon SoCs to provide hardware-based security features.

Privilege Levels `(230329.TrustZone.yaml)`

Question: `[14:42]` What is exception level? `(TrustZone.0329.md)`

Answer:

Exception level (EL) is a concept in the ARM architecture that defines different privilege levels for executing software. The ARMv8-A architecture, for instance, supports four exception levels: EL0, EL1, EL2, and EL3. Each level has different access rights and capabilities, which provide a mechanism for isolating different types of software and ensuring that less privileged software cannot interfere with more privileged software.

Here's a brief overview of the exception levels in ARMv8-A architecture:

- EL0: This is the least privileged level and is used for running user applications. Software at this level has limited access to system resources and cannot directly access privileged hardware operations.
- EL1: This level is used for running operating system kernels and is more privileged than EL0. Software at this level can access system resources and control hardware operations but is still restricted from certain sensitive operations.
- EL2: This level is used for running hypervisors, which manage and control virtual machines. Software at this level has more privileged access to hardware resources and can control the operation of multiple EL1 instances (virtual machines).
- EL3: This is the most privileged level and is typically used for secure monitor or firmware code, which manages the TrustZone security features. Software at this level has full control over the hardware resources and can enforce secure access policies across the entire system.

These exception levels provide a structured way to separate software components based on their required privileges and ensure a secure execution environment.

Question: `[14:43]` How does a software component know which EL it owns? `(TrustZone.0329.md)`

Answer:

A software component does not inherently "know" its exception level (EL). However, when it is executed, it runs at the EL designated by the system configuration and the control registers set by more privileged software.

Typically, software components are designed to operate at specific exception levels. The boot process and the system's firmware, which run at the highest privilege level (e.g., EL3 in ARMv8-A), are responsible for setting up the initial exception levels and transitioning to lower-privileged levels as needed. During this process, they load and configure less privileged software components (such as the hypervisor, operating system kernel, and user applications) to execute at the appropriate EL.

At runtime, a software component can determine its current exception level by querying the CurrentEL register. This register contains a field that indicates the current exception level, allowing the software to determine its current privilege level and act accordingly.

It's important to note that this query can only be performed if the software component is designed to check its own exception level, and it is assumed that the component has been correctly loaded and configured by the more privileged software in the system.

Question: `[14:45]` Can a software component change the EL? `(TrustZone.0329.md)`

Answer:

A software component can't directly change its own exception level (EL), but it can initiate a process that results in a change of EL. This typically happens through a specific event or instruction that triggers a change in privilege level, such as an exception, interrupt, or a special instruction like the "Secure Monitor Call" (SMC) in ARM systems.

For example, when an exception occurs, the processor switches to a higher EL to handle the exception. Once the exception has been dealt with, the software running at the higher EL can return to the previous, lower EL using the "Exception Return" (ERET) instruction in ARM systems or a similar mechanism in other architectures.

In the ARM architecture, a lower EL software can also request services from a higher EL software (e.g., from the normal world to the secure world in TrustZone) using the SMC instruction. The SMC instruction causes a switch to a higher EL, where the requested service is provided, and then the control is returned back to the lower EL.

Keep in mind that a software component can only initiate a change to a higher EL if it has been designed to do so and the system is configured to allow such transitions. The actual EL transition is managed by the hardware and the more privileged software components.

Question: `[14:47]` Are ELs of TrustZone? `(TrustZone.0329.md)`

Answer:

Exception Levels (ELs) are a concept in the ARM architecture, not specific to TrustZone. However, TrustZone does make use of the ELs to separate and manage privileges between the secure and non-secure worlds.

In ARMv8-A architecture, there are four ELs (EL0 to EL3). TrustZone operates on EL3 and S-EL1. EL3 is the Secure Monitor level, responsible for managing the transition between the secure and non-secure worlds. S-EL1 is used for secure operating systems or hypervisors.

In summary, while ELs are part of the ARM architecture, TrustZone does make use of them to manage privileges and ensure security between the two worlds (secure and non-secure).

TrustZone `(230410.Arm.yaml)`

Question: `[14:50]` Which types of ARM CPU have TrustZone? `(TrustZone.0329.md)`

Answer:

TrustZone technology is available in various ARM CPU cores, particularly in the ARM Cortex-A and ARM Cortex-M series.

1. ARM Cortex-A series: TrustZone is supported in many Cortex-A cores, including Cortex-A5, Cortex-A7, Cortex-A8, Cortex-A9, Cortex-A12, Cortex-A15, Cortex-A17, Cortex-A32, Cortex-A35, Cortex-A53, Cortex-A55, Cortex-A57, Cortex-A72, Cortex-A73, Cortex-A75, Cortex-A76, and Cortex-A77.
2. ARM Cortex-M series: The ARMv8-M architecture introduces TrustZone technology for Cortex-M processors. Examples of Cortex-M cores with TrustZone support include Cortex-M23 and Cortex-M33.

Keep in mind that ARM continues to develop new processor cores, and the list above might not be exhaustive. Always refer to the official ARM documentation for the most up-to-date information on specific cores and their features.

Question: `[14:51]` Does ARM Cortex-R series have TrustZone? `(TrustZone.0329.md)`

Answer:

Yes, some ARM Cortex-R series processors also have TrustZone support. TrustZone for Cortex-R processors is designed for real-time systems that require high performance, safety, and security. One example of a Cortex-R processor with TrustZone support is the Cortex-R52.

As with other series, it’s essential to consult the official ARM documentation or the specific processor datasheet to verify TrustZone support for each Cortex-R processor.

Boot `(230329.TrustZone.yaml)`

Question: `[15:20]` What is FF-A? `(TrustZone.0329.md)`

Answer:

FF-A, or Firmware Framework for Armv8-A, is a specification that defines a standard interface for Armv8-A-based systems to support secure firmware components, such as Secure Partitions and Secure Partition Managers. This framework aims to promote interoperability between different firmware components from different vendors and provides a consistent method for handling security, communication, and resource management.

FF-A enables the development of secure, modular firmware components that can be used across multiple platforms and SoCs, simplifying firmware development and reducing the need for custom solutions. It supports various use cases, including secure boot, trusted execution environments, and virtualization.

Question: `[15:21]` Does TF-A rely on FF-A? `(TrustZone.0329.md)`

Answer:

Yes, Arm Trusted Firmware-A (TF-A) relies on and implements the Firmware Framework for Armv8-A (FF-A) specification. TF-A provides a reference implementation of secure world software for Armv8-A platforms, including support for Secure Partition Manager (SPM) and Secure Partitions (SPs) as defined by the FF-A specification.

By relying on FF-A, TF-A ensures that it follows a standardized interface for secure firmware components, enabling compatibility and interoperability with various platforms and SoCs. This helps simplify firmware development and promote the use of secure, modular firmware components across different Armv8-A-based systems.

Question: `[15:24]` Does TF-A rely on TrustZone? `(TrustZone.0329.md)`

Answer:

Yes, Arm Trusted Firmware-A (TF-A) relies on and utilizes the TrustZone technology for providing a secure execution environment on Armv8-A platforms. TrustZone is a hardware-based security feature that creates a secure and isolated environment (Secure World) for running sensitive applications and handling secure data.

TF-A is a reference implementation of secure world software that provides various security services, such as secure boot, runtime firmware services, and a Secure Partition Manager (SPM) for managing Secure Partitions (SPs). By leveraging TrustZone, TF-A ensures that these security services and sensitive applications are protected from the less secure, normal world applications and potential vulnerabilities.

FF-A, TF-A, and TrustZone `(230409.FF-A.yaml)`

Question: `[15:28]` What is TF-A? `(FF-A.0409.md)`

Answer:

TF-A, or Trusted Firmware-A, is a reference implementation of secure world software for Arm A-profile processors. It provides a platform for implementing security features and building trusted execution environments in devices that use Armv8-A processors, such as smartphones, tablets, and other embedded systems.

TF-A includes implementations of various Arm security technologies, such as Secure Boot, Secure Monitor (for switching between secure and non-secure worlds), and runtime services. The project is open-source, allowing developers to use and modify the code to develop secure firmware solutions that meet the requirements of their specific applications.

Question: `[15:29]` Please draw the relationship among TrustZone, TF-A, and FF-A. `(FF-A.0409.md)`

Answer:

TrustZone, TF-A, and FF-A are technologies and frameworks related to the security aspects of Arm-based systems. Here’s a simple representation of their relationships:

```
+-----------------+
|   TrustZone     |
|  (Technology)   |
| +-------------+ |
| |    TF-A     | |
| | (Firmware)  | |
| +-------------+ |
| +-------------+ |
| |    FF-A     | |
| |  (Framework)| |
| +-------------+ |
+-----------------+
```

1. TrustZone is a hardware-based security technology built into Arm processors. It provides a secure execution environment by creating two separate worlds: a Secure World and a Non-secure World. TrustZone ensures that the Secure World's data and code are protected from unauthorized access.
2. Trusted Firmware-A (TF-A) is an open-source reference implementation of secure world software for Arm A-profile processors. It leverages TrustZone technology and provides secure boot, runtime services, and a secure monitor for switching between the secure and non-secure worlds.
3. Firmware Framework for Armv8-A (FF-A) is a software architecture specification that aims to standardize the way partitions, services, and communication are managed in Arm A-profile systems. It defines the interactions between the Secure Partition Manager (SPM), partitions, and the Non-secure world. FF-A can be used in conjunction with TrustZone and TF-A to build a complete secure software solution.

Question: `[15:36]` What is relationship between FF-A and TF-A? `(FF-A.0409.md)`

Answer:

The relationship between FF-A (Firmware Framework for Armv8-A) and TF-A (Trusted Firmware-A) is that FF-A provides a standardized software architecture specification for managing partitions, services, and communication on Arm A-profile systems, while TF-A is an open-source reference implementation of secure world software that leverages TrustZone technology.

TF-A can be used as a base for implementing the secure world software stack on Arm A-profile systems, and it can be extended or adapted to conform to the FF-A architecture. By aligning the TF-A implementation with the FF-A specification, developers can achieve a more modular, maintainable, and interoperable software stack that can better address the needs of secure systems.

In summary, the relationship between FF-A and TF-A is complementary, with FF-A providing the architectural guidelines, and TF-A serving as a reference implementation that can be tailored to adhere to those guidelines.

Question: `[15:37]` Could I say that TF-A refers FF-A? `(FF-A.0409.md)`

Answer:

Yes, you can say that TF-A refers to FF-A, as TF-A can be extended or adapted to conform to the FF-A architecture. The FF-A provides the architectural guidelines for secure systems, while TF-A serves as a reference implementation that can be tailored to adhere to those guidelines. By aligning the TF-A implementation with the FF-A specification, developers can achieve a more modular, maintainable, and interoperable software stack for secure systems.

Basic Concept `(230402.DRM.yaml)`

Question: `[21:08]` What is license key? `(DRM.0402.md)`

Answer:

A license key, in the context of DRM (Digital Rights Management) systems, is a unique alphanumeric code or data object that grants users or devices access to protected digital content. The license key is typically generated by the DRM system and contains information about usage rights, restrictions, and other conditions associated with the content.

When a user purchases or acquires access to DRM-protected content, they receive a license key associated with that content. The license key enables the authorized user or device to access, decrypt, and play the content according to the specified rules and restrictions. These rules may include limitations on playback duration, the number of devices the content can be accessed on, or the ability to make copies or modifications.

In addition to granting access to the content, the license key serves as a method of enforcing content protection by ensuring that only authorized users or devices can access and use the protected content. The DRM system checks the validity of the license key before allowing access to the content, preventing unauthorized access, copying, or sharing.

It's important to note that the term "license key" can also refer to a code used to activate or unlock software products, but the concept is similar – it is a unique identifier that grants access to a specific product, feature, or service.

Question: `[21:10]` What if the license key is stolen? `(DRM.0402.md)`

Answer:

If a license key is stolen or obtained by unauthorized users, it can potentially compromise the DRM (Digital Rights Management) system's content protection. Unauthorized users may gain access to the protected content, violating the intended usage rights and restrictions. This can result in revenue loss for content providers, as well as unauthorized distribution or copying of the content.

To mitigate the risks associated with stolen license keys, DRM systems employ various security measures:

1. Secure key exchange: DRM systems use secure protocols and encryption methods to transmit license keys between the content server and the client device. This minimizes the risk of license keys being intercepted or stolen during transmission.

2. Device binding: DRM systems often bind license keys to specific devices or user accounts, ensuring that the license key can only be used on authorized devices or by authorized users. If a license key is stolen, it would not work on an unauthorized device or account.

3. License key revocation: In cases where a license key is known to be compromised, content providers or DRM systems may revoke the license key, rendering it invalid. This prevents unauthorized users from accessing the content using the stolen key. Revocation can also be used to disable access to the content on specific devices if they are suspected of being compromised.

4. Monitoring and analytics: Content providers can use monitoring and analytics tools to track the usage of license keys and detect suspicious activity. If abnormal usage patterns or multiple access attempts from different devices are detected, the content provider can investigate the issue and take appropriate action, such as revoking the license key or blocking access from specific devices.

5. Hardware-based security: In more advanced DRM systems, like PlayReady SL3000, hardware-based security features are used to further protect license keys and content. These features make it more difficult for attackers to extract or tamper with license keys, reducing the risk of keys being stolen.

While these security measures can reduce the risk of license key theft, no system is completely foolproof. Content providers must continuously monitor, update, and improve their DRM systems to address new threats and vulnerabilities.

Question: `[21:11]` What is content key? `(DRM.0402.md)`

Answer:

A content key, in the context of DRM (Digital Rights Management) systems, is a cryptographic key used to encrypt and decrypt protected digital content. Content keys help ensure that only authorized users or devices can access and consume the content, as they need the corresponding content key to decrypt it.

When a content provider wants to protect their digital content, such as audio, video, or multimedia files, they use encryption algorithms and a content key to encrypt the data. The encrypted content is then distributed to users or devices, but it remains unreadable without the content key.

When a user or device wants to access the protected content, they must first obtain a license from the DRM system. This license typically includes the content key, along with usage rights and restrictions for the content. The user's device uses the content key to decrypt the content, allowing it to be played or consumed according to the specified rules and limitations.

To maintain security, DRM systems often use secure key exchange protocols and encryption methods to transmit content keys between the content server and the client device. Additionally, the content key may be further protected by binding it to a specific device or user account, ensuring that only authorized users or devices can use the key to decrypt the content.

In summary, a content key is a critical component of DRM systems, as it enables encryption and decryption of protected content, ensuring that only authorized users or devices can access and consume the content according to the rules and restrictions set by the content provider.

Question: `[21:13]` What if the content key is stolen? `(DRM.0402.md)`

Answer:

If a content key is stolen or obtained by unauthorized users, it can compromise the DRM (Digital Rights Management) system's content protection. Unauthorized users with the content key can potentially decrypt and access the protected content, violating the intended usage rights and restrictions. This can lead to unauthorized distribution, copying, or sharing of the content, and revenue loss for content providers.

To mitigate the risks associated with stolen content keys, DRM systems employ various security measures:

1. Secure key exchange: DRM systems use secure protocols and encryption methods to transmit content keys between the content server and the client device. This minimizes the risk of content keys being intercepted or stolen during transmission.
2. License management: Content keys are usually delivered as part of a license that contains usage rights and restrictions. DRM systems control access to content keys by granting licenses only to authorized users or devices. This helps prevent unauthorized access to content keys.
3. Device binding: DRM systems often bind content keys to specific devices or user accounts, ensuring that the content key can only be used on authorized devices or by authorized users. If a content key is stolen, it would not work on an unauthorized device or account.
4. Monitoring and analytics: Content providers can use monitoring and analytics tools to track the usage of content keys and detect suspicious activity. If abnormal usage patterns or multiple access attempts from different devices are detected, the content provider can investigate the issue and take appropriate action, such as revoking the content key or blocking access from specific devices.
5. Hardware-based security: Advanced DRM systems, like PlayReady SL3000, use hardware-based security features to protect content keys and content. These features make it more difficult for attackers to extract or tamper with content keys, reducing the risk of keys being stolen.

While these security measures can reduce the risk of content key theft, no system is completely foolproof. Content providers must continuously monitor, update, and improve their DRM systems to address new threats and vulnerabilities.

Question: `[21:35]` What is identification attacker? `(DRM.0402.md)`

Answer:

An identification attacker, also known as an identity spoofing attacker or impersonation attacker, is a malicious actor who attempts to assume the identity of a legitimate user, device, or system to gain unauthorized access, privileges, or sensitive information. Identification attacks can target various entities, such as individuals, servers, devices, or even Trusted Execution Environments (TEEs), as previously mentioned in the context of TEE identification.

Identification attackers employ various techniques to achieve their goals, including:

1. Phishing: Attackers may use phishing emails, messages, or websites to trick users into providing their login credentials or other sensitive information, which can then be used to impersonate the victim.
2. Man-in-the-middle (MITM) attacks: In this type of attack, the attacker intercepts communication between two parties, such as a user and a server, and can potentially modify, inject, or steal data. This allows the attacker to impersonate one or both parties, gaining access to sensitive information or resources.
3. Password attacks: Attackers may use methods like brute-force, dictionary attacks, or credential stuffing to guess or obtain a user's password, enabling them to assume the user's identity.
4. Social engineering: Social engineering tactics can be used to manipulate individuals into providing sensitive information or access, which can then be used to impersonate the victim.
5. Key theft or compromise: If an attacker can obtain a user's or device's cryptographic keys, they can potentially impersonate that entity in secure communications, gaining unauthorized access to protected resources or information.
6. Device spoofing: An attacker may attempt to spoof a device's unique identifiers, such as MAC addresses or hardware-based identifiers, to impersonate the device and gain access to secure networks or resources.

To defend against identification attacks, organizations and individuals should employ strong authentication mechanisms, such as multi-factor authentication (MFA), use secure communication protocols like TLS, keep software and systems up-to-date, and educate users about safe online practices and how to recognize phishing attempts or other social engineering tactics.

Question: `[21:36]` What is exploitation attacker? `(DRM.0402.md)`

Answer:

An exploitation attacker is a malicious actor who seeks to take advantage of vulnerabilities, weaknesses, or misconfigurations in software, hardware, or systems to gain unauthorized access, escalate privileges, steal sensitive information, or disrupt services. Exploitation attackers typically target known or unknown security flaws (also called “zero-day vulnerabilities”) to compromise systems, install malware, or perform other malicious activities.

Exploitation attackers use various techniques and tools to carry out their attacks, including:

1. Exploit kits: These are software tools or packages that contain a collection of exploits targeting different vulnerabilities in popular software applications, such as web browsers or operating systems. Attackers can use exploit kits to automate the exploitation process and compromise vulnerable systems.
2. Buffer overflow attacks: This type of attack occurs when an attacker overflows a memory buffer with more data than it can handle, potentially causing the program to crash or execute arbitrary code provided by the attacker.
3. Injection attacks: Injection attacks, such as SQL injection or cross-site scripting (XSS), involve inserting malicious code or data into a vulnerable application or system. This can allow the attacker to gain unauthorized access, steal sensitive information, or manipulate the targeted system.
4. Privilege escalation: Exploitation attackers may seek to exploit vulnerabilities that allow them to escalate their privileges within a system, granting them greater access and control.
5. Social engineering: Attackers can use social engineering tactics to manipulate users into installing malware, revealing sensitive information, or granting access to secure systems.

To defend against exploitation attackers, organizations and individuals should adopt strong security practices, such as keeping software and systems up-to-date with the latest security patches, implementing strong access controls and authentication mechanisms, monitoring systems for signs of compromise, and educating users about safe online behavior and potential attack methods. Additionally, employing intrusion detection and prevention systems (IDPS), firewalls, and antivirus software can help detect and block exploitation attempts.

Question: `[21:38]` Please summarize both in a table. `(DRM.0402.md)`

Answer:

Here’s a table summarizing the key aspects of identification attackers and exploitation attackers:

Category	Identification Attacker	Exploitation Attacker
Primary Goal	Assume the identity of a legitimate user, device, or system to gain unauthorized access or privileges	Exploit vulnerabilities, weaknesses, or misconfigurations to gain unauthorized access, escalate privileges, or disrupt services
Attack Techniques	Phishing, Man-in-the-middle attacks, Password attacks, Social engineering, Key theft or compromise, Device spoofing	Exploit kits, Buffer overflow attacks, Injection attacks, Privilege escalation, Social engineering
Defenses	Multi-factor authentication, Secure communication protocols (e.g., TLS), Software and system updates, User education	Security patch management, Access controls and authentication mechanisms, System monitoring, User education, Intrusion detection and prevention systems, Firewalls, Antivirus software

This table provides a high-level overview of the differences between identification attackers and exploitation attackers, focusing on their primary goals, attack techniques, and recommended defenses. While there may be some overlap in the techniques used by both types of attackers, their objectives and methods of compromising systems differ.

TEE Identification `(230328.TEE.yaml)`

Question: `[21:20]` What is TEE identification? `(TEE.0328.md)`

Answer:

TEE identification refers to the process of uniquely identifying a Trusted Execution Environment (TEE) within a device or across multiple devices. This identification can be used for various purposes, such as attestation, secure communication, and device management. TEE identification typically involves the use of a unique identifier or a set of identifiers that can be securely associated with a specific TEE instance.

Some common methods for TEE identification are:

1. Unique Device Identifier (UDID): A unique identifier assigned to the device that hosts the TEE. This identifier can be used to differentiate between devices, but it may not be specific to the TEE itself.

2. TEE Instance Identifier: A unique identifier assigned to a specific instance of a TEE. This identifier is typically generated during the TEE's initialization or secure boot process and can be used to uniquely identify the TEE within the device.

3. Attestation Identity Key (AIK): In some TEE implementations, such as Intel SGX, the TEE generates a unique asymmetric key pair called the Attestation Identity Key. The AIK can be used to identify the TEE during remote attestation processes, allowing other parties to verify the authenticity and integrity of the TEE.

4. Certificates: Some TEE implementations may use certificates issued by trusted Certificate Authorities (CAs) to identify the TEE. These certificates can be used to establish the authenticity of the TEE and to securely associate it with a specific device or entity.

TEE identification plays a crucial role in ensuring the security and trustworthiness of the TEE and the applications running within it. It enables secure communication between the TEE and other components, remote attestation of the TEE, and secure management of devices and services that rely on the TEE.

Question: `[21:22]` Is TEE identification obversable? `(TEE.0328.md)`

Answer:

TEE identification is typically not directly observable by normal applications running in the non-secure world (also called the Rich Execution Environment, or REE). The main purpose of a TEE is to provide a secure and isolated environment for sensitive operations, which includes protecting the TEE's identification information from unauthorized access.

However, some TEE implementations may provide APIs or services that allow authorized applications in the normal world to interact with the TEE and obtain some identification-related information, such as public keys or certificates, for specific purposes like establishing a secure communication channel or performing remote attestation. Access to this information is usually tightly controlled and limited to authorized applications that meet specific security requirements.

For example, in Intel SGX, an application in the normal world can request a remote attestation report, which includes a public key called the Attestation Identity Key (AIK) that can be used to identify the SGX enclave. The remote attestation process involves communication with Intel's Attestation Service, which verifies the enclave's authenticity and provides a signed attestation report.

In summary, while TEE identification information is not directly observable by normal applications, authorized applications may be able to access certain identification-related information through specific APIs or services provided by the TEE, under controlled conditions and for specific purposes.

Question: `[21:26]` Can I use DID as TEE identification? `(TEE.0328.md)`

Answer:

Decentralized Identifiers (DIDs) can potentially be used as a TEE identification mechanism, although this would require integrating the DID concept into the TEE framework. DIDs are a new type of identifier designed for decentralized identity management systems, which allow entities to create, manage, and control their own identifiers without relying on centralized authorities.

To use DIDs as TEE identification, you would need to consider the following steps:

1. Generate DID for TEE: When a TEE is initialized, generate a DID to uniquely identify the TEE instance. This DID should be securely associated with the TEE and stored within the secure environment.
2. Create DID Document: A DID Document is a JSON-LD file that contains the public keys, authentication mechanisms, and service endpoints associated with the DID. In the context of TEE identification, the DID Document should include the TEE's public keys, supported authentication methods, and any secure communication endpoints.
3. Associate DID with TEE: The DID should be securely associated with the TEE and its related cryptographic keys. This can be done through secure storage within the TEE, attestation mechanisms, or other secure binding techniques.
4. Integrate DID APIs and Protocols: To use DIDs for TEE identification, you would need to integrate DID APIs and protocols into the TEE framework. This may involve modifying the TEE APIs to support DID-based identification, as well as implementing DID resolution and verification mechanisms within the TEE or through external services.
5. Leverage Decentralized Identity Ecosystem: By adopting DIDs for TEE identification, you can potentially leverage the growing ecosystem of decentralized identity solutions, such as Verifiable Credentials, DID-based authentication, and secure communication protocols.

Using DIDs for TEE identification is an innovative idea that could potentially enhance the security, privacy, and interoperability of TEE-based systems. However, it is important to note that integrating DIDs into the TEE framework would require significant development effort, as well as coordination with existing TEE implementations and standards. Moreover, it would be necessary to carefully evaluate the security implications and trade-offs of using DIDs in this context.

Revocation of TA `(230328.TEE.yaml)`

Question: `[22:07]` What is revocation for Trusted Application? `(TEE.0328.md)`

Answer:

Revocation for Trusted Applications (TAs) refers to the process of invalidating, removing, or disabling a TA from a Trusted Execution Environment (TEE). The revocation process may be necessary under various circumstances, such as when a TA has been compromised, contains vulnerabilities, or is no longer needed.

Revocation can help maintain the security and integrity of the TEE by ensuring that only trusted and up-to-date TAs are allowed to execute within the secure environment. The specific process for revoking a TA depends on the TEE implementation and the device management policies.

Here are some common steps involved in the revocation of a Trusted Application:

1. Detection: Identify the need for revocation, such as discovering a security vulnerability in the TA or determining that the TA is no longer required.
2. Notification: Notify the relevant parties, such as the device owner, system administrator, or TEE vendor, of the need for revocation.
3. Distribution: Distribute a revocation list, update, or patch that includes the necessary information to revoke the affected TA. This could be in the form of a signed revocation list or a firmware update containing the new list of authorized TAs.
4. Revocation: The TEE or the device management system applies the revocation, which may involve disabling the TA, removing its associated keys and data, or preventing it from being loaded into the TEE.
5. Verification: Verify that the TA has been successfully revoked and can no longer be executed within the TEE.
6. Reporting: Notify relevant parties of the successful revocation and provide any necessary information for further action, such as updating other affected devices or systems.

It is important to note that the specific revocation process and mechanisms will depend on the TEE implementation, the device management policies, and the security requirements of the system. Some TEEs may provide built-in support for revocation, while others may rely on external device management systems or custom solutions.

DRM `(230327.OP-TEE.yaml)`

Question: `[22:31]` Below text comes from “BKK16-201 - PlayReady OPTEE Integration with Secure Video Path”`(OP-TEE.0327.md)`

Answer:

This text appears to be an excerpt from a presentation, “BKK16-201 - PlayReady OPTEE Integration with Secure Video Path.” The presenter is Zoltan Kuscsik, PhD, and the presentation took place on March 8, 2016, during the Linaro Connect BKK16 event. The presentation deals with the integration of key components such as the W3C EME Working Draft, Microsoft PlayReady DRM Porting Kit v3.0, OP-TEE OS, OpenCDM/OpenCDMI, and Chromium v45.

The talk provides updates since SFO15, such as the progress on the proof of concept for Secure Data Path with OP-TEE/Playready on STM B2120, the integration of the Secure Memory Allocator Framework (SMAF), and the implementation of Playready TA with support for the Playready Interface For TEE (PRiTEE).

Supported boards mentioned in the presentation include STM B2120 and 96boards - HiKey. The presentation also discusses various aspects related to EME, secure memory allocation, Playready integration, Open CDM, and the implementation overview for Chromium.

There are several links provided at the end of the presentation for further information on Playready, OP-TEE, and Linaro OpenCDM.

Question: `[22:33]` What is secure data path? `(OP-TEE.0327.md)`

Answer:

A Secure Data Path (SDP) is a mechanism that ensures the confidentiality, integrity, and protection of sensitive data while it is being processed, transmitted, or stored. In the context of multimedia applications and Digital Rights Management (DRM) systems, a secure data path ensures that the content is protected from unauthorized access or tampering at all stages of its life cycle, from decoding to rendering on the display.

The SDP is designed to prevent unauthorized access and data leaks by isolating the sensitive data and processing it within a secure environment, such as a Trusted Execution Environment (TEE) or hardware-protected memory regions. This isolation ensures that even if the rest of the system is compromised, the sensitive data remains protected.

In the case of multimedia applications, the secure data path provides end-to-end protection for encrypted content. It ensures that decrypted data (e.g., video or audio frames) is not exposed to the untrusted parts of the system, such as the general-purpose operating system, applications, or untrusted hardware components. This protection is critical for DRM systems to prevent content piracy and maintain the trust of content providers.

Question: `[22:35]` So the video is played in secure world. Right? `(OP-TEE.0327.md)`

Answer:

Not exactly. The video is not played entirely within the secure world. Instead, the sensitive parts of the video processing, such as decryption and decoding, are performed within the secure world, using technologies like Trusted Execution Environment (TEE) or hardware-protected memory regions. Once the video data is processed securely, it is then passed to the normal world for rendering and display.

The purpose of this approach is to protect sensitive data and operations from unauthorized access or tampering while minimizing the performance overhead of the secure world. By only processing the critical parts of the video pipeline securely, the system can maintain a balance between security and performance.

Question: `[22:37]` How do we protect the video that displays in normal world? `(OP-TEE.0327.md)`

Answer:

Protecting the video data when it is displayed in the normal world is a crucial aspect of maintaining the security of the content. Here are some techniques and technologies used to achieve this protection:

1. Secure Data Path (SDP): SDP is a technique that ensures the secure handling of sensitive data throughout the entire video pipeline. In SDP, the decrypted and decoded video data is transferred from the secure world to the normal world through a protected memory path. This prevents unauthorized access or tampering during the data transfer.
2. Hardware-based content protection: Certain hardware components, such as display controllers and graphics processors, can offer built-in content protection features. These features ensure that the video data remains secure when it is passed to the display device.
3. High-bandwidth Digital Content Protection (HDCP): HDCP is a widely used technology for protecting video content transmitted over digital interfaces like HDMI, DisplayPort, or DVI. HDCP encrypts the video data during transmission, preventing unauthorized access or copying.
4. Output Protection Management (OPM): OPM is a system that enforces the use of HDCP or other content protection mechanisms when video data is transmitted from a source device to a display device. OPM ensures that the video data is protected throughout the entire transmission process.
5. Frame buffer encryption: Some systems support the encryption of frame buffers, which store the video data before it is rendered on the display. This feature provides an additional layer of protection for the video data as it is stored in the memory.

By employing a combination of these techniques and technologies, the video data can be adequately protected even when it is displayed in the normal world.

Question: `[22:41]` What is “EME with OP-TEE on Hikey”? `(OP-TEE.0327.md)`

Answer:

“EME with OP-TEE on Hikey” refers to the implementation of the Encrypted Media Extensions (EME) using the OP-TEE (Open Portable Trusted Execution Environment) on the HiKey development board.

EME is a W3C (World Wide Web Consortium) specification that provides a standard interface for content protection in web browsers. It allows web applications to communicate with Content Decryption Modules (CDMs) to enable the playback of encrypted multimedia content.

OP-TEE is an open-source Trusted Execution Environment (TEE) that provides a secure environment for the execution of sensitive code and data protection. OP-TEE leverages ARM TrustZone technology to isolate the secure world from the normal world.

HiKey is a development board built around a 64-bit ARM-based SoC (System on Chip) and is part of the 96Boards family of development platforms. It is widely used for prototyping and development purposes, including the implementation of secure solutions.

In summary, “EME with OP-TEE on Hikey” refers to a solution that integrates EME and OP-TEE on the HiKey development board to provide secure playback of encrypted multimedia content within web applications. This implementation ensures the protection of sensitive data and operations related to content decryption and key management within the secure world of the TEE.

Question: `[22:43]` What is AES OCDMI? `(OP-TEE.0327.md)`

Answer:

AES OCDMI refers to an implementation of the Open Content Decryption Module Interface (OCDMI) that uses the Advanced Encryption Standard (AES) algorithm for content decryption.

OCDMI is a part of the OpenCDM project, which aims to provide an open-source and modular Content Decryption Module (CDM) solution for various DRM systems. CDMs are used in conjunction with the Encrypted Media Extensions (EME) to enable the playback of encrypted multimedia content in web browsers.

AES (Advanced Encryption Standard) is a widely used symmetric encryption algorithm that provides strong security for data encryption and decryption. In the context of AES OCDMI, the AES algorithm is used to decrypt the protected multimedia content within the secure environment provided by the OpenCDM framework.

To sum up, AES OCDMI is an open-source solution for securely decrypting protected multimedia content using the AES encryption algorithm within the OpenCDM framework.

Question: `[23:01]` What is OCDM? `(OP-TEE.0327.md)`

Answer:

OCDM stands for Open Content Decryption Module. It is an open-source implementation of a Content Decryption Module (CDM) designed to work with the Encrypted Media Extensions (EME) standard, which is part of the HTML5 specification. EME provides a framework for playing encrypted multimedia content in web browsers without the need for third-party plugins such as Adobe Flash or Microsoft Silverlight.

The primary purpose of OCDM is to enable the decryption and playback of protected multimedia content in web browsers, while still maintaining the necessary security and Digital Rights Management (DRM) requirements set by content providers. OCDM aims to provide a standard, cross-platform interface for integrating various DRM systems into web browsers, making it easier for developers to create and deploy secure media solutions.

OCDM is designed to work with various DRM systems and can be extended to support new DRM technologies as they emerge. This allows content providers to use a single, unified solution for delivering protected content to multiple platforms and devices, reducing the complexity of managing multiple DRM systems.
