2023-04-17

Created: 2023-04-22 01:02:45

Questions

- 2022 WATZ @ Papers.OP-TEE.0406.md
  - `[22:34]` Paper: WATZ: A Trusted WebAssembly RuntimeEnvironment with Remote Attestation for TrustZone
  - `[22:40]` What is the benefit of using WASM in TEE?
  - `[22:44]` Is WATZ extended from OP-TEE?
  - `[22:55]` What is Remote Attestation?
  - `[22:59]` So the software is provided by the remote device and the party verify the signature of the software before run it. Right?
  - `[23:01]` Is remote attestation only used to verify software?
  - `[23:03]` How does a remote device declare that secure boot is enabled?
  - `[23:07]` But how does the party(verifier) determine if the report is fake if the device is breached?
  - `[23:11]` How does the party(verifier) verify the root of trust of the device?
  - `[23:13]` Can remote attestation help the party know if the remote device is breached?
  - `[23:19]` Can TEE guarantee the code?

Q & A

2022 WATZ @ Papers.OP-TEE.0406.md

Question: `[22:34]` Paper: WATZ: A Trusted WebAssembly RuntimeEnvironment with Remote Attestation for TrustZone

Answer:

The paper “WATZ: A Trusted WebAssembly Runtime Environment with Remote Attestation for TrustZone” presents WATZ, a runtime environment designed for the efficient and secure execution of WebAssembly (Wasm) code within Arm’s TrustZone trusted execution environment (TEE). The runtime also incorporates a lightweight remote attestation system tailored to Wasm applications running in TrustZone, as TrustZone lacks built-in mechanisms for attestation.

The primary research questions that WATZ aims to address include: 1. System challenges in embedding Wasm into Arm TrustZone. 2. The trustworthiness of remote execution of Wasm applications. 3. Performance comparison of Wasm applications when hosted in the trusted environment.

WATZ extends OP-TEE, a popular open-source trusted OS, and the prototype is validated with Arm hardware. The system successfully hosts Wasm applications in TrustZone, leveraging the WASI standard, a POSIX-like layer for Wasm, to interact with the TEE facilities. The paper also proposes a remote attestation protocol to attest Wasm code embedded in the trusted environment, ensuring that executed Wasm binaries are trustworthy, similarly to Intel SGX.

In terms of performance, WATZ is on par with Wasm applications executed outside of the TEE, and up to 2.12 times slower compared to native execution. This difference is considered negligible given the security benefits provided by WATZ.

Question: `[22:40]` What is the benefit of using WASM in TEE?

Answer:

Using WebAssembly (WASM) in a Trusted Execution Environment (TEE) provides several benefits, making it a valuable choice for enhancing security and interoperability in various applications:

1. Portability: WASM is platform-agnostic, allowing developers to write code once and run it on various platforms, including

TEEs. This increases the reusability and flexibility of applications running in TEEs.

2. Isolation and security: WASM provides a sandboxed execution environment, which ensures that applications are isolated from each other and from the host system. This isolation helps mitigate potential security risks, such as privilege escalation or lateral attacks.
3. Interoperability: WASM allows developers to build software components using modern programming languages and leverage existing code, making it easier to deploy applications in TEEs without the need to rewrite or recompile them.
4. Efficiency: WASM is designed for near-native performance, which means that applications running within TEEs can benefit from the efficiency and speed of WASM execution, even in resource-constrained environments like IoT devices.
5. Support for non-standard system interfaces: WASM can interact with non-standard system interfaces, making it suitable for TEEs that might have unique APIs or restrictions.

Overall, integrating WASM in TEEs allows for enhanced security, performance, and interoperability in various use cases, including smartphones, automotive applications, IoT devices, and edge computing.

Question: `[22:44]` Is WATZ extended from OP-TEE?

Answer:

Yes, WATZ is extended from OP-TEE. OP-TEE (Open Portable Trusted Execution Environment) is a popular open-source trusted operating system for Arm TrustZone. WATZ extends OP-TEE by adding support for running WebAssembly (WASM) code inside Arm's TrustZone TEE and providing a lightweight remote attestation system optimized for WASM applications running in TrustZone. This extension allows WATZ to offer a secure and efficient runtime environment for WASM applications within the TrustZone TEE.

Question: `[22:55]` What is Remote Attestation?

Answer:

Remote attestation is a security mechanism used to verify the authenticity and integrity of software running on a remote device, typically within a trusted execution environment (TEE) or a secure enclave, like Intel SGX or Arm TrustZone. It helps establish trust between a remote device and a verifying party by providing evidence that the software running on the remote device is genuine, unaltered, and operating in a secure environment.

In a remote attestation process, the remote device generates a cryptographic signature or proof, which includes information about the software's state and the secure environment. This proof is then sent to the verifying party, who can validate the signature using a public key or a pre-shared key. If the signature is valid and the software's state meets the expected criteria, the verifying party can trust the remote device and its software.

Remote attestation is particularly useful in scenarios where devices are deployed in untrusted or potentially compromised environments, such as IoT devices or cloud servers. It enables the establishment of trust between distributed devices and helps ensure the confidentiality and integrity of sensitive data and processes.

Question: `[22:59]` So the software is provided by the remote device and the party verify the signature of the software before run it. Right?

Answer:

Yes, that's correct. In remote attestation, the software running on the remote device generates a cryptographic signature or proof, which includes information about the software's state and the secure environment. This proof is then sent to the verifying party.

The verifying party checks the signature using a public key or a pre-shared key, and if the signature is valid and the software's state meets the expected criteria, the verifying party can trust that the software running on the remote device is genuine and unaltered.

In other words, remote attestation allows the verifying party to confirm the authenticity and integrity of the software running on the remote device before deciding to trust it and establish secure communication or share sensitive data.

Question: `[23:01]` Is remote attestation only used to verify software?

Answer:

Remote attestation is primarily used to verify the authenticity and integrity of software running in a secure environment, such as a trusted execution environment (TEE) or secure enclave. However, remote attestation can also be extended to cover aspects of the underlying hardware or firmware.

In some cases, remote attestation can be used to verify the configuration and state of the hardware or firmware to ensure it meets specific security requirements or complies with certain policies. For example, it can be used to verify that secure boot has been enabled, or that a specific hardware component is genuine and not tampered with.

So, while remote attestation is primarily focused on software, it can also be applied to hardware and firmware components in certain scenarios to provide a higher level of assurance about the security and trustworthiness of the entire system.

Question: `[23:03]` How does a remote device declare that secure boot is enabled?

Answer:

A remote device can declare that secure boot is enabled by providing evidence in the form of a signed measurement or attestation report. This process involves the following steps:

1. Secure boot process: When the device boots up, the secure boot mechanism ensures that each component in the boot chain (bootloader, firmware, and operating system) is verified using digital signatures before being executed. This process creates a series of measurements (hashes) of the loaded components.
2. Platform Configuration Registers (PCRs): The measurements obtained during the secure boot process are stored in the device's Trusted Platform Module (TPM) or another secure element, in special registers called Platform Configuration Registers (PCRs).
3. Attestation request: A remote party (verifier) can request an attestation report from the device to verify that secure boot is enabled. The device generates a report that includes the PCR values, a nonce provided by the verifier, and other relevant information about the device's state.
4. Signing the report: The device signs the attestation report using a private key that corresponds to a public key known to the verifier. This key can be an endorsement key, an attestation identity key, or another key tied to the device's secure element.
5. Verifying the report: The remote party receives the signed report and verifies the signature using the corresponding public key. If the signature is valid, the remote party can examine the PCR values and other information in the report to determine if secure boot is enabled and if the device's configuration meets the required security policies.

This process allows the remote party to verify that secure boot is enabled on the device and that the device's boot chain components are genuine and have not been tampered with.

Question: `[23:07]` But how does the party(verifier) determine if the report is fake if the device is breached?

Answer:

The remote attestation process relies on the hardware-backed security provided by secure elements like TPM (Trusted Platform Module) or TEE (Trusted Execution Environment) to ensure the integrity and authenticity of the attestation report. However, if the device is breached and the attacker can compromise these secure elements, the remote attestation process could potentially be fooled.

To minimize the risk of accepting fake reports, the verifier should follow these best practices:

1. Verify the root of trust: Ensure that the device's secure element (e.g., TPM or TEE) has a legitimate and trusted root of trust, such as a hardware-based root of trust or a known and trusted manufacturer's certificate.
2. Use a strong attestation protocol: Employ a secure and well-established attestation protocol, which has been formally verified and widely reviewed by the security community.

3. Verify the attestation report's signature: Ensure that the report is signed with a valid and trusted key tied to the device's secure element. The verifier must have access to the corresponding public key and validate the signature.
4. Check the nonce: The verifier should provide a unique nonce (number used once) during the attestation request, which the device must include in the attestation report. This ensures that the report is fresh and generated for the specific request, mitigating replay attacks.
5. Evaluate the report's content: The verifier should carefully analyze the attestation report's contents, including PCR values and other device state information, to ensure that it meets the required security policies and that the device's configuration is as expected.

It is important to note that no security measure is foolproof, and sophisticated attackers might still find ways to breach a system. However, following best practices and employing a defense-in-depth approach can significantly increase the difficulty for an attacker to compromise the remote attestation process.

Question: `[23:11]` How does the party(verifier) verify the root of trust of the device?

Answer:

To verify the root of trust (RoT) of a device, the verifier (the party receiving the attestation report) must follow these steps:

1. Obtain the device's attestation report: The attestation report, generated by the device's secure element (e.g., TPM or TEE), includes information about the device's hardware, firmware, and software configuration, as well as a digital signature created using the device's private attestation key.
2. Verify the attestation key: The device's attestation key must be tied to a trusted certificate issued by the device manufacturer or another trusted authority. The verifier should validate the attestation key's certificate chain up to a trusted root certificate that is pre-installed or obtained from a trusted source. This process ensures the attestation key is genuine and belongs to a legitimate device.
3. Verify the attestation report's signature: The verifier must use the device's public attestation key (corresponding to the private attestation key used to sign the report) to verify the digital signature on the attestation report. If the signature is valid, it means the report was generated by the device with the genuine attestation key.
4. Check the nonce: To ensure the freshness of the attestation report and prevent replay attacks, the verifier should provide a unique nonce (number used once) during the attestation request. This nonce must be included in the attestation report, and the verifier should confirm its presence and correctness.
5. Examine the report's content: The verifier should carefully analyze the attestation report's contents to ensure that the device's hardware, firmware, and software configuration meet the required security policies. This includes checking measurements of the secure boot process, firmware versions, and other relevant security properties.

By following these steps, the verifier can establish trust in the device's root of trust, making it difficult for an attacker to generate a fake attestation report. However, it is essential to maintain up-to-date trust anchors (e.g., root certificates) and follow best practices to ensure the security of the remote attestation process.

Question: `[23:13]` Can remote attestation help the party know if the remote device is breached?

Answer:

Remote attestation can help the party (verifier) determine if the remote device might be breached, but it does not provide a 100% guarantee. Remote attestation provides information about the device's hardware, firmware, and software configuration, as well as measurements of the boot process and the integrity of the executed code.

If the device is compromised, there is a chance that the integrity measurements included in the attestation report will not match the expected values. The verifier can then detect the discrepancy and decide not to trust the device. However, if the breach is sophisticated enough and can tamper with the attestation process itself, it might be challenging to detect the breach using remote attestation alone.

To further enhance the security of remote attestation and reduce the chances of not detecting a breach, it is essential to follow best practices:

1. Use hardware-based roots of trust (e.g., TPM or TEE) to increase the difficulty for attackers to tamper with the attestation process.
2. Keep trust anchors (e.g., root certificates) up-to-date and manage them securely.
3. Regularly update the device's firmware, software, and security configurations to minimize vulnerabilities.
4. Use secure communication channels to transmit attestation reports and other sensitive information.
5. Combine remote attestation with other security mechanisms (e.g., intrusion detection systems, secure logging, and monitoring) to increase the chances of detecting breaches.

By employing a comprehensive security approach and using remote attestation as one of the security mechanisms, the verifier can have a higher level of confidence in detecting if a remote device is breached. However, it is essential to remember that no security measure can provide absolute certainty.

Question: `[23:19]` Can TEE guarantee the code?

Answer:

A Trusted Execution Environment (TEE) can provide a high level of assurance regarding the integrity and confidentiality of the code executed within it, but it cannot provide an absolute guarantee. A TEE isolates sensitive code and data from the rest of the system, protecting them from unauthorized access or tampering by malicious software or external attackers.

TEEs, such as ARM TrustZone or Intel SGX, leverage hardware-based security features to create a secure enclave where sensitive operations can be performed. These enclaves provide a range of security properties, including:

1. Code isolation: The code running in the TEE is isolated from the rest of the system, ensuring that other applications or the OS cannot access or modify it.
2. Data protection: Data stored or processed within the TEE is encrypted and protected from unauthorized access or tampering.
3. Secure boot: TEEs often support secure boot, which ensures that the code loaded into the TEE has not been tampered with and comes from a trusted source.
4. Remote attestation: TEEs can provide remote attestation capabilities, allowing a remote party to verify the integrity and authenticity of the code running in the TEE.

Despite these security properties, TEEs are not infallible, and vulnerabilities have been discovered in the past. For example, side-channel attacks have been demonstrated against Intel SGX, potentially allowing an attacker to extract sensitive information from the enclave.

While TEEs can provide a high level of assurance for the code they execute, it is essential to remember that no security measure can offer absolute certainty. Regularly updating the software, firmware, and security configurations of the TEE, as well as employing a comprehensive security approach that includes other security mechanisms, can help reduce the risk of compromise.
