CONFIDENTIAL COMPUTING CONSORTIUM
Governance Risk & Compliance Special Interest Group

Confidential Workload Governance

August 2025

Context

When implementing code to run as a Confidential Workload, the application developer is faced with a set of concerns around the additional security properties that Confidential Computing would deliver. It may not be sufficient to run a Confidential Workload inside a TEE, even coupled with Remote Attestation, as all desired Confidential Computing-related requirements may not be automatically met. A properly governed Confidential Workload must be subjected to a set of Confidential Computing-specific Control Objectives in order to satisfy the requirements of each Persona **[1]**.

Problem

Trusted Execution Environments by themselves deliver only some of the properties needed to keep code and data properly safeguarded while in-use. These default properties include isolation from the hosting environment and the ability to perform Remote Attestation, as well as other platform-level functions such secure random number generation and sealing data to the platform. These facilities, while essential, are only a subset of the requirements for secure Confidential Workload execution. Code not developed specifically for execution inside TEEs may not fully protect data in use even when run inside a TEE. Moreover, it is generally not sufficient to develop the code itself with Confidential Computing-specific features in mind: additional non-trivial considerations apply to the build chains and deployment processes.

Forces

Vulnerabilities may be present in deployed code due to a wide variety of factors. Some relate to the code development itself. However, even the most perfectly written code may still be vulnerable due to supply chain, build tooling, and deployment issues. These are all covered in this section.

In total, there are four main areas of concern for secure Confidential Workload governance:

1. **Secure Code Design & Development:** Code written in traditional ways for a non-confidential environment and then naïvely executed as-is inside a TEE may contain security vulnerabilities that would weaken the protections offered by Confidential Computing. Some relevant potential vulnerabilities in this context include:
    - Information leakage and tampering due to unsecured incoming and outgoing data — such as network, storage, logs, environment/configuration, and metrics.
    - Use of a poor source of entropy in generation of random numbers leading to inadequate randomness in cryptographic operations.
    - Issuing bearer tokens leading to the tokens being leaked (e.g., by some relying party), allowing any attacker in possession of the leaked tokens to impersonate the identity of the code that the tokens represent. This attack is not prevented by the use of Confidential Computing code running in a TEE, and secrets that are only supposed to be made available to the TEE are no more protected from bearer token leaks than code running outside TEEs.
2. **Secure Code Build**
    - Code compilation is sabotaged by a malicious actor, including by tampering with compilation settings, making unauthorized code changes preceding compilation, or modifying the compiler output.
3. **Secure Supply Chain**
    - The build tools themselves may be vulnerable or actively malicious.
    - Vulnerabilities already present in or maliciously inserted into dependencies, for example, open-source libraries or compiled artifacts introduced during the build process.
4. **Secure Packaging, Integration, Configuration and Deployment:** The final class of issues that should concern owners of data entrusted to Confidential Workloads relates to the configuration and environment settings during execution:
    - **Unnecessarily large TCBs:** Lift-and-shift of an unmodified code execution environment such as a VM may be less secure than repartitioned workloads that explicitly minimize the trusted computing base for sensitive operations.
    - **Security-sensitive configuration:** As one example, trusted roots are essential for establishing and verifying secure communication across systems and applications, and certificate signature integrity and authenticity are fundamental to secure operation and infrastructure trust. Polluted root certificate stores can cause the Workload to trust the wrong entities, e.g., terminating TLS connections with the wrong parties or trusting the wrong signatures. As another example,

misconfiguration of cryptographic parameters, such as poor choices of ciphers, key lengths, and modes may introduce vulnerabilities into exchanged data flows.

- **Verifier Hygiene:** The Verifier policies* may allow older, vulnerable versions of the code to continue being treated as valid**

Solution

The terms MUST/SHOULD/MAY etc. below are used in accordance with **[2]**. Every SHOULD recommendation is explained separately in the "SHOULD vs. MUST Clarifications" section towards the end of this document.

For each of the main concern areas listed in the Forces section above, the proposed solution is listed below:

- **Secure Code Design & Development**
  - **Encryption of all data in transit and at rest:** The code SHOULD **[a]** be implemented to safeguard all data exchanges with the outside world (e.g., network, storage, and peripherals such as Graphics Processing Units and smart NICs) to prevent information leakage/tampering, and cryptographic keys MUST be securely generated/procured/provisioned for all such operations. Data flows that may leak sensitive information to attackers include parameters passed to APIs exposed by the TEE as well as their corresponding return values.
  - **Secure RNG:** The platform RNG (functionality available to code executing within TEEs via a special platform API) MUST be used for all secure random number generation within TEEs.
  - **Check all outputs for data leaks:** The developer MUST reassess the trust relationship that code running confidentially has with its counterparties; for instance, this may include ensuring that generated logs and metrics do not leak information that cannot be exposed to unauthorized parties and all cached data is sealed to the platform.
  - **Secure data exchange with smart accelerators such as GPUs, NICs, local storage, etc.:** Any device or accelerator that handles decrypted data MUST be considered part of the TCB. The implementation MUST only pass data to attested accelerators which themselves implement a TEE. The implementation MUST establish a secure channel through device attestation. All other uses of devices should treat the device as an untrusted pass-through and use end-to-end encryption with another trusted endpoint.
  - **Use secure credentials:** design SHOULD **[b]** avoid reliance on bearer tokens in authentication protocols.
- **Secure & Attestable Build Environment**
  - **Establish, Maintain and Produce Evidence of Integrity and Authenticity of the Build Tools and Environment:**
    - **Secure and Isolated Build Practices:** The build process SHOULD **[c]** incorporate integrity and authenticity checks and strongly isolated compilation processes to prevent malicious tampering. The **SLSA [3] Build track** describes specific practices to enhance build process integrity. The build environment SHOULD **[d]** be integrity checked to detect tampering in the underlying compute system prior to running a build pipeline; the future **SLSA Build Environment track** will define specific practices to achieve this, including the use of Confidential Computing (CC). CC or alternative data-

* *Verifier policies* as used in this document is a shorthand for Endorsements, Reference Values and Appraisal Policy for Evidence in RATS **[5]** parlance.

** Blue-green deployments may allow both newer and older versions; this is covered under the "Workload Upgrades" pattern **[6]**.

in-use protection technology SHOULD be used to safeguard a build environment at runtime.

    - **Secure and Documented Build Tool Settings:** The build process MUST use recommended build tool settings as well as up-to-date versions of tools and dependencies. The configuration, inputs and outputs of the build process SHOULD be documented (e.g., via SLSA Provenance) and authenticated using a framework such as in-toto [4]. Failure to do either can lead to the generation of vulnerable executable code and make it more difficult to determine the integrity and trustworthiness of the produced code.

  - **Reproducible Builds:** Builds SHOULD **[e]** be reproducible to ensure that the built binaries can be independently verified to match the source code. This approach provides additional measures for detecting tampering or unintended modifications during the build process.

- **Secure and Attested Supply Chain Management**

  - Automated tools MUST be used to continuously monitor and manage third-party software libraries and dependencies for known vulnerabilities. Dependencies MUST be kept updated and strict vetting processes applied, including code signature and build metadata checks if available, when ingesting dependencies. Dependency management MUST continue for the duration of the application's deployment.

  - Include Bills of Materials as part of generated software; the BOMs SHOULD **[f]** be verified by the Verifier post-deployment.

- **Secure Integration, Configuration and Deployment**

  - **Minimizing the Workload TCB:** Care MUST be taken to include only the minimum amount of code in each TEE that is necessary for the functionality encapsulated by that TEE.

  - **Trusted Root Store Hygiene:** The contents of the Root Stores, if any, used by the Confidential Workload, MUST be carefully curated and safeguarded against tampering. If used as a configuration parameter (as opposed to being hard-coded into the Workload), the Trusted Root Store measurement MUST be included in the remote attestation process.

  - **Cryptography Hygiene:** The choice of cryptographic ciphers, key lengths and modes MUST be carefully curated and safeguarded against tampering. If configurable (as opposed to being hard-coded into the Workload), these choices MUST be included in the remote attestation process.

  - **Cryptographic Key Hygiene:** All cryptographic keys procured from external Key Vaults MUST only be released to properly authorized requesting TEEs, usually based on the results of successful Remote Attestation and delivered to the requesting TEEs using a secure transport.

  - **Security-Sensitive Configuration Hygiene:** All security-sensitive configuration MUST be included in the remote attestation process.

  - **Verifier Hygiene:** The Verifier policies MUST match the most recent versions of deployed Workloads; older vulnerable Workloads MUST be phased out in a timely fashion following successful deployment of up-to-date versions.

Governance Expectations Summary

The numbers in the left column below refer to **[1]**. Rows listed as N/A indicate that corresponding expectations are listed under different Patterns documents.

#	DESCRIPTION
2–6, 8, 10–11, 13	N/A
1	Supply accurate and up-to-date component-specific guidance for secure software development.
7	Update Verifier policies with the Reference Values of newly deployed packages.
9, 14	Evidence of establishment of and adherence to secure design & development practices Evidence of securely managed build environment Evidence of robust supply chain and dependency management Evidence of secure integration, configuration and deployment practices.
12	Provide evidence of measurements of newly deployed packages being recognized as valid.
15	Evidence of requiring and validating that expectations set out in (9, 14) are satisfied.

“SHOULD” vs. “MUST” Clarifications

a. Extreme care must be taken to ensure that the inputs and outputs of computations, whether sent over the network or to/from storage, are properly secured against tampering and disclosure. Failure to follow this recommendation can have a severe impact on the security of the application and its consumers.

b. Failure to bind tokens to the TEE may cause them to be leaked, undermining the security of a relying party for which they are intended.

c. Failure to secure the build environment may cause vulnerabilities to be introduced into the compiled/generated artifacts and open additional avenues to attack that running these artifacts inside a TEE would be unable to mitigate.

d. If the build tools themselves are not running inside TEEs, other compensating controls, such as physical security or strongly segregated environments, should be considered.

e. Non-reproducible builds make it more difficult to ensure and subsequently prove that the inputs into the build process map exactly to the outputs, thus requiring compensating controls such as cryptographic signing and timestamping of generated artifacts.

f. Failure to check the BOMs of Attesters by the Verifier may create situations where Workloads containing newly discovered vulnerabilities, that could be discovered by cross-checking BOMs against known vulnerabilities, continue to attest successfully.

References

1. Expectations of Ecosystem Participants:
./Expectations of Ecosystem Participants
2. Key Words for Use in RFCs to Indicate Requirement Levels:
https://datatracker.ietf.org/doc/rfc2119/
3. Supply-chain Levels for Software Artifacts (SLSA)
https://slsa.dev
4. In-Toto Attestation Framework
https://github.com/in-toto/attestation
5. Remote Attestation Procedures (RATS) Architecture RFC:
https://datatracker.ietf.org/doc/rfc9334/
6. Confidential Workload Upgrade Governance Pattern:
https://github.com/confidential-computing/governance/blob/main/SIGs/GRC/publications/Confidential_Workload_Upgrade_Governance.md
7. Confidential Computing Glossary:
https://github.com/confidential-computing/glossary/
8. NIST SP 800-204D "Strategies for the Integration of Software Supply Chain Security in DevSecOps CI/CD Pipelines":
https://csrc.nist.gov/pubs/sp/800/204/d/final
9. Picture of likely Workload slices on various hardware architectures: **https://github.com/confidential-computing/governance/blob/main/terminology/Full-Table.jpg**