Attestable Build Chain: Enabling Trust in Reproducible Builds

Kenichiro Muto
Institute of Information Security
Yokohama, Japan
dgs247101@iisec.ac.jp

Kuniyasu Suzaki
Institute of Information Security
Yokohama, Japan
suzaki@iisec.ac.jp

***Abstract*—Ensuring trust in software supply chains requires verifying not only artifacts but also the processes that produce them. Although Reproducible Builds (R-B) require rebuilding to validate artifacts, they cannot verify whether the build was executed with the intended toolchain and inputs and may reproduce unintended or compromised builds without detection. We present *Attestable Build Chain*, a framework for externally verifying build-time execution without rebuilding. Rather than preventing compromise, it provides verifiable, tamper-evident evidence of actual build-time execution, enabling verification of build process integrity from observed file accesses during the build. Our system records execution evidence at the system level using Linux IMA and a virtual TPM inside a Confidential Virtual Machine (CVM), anchored in a hardware-rooted chain of trust, and produces a unified *Build Chain Report*. Evaluation in multiple open-source projects demonstrates practicality and enables artifact validation without rebuilding.**

***Index Terms*—Remote Attestation, Reproducible Builds, Software Bill of Materials, Confidential Computing, Trusted Platform Module, Software Supply Chain**

I. Introduction

Ensuring trust in the software supply chain is a critical challenge in modern systems. Each stage, from source code development to building and distribution, can be a target of tampering, and compromise at any single point can invalidate the integrity of the final artifact. Incidents such as SolarWinds, XcodeGhost, and repository compromises in PyPI/npm demonstrate that malicious code can be injected into legitimate distributions through compromised build environments or toolchains [1], [2]. These incidents highlight the need to establish trust not only in artifacts but also in the processes that produce them [3]–[6].

Reproducible Builds (R-B) [7] and Software Bill of Materials (SBOM) [8] have been proposed to improve the reproducibility and transparency of the artifacts. R-B enables verification of output equivalence, but it does not reveal which toolchain components and build-time files were actually used during the build; thus, even unintended or compromised builds can be reproducibly regenerated, allowing attacks that preserve output equivalence to remain undetected, including Trusting Trust [9], [10] and OS-level manipulation (e.g., Iago Attack [11]–[15]). Similarly, SBOMs provide visibility into the composition of software, but cannot ensure integrity if their generation process or execution environment is compromised [16]. Although Confidential Computing [17], [18] and SLSA [19] strengthen the trust base, mechanisms that allow third parties to verify build-time execution remain limited.

In this work, we define the end-to-end path from source code to binary generation as the *Build Chain* and propose *Attestable Build Chain*, a two-phase attestation framework that enables third-party verification of both the build environment and build-time execution. Builds run inside a Confidential Virtual Machine (CVM) as a practical attestation and isolation environment: Phase 1 verifies the CVM configuration, while Phase 2 records build-time execution evidence using Linux Integrity Measurement Architecture (IMA). This evidence is collected at the kernel level within the attested CVM and protected by a virtual Trusted Platform Module (vTPM), making it tamper-resistant even against compromised build components. It is then externally presented, together with the build artifact and SBOM, as a verifiable evidence package called the *Build Chain Report*. This extends previous Attestable Builds [20] by enabling verification of actual file accesses and executed toolchain components during the build, rather than relying solely on user-level reports of declared build steps. By combining signed SBOMs with the *Build Chain Report*, downstream users can verify both the build environment and the executed toolchain without rebuilding.

Our key contribution is not merely the combination of existing primitives, but enabling externally verifiable, system-level observation of actual build-time execution, rather than relying on declared workflows or output equivalence. Our main contributions are as follows.

- **Verifiable Build Execution:** We propose *Attestable Build Chain*, a two-phase attestation mechanism that enables third parties to verify both the build environment and build-time execution, in addition to artifact reproducibility.
- **Hardware-rooted Execution Evidence:** We show how Linux IMA and vTPM provide tamper-resistant kernel-level evidence of accessed files and executed toolchain components, thereby making executed toolchain usage externally verifiable beyond output-based verification.
- **Practical System and Evaluation:** We implemented the system on an AMD SEV–SNP-based CVM and evaluated its applicability and performance on multiple OSS projects.

II. Background

A. Reproducible Builds (R-B)

R-B guarantees that identical binaries can be generated from the same source code and build environment, allowing third-party verification of artifacts [7]. However, reproducibility alone does not reveal which toolchain components and build-time inputs were actually used during the build process. As shown by Trusting Trust attacks [9], malicious binaries can be generated even from clean sources; therefore, mechanisms are needed to make the build process itself externally verifiable [21].

B. Software Bill of Materials (SBOM)

An SBOM is structured information that describes the components and dependencies that comprise software, supporting vulnerability management and license compliance [8]. However, the correctness and integrity of an SBOM are not guaranteed; if the generation environment or tools are compromised, an SBOM may appear valid but does not reflect the actual build-time behavior [16]. Hence, trust in an SBOM depends on the integrity of both the execution environment and the generation process.

C. Confidential Virtual Machine (CVM)

A CVM provides an isolated execution environment as a virtual machine, protected from the host through hardware-assisted encryption. A representative implementation is AMD SEV-SNP, which prevents observation by privileged software outside the guest, including the host OS [22]. To verify the integrity of software running inside a CVM, one can use a Secure VM Service Module (SVSM) and a virtual TPM (vTPM), together with Linux IMA, to record file integrity measurements at the kernel level in a cryptographically verifiable, tamper-evident form [23]–[25]. Linux IMA is a kernel subsystem that records integrity measurements of accessed files and executed binaries, forming append-only, ordered execution evidence. In this work, we treat such measurements as build-time execution evidence and use them for external verification of the build process.

D. Remote Attestation (RA)

Remote Attestation (RA) provides cryptographic evidence of the configuration and state of an execution environment, enabling external verification [26]. With SEV-SNP, the CVM configuration is provided as an attestation report verifiable under AMD's endorsement chain. In addition, combining vTPM and Linux IMA allows dynamic file integrity measurements to be verified. We extend this model by attesting not only static configuration but also build-time execution evidence, enabling artifact validation based on evidence obtained during the build, without rebuilding.

III. Threat Model and Research Questions

A. Threat Model

We assume a typical software supply chain in which source code with a source-level SBOM is used and R-B produces a binary artifact together with its corresponding SBOM. In software supply chains, it is often necessary to verify, after the fact, whether an artifact was built not only from the intended source but also with the intended toolchain. However, R-B alone is often insufficient for this purpose, as it requires an uncompromised reference build and does not reveal what was actually executed during the build process. Figure 1 illustrates representative points at which integrity can be compromised in this workflow.

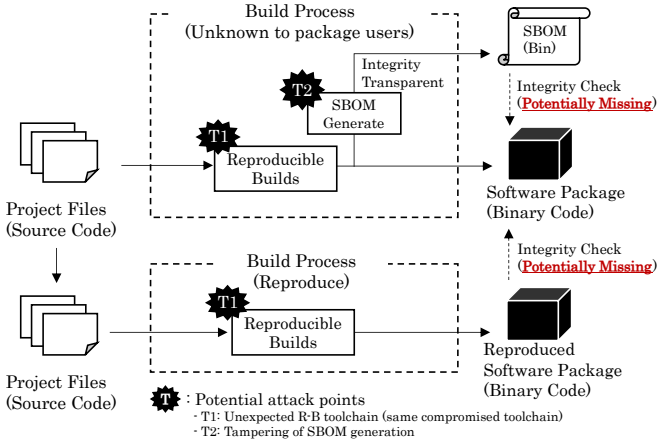

Fig. 1. Threat Model: Even when R-B produces identical binaries, tampering with the toolchain (T1) or SBOM generation process (T2) may remain undetected, as output-based verification does not capture build-time execution.

Specifically, as shown in T1, if the toolchain (e.g., compiler or linker) is tampered with, a Trusting Trust attack can produce malicious binaries without modifying the source code [9]. Build-time behavior may also be influenced by the execution environment; for example, OS-level manipulation during execution (e.g., Iago attacks [11]–[15]) can affect runtime behavior while remaining unobservable through output equivalence. As a result, even when both the original build and the reproduction use the same compromised toolchain, identical binaries can still be produced, causing R-B verification to succeed even though the executed toolchain is not the intended one. As shown in T2, if the SBOM generation process is compromised, attackers can insert malicious dependencies or forge metadata, producing seemingly valid SBOMs and artifacts [16]. Because SBOMs describe declared components rather than actual execution, they may fail to reflect true build-time behavior when the generation process is tampered with. In both cases, the build process becomes effectively a black box for external verifiers. Although output equivalence can be confirmed, it remains unclear whether the intended toolchain and inputs were actually used during the build. Therefore, integrity checks based solely on artifacts or SBOMs can miss attacks involving unexpected or compromised toolchain components, even when the output equivalence holds ('Potentially Missing' in Figure 1).

Importantly, this limitation is not merely due to imperfections of R-B. Even if R-B were ideal and always produced identical outputs, external verifiers would still be unable to

confirm whether the build was executed with the intended toolchain and inputs, because the build process itself remains unobservable. Therefore, securing the software supply chain requires mechanisms that provide verifiable evidence of build-time execution, including accessed files and executed toolchain components, rather than relying solely on output equivalence.

B. Research Questions

R-B assumes that a third party can rebuild from source to verify that an artifact was produced from the intended input. In practice, however, constraints on execution environments and distribution often require users to rely on third-party-built artifacts as-is. For example, reproducing historical build environments may be difficult due to toolchain updates, dependency changes, or operational constraints. In such cases, although R-B can confirm output equivalence and SBOMs provide compositional transparency, neither can reveal whether the build was actually executed with the intended toolchain and inputs. As discussed in Section III-A, the build process remains effectively unobservable to external verifiers, making it difficult to detect tampering such as T1 and T2.

Ensuring the correctness of the build process alone is therefore insufficient, because external verifiers cannot directly observe or verify how the build was executed. In other words, integrity must not only be ensured, but also be externally verifiable. Our goal is not to prevent tampering itself, but to make build-time deviations externally verifiable.

This leads to the following research questions.

RQ1: Build Process Visibility
Can we externally verify which files and toolchain components were actually used during the build process?

RQ2: Integrity of Build Evidence
Can we ensure that the recorded build-time execution evidence is protected against tampering and reflects the actual build process?

RQ3: Verification without Rebuilding
Can an external verifier validate that the build and SBOM generation were executed as intended, using only attested execution evidence without re-executing R-B?

C. Assumptions

We aim to enable external verification of build-time execution using attested execution evidence. Our assumptions are intended to isolate the problem of verifying build-time behavior from lower-level trust establishment.

We consider a setting in which builds are executed inside a CVM, referred to as a Reproducible Build CVM (R-CVM), whose configuration is verified via RA. We assume that the CVM environment, including its initial software stack, matches pre-registered reference values. Such reference values can be established using techniques such as Diverse Double-Compiling (DDC) [27], which has been studied as a foundation for the construction of verifiable toolchains.

We further assume that the hardware root of trust (e.g., SEV-SNP) and attestation mechanisms (including vTPM) are correctly implemented and not compromised. Risks related to trust boundaries and infrastructure dependencies in cloud environments have been extensively discussed in previous work [28], [29], and are assumed to be mitigated in our setting. Finally, we assume that signature verification for SBOMs and attestation reports is performed correctly, that the supporting public-key infrastructure (PKI) is securely operated, and that cryptographic primitives are secure.

Under these assumptions, the external verifier still does not have visibility into the internal state of the R-CVM. Even if the R-CVM is securely constructed and its configuration is correctly attested via RA, the verifier cannot determine from artifacts or SBOMs alone whether the intended toolchain and inputs were actually used during the build. Therefore, we focus on external verification of build-time execution.

IV. Design

A. Security Mechanism

Based on RQ1–RQ3, we design a scheme that extends artifact-level verification to verification of build-time execution. Although R-B enables output equivalence checking, it does not provide visibility into how the artifact was produced or which toolchain components were actually executed. To address this limitation, we introduce the *Attestable Build Chain*, an extended attestation mechanism that provides verifiable evidence of build-time execution based on file accesses and toolchain usage. Our approach shifts the verification target from output equivalence to execution evidence, recording which files and toolchain components were actually used during the build. Figure 2 illustrates the overall mechanism.

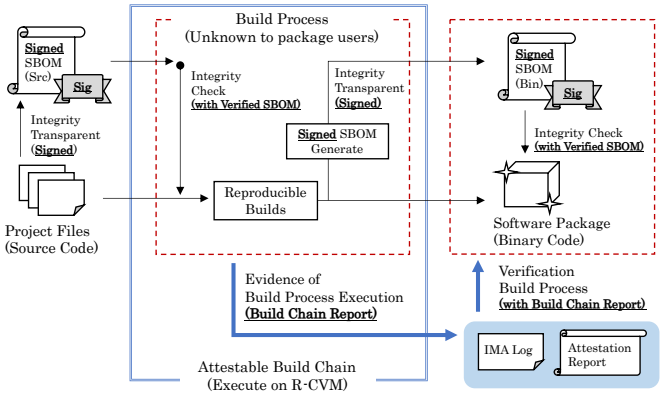

Fig. 2. Attestable Build Chain: Extending verification from output equivalence to build-time execution evidence. Build execution is measured inside an R-CVM using Linux IMA and vTPM, and verified externally via a hardware-rooted chain of trust.

The scheme integrates multiple security primitives within an R-CVM to ensure integrity across the workflow from source code to binary generation. The input source code and SBOM are verified through signatures. Build execution is isolated inside the R-CVM, whose initial configuration is attested using RA and rooted in Trusted Boot measurements. During execution, Linux IMA records file accesses and tool invocations at the kernel level, providing visibility into actual

build-time execution. Because these measurements are independent of the build process itself, they cannot be forged by compromised build components. The vTPM ensures that the measurements are append-only and tamper-evident, forming ordered execution evidence.

All evidence is aggregated into a *Build Chain Report*, signed under the SEV-SNP trust base. External verifiers validate the report by comparing recorded execution evidence against expected reference values, such as known toolchain binaries and declared SBOM components. This enables verification of deviations such as toolchain substitution or inconsistencies between declared and executed components without requiring rebuilding. As a result, integrity can be verified across the input, the environment, the execution, and the output, allowing verification beyond the output equivalence. The key management details required for the signing and verification of the *Build Chain Report* are described in Section IV-C.

B. System Architecture and Attestation Procedure

Figure 3 shows the system architecture and attestation workflow of the *Attestable Build Chain*. The scheme consists of two phases: traditional attestation and extended attestation.

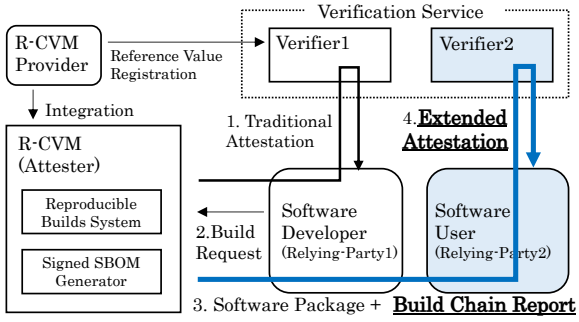

Fig. 3. Two-phase attestation workflow of Attestable Build Chain. Phase 1 verifies the execution environment, while Phase 2 provides verifiable evidence of build-time execution.

In Phase 1 (Traditional Attestation), Verifier1 confirms that the R-CVM configuration matches pre-registered reference values. This ensures that the R-CVM is instantiated with an attested configuration, providing an isolated and integrity-protected execution environment. Next, signed SBOM and the corresponding source code are delivered to the R-CVM, and the input integrity is confirmed by SBOM signature verification.

In Phase 2 (Extended Attestation), the build execution becomes observable and verifiable. During the build, Linux IMA records the file accesses and executed binaries and extends these measurements to vTPM PCRs. Because the measurements are chained and signed, the resulting execution evidence is tamper-evident and ordered.

The collected evidence is packaged into the *Build Chain Report*, together with the generated binary and signed SBOM. External verifiers validate that recorded toolchain components and accessed files match expected reference values, enabling detection of substituted or unexpected toolchain execution without re-executing the build. Unlike approaches that rely solely on output equivalence, this mechanism provides visibility into the build process itself. Even when identical binaries are produced, discrepancies in toolchain usage or execution behavior can be verified from attested evidence.

C. Key Management

A key-management framework is essential to realize the security mechanisms in Section IV-A. As shown in Figure 4, the *Attestable Build Chain* manages three types of keys: signed SBOM Keys, Attestation Keys (SEV-SNP and vTPM) and Verifier Keys, clarifying the issuer of each piece of data.

The signed SBOM Key is generated by the R-CVM provider and distributed as a public-key certificate through a CA. Its private key is kept inside the R-CVM and is used exclusively for SBOM signing. For Attestation Results, the verifier's signing key is also trusted via a CA-issued public-key certificate.

Attestation Keys are securely provisioned within the TEE of the R-CVM equipped with SEV-SNP. The corresponding public keys are distributed to verifiers via a key distribution service and are used to verify the attestation reports. In addition, vTPM keys are generated at each R-CVM boot and are endorsed by the SEV-SNP Attestation Key, allowing verification. This composite key management guarantees the integrity and trust of SBOMs and attestation evidence.

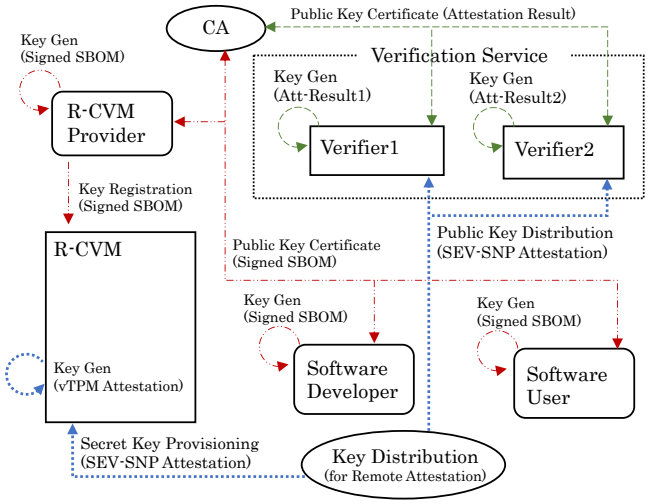

Fig. 4. Key Management Architecture for *Attestable Build Chain*: Red flows show signed SBOM Keys, generated by the R-CVM provider and certified by CA. Blue flows represent Attestation Keys (SEV-SNP and vTPM), provisioned and generated under the chain of trust rooted in SEV-SNP with SVSM. Green flows indicate Verifier keys, certified by CA.

V. Implementation and Evaluation

A. Prototype System Implementation

We implemented a prototype integrating AMD SEV-SNP, vTPM, Linux IMA, and Bazel to enable verifiable build-time execution inside an R-CVM. The R-CVM runs on QEMU with an SEV-SNP–enabled kernel based on COCONUT-SVSM [30], and the builds are executed using Bazel [31]. During the build, Linux IMA records the file accesses and executed

binaries, extending the measurements to vTPM PCRs. At launch, an SEV-SNP attestation report is obtained via VirTEE `snpguest` [32], [33], embedding the hash of the vTPM Endorsement Key (EK) into `report_data` to bind the CVM and the vTPM. All evidence is aggregated into a *Build Chain Report*, and a Flask-based API enables external verification. We configured IMA to record all file accesses, including repeated events, to capture complete execution behavior. This prototype enables external verifiers to verify not only artifact integrity but also build-time execution without rebuilding.

The prototype system realizes the three roles defined above: Relying-Party (Software Developer and Software User), Attester (R-CVM) and Verifier (Verification Service). For simplicity, we implemented Relying-Party as a single client and integrated Verifier_1 and Verifier_2 on one verification server. Table I lists the OS and the main configurations of the tools of each component.

TABLE I
MACHINE SPEC WITH OS AND TOOLS

Component (Role)	Machine Spec with OS and Tools (Version)
Build Client (Relying-Party_1&2)	AMD Ryzen 3 7330U Processor Virtual Box (7.1.4) - 2CPU (2295.6 MHz × 2), Memory 2048 MB Ubuntu Server (24.04.2 LTS) Kernel (6.8.0-59-generic) Python3 (3.12.3), Requests(2.32.3) OpenSSL (3.0.13 30 Jan 2024) tpm2-tools (5.6-1build4)
Build Server (Attester)	AMD EPYC 7313P Processor qemu (v8.2.0-81-g260df2b36b-dirty) - 4CPU (3000.0 MHz × 4), Memory 8192MB Ubuntu Server (24.04 LTS) Kernel (6.8.0-coconut-svsm-sevsnp+) *customized Python3 (3.12.3), Flask (3.1.0) OpenSSL (3.0.13 30 Jan 2024) VirTEE snpguest (0.5.1) tpm2-tools (5.7-12-gbd832d3f) Bazel (7.6.1)
Verification Server (Verifier_1&2)	AMD Ryzen 3 7330U Processor Virtual Box (7.1.4) - 2CPU(2295.6 MHz × 2), Memory 2048 MB Ubuntu Server (24.04.2 LTS) Kernel (6.8.0-59-generic) Python3 (3.12.3), Flask (3.1.0), PyJWT (2.10.1) OpenSSL (3.0.13 30 Jan 2024) VirTEE snpguest (0.8.3) tpm2-tools (5.6-1build4)

B. Build Chain Report Implementation

The *Build Chain Report* bundles evidence for external verification, without rebuilding, of artifact integrity, process integrity, and hardware/environment integrity into a single package. Figure 5 illustrates the structure of a *Build Chain Report* for the `hello_world` example and how each component maps to its verification target (artifact, process, hardware).

The figure shows that evidence for different targets (artifact, build process, and execution environment) is organized so that it can be verified stepwise from a hardware root of trust, beginning with Trusted Boot and extending to kernel-level runtime measurements. We categorize the components of the *Build Chain Report* into three groups.

Artifacts and SBOMs

The artifact (e.g., `hello_world.tar`) and its signed SBOM (e.g., `*.spdx.json`, `*.sig`, `public_key.pem`) enable verification of the correspondence between the artifact and its composition information (artifact integrity).

Runtime measurement evidence

The IMA log (e.g., `ascii_runtime_measurements`) and the vTPM PCR quote (e.g., `pcr_quote.*`, `nonce_2`, `ak.*`) enable verification that the history of referenced/generated files and execution events during the build and SBOM generation has not been tampered with (process integrity).

Hardware-rooted attestation evidence

The SEV-SNP attestation report (e.g., `snp_report.bin`) and the boot measurement log (e.g., `binary_bios_measurements`) enable verification that the initial state and TCB of the R-CVM match reference values (hardware/environment integrity). Furthermore, embedding the vTPM EK hash into `report_data` binds the CVM and vTPM evidence to the same trust anchor, allowing external verification of the linkage between environment attestation and runtime measurements.

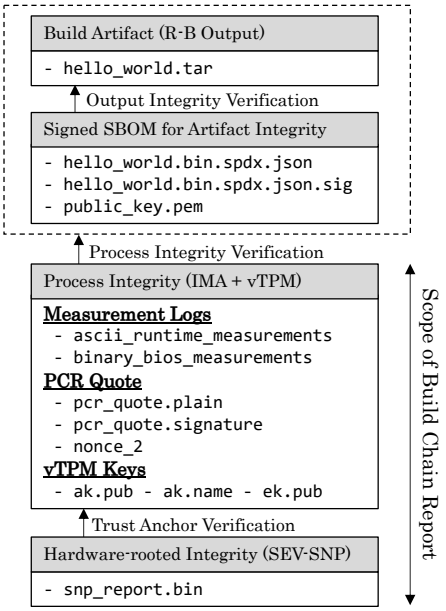

Fig. 5. Structure of Build Chain Report: Components and their roles in verifying artifact, process, and hardware integrity (`hello_world` package example).

The verification proceeds from the bottom-up following the structure shown in Figure 5. First, SEV-SNP attestation establishes a hardware-based trust anchor for the execution environment. Next, the verifier checks the vTPM quote and reconstructs PCR values from Trusted Boot and IMA logs to ensure that the recorded execution trace is consistent and tamper-evident. Finally, artifact integrity is validated by verifying SBOM signatures and matching file hashes with the produced artifact.

Concretely, verification consists of three steps corresponding to the components of the *Build Chain Report*: (i) hardware-rooted environment integrity: validating that the attested ex-

ecution environment matches reference values, (ii) process integrity: confirming that the execution trace derived from IMA measurements and PCR values is consistent and tamper-evident, and (iii) artifact and SBOM integrity: checking that binaries and files observed in the IMA measurements correspond to expected (i.e. reference values) toolchain components and declared SBOM entries.

C. Security Evaluation against the Threat Model

We evaluate whether the proposed approach enables verification of integrity deviations corresponding to the Threat Model (T1 and T2). This evaluation examines whether the proposed verification mechanism operates as intended in practice.

We confirm that files accessed and binaries executed during the build are recorded in the *Build Chain Report*, and that verification fails when tampering corresponding to T1/T2 is introduced, including substitution of executed toolchain binaries, modification of SBOM contents, and manipulation of measurement logs. These results demonstrate that deviations in build-time execution lead to verifiable inconsistencies in the recorded evidence.

For toolchain tampering (T1), the verifier compares hashes of executed binaries recorded in the IMA log with pre-registered reference values, allowing identification of substituted compiler or linker components. Because this verification is based on the binaries actually executed during the build, it captures runtime deviations that are not observable through the output equivalence in R-B. In particular, even if the same compromised toolchain is consistently used, its execution is exposed by a mismatch against expected toolchain references.

For SBOM tampering (T2), the verifier compares the artifact and dependency hashes declared in the SBOM with the corresponding file access measurements recorded in the IMA log. Because the SBOM represents declared composition while the IMA log reflects actual execution, mismatches provide verifiable evidence of inconsistency in the SBOM generation process.

These properties rely on the integrity of the execution evidence. IMA measurements are extended into vTPM PCRs and bound to the SEV-SNP attestation report, ensuring append-only and tamper-evident logs. Binding the vTPM public key hash to the attestation report further ensures linkage between the CVM and runtime measurements. Overall, the proposed mechanism enables external verification of executed toolchain usage and SBOM consistency solely from the *Build Chain Report*, without rebuilding.

D. Applicability and Performance Evaluation

We evaluate whether our approach can be applied to real-world OSS projects and measure its performance overhead. We selected four OSS projects with different languages and build systems: C++ *guetzli* and *abseil-cpp*, Rust *fd*, and Go *buildifier* [34]–[37]. All projects were obtained from GitHub and were successfully built within the R-CVM with signed SBOM verification and IMA-based execution tracking. To enable R-B within the R-CVM, Bazel build configurations were applied using `rules_pkg` [38]. Even for *fd*, which does not natively use Bazel, only minor modifications to its `BUILD` files were required. This suggests that our approach can be applied to a range of OSS projects with modest adaptation effort.

For performance evaluation, we compared build times in three environments: (1) baseline without SEV-SNP, (2) CVM with SEV-SNP only, and (3) R-CVM with SEV-SNP and IMA measurement. As shown in Table II, the SEV-SNP CVM introduces negligible overhead compared to the baseline. In contrast, the R-CVM incurs additional overhead as a result of IMA-based file-access measurement. The Bazel build ratio ranges from 80–97%, indicating that the dominant overhead originates from file-access latency during measurement. Although this overhead may limit applicability in iterative development scenarios, it remains practical for security-critical release builds, where verifiable build execution is required and justifies the additional measurement overhead.

TABLE II
BUILD TIME (AVERAGE ACROSS 5 BUILDS)

Environment	[34]	[35]	[36]	[37]
Baseline (sec.)	7.83	17.61	165.32	160.24
SNP-only CVM (sec.)	7.87	17.75	168.67	164.95
R-CVM (sec.)	39.98	61.67	891.87	313.81
— Breakdown of R-CVM —				
Build Execution (sec.)	33.29	49.75	865.34	278.52
Auxiliary Tasks (sec.)	6.69	11.92	26.53	35.29
Bazel build ratio (%)	83.3	80.7	97.0	88.8

[34]:*guetzli*, [35]:*abseil-cpp*, [36]:*fd*, [37]:*buildifier*

VI. DISCUSSIONS AND RELATED WORK

A. Limitations and Future Work

This work assumes a pre-verified trusted computing base (TCB) for executing R-B and does not address stepwise or recursive validation of the TCB itself. In principle, a TCB could be constructed with verifiable provenance and derive reference values using DDC [27], but the feasibility and selection of appropriate trust anchors remain open challenges.

From a system perspective, Linux IMA captures a broad range of activities inside the R-CVM. As a result, the *Build Chain Report* may include unrelated file accesses to the build process, introducing noise in execution evidence. This can reduce clarity when interpreting build-time behavior. Although such measurements may include unrelated file accesses and may vary due to build-time non-determinism, verification still focuses on expected toolchain components and relevant execution paths; however, reducing this noise is important for improving the clarity and practical usability of the resulting evidence, especially for large-scale builds. Mitigation strategies include re-initializing the R-CVM per build, refining IMA measurement policies, and augmenting logs with contextual information such as `uid` and `pid`. Reducing TCB through lightweight or separation-based kernels (e.g., Asterinas [39] or Framekernel [40]) is another promising direction.

Our implementation is currently limited to Bazel-based builds. Although the proposed mechanism is independent of specific build systems and does not require R-B, its applicability to other build systems has not yet been fully evaluated. Future work includes extending support to additional build systems, integrating with SBOM standards such as SPDX [41] and CycloneDX [42], and aligning with CI/CD pipelines and supply-chain frameworks such as OpenSSF S2C2F [43], MITRE SoT [44] and Binary Transparency [45].

B. Related Work

We compare representative related work to clarify the position of *Attestable Build Chain*. Table III presents a comparison across five criteria (C1–C5): protection of the execution-environment, root of trust, build integrity, transparency of provenance (e.g., via SBOMs) and verification without rebuild.

TABLE III
COMPARISON WITH RELATED WORK

Related Work	C1	C2	C3	C4	C5
Drexel et al. [46]	○	○	◐	◐	○
in-toto [47]	○	○	◐	●	○
Revelio [48]	●	●	○	○	○
Attestable Builds [20]	●	●	◐	○	●
Attestable Build Chain	●	●	●	●	●

●: Supported ◐ : Partially Supported ○: Not supported

C1: Execution Env. Protection **C2**: Root of Trust & RA
C3: Build Process Integrity **C4**: Provenance Transparency
C5: Verify w/o Rebuild

In terms of build integrity, prior work has focused on Reproducible Builds (R-B) and Diverse Double-Compiling (DDC). R-B enables third-party verification through output equivalence [46], but does not provide visibility into which components were actually executed during the build. As a result, attacks that preserve output equivalence remain unobservable. DDC [27] provides a foundation for compiler trust, and previous work has explored provenance validation and quantitative assurance of toolchains [21], [49]. However, these approaches primarily focus on validating compilers or final artifacts, rather than observing build-time execution.

In the context of software supply-chain security, in-toto [47] defines verifiable build workflows, and SLSA [19] organizes them into assurance levels. SBOM-related efforts such as SPDX and Sigstore/Cosign [50] improve transparency and artifact authenticity. However, these approaches rely on declared metadata or workflow descriptions and cannot independently verify whether the reported execution matches the actual system-level build execution.

Remote attestation has been widely used to verify execution environments using TPMs and CVMs. Systems such as Revelio [48], SNPGuard [51] and Narayanan et al. [23] verify the system configuration and the runtime integrity of CVMs. However, they focus on environment integrity and do not provide visibility into build-time execution behavior.

Attestable Builds [20] is the closest related work, enabling the attestable execution of the build within TEEs. Their approach verifies the declared build steps and tool execution through user-level reporting, where execution evidence is produced by the build tools themselves. As a result, a compromised toolchain may still produce execution records that cannot be independently verified against actual system-level behavior.

In contrast, our approach measures file accesses at the kernel level using Linux IMA and anchors them in a hardware-rooted chain of trust (vTPM and SEV-SNP). By measuring actual file accesses independently of the build process, this reduces reliance on potentially compromised components and enables external verification of which toolchain components were actually executed during the build. Overall, our approach complements prior work by providing independently verifiable execution evidence of the build process itself, rather than relying solely on output equivalence or declared execution.

VII. CONCLUSION

We proposed and implemented the *Attestable Build Chain*, a mechanism that enables third-party verification of software build processes. Although R-B verifies output equivalence, it does not reveal whether the build was executed with the intended toolchain and environment. By combining SEV–SNP-based CVMs, vTPM-rooted trust, and Linux IMA measurements, our approach extends attestation from static configurations to build-time execution.

The resulting *Build Chain Report* enables external verification of both artifacts and build-time execution without rebuilding. Our evaluation demonstrates applicability across OSS projects and shows that the overhead of IMA-based measurement is acceptable for security-critical release builds. Overall, our work complements R-B and SBOMs by providing verifiable execution evidence of the build process, addressing a key gap in software supply-chain assurance. Future work includes reducing the TCB, supporting additional build systems, and integrating with CI/CD pipelines and SBOM ecosystems.

ACKNOWLEDGMENT

We acknowledge NTT Social Informatics Laboratories for their cooperation in the early stages of this research, and thank Professor Eiji Kuwana and Professor Atsuhiro Goto of Institute of Information Security for their support in enabling this cooperation. Portions of this work were conducted with the assistance of ChatGPT.

REFERENCES

[1] B. M. Reichert and R. R. Obelheiro, "An Integrity-Focused Threat Model for Software Development Pipelines," *arXiv preprint arXiv:2211.06249*, 2022.

[2] M. Ohm, H. Plate, A. Sykosch, and M. Meier, "Backstabber's Knife Collection: A Review of Open Source Software Supply Chain Attacks," in *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment (DIMVA 2020)*. Springer, 2020, pp. 23–43.

[3] P. Ladisa, H. Plate, M. Martinez, and O. Barais, "SoK: Taxonomy of Attacks on Open-Source Software Supply Chains," in *IEEE Symposium on Security and Privacy (S&P 2023)*. IEEE, 2023, pp. 1509–1526.

[4] W. Enck and L. Williams, "Top Five Challenges in Software Supply Chain Security: Observations From 30 Industry and Government Organizations," *IEEE Security & Privacy*, vol. 20, no. 2, pp. 96–100, 2022.

[5] L. Williams, G. Benedetti, S. Hamer, R. Paramitha, I. Rahman, M. Tamanna, G. Tystahl, N. Zahan, P. Morrison, Y. Acar, M. Cukier, C. Kästner, A. Kapravelos, D. Wermke, and W. Enck, "Research Directions in Software Supply Chain Security," *ACM Transactions on Software Engineering and Methodology*, vol. 34, no. 5, pp. 1–38, 2025.

[6] B. Hammi, S. Zeadally, and J. Nebhen, "Security Threats, Countermeasures, and Challenges of Digital Supply Chains," *ACM Computing Surveys*, vol. 55, no. 14s, pp. 1–40, 2023.

[7] C. Lamb and S. Zacchiroli, "Reproducible Builds: Increasing the Integrity of Software Supply Chains," *IEEE Software*, vol. 39, no. 2, pp. 62–70, 2021.

[8] É. Ó. Muirí, "Framing Software Component Transparency: Establishing a Common Software Bill of Material (SBOM)," *NTIA, Nov*, 2019.

[9] K. Thompson, "Reflections on Trusting Trust," *Communications of the ACM*, vol. 27, no. 8, pp. 761–763, 1984.

[10] B. McDanel, "Beyond Trusting Trust: Multi-Model Validation for Robust Code Generation," *arXiv preprint arXiv:2502.16279*, 2025.

[11] S. Checkoway and H. Shacham, "Iago Attacks: Why the system call API is a bad untrusted RPC interface," *ACM SIGARCH Computer Architecture News*, vol. 41, no. 1, pp. 253–264, 2013.

[12] R. Cui, L. Zhao, and D. Lie, "Emilia: Catching Iago in Legacy Code," in *Network and Distributed System Security Symposium (NDSS 2021)*, 2021.

[13] R. Cui, *Detect Iago Vulnerabilities in Legacy Code with Reverse Syscall Fuzzing*. University of Toronto (Canada), 2021.

[14] Y. Xu, W. Cui, and M. Peinado, "Controlled-Channel Attacks: Deterministic Side Channels for Untrusted Operating Systems," in *IEEE Symposium on Security and Privacy (S&P 2015)*. IEEE, 2015, pp. 640–656.

[15] O. S. Hofmann, S. Kim, A. M. Dunn, M. Z. Lee, and E. Witchel, "InkTag: secure applications on an untrusted operating system," in *International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS 2013)*, 2013, pp. 265–278.

[16] C. Özkan, X. Zou, and D. Singelée, "Supply Chain Insecurity: The Lack of Integrity Protection in SBOM Solutions," *arXiv preprint arXiv:2412.05138*, 2024.

[17] C. Kocaoğullar, T. Marjanov, I. Petrov, B. Laurie, A. Cutter, C. Kern, A. Hutchings, and A. R. Beresford, "A Confidential Computing Transparency Framework for a Comprehensive Trust Chain," *arXiv preprint arXiv:2409.03720*, 2024.

[18] A. Delignat-Lavaud, C. Fournet, K. Vaswani, S. Clebsch, M. Riechert, M. Costa, and M. Russinovich, "Why Should I Trust Your Code?" *Communications of the ACM*, vol. 67, no. 1, pp. 68–76, 2024.

[19] M. Tamanna, S. Hamer, M. Tran, S. Fahl, Y. Acar, and L. Williams, "Unraveling Challenges with Supply-Chain Levels for Software Artifacts (SLSA) for Securing the Software Supply Chain," *arXiv preprint arXiv:2409.05014*, 2024.

[20] D. Hugenroth, M. Lins, R. Mayrhofer, and A. R. Beresford, "Attestable builds: compiling verifiable binaries on untrusted systems using trusted execution environments," in *Proceedings of the 2025 ACM SIGSAC Conference on Computer and Communications Security (CCS 2025)*, 2025, pp. 4514–4528.

[21] X. de Carné de Carnavalet and M. Mannan, "Challenges and implications of verifiable builds for security-critical open-source software," in *Annual Computer Security Applications Conference (ACSAC 2014)*, 2014, pp. 16–25.

[22] AMD, "AMD SEV-SNP: Strengthening VM Isolation with Integrity Protection and More," *White Paper*, January 2020.

[23] V. Narayanan, C. Carvalho, A. Ruocco, G. Almasi, J. Bottomley, M. Ye, T. Feldman-Fitzthum, D. Buono, H. Franke, and A. Burtsev, "Remote attestation of confidential VMs using ephemeral vTPMs," in *Annual Computer Security Applications Conference (ACSAC 2023)*, 2023, pp. 732–743.

[24] A. Steffen, "The Linux Integrity Measurement Architecture and TPM-Based Network Endpoint Assessment," in *Linux Security Summit*, 2012.

[25] Linux Kernel Community, "Integrity Measurement Architecture (IMA) - Event Log Format," https://ima-doc.readthedocs.io/en/latest/event-log-format.html, 2023, (Accessed: 2026-04-06).

[26] H. Birkholz, D. Thaler, M. Richardson, N. Smith, and W. Pan, "RFC 9334: Remote ATtestation procedureS (RATS) Architecture," 2023.

[27] D. A. Wheeler, "Countering Trusting Trust through Diverse Double-Compiling," in *Annual Computer Security Applications Conference (ACSAC 2005)*. IEEE, 2005, pp. 28–40.

[28] G. Scopelliti, C. Baumann, and J. T. Mühlberg, "Understanding Trust Relationships in Cloud-Based Confidential Computing," in *IEEE European Symposium on Security and Privacy Workshops (EuroS&PW 2024)*. IEEE, 2024, pp. 169–176.

[29] J. Eisoldt, A. Galanou, A. Ruzhanskiy, N. Küchenmeister, Y. Baburkin, T. Dai, I. Gudymenko, S. Köpsell, and R. Kapitza, "SoK: A cloudy view on trust relationships of CVMs–How Confidential Virtual Machines are falling short in Public Cloud," *arXiv preprint arXiv:2503.08256*, 2025.

[30] COCONUT-SVSM Project, "COCONUT Secure VM Service Module (SVSM)," https://github.com/coconut-svsm, 2023, (Accessed: 2025-03-31).

[31] Google, "Bazel: Fast, Reliable, Scalable Build System," https://bazel.build/, 2015, (Accessed: 2025-03-31).

[32] AMD, "SEV-SNP Platform Attestation Using VirTEE/SEV," https://www.amd.com/content/dam/amd/en/documents/developer/58217-epyc-9004-ug-platform-attestation-using-virtee-snp.pdf, 2023, document No. 58217, Version 1.2.

[33] VirTEE, "snpguest: A CLI tool for interacting with SEV-SNP guest environment," https://github.com/virtee/snpguest, 2023, (Accessed: 2025-03-31).

[34] Google, "Guetzli: Perceptual JPEG encoder," https://github.com/google/guetzli, 2017, (Accessed: 2025-04-12).

[35] Abseil, "Abseil C++ Common Libraries," https://github.com/abseil/abseil-cpp, 2017, (Accessed: 2025-04-12).

[36] D. Peter, "fd: A simple, fast and user-friendly alternative to 'find'," https://github.com/sharkdp/fd, 2017, (Accessed: 2025-04-12).

[37] Bazel Project - buildtools, "A bazel BUILD file formatter and editor," https://github.com/bazelbuild/buildtools, 2019, (Accessed: 2025-04-12).

[38] Bazel Project - rules_pkg, "Bazel packaging rules," https://github.com/bazelbuild/rules_pkg, 2018, (Accessed: 2025-04-12).

[39] Y. Peng, H. Tian, J. Zhang, R. Li, C. Chen, J. Jiang, J. Xian, X. Wang, C. Xu, D. Zhou, Y. Luo, S. Yan, and Y. Zhang, "Asterinas: A Linux ABI-Compatible, Rust-Based Framekernel OS with a Small and Sound TCB," in *USENIX Annual Technical Conference (USENIX ATC 2025)*, 2025.

[40] Y. Peng, H. Tian, J. Xian, S. Zhou, S. Yan, and Y. Zhang, "Framekernel: A Safe and Efficient Kernel Architecture via Rust-based Intra-kernel Privilege Separation," in *ACM SIGOPS Asia-Pacific Workshop on Systems (APSys 2024)*, 2024, pp. 31–37.

[41] Linux Foundation SPDX Project, "SPDX Specification," https://spdx.dev/specifications/, 2024, (Accessed: 2026-04-06).

[42] CycloneDX Core Working Group, "OWASP CycloneDX: Authoritative Guide to SBOM Second Edition," OWASP Foundation, Technical Guide, 2024. [Online]. Available: https://cyclonedx.org/guides/OWASP_CycloneDX-Authoritative-Guide-to-SBOM-en.pdf

[43] Open Source Security Foundation (OSSF), "Secure Supply Chain Consumption Framework (S2C2F)," https://github.com/ossf/s2c2f, 2022, (Accessed: 2026-04-06).

[44] MITRE, "System of Trust Framework," https://sot.mitre.org/framework/system_of_trust.html, 2022, (Accessed: 2026-04-06).

[45] Google, "Building Trust in the Software Supply Chain," https://binary.transparency.dev/, 2021, (Accessed: 2026-04-06).

[46] J. Drexel, E. Hänggi, and I. M. Veiga, "Reproducible Builds and Insights from an Independent Verifier for Arch Linux," in *Sicherheit 2024*. Gesellschaft für Informatik eV, 2024, pp. 243–257.

[47] S. Torres-Arias, H. Afzali, T. K. Kuppusamy, R. Curtmola, and J. Cappos, "in-toto: Providing farm-to-table guarantees for bits and bytes," in *USENIX Security Symposium (USENIX Security 2019)*, 2019, pp. 1393–1410.

[48] A. Galanou, K. Bindlish, L. Preibsch, Y.-A. Pignolet, C. Fetzer, and R. Kapitza, "Trustworthy confidential virtual machines for the masses," in *International Middleware Conference (Middleware 2023)*, 2023, pp. 316–328.

[49] D. Volpano, D. Malzahn, A. Pareles, and M. Thober, "Quantitative Toolchain Assurance," *arXiv preprint arXiv:2308.16275*, 2023.

[50] Linux Foundation SPDX Project, "A Step-by-Step Guide to Signing an SPDX SBOM with Sigstore's Cosign," https://spdx.dev/a-step-by-step-guide-to-signing-an-spdx-sbom-with-sigstores-cosign/, 2023, (Accessed: 2026-04-06).

[51] L. Wilke and G. Scopelliti, "SNPGuard: Remote Attestation of SEV-SNP VMs Using Open Source Tools," in *IEEE European Symposium on Security and Privacy Workshops (EuroS&PW 2024)*. IEEE, 2024, pp. 193–198.