From Hardware to Artifact: Trusted Software Builds with Remote Attestation

Behnaz Hassanshahi
Oracle
Brisbane, Queensland, Australia
behnaz.hassanshahi@oracle.com

Rohan Kollambalath*
Oracle
Brisbane, Queensland, Australia
rohankollambalath@gmail.com

Trong Nhan Mai
Oracle
Brisbane, Queensland, Australia
trong.nhan.mai@oracle.com

Jagannathan Raman
Oracle
Westford, USA
jag.raman@oracle.com

Ian Chin Wang
Oracle
Nashua, New Hampshire, USA
ian.chin.wang@oracle.com

Abstract

Software supply chain attacks, such as the SolarWinds breach, have exposed the risks of compromised build environments, where attackers can inject malicious code or tamper with provenance records. While build provenance helps track artifact creation, it cannot guarantee trust unless the build environment itself is secure, particularly in cloud-based virtualized systems that are vulnerable to host or hypervisor-level attacks.

We present a hardware-backed architecture designed to secure cloud-based virtual machine (VM) build environments by leveraging confidential computing to minimize the Trusted Computing Base (TCB). Our solution builds upon AMD SEV-SNP and Secure Boot to establish a strong root of trust for the build platform. We combine hardware-based remote attestation for verifiable system state reporting with runtime enforcement using Linux Integrity Measurement Architecture (IMA) and Security-Enhanced Linux (SELinux). This ensures that builds are executed only within trusted, tamper-resistant environments, and that provenance records are cryptographically bound to a hardware root of trust. We evaluate our approach on confidential VMs and demonstrate that it offers strong security guarantees with low performance overhead.

ACM Reference Format:
Behnaz Hassanshahi, Rohan Kollambalath, Trong Nhan Mai, Jagannathan Raman, and Ian Chin Wang. 2025. From Hardware to Artifact: Trusted Software Builds with Remote Attestation. In *Proceedings of the 2025 Workshop on Software Supply Chain Offensive Research and Ecosystem Defenses (SCORED '25), October 13–17, 2025, Taipei, Taiwan.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3733827.3765524

1 Introduction

In recent years, software supply chain security has emerged as a critical concern, with large-scale cyber attacks such as the SolarWinds breach [19] demonstrating the potential for adversaries to infiltrate and compromise the software development process. These attacks have underscored the vulnerabilities inherent in modern software supply chains, where build systems, dependencies, and third-party libraries present potential vectors for exploitation. As software development increasingly relies on complex, multi-tiered supply chains, ensuring the integrity of the build process has become an essential requirement for securing software artifacts.

To mitigate these risks, several frameworks have been developed to ensure that software is built securely and that its integrity can be verified. One widely adopted approach is the generation of provenances proposed by frameworks like Supply Chain Levels for Software Artifacts (SLSA) [20]. SLSA addresses these risks by defining security levels, emphasizing on provenance generation and verification, with higher levels requiring builds on trusted infrastructure and cryptographic signatures to prevent forgery. Such provenances help establish trust in software components and allows for post-build verification to detect potential tampering. However, the effectiveness of provenance ultimately depends on the trustworthiness of the underlying system where the build takes place. If an attacker compromises the build machine or the environment in which the provenance is generated, the provenance itself may be manipulated, making it unreliable.

In today's computing landscape, most build processes are executed on cloud-based VMs, introducing additional security risks. Cloud environments operate under a shared responsibility model, where infrastructure security is managed by cloud providers, but virtualized workloads are the responsibility of tenants. This model introduces potential security risks: an attacker or insider threat actor who launches VM escape attacks, gains control of a hypervisor, exploits security flaws in the host operating system (OS) or shared resources may compromise virtual machines running on the platform. A successful attacker could manipulate build artifacts, inject malicious code, or alter provenance records, all while maintaining the appearance of a legitimate build process. They might either tamper with an authentic build provenance or forge an illegitimate one to make it appear valid, effectively bypassing verification and compromising the integrity of the software supply chain.

Efforts such as reproducible builds aim to detect malicious behavior by ensuring that the rebuilt artifact is bitwise identical to the original [14]. While this approach verifies that the artifact matches the expected output, it does not address the case where the original build components themselves may have been compromised. Consequently, there is a clear need for a verifiable mechanism to audit previous builds, enabling the detection of such compromises.

*This work was done while the author was at Oracle.

SCORED '25, Taipei, Taiwan

ACM ISBN 979-8-4007-1915-8/25/10
https://doi.org/10.1145/3733827.3765524

Given these security threats, it is essential to move toward a hardware-based, zero-trust architecture that secures the build platform from the ground up. A secure build system must be resistant to tampering, capable of detecting unauthorized modifications, and able to provide strong guarantees about the provenance of generated artifacts. Moreover, it should offer remote attestation mechanisms that allow third parties to independently verify the state of the system at the time of build execution. Without such assurances, there is no reliable way to trust that a given artifact has been produced in a secure and uncompromised environment.

To tackle these challenges, we propose an architecture with minimal trust assumptions, built on a hardware-based attestation framework to secure remote build environments. Our approach integrates confidential computing technologies, remote attestation, and fine-grained security policies to create a robust foundation for performing secure builds. By leveraging modern hardware features, such as AMD Secure Encrypted Virtualization-Secure Nested Paging (SEV-SNP) [1], our framework ensures that a build machine's execution environment remains protected, even in the presence of a compromised host OS or hypervisor. Additionally, by incorporating Linux Integrity Measurement Architecture (IMA) and Security-Enhanced Linux (SELinux) policies, we further enforce runtime security controls that safeguard against unauthorized modifications and privilege escalation attacks. Our contributions in this paper are threefold:

(1) **Secure-by-design architecture with minimal trust assumptions.** Our architecture is tamper-resistant by design, built on a hardware root of trust (e.g., AMD SEV) and independently verified attestation (e.g., Veraison [23]). Trust is reduced to the minimum necessary. We use Secure Boot, Linux IMA and SELinux to enforce runtime policies that protect critical binaries and system files, ensuring integrity and security throughout execution.
(2) **Holistic and layered trust establishment.** Our system establishes trust in a layered and progressive manner, starting from the hardware level and extending through the entire software stack. This approach incorporates chained attestations and continuous verification, enabling to move toward a zero-trust model when the build process is untrusted.
(3) **Advanced verification through deep system insight.** We introduce novel techniques for artifact verification by analyzing diverse, underutilized data sources, such as system logs, configuration states, and runtime behaviors. This rich context is evaluated by a policy engine, enabling a more comprehensive and proactive security posture that extends and enhances traditional confidential computing techniques. The system is designed to detect threats across the software lifecycle.

2 Background

In this section, we introduce key terminologies and concepts that form the foundation of our system design.

2.1 Provenances and Attestations

Provenance refers to a detailed, record documenting how a software artifact (e.g., a binary) was built, including its inputs, the build process, and the entities involved. Attestation is the cryptographic proof that the provenance is trustworthy. Attestation makes provenance actionable for security and compliance. It is a cornerstone of modern software supply chain security [8, 20], offering transparency into the components and procedures used during artifact creation. By enabling verification against expected inputs and workflows, provenance helps ensure builds are trustworthy and reproducible. However, maintaining the integrity of provenance remains a challenge. Trust in the provenance ultimately depends on the trustworthiness of the machine and environment where it is generated. This underscores the need for robust mechanisms to validate the security and integrity of build systems.

2.2 Confidential Computing

Confidential computing, particularly with technologies like AMD SEV-SNP [1], offers a framework that can be used for securing build systems. By enabling workloads to run within encrypted virtual machines, confidential computing ensures that sensitive data and processes remain protected from tampering and unauthorized access even in environments where the host may be compromised. A critical feature of AMD SEV is its ability to generate hardware-based cryptographic keys unique to each CPU. These keys enable the signing of attestations bound to the physical hardware, establishing a root of trust from the hardware manufacturer to the runtime environment. Remote attestation allows external verification servers to validate these signed reports, ensuring the integrity of the machine running the build. Veraison [23], an open-source, standards-based framework based on the IETF's Remote ATTestation ProcedureS (RATS) [3] architecture, simplifies this process by providing modular components adaptable to diverse attestation use cases.

By leveraging confidential computing and remote attestation, we can ensure that the provenance of a build is both accurate and securely tied to the specific virtual machine (VM) in which the build occurred. However, the provenance of the build alone is insufficient unless the underlying system can also be demonstrated as secure. This requires comprehensive mechanisms to verify and maintain the integrity of the build system itself, ensuring that every layer of the hardware, firmware, operating system, and application adheres to a strong security posture.

2.3 Secure Boot Enforcement

Secure Boot enforcement works by ensuring that only software signed with trusted cryptographic keys is allowed to execute during the boot process [7]. In cloud and virtualized environments Secure Boot starts with a hardware-based root of trust, typically stored in immutable memory (like fuses or a secure chip), which contains the public key(s) of trusted firmware or bootloader signers. At each stage of the boot process (firmware, bootloader, OS kernel), the system checks the digital signature of the next component before allowing it to execute. If the signature matches a trusted key in the system's database, the component is loaded; otherwise, booting is halted or a recovery process is initiated. The system maintains databases of allowed ('db') and forbidden ('dbx') signatures. Only components signed by keys in the allowed database can be loaded, while those in the forbidden database are blocked. The enforcement mechanism itself is designed to be tamper-resistant, so attackers cannot bypass or modify the verification process.

2.4 Linux Integrity Measurement Architecture

The Linux Integrity Measurement Architecture (IMA) [21] is a kernel-based framework that provides a mechanism to verify the integrity of files and binaries on a system. IMA ensures that files are not modified without authorization by checking their cryptographic hash against a previously recorded value before granting access. If a mismatch is detected, access to the file is denied. Similarly, IMA verifies the digital signatures of binaries to ensure they have not been tampered with, providing an additional layer of assurance. IMA is well-suited for protecting static files and binaries that are critical to the operation of a secure build environment. By enforcing well-defined IMA policies, we can provide verifiable guarantees about the integrity of the build system, ensuring that the system operates in a known and trusted state. While recent work [5] identified challenges in verifying IMA hashes for all materials in the SBOM and proposed using eBPF to prevent attribute modifications during builds on certain Linux distributions, newer versions of Oracle Linux [1] offer significantly improved IMA support. For example, Oracle Linux 9 (OL9) [6] enables digital signature-based IMA appraisal, eliminating the need for the Extended Verification Module (EVM). EVM was previously vulnerable to offline attacks, as hashes could be generated and written directly into extended attributes. In contrast, OL9 protects extended attributes using Oracle-signed digital signatures, preventing offline tampering. This approach simplifies system administration and facilitates rapid deployment, thereby strengthening the overall security and integrity posture.

2.5 SELinux

Security-Enhanced Linux (SELinux) [9] is another critical technology that can be used to enhance the security of build environments. SELinux enables fine-grained control over how processes, users, and resources interact on a system. By assigning security labels (types) to system resources and enforcing strict access control policies, SELinux reduces the attack surface and minimizes the risk of privilege escalation or unauthorized access. When used in conjunction with IMA, SELinux provides complementary security capabilities. SELinux focuses on access control and process isolation, while IMA ensures the integrity of the files and binaries being accessed. The state of these policies and their logs could also be included in hardware signed attestation reports to then provide assurance about the state of the system.

3 Threat Model and Proposed Solution

Our design assumes that the build processes occur within ephemeral containers on a guest VM in a cloud environment, though the guest confidential VM itself may or may not be ephemeral. In our threat model, we consider an attacker who may gain access to the hypervisor or be granted access to the confidential VM, but without root privileges. In this scenario, the attacker could execute untrusted build processes within the VM by exploiting vulnerabilities in the container environment where the build takes place. Alternatively, the attacker could move laterally to gain unauthorized access to the host or guest VM, operating as an insider threat. Although the attacker does not have full control over the system, they can still manipulate critical aspects of the build environment, such as tampering with build artifacts or injecting malicious code, to undermine the integrity of the build process.

Trusted Computing Base (TCB). : We aim to protect every layer of the build environment, from the hardware up to the final artifact, while keeping the TCB as minimal as possible. The TCB refers to the set of components within the system that are trusted and operate without additional protective measures. To minimize the TCB, our design trusts only the hardware, the attestation server, and the guest Operating System (OS) image. We trust the AMD SEV-SNP feature and its ability to generate hardware-based cryptographic keys unique to each CPU. The attestation server is an open-source component [23], designed to be transparent and verifiable by multiple parties. It is actively maintained and improved by a diverse set of contributors providing security enhancements, and ongoing audits. We assume that the original guest OS image, which is the trusted operating system used within the VM, is secure when it is first deployed. However, to address the possibility of deviation or compromise once the VM is booted, we provide mechanisms to detect any unauthorized changes to the system's configuration as explained in this section.

We have identified security levels based on the critical assets that need protection throughout the software build process. At each level, we rely on remote attestation to validate the integrity of the environment before proceeding to the next stage of the build. To reach and verify a given security level, each component in the stack must communicate specific evidence to the attestation server, which enforces security policies and verifies the integrity of the environment through attestations. Figure 1 illustrates this layered architecture and its corresponding TCB, while Table 1 outlines potential attack vectors and their mitigations across six levels, from host and guest operating systems (Level 1) to the final build artifact (Level 6). In the following sections, we will explain how each mitigation strategy addresses the attacks listed at the corresponding level.

While we provide protection mechanisms at each level to mitigate targeted attacks, we also leverage chained attestations, built upon remote attestation techniques, to gradually establish trust throughout the stack. Each level can access the attested evidence from the previous one, including details about its protection mechanisms. This enables a trust chain anchored to the hardware, minimizing the system's attack surface and strengthening the credibility of the provenance.

3.1 Level 1: Host and Guest Operating Systems

To address the adversaries outlined in our threat model, who may gain access to the hypervisor or host operating system, our solution leverages the principles of confidential computing (see Section 2.2). Specifically, we utilize hardware-based Trusted Execution Environments (TEEs) to isolate and encrypt the memory of the guest virtual machine (VM), ensuring that even privileged entities, such as the host OS or hypervisor, cannot access or tamper with the build environment running inside the guest VM. Complementing

[1]Oracle Linux and Java are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

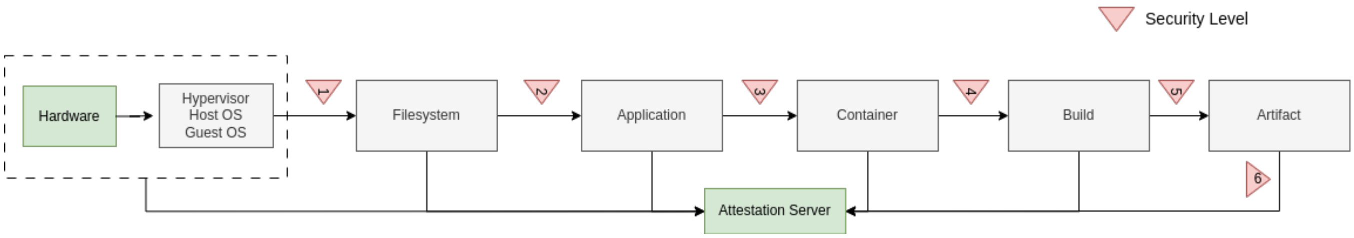

Figure 1: Trusted Computing Base and Attestation Levels

Level	Potential Attacks	Mitigations
Host and Guest Operating Systems (Level 1)	• Hypervisor exploits • VM escape attacks • Bootloader or firmware attacks (e.g., BIOS/UEFI rootkits) • Cloud management API exploits • Host OS attacks	• Secure Boot Enforcement • Confidential Computing • Remote Attestation of the underlying bootloaders and operating systems
Filesystem (Level 2)	• Tampering of system binaries • Tampering of essential system configurations and files	• Linux IMA • Remote Attestation of configurations and policies
Application (Level 3)	• Attackers running malicious executables on the machine • Tampering of application code • Unauthorized access to resources • Attacker generates fake attestation reports • Application compromise risking machine integrity	• SELinux for process and resource confinement • IMA for static code protection • IMA to ensure only signed binaries may run on the machine
Container (Level 4)	• Container escape attacks tampering guest VM and build resources • Attackers attempt to enter containers	• SELinux to isolate containers and restrict users' control over containers
Build (Level 5)	• Malicious build code injecting unexpected code • Malicious source pulled from version control	• Partial support with build log monitoring • Source code analysis and review
Artifact (Level 6)	• Attacker compromises the artifact repository • Impersonation attacks	• The artifact hash in the attestation does not match the malicious artifact • Check the availability of a verifiable attestation for the artifact

Table 1: Security Levels, Attacks, and Mitigations

this, we use the Secure Boot mechanism, as described in Section 2.3, which ensures that only signed, trusted images can be launched, thereby maintaining the integrity of both the firmware and the boot chain. While these techniques mitigate certain cloud threats, such as those targeting the hypervisor or the guest OS, they do not provide a complete defense against all potential attacks (e.g., management API exploits). Nevertheless, they offer robust protection against hypervisor escapes, firmware tampering, and other host-level compromises.

We use remote attestation to verify the configuration and state of key system components. Immediately after boot, an attestation is performed on the firmware and kernel to confirm their integrity. The attestation server defines acceptable configurations and verifies that reports originate from the correct VM, forming the basis for a secure chain of trust.

This chain extends to the execution of the build and artifact generation, enabling the system to detect and respond to malicious behavior in real time, such as halting untrusted builds. Each attestation includes a reference to the previous one, ensuring that if any lower-level attestation fails, the system will prevent further progress at higher levels. To avoid redundancy, we will omit further mention of this process at subsequent layers.

3.2 Level 2: Filesystem

After boot completes and a root of trust is established through mechanisms, such as Secure Boot and confidential computing, critical system files and configurations remain vulnerable to tampering. Attackers commonly target these components (e.g., `/bin/bash`, `/usr/bin/sudo`, `/etc/shadow`, and cloud agents) to escalate privileges, persist on the system, or alter behavior. To protect these static assets, we leverage Linux's Integrity Measurement Architecture (IMA) in combination with remote attestation.

IMA, as described in Section 2.4, allows for the definition of measurement, appraisal, and audit policies that specify which files are measured and how their integrity is verified. These policies are typically loaded at boot, cryptographically signed, and validated using the IMA keyring. Our attestation process verifies not only the measurement list but also the integrity of the IMA policy itself, ensuring a consistent and trusted security posture.

Through this approach, we can confirm that all critical binaries and configurations are signed and protected from unauthorized modification. If any protected file is altered (e.g., `/etc/sudoers`), IMA will block access or execution. Additionally, IMA can safeguard static components of our build orchestrator and validate SELinux configuration files, which are expected to remain unchanged throughout the build. Importantly, IMA policies cannot be disabled without rebooting, reinforcing continuous enforcement during the machine's operation.

3.3 Level 3: Application

While IMA secures static files and binaries, SELinux enforces fine-grained, mandatory access control over users, processes, and system resources. By tightly restricting the actions applications can perform and the files they can access, SELinux effectively mitigates the impact of adversaries who gain access to the guest VM, acting as insider threats. SELinux also limits access to security-critical resources, such as those involved in attestation, helping to protect the integrity of the IMA subsystem itself. Together, IMA and SELinux create a layered defense: IMA ensures the system runs only verified code, while SELinux enforces strict containment of process behavior, even in the presence of a breach. The system generates an attestation including both SELinux and IMA logs as evidence payload, which can be verified by a policy engine.

3.4 Level 4: Container

To defend against adversaries who inject malicious code into build scripts and attempt to escape the container during the build process, we leverage SELinux to protect both the containers and dynamically modified files from tampering. SELinux allows us to define strict policies that govern which processes and resources are authorized to perform specific actions, ensuring that build containers remain isolated from the host system and from each other. Additionally, the integrity of the SELinux policy can be remotely verified through attestation before the build begins, further strengthening trust in the system.

SELinux assigns security labels to processes and resources, e.g., by labeling container processes as `container_t` to enforce mandatory access control. This ensures that even if an attacker escapes the container's namespace, they are still blocked from accessing host files or processes with different labels. Policies strictly govern what files and directories a container can access, limiting the attacker's ability to modify build artifacts or interact with other containers. Additionally, SELinux enforces separation between containers and restricts which host processes can interact with them, significantly reducing the risk of lateral movement or unauthorized access, even in the presence of vulnerabilities in the container runtime or orchestration layer.

3.5 Level 5: Build

To mitigate attacks such as malicious build code injecting unauthorized behavior or compromised source code from version control, we collect and include build logs in the attestation report. This provides partial protection by enabling post-build monitoring and auditability. However, this mechanism does not prevent the generation of malicious artifacts when the source code itself is malicious. In such cases, complementary techniques, such as static analysis [18], code reviews, and source integrity verification are necessary to ensure end-to-end trust in the build pipeline.

3.6 Level 6: Artifact

Once a build completes, a detailed provenance record is generated for each artifact, capturing the hash of the artifact itself and all its components. This provenance is included within an attestation report, which, like the attestations at lower levels, is cryptographically signed using the hardware root of trust from the build machine. The attestation report, which also contains a hash of the provenance is sent to an attestation server for verification and stored securely alongside the artifact.

By publishing this signed metadata with the artifact, we establish a strong, verifiable link between the artifact and the trusted machine that built it. Since the build machine is hardened and its root of trust is protected, we can be confident in the integrity of the resulting artifact. Additionally, storing signed attestation reports provides an immutable audit log of the build process, enabling retrospective analysis in case of suspected tampering.

To protect against threats such as repository compromise or impersonation, the system enforces strict attestation requirements. If an attacker replaces a legitimate artifact in the repository with a malicious one, the artifact hash in the attestation will no longer match, and the system will reject the compromised artifact. Similarly, if an attacker attempts to inject an artifact without a valid attestation, such as through impersonation, the system will block its use. Only artifacts with properly signed and verified attestations are accepted, ensuring that only trusted, unaltered artifacts are deployed.

4 Implementation

We developed a proof-of-concept system that hardens Reproducible Central (RC) [15], an open-source platform for building and verifying Java artifacts in containerized environments. Its build model aligns well with our architecture, allowing us to integrate policy enforcement and secure provenance tracking. We configured Linux IMA to protect the static source code and binaries, while SELinux isolates temporary files and processes during the build. The provenance generated is secured on the machine and inaccessible to users. Access to the attestation generation tool is strictly limited to the program that runs the build, which is the only entity permitted to run containers or initiate attestations.

Our foundation for zero trust begins with hardware-backed attestation using AMD SEV-SNP. This technology provides cryptographic verification of the build VM's state, using Versioned Chip

Endorsement Keys (VCEKs) signed by AMD's key infrastructure. This ensures reports originate from a genuine SEV-SNP-enabled VM, isolating it from the host OS and hypervisor.

Attestation evidence is encoded using Entity Attestation Tokens (EAT) [10], which encapsulate configuration data, hardware keys, and signed system state reports. Communication between the build VM and the attestation server occurs over HTTPS, and all data is cryptographically signed to prevent tampering or replay attacks. Each session uses a time-limited nonce signed by the VM's hardware, ensuring freshness and authenticity.

Our attestation flow begins by verifying the guest operating system kernel, then enforcing IMA and SELinux policies. IMA verifies critical files using signed signatures, and SELinux restricts access based on predefined type-enforced rules. Only the build driver, which is also protected by IMA, is authorized to execute the attestation generator. Currently, the tool used for performing remote attestation is called EATGEN. EATGEN is the sole resource on the system authorized to generate attestation reports. This exclusivity is enforced through SELinux policies, ensuring that only the EATGEN binary has access to the `/dev/sev-guest` file, which is essential for generating attestations.

Additionally, EATGEN requires access to several root-owned system files for logging and performing specific actions. To enable its operation as a non-privileged user, we implemented the SUID mechanism, allowing users to execute EATGEN with root privileges. While alternative configurations could eliminate the need for SUID, such changes could introduce new security weaknesses, such as non-root users gaining access to root-protected resources or increasing the complexity of the security framework.

Figure 2 shows how SELinux confines all components, allowing only a restricted build user to invoke the build process. The build driver interacts with the source code and attestation components, while all other access is denied by policy. Any attempt to subvert the build or tamper with configurations triggers IMA or SELinux violations, which are included in attestation logs and can invalidate the build.

We use Veraison [23] as our attestation server to verify attestation reports, enforce configuration policies, and halt the build process upon detecting integrity violations. Using Veraison, we check kernel and policy measurements, ensuring that compromised builds cannot disable or modify protections. Builds occur in isolated containers, with system-level protections validated continuously from boot. It is important to note that for hermetic builds [20], the Veraison attestation server must be included in the list of permitted entities.

Upon build completion, the resulting artifact is hashed and included in a final hardware-signed attestation report. This report is stored as part of the artifact's metadata. Further verification is possible by comparing the artifact hash with the attested hash, using the public key associated with the originating hardware, ensuring strong, cryptographically verifiable provenance.

5 Evaluation

In this section, we present case studies to assess the effectiveness of our approach in mitigating real-world threats and evaluate the performance overhead introduced by our system.

The experiments were conducted on a confidential-computing-enabled VM with the following specifications: an `x86_64` AMD EPYC-Milan-v2 processor, 4 GB of RAM, and a hypervisor using QEMU with KVM support. The operating system used was Oracle Linux 9.5, and the container engine was Podman. The build target was a Java artifact [2], compiled using Java 17. An initial warm-up build populated the cache, and the `-pull-never` flag was used with Podman to minimize network effects. The IMA policy is tailored for Reproducible Central and excluded frequently changing temporary files. SELinux policies ensures type isolation for those files and enforced access control during builds. The Reproducible Central script was used for executing the builds.[3]

Next, we present two representative case studies; however, our system supports a wide range of scenarios that could be similarly explored.

5.1 Case Study 1

This case study demonstrates a scenario where an attacker attempts to compromise the provenance of a software build and tamper with the build artifacts. The goal is to show how our system securely generates, signs, and verifies provenance using hardware-backed attestation and ensures that tampering is detected, even if an attacker manages to modify the artifact.

In this scenario, the user operates with unprivileged access (UID 1000), and SELinux is enabled to enforce strict security policies on the system. The source code for Reproducible Central is assumed to be pre-loaded, and all required setups are completed. SELinux will help ensure the safety of dynamic files and will provide granular control over which processes can access resources. Additionally, Linux IMA is used to ensure static files and binaries are untampered and trusted.

The primary goal in this case study is to demonstrate how to securely generate and verify build provenance using remote attestation. The build process begins with running a Maven build, which produces a build provenance. This provenance is encapsulated in an attestation report within an Entity Attestation Token (EAT), signed by the machine's hardware root of trust. The EAT is then sent to the remote attestation server (Veraison), which verifies the provenance and returns an Entity Attestation Response (EAR), confirming the build's integrity and security. With SELinux enabled, the unprivileged user (UID 1000) is restricted from accessing directories, modifying the source code, or altering the build script. The user is also unable to generate attestations or tamper with the build process, ensuring that the system remains secure and trusted throughout the process.

Mitigation (Provenance Tampering Detection). Once the build is complete, the attestation report is stored locally on the machine, ensuring that it cannot be tampered with without detection.

- Any tampering with the artifact causes a mismatch in the hash values.

[2] `com.io7m.immutables-style-1.0.0`

[3] RB_OCI_ENGINE="podman" RB_OCI_ENGINE_RUN_OPTS="–pull=never –userns=keep-id" RB_OCI_VOLUME_FLAGS=":Z,rw" ./reproducible-central/rebuild.sh reproducible-central/content/com/io7m/immutables-style/ com.io7m.immutables-style-1.0.0.buildspec

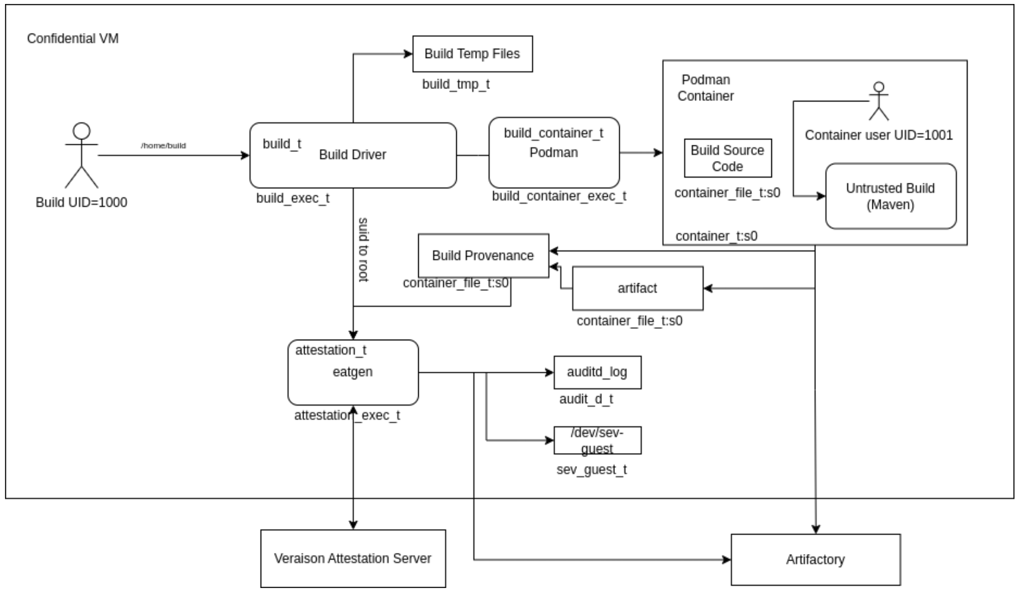

Figure 2: Hardening the build system with SELinux

- The integrity of the provenance and artifact is validated by comparing the hash stored within the EAT/EAR tokens, ensuring that any change in provenance is detected.
- Changing the provenance breaks the hash comparison, and the attestation report becomes invalid.
- When attempting to send the modified EAT to the verification server, the server responds with an error, indicating a failed attestation, as the hardware signature is no longer valid.

This experiment illustrates the critical role of hardware-backed attestation in ensuring the integrity of software provenance. By tying the provenance hash to a secure root of trust, we prevent attackers from tampering with build artifacts or forging attestation reports. Our approach ensures that only trusted builds are recognized, providing a reliable mechanism to validate software supply chain security.

5.2 Case Study 2

This case study demonstrates an attack where a compromised build container attempts to escape its isolated environment and tamper with host binaries. The attacker assumes control of a privileged container and tries to modify host files by mounting the host filesystem into the container. To simulate this, SELinux was temporarily disabled, and a Podman container was launched with elevated privileges and a host directory mounted. This container represents an untrusted build attempting to interfere with the build pipeline. The attacker modifies a trusted system binary (e.g., `vim`) from within the container.

Observed Behavior (SELinux Off). With SELinux disabled:

- The attacker successfully modified `/usr/bin/vim` on the host.
- Subsequent invocations of `vim` by other users resulted in modified behavior, indicating successful tampering.
- IMA, which appraises signed binaries, denied execution of the tampered binary and logged the violation.

Mitigation (SELinux Enabled). After re-enabling SELinux:

- Attempts to tamper with host binaries from the container failed with permission errors.
- SELinux enforced mandatory access controls, preventing the container (labeled as `container_t`) from writing to host-mounted files labeled with different SELinux contexts.
- IMA continued to monitor file integrity, while SELinux blocked unauthorized write operations before tampering could succeed.

This experiment highlights the importance of defense-in-depth. IMA detects tampering post-modification, while SELinux prevents such modifications from occurring in the first place. Together, they provide layered protection against container breakout and host compromise. Attackers exploiting CVEs in the container stack often rely on filesystem access, SELinux policies effectively mitigate this attack vector by isolating container processes from host resources.

5.3 Performance

We evaluated the performance impact of IMA and SELinux policies during software builds. Both mechanisms introduce some overhead by performing runtime checks on file access and process execution. To quantify this, we measured build times using the `time` command across different policy configurations. Each build was run five times, and the average was computed. Remote attestation was excluded from this evaluation to isolate the effects of IMA and SELinux. While attestations may introduce some latency, it is largely network-dependent and infrequent enough to be negligible during builds.

We observed no consistent or significant increase in build times when enabling IMA and SELinux. IMA had a negligible overhead due to its role in validating static files at access time. Since most build operations occur inside containers and static files are read early, its runtime impact was minimal. SELinux, though known to impose 10–20% overhead [4], showed limited performance impact in our case, likely due to the I/O-light nature of the build. These results suggest that for containerized build environments with minimal file access, both IMA and SELinux can be enabled with negligible impact on performance. Further evaluation on more I/O-intensive workloads could better characterize their overhead in general-purpose scenarios.

6 Related Work

A number of systems have been proposed for remote attestation and system integrity verification, each targeting different aspects of system security. However, none offer an end-to-end attestation solution that spans the entire build stack as we propose in this paper.

Tapcon [24] enables attestation for trusted execution environments (TEEs) but falls short when the TEE itself is compromised. By relying on the execution environment without validating its supporting layers, Tapcon introduces security risks. Our system avoids this pitfall by employing a layered defense strategy that includes hardware-backed attestation, Linux Integrity Measurement Architecture (IMA) for binary integrity enforcement, and SELinux for mandatory access control, ensuring that no software layer is implicitly trusted.

Keylime [16] offers remote attestation using Trusted Platform Modules (TPMs) to verify OS configurations. While it enhances OS-level security, it does not cover higher-level components such as user-space binaries or container isolation. However, our approach extends attestation to all relevant components of the build system.

Gianluca et al. [17] introduce a notion of "trustworthiness" and define a hierarchy of attestation levels, from Level 0 (no attestation) to Level 4 (fully measured boot). While our work adopts a similar multi-level view, we specifically focus on securing an untrusted build process, introducing levels that are designed to meet the challenges of integrity enforcement in containerized, user-driven environments.

Other notable works [2, 11–13, 22] address adjacent domains but differ fundamentally in scope and assumptions. VRASED [11], for instance, focuses on hardware-based remote attestation for embedded devices by verifying register states. This hardware-centric and low-level approach is tailored for constrained environments, whereas our work targets OS-level and application-level integrity in general-purpose systems. On the other hand, Semantic Remote Attestation [12] uses language virtual machines to validate runtime properties of code. It requires trusting all software within the VM and relies heavily on high-level semantic information, unlike our design, which maintains a strict separation between trusted and untrusted code and does not assume source-level visibility. SCARR [22] tackles control-flow attacks and attests source code integrity using static analysis techniques. Although useful for certain threat models, it does not address dynamic runtime verification or enforcement mechanisms such as SELinux and IMA, which form the backbone of our trusted computing environment.

7 Conclusion

In this paper, we presented a hardware-rooted attestation framework designed to secure remote build environments against tampering and supply chain attacks. By combining confidential computing, remote attestation, and runtime security enforcement through IMA and SELinux, our approach ensures that software artifacts are built within a verifiably secure and policy-enforced environment. Our implementation on AMD SEV-enabled virtual machines demonstrates that these strong security guarantees can be achieved with minimal performance overhead. Ultimately, our work shows that end-to-end trust in software builds is achievable when provenance is anchored to a cryptographically verifiable and tightly controlled build system. This lays a strong foundation for securing modern build pipelines against increasingly sophisticated threats.

References

[1] AMD Secure Encrypted Virtualization (SEV). [n. d.]. https://www.amd.com/en/developer/sev.html. last accessed on August 30, 2025.

[2] Andrew Baumann, Marcus Peinado, and Galen Hunt. 2015. Shielding Applications from an Untrusted Cloud with Haven. *ACM Transactions on Computer Systems* 33, 3 (Sept. 2015), 1–26. doi:10.1145/2799647

[3] Henk Birkholz, Dave Thaler, Michael Richardson, Ned Smith, and Wei Pan. 2023. Remote ATtestation procedureS (RATS) Architecture. https://www.rfc-editor.org/info/rfc9334. doi:10.17487/RFC9334 last accessed on August 30, 2025.

[4] Brennon Brimhall, Justin Garrard, Christopher De La Garza, and Joel Coffman. 2023. A Comparative Analysis of Linux Mandatory Access Control Policy Enforcement Mechanisms. In *Proceedings of the 16th European Workshop on System Security* (Rome, Italy) *(EUROSEC '23)*. Association for Computing Machinery, New York, NY, USA, 1–7. doi:10.1145/3578357.3589454

[5] Alex Crawford, Eugene Yakubovich, and Rob Szumski. 2023. Enforcing SBOMs through the Linux kernel with eBPF and IMA. In *Proceedings of the 2023 Workshop on Software Supply Chain Offensive Research and Ecosystem Defenses* (Copenhagen, Denmark) *(SCORED '23)*. Association for Computing Machinery, New York, NY, USA, 77–78. doi:10.1145/3605770.3625206

[6] Enforcing Runtime Integrity Checks in OL9 with IMA. 2025. https://blogs.oracle.com/linux/post/runtime-integrity-checks-with-ima. last accessed on August 30, 2025.

[7] Hardware Secure Boot, OCP Security workgroup. 2025. https://www.opencompute.org/documents/secure-boot-2-pdf. last accessed on August 30, 2025.

[8] Behnaz Hassanshahi, Trong Nhan Mai, Alistair Michael, Benjamin Selwyn-Smith, Sophie Bates, and Padmanabhan Krishnan. 2023. Macaron: A Logic-based Framework for Software Supply Chain Security Assurance. In *Proceedings of the 2023 Workshop on Software Supply Chain Offensive Research and Ecosystem Defenses* (Copenhagen, Denmark) *(SCORED '23)*. Association for Computing Machinery, New York, NY, USA, 29–37. doi:10.1145/3605770.3625213

[9] What is SELinux? 2025. https://www.redhat.com/en/topics/linux/what-is-selinux. last accessed on August 30, 2025.

[10] Laurence Lundblade, Giridhar Mandyam, and Jeremy O'Donoghue. 2021. *The Entity Attestation Token (EAT)*. Internet Draft draft-ietf-rats-eat-11. Internet Engineering Task Force. https://datatracker.ietf.org/doc/draft-ietf-rats-eat-11 Num Pages: 91.

[11] Ivan De Oliveira Nunes, Norrathep Rattanavipanon, and Gene Tsudik. [n. d.]. VRASED: A Verified Hardware/Software Co-Design for Remote Attestation. ([n. d.]).

[12] VM '04 — Technical Paper. 2025. https://www.usenix.org/legacy/publications/library/proceedings/vm04/tech/haldar/haldar_html/index.html. last accessed on August 30, 2025.

[13] Lukas Petzi, Ala Eddine Ben Yahya, and Alexandra Dmitrienko. [n. d.]. SCRAPS: Scalable Collective Remote Attestation for Pub-Sub IoT Networks with Untrusted Proxy Verifier. ([n. d.]).

[14] Reproducible Builds. 2025. https://reproducible-builds.org/. last accessed on August 30, 2025.

[15] Reproducible Central: Rebuild instructions for artifacts published to (Maven) Central Repository. 2025. https://github.com/jvm-repo-rebuild/reproducible-central. last accessed on August 30, 2025.

[16] Nabil Schear, Patrick T. Cable, Thomas M. Moyer, Bryan Richard, and Robert Rudd. 2016. Bootstrapping and maintaining trust in the cloud. In *Proceedings of the 32nd Annual Conference on Computer Security Applications (ACSAC '16)*. Association for Computing Machinery, New York, NY, USA, 65–77. doi:10.1145/2991079.2991104

[17] Gianluca Scopelliti, Christoph Baumann, and Jan Tobias Mühlberg. 2024. Understanding Trust Relationships in Cloud-Based Confidential Computing. In *2024 IEEE European Symposium on Security and Privacy Workshops*. 169–176. doi:10.1109/EuroSPW61312.2024.00023

[18] Ridwan Salihin Shariffdeen, Behnaz Hassanshahi, Martin Mirchev, Ali El Husseini, and Abhik Roychoudhury. 2025. Detecting Python Malware in the Software Supply Chain with Program Analysis. In *ICSE SE In Practice (SEIP)*.

[19] SolarWinds Cyber Attack. 2025. https://www.fortinet.com/resources/cyberglossary/solarwinds-cyber-attack. last accessed on August 30, 2025.

[20] Supply Chain Levels for Software Artifacts (SLSA). 2025. https://slsa.dev. last accessed on August 30, 2025.

[21] How to use the Linux kernel's Integrity Measurement Architecture. 2025. https://www.redhat.com/en/blog/how-use-linux-kernels-integrity-measurement-architecture. last accessed on August 30, 2025.

[22] Flavio Toffalini, Eleonora Losiouk, Andrea Biondo, Jianying Zhou, and Mauro Conti. [n. d.]. ScaRR: Scalable Runtime Remote Attestation for Complex Systems. ([n. d.]).

[23] Veraison Remote Attestation Service. 2025. https://github.com/veraison. last accessed on August 30, 2025.

[24] Yan Zhai, Qiang Cao, Jeffrey Chase, and Michael Swift. 2017. TapCon: practical third-party attestation for the cloud. In *Proceedings of the 9th USENIX Conference on Hot Topics in Cloud Computing* (Santa Clara, CA) *(HotCloud'17)*. USENIX Association, USA, 19.