CARVE: Container-Aware Runtime Verification Evidence for Confidential VM Attestation

Hajeong Jeon, Johannes Lohmöller, Luca Hatz, Klaus Wehrle
Communication and Distributed Systems, *RWTH Aachen University*, Aachen, Germany
{*lastname*}@comsys.rwth-aachen.de

***Abstract*—Confidential virtual machines (CVMs) protect cloud workloads from untrusted infrastructure. However, existing CVM attestation mechanisms primarily establish trust at launch time and do not provide container-scoped evidence of runtime software integrity. This gap in the trust chain makes it difficult for container owners to verify that containers continue to execute the expected software stack, especially when the cloud operator or underlying host infrastructure may be compromised. In this work, we propose CARVE, a container-specific runtime integrity attestation mechanism for AMD SEV-SNP CVMs, extending the trust established at boot time to runtime workloads within containers. Our approach leverages a cgroup-aware extension of the Linux Integrity Measurement Architecture (IMA) to continuously collect integrity measurements and generate container-scoped runtime evidence. We further extend the CVM's remote attestation mechanism to enable container-specific runtime attestation and integrity verification of running containers. Our prototype demonstrates the feasibility and scalability of CARVE: container-specific IMA measurement processing incurs an average of $224\,\mu s$ per measurement event, the architecture adds $1.01\,s$ to the CVM boot time compared to a plain CVM, and its indexed evidence collection limits median attestation latency to $610\,ms$ for a heavyweight workload under 32 concurrent containers.**

***Index Terms*—Confidential computing, integrity measurement architecture, remote attestation**

I. Introduction

In cloud computing, containerization has become a widely adopted technology [1], [2]. Containers are increasingly used for security-sensitive workloads in addition to traditional applications. For example, in healthcare, privacy-preserving analysis of distributed patient data has been implemented using containerized federated learning [3]. Machine learning workloads have also been containerized on trusted processors to protect sensitive computations [3], [4]. Such deployments are often multi-tenant, colocating mutually independent workloads on shared cloud infrastructure, which introduces additional security concerns. However, container security in such shared environments remains a challenge and is often treated as "out of scope" [2]. This leaves the trustworthiness of the container execution uncertain [2], [5], making it difficult for a workload owner to determine whether a deployed container continues to execute its expected software stack. Such uncertainty weakens assurance in the workload integrity and may expose sensitive data by allowing unauthorized code execution [1], [5].

To address these concerns, confidential virtual machines (CVMs) built on technologies such as AMD SEV-SNP [6] and Intel TDX [7] provide hardware-backed isolation against an untrusted host operating system and hypervisor, even for data in use [8]. Corresponding CVM instances are readily available at all major cloud providers [9]. Beyond isolation, CVMs provide attestation mechanisms that allow a remote verifier to validate hardware-backed claims about the initial CVM state.

Nevertheless, the trust established by CVM attestation is confined to launch time and to the granularity of the entire CVM [6], a limitation that operators may overlook, as organizations tend to overestimate the protections that deployed privacy technologies provide [10]. This limitation is particularly relevant to multi-tenant CVMs, where several mutually independent containers share a single attested guest VM runtime. In detail, a container that executes an unauthorized file not only violates its expected software stack but may also compromise the shared guest environment and, thereby, the integrity of colocated containers.

While container security has seen further developments to address this limitation, existing approaches [11]–[14] cover only parts of the problem. For instance, Parma [11] and Confidential Containers (CoCo) [12] provide attested container launch and secure execution mechanisms, but they do not provide incremental, container-scoped evidence of files used after launch. Thus, they do not provide the remote verifier with fresh (i.e., real-time) container-scoped measurements of the software used during execution. Orthogonally, in Linux, runtime integrity monitoring can be achieved using the Integrity Measurement Architecture (IMA) [15], in which the kernel continuously measures and records file events according to predefined policies. However, IMA operates at the system-wide scope and does not natively distinguish measurements from different containers [13], [14].

We thus present CARVE, an architecture for *end-to-end container integrity verification* that establishes a verifiable chain of trust from CVM boot through container launch and runtime execution. CARVE continuously records file events and periodically binds newly collected container-specific measurements to a fresh CVM attestation report. Thereby, it enables a remote verifier to validate fresh, container-specific runtime evidence in multi-tenant CVMs. We also support efficient evidence collection even with concurrent workloads.

In summary, we make the following contributions:

- We design an architecture that extends CVM launch-time attestation to individual container workloads by combining measured boot, container-aware IMA measurements, and hardware-backed binding of runtime evidence.

- We introduce a cgroup-aware IMA measurement template that records container-specific file events, enabling fine-grained integrity monitoring of container workloads.
- We design an in-memory, per-container indexing mechanism that avoids repeated scanning of the append-only IMA measurement log, supporting efficient retrieval of newly generated measurements.

CARVE provides container-specific runtime integrity attestation without relying on a vTPM, while adding an average of $224\,\mu s$ per IMA measurement event and a median attestation latency of $610\,ms$ with 32 concurrent containers. Overall, CARVE connects the CVM-level and container-level integrity, extending the hardware-rooted trust of CVM attestation from a single launch-time snapshot to fresh, container-specific runtime evidence.

II. Background and Related Work

A. AMD SEV-SNP

AMD Secure Encrypted Virtualization-Secure Nested Paging (SEV-SNP) [6] is a VM-based TEE [8] that employs hardware-based memory encryption and integrity protection. In detail, the guest SEV-SNP VMs are isolated from the host software stack, preventing untrusted components from reading or modifying guest VM memory in plaintext. When a SEV-SNP-enabled VM is successfully launched, its memory is encrypted with a VM-specific key managed by the AMD Secure Processor (SP). To verify the integrity of guest VMs, SEV-SNP provides a remote attestation mechanism, in which the guest VM can generate an attestation report that includes a measurement of the VM's initial firmware state [6]. Our work extends this attestation mechanism by employing measured direct boot [16], allowing the remote verifier to validate the integrity of the guest kernel, initramfs, and kernel command-line parameters in addition to the firmware state. In this paper, we refer to *confidential VMs (CVMs)* as SEV-SNP-enabled guest VMs that are launched with measured direct boot and pass the initial remote attestation verification.

Nevertheless, the trust chain of SEV-SNP is limited to the guest VM at launch time, so that the integrity of containers running within the guest VM can still be compromised after launch without affecting the attestation report. Hence, the attestation report must include runtime integrity evidence to demonstrate that container workloads execute the expected software stack.

B. Linux IMA

Linux Integrity Measurement Architecture (IMA) [15] is a kernel subsystem that measures and appraises (only when configured) the file integrity during process execution. IMA provides a flexible policy and template mechanism to define which file events should be measured (policy) and how the measurements should be recorded (template). Depending on the defined policy and template, IMA records the cryptographic hash (i.e., file digest) and the path of the file events, such as file reads, executions, and kernel module loads. Each measurement is stored in the kernel-maintained list, which is append-only and can only be reset by a system reboot. Once recorded, each measurement cannot be modified or deleted. In our threat model (Section III-B), we include the measured guest kernel and its IMA subsystem in the CVM trusted computing base (TCB), which is then protected by the SEV-SNP hardware-based memory encryption (cf. Section II-A).

However, the original IMA implementation does not provide distinct measurements for different containers running in the same system, because the measurements are recorded at a system-wide scope. In particular, standard IMA entries do not contain container identity. Therefore, measurements with the same digest and path cannot be attributed to the container that triggered the measurement.

C. Related Work

Previous work [11]–[14], [16]–[22] has explored various approaches for confidential computing and container security. We observe three main research directions in the literature: (i) CVM-level attestation, (ii) confidential container attestation, and (iii) IMA-based runtime monitoring and attestation.

CVM-level attestation. Several works have proposed mechanisms that extend the attestation capabilities of SEV-SNP to measured boot, secret provisioning, and post-boot state [9], [16], [17], [19]. For instance, SNPGuard [16] extends the basic SEV-SNP attestation to include measured boot and secret provisioning, focusing on the VM boot chain integrity. Galanou et al. [17] capture post-boot in-guest state via eBPF and Linux Security Module hooks under verifier-specific attestation policies. SVSM-vTPM [18] strengthens SEV-SNP CVMs with an ephemeral vTPM, enabling existing TPM-based attestation and sealing mechanisms. The attestation target of these works remains the VM as a whole; they do not address how to attribute runtime integrity evidence to individual container workloads after the CVM has booted.

Confidential container attestation. Research in confidential computing and CVMs has also been extended to containerized workloads. In VM-based TEEs (i.e., CVMs), Parma [11] executes unmodified container groups inside a CVM using attested execution policies, providing container attestation and execution integrity against an untrusted host and hypervisor. Similarly, CoCo [12] focuses on confidential computing in a cloud-native environment by combining container abstractions with hardware-backed VM isolation. However, these approaches do not provide container-scoped evidence of file events that can be used to verify the runtime integrity of the containers.

IMA-based runtime monitoring and attestation. For runtime integrity monitoring, Linux IMA is widely used to measure security-sensitive file operations [13], [14], [21], [22]. Previously, IMA has been extended to container-specific measurements by introducing container-specific PCRs [13] or namespacing IMA policies and measurement states [14]. Ferro et al. [21] further build on IMA namespaces to provide privacy-preserving container attestation in multi-tenant environments. More recent work has proposed to combine IMA with eBPF to block unauthorized binaries at runtime [22]. Such approaches, however, overlook the gap between container-specific runtime

TABLE I
COMPARISON OF RELATED WORK WITH CARVE. CARVE ENSURES CONTAINER-SPECIFIC RUNTIME INTEGRITY THAT IS BOUND TO THE ATTESTED CVM, WITHOUT RELYING ON A TPM OR VTPM.

System	CVM attestation	Container-specific evidence	CVM-bound evidence	TPM/vTPM Dependency
SNPGuard [16]	✓	×	×	×
SVSM-vTPM [18]	✓	×	×	✓
Parma [11]	✓	△	×	×
CoCo [12]	✓	△	×	△
Container-IMA [13]	×	✓	×	✓
Security Namespace [14]	×	✓	×	×
Ferro et al. [21]	×	✓	×	✓
RICe [22]	×	△	×	×
CARVE	✓	✓	✓	×

×: not supported/not dependent, ✓: fully supported/fully dependent, △: partially supported/partially dependent

integrity and CVM integrity attestation, which is necessary to establish a complete trust chain from the root of trust (RoT) to the container workloads.

TPM/vTPM dependency. Several approaches [12], [13], [18], [21] rely on TPM or vTPM. Container-IMA [13] and Ferro et al. [21] anchor container-specific measurements to a physical TPM, and Security Namespace [14] avoids this dependency only for measurement and appraisal, as its remote attestation still assumes a TPM. Use of a TPM or vTPM expands the attack surface and introduces additional complexity, as the TPM or vTPM must be trusted and properly configured.

Table I compares the related work with CARVE in terms of CVM attestation, container-specific evidence, CVM-bound evidence, and TPM/vTPM dependency. CARVE thus serves as a complementary approach to prior work, combining the benefits of CVM attestation and container-specific runtime integrity evidence, while eliminating the need for a vTPM.

III. DESIGN GOALS AND THREAT MODEL

A. Design Goals

The runtime integrity of containerized workloads is not reliably covered by the original CVM trust boundaries (cf. Section II-A). Importantly, a remote verifier cannot directly determine whether a container executes its expected software stack, even if the CVM itself has been attested to be in a trusted state at launch time. We thus formalize our design goals as follows:

- **G1 Trusted container-specific runtime integrity evidence.** To ensure that a container has used only expected files covered by the IMA policy, the system must provide reliable evidence attributable to the corresponding container. In particular, the evidence must be scoped to the target container only, so that measurements from different containers within the same CVM are distinguishable and attributable to their respective containers.
- **G2 Extension of the CVM trust chain and attestation binding.** The container-specific evidence must be bound to the attested CVM to include it in the original CVM trust chain. Cryptographically binding container-scoped measurements to the attestation report enables the remote verifier to establish trust in the containers.
- **G3 No stale evidence.** At the time of attestation, the presented container evidence must be fresh and reflect the file events that have occurred since the last successful attestation. Thus, any stale evidence should be rejected and considered invalid to ensure the integrity of the attestation procedure.
- **G4 Verifiable attestation and enforcement path.** Components involved in the attestation flow (i.e., evidence collection, binding, verification) must be included in the measured CVM software stack or reside at the trusted remote verifier.

To fulfill G1–G4, we propose a design that extends trust from the hardware RoT to the container runtime. In the following, we describe which components this design may rely upon and which capabilities an attacker retains.

B. Trust Assumptions and Threat Model

Based on our design goals (G1–G4), we define the trusted components and detailed attacker capabilities in the system. On the host platform, we assume only the CPU hardware and the AMD SP to be inherently trusted. The guest VM image, which includes the guest kernel, initramfs, root filesystem, and our system components, is not trusted by default before deployment. Instead, measured direct boot and early-boot integrity verification establish a guest TCB comprising the guest kernel (including the IMA subsystem), the initramfs-based verification logic, and the attestation daemon. We assume that these components remain uncompromised during the lifetime of the CVM. Additionally, the remote verifier is considered trusted as an external entity that is not part of the CVM or the host platform; attacks on it are out of scope. The remote verifier maintains an expected container manifest that binds each approved image digest to the file digests expected in the IMA measurement list. We assume that these manifests are correct and up to date as the remote verifier is trusted; securing the container-image supply chain that produces them is orthogonal to CARVE.

Other components running on the host platform, including the host OS software stack and hypervisor, are considered untrusted and may be compromised by malicious actors. We consider two types of attackers: (i) a *malicious cloud operator* with administrative control over the host OS, hypervisor, and cloud management stack, but without root privileges inside the CVM, and (ii) a *container-level adversary* with root privileges within its own container, but without guest-root or kernel-level privileges that could compromise the CVM TCB. A *malicious cloud operator* may attempt to inspect or modify the VM image, inject malicious inputs, or replay stale states. CARVE relies on SEV-SNP to protect the memory of the running CVM from the untrusted host and on measured direct boot to detect modifications to the initial guest software stack. Then, a *container-level adversary* may execute unexpected binaries, load unauthorized libraries, or modify the container software stack. Such changes are not reflected in the launch-

time CVM attestation and may cause the container to deviate from its expected software stack while the guest TCB remains unchanged.

In our threat model, we consider physical attacks on the host platform or side-channel attacks on the CVM as out of scope. We also exclude denial-of-service (DoS) attacks against the CVM or containers, in-memory attacks, and misuse of authorized binaries that do not generate IMA measurements. Further, attacks that compromise the measured guest TCB or obtain guest-root or kernel-level privileges, including post-boot tampering with IMA or the attestation daemon, are excluded. These attacks require additional protections for the guest TCB itself. They are orthogonal to our design goals, which focus on attributing runtime integrity evidence to individual containers and binding that evidence to the attested CVM. In summary, CARVE must uphold G1–G4 against a compromised host software stack and tampered container workloads, while relying only on the AMD SP, the measured guest software stack, and the remote verifier.

IV. System Architecture

A. CARVE Design

Based on our threat model (cf. Section III-B), we design a flow that records, binds, and verifies container-specific file-integrity evidence throughout the container lifecycle (Fig. 1). Our system workflow includes three parts: (i) boot-time trust establishment of the guest kernel, customized initramfs, and attestation daemon, (ii) runtime integrity monitoring of the container workloads using cgroup-aware IMA measurements, and (iii) container-specific runtime attestation by binding the container evidence to the CVM's attestation report. We further describe each part of the workflow in detail.

Boot-time trust establishment. The CVM is launched with a measured kernel containing the customized IMA template, a customized initramfs, and a measured kernel command line containing the expected dm-verity root hash. As the attestation daemon is responsible for collecting and indexing the container-specific runtime evidence, it is thus included in the measured software stack of the CVM. During early boot, the attestation daemon's measurement is verified against the expected value provisioned at launch time (i.e., local integrity verification). The CVM can be launched only if the local verification succeeds; otherwise, the boot process is aborted. When the CVM is successfully launched, the attestation daemon performs an initial remote attestation verifying the measurements of the firmware, guest kernel, initramfs, and command line. If the initial remote attestation fails, the daemon enters a fail-closed state and triggers guest shutdown before any container starts. Overall, our architecture ensures a fail-closed behavior, preventing the CVM from running in an untrusted state (G4).

Runtime integrity monitoring with cgroup-aware IMA measurements. Standard IMA provides integrity measurements in a system-wide scope (cf. Section II-B), which prevents the attribution of measurements to individual containers. Our architecture thus employs a customized IMA template to record cgroup information during runtime, enabling fine-grained integrity monitoring of container workloads (①). Additionally, the attestation daemon maintains an index of the IMA measurements to enable efficient collection and retrieval of container-specific evidence, as the IMA measurement log can grow significantly during CVM runtime (cf. Section II-B). Fig. 1 illustrates the indexing mechanism (②), and we discuss its implementation in Section V-C1.

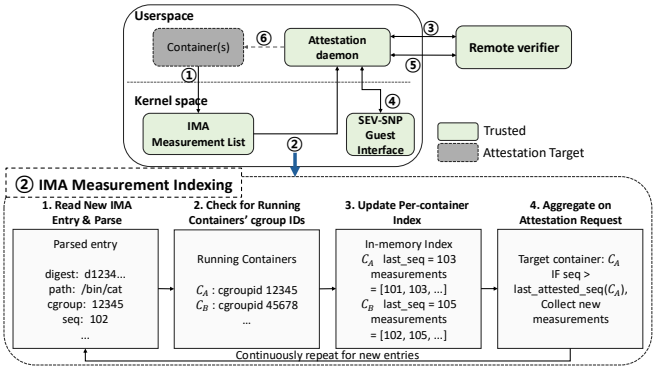


Fig. 1. CARVE overview with the attestation and verification flow. Indexing of the IMA measurements is performed by the attestation daemon, which collects and generates container-specific runtime integrity evidence.

Container-specific runtime attestation and verification. During CVM runtime, the attestation daemon continually monitors running containers and collects the IMA measurements and container information. Then, the attestation daemon requests a nonce from the remote verifier to ensure freshness (G3) of the evidence (③). Once the nonce is received, the attestation daemon requests an attestation report that binds the container's IMA measurements, container information (G1), and the nonce from the remote verifier (④). The AMD SP signs the attestation report containing the evidence digest, thereby binding the container evidence to the attested CVM (G2). The attestation report is then sent to the remote verifier for verification (⑤). If the verification is successful, the container is considered to be in a trusted state, and the attestation daemon allows it to continue running (⑥). Otherwise, the attestation daemon stops the container to prevent further execution in an untrusted state (⑥).

Together, these three stages extend the launch-time trust of the CVM into container-specific runtime integrity attestation.

B. Security Analysis

We now analyze how CARVE withstands the attacks admitted by our threat model (cf. Section III-B) with respect to our design goals (G1–G4). Table II summarizes the potential attacks and their corresponding mitigation strategies.

Unauthorized Runtime Execution. A container-level adversary (cf. Section III-B) may attempt to execute unauthorized files that are not part of the expected software stack of the container. CARVE records the file digest and container identity for each measured file event (cgroup-aware IMA measurement), allowing the remote verifier to compare the measurements

TABLE II
POTENTIAL ATTACKS AND CARVE'S MITIGATION STRATEGIES. EACH ATTACK IS MAPPED TO THE DESIGN GOAL (G1–G4) IT VIOLATES, WHICH THE CORRESPONDING MITIGATION FULFILLS.

Design Goal	Possible Attack	CARVE Mitigation
G1	Unauthorized Runtime Execution	Cgroup-aware IMA measurements
G2	Evidence Forgery or Substitution	Evidence binding to attestation reports
G3	Replay Attack	Nonce inclusion in attestation reports
G4	Boot-chain & Attestation-path Tampering	Fail-closed enforcement

against the expected file digests in the approved manifest for the target container.

Evidence Forgery or Substitution. Forged or substituted evidence can be presented to the remote verifier to mislead the attestation process, where an attacker may try to tamper with or replace the container-specific evidence to make the container appear to be in a trusted state. In our design, a trusted component (i.e., attestation daemon) hashes the container-specific evidence and binds the resulting digest to the attestation report, which is then signed by the AMD SP. Then, the remote verifier validates the report signature and recomputes the digest from the accompanying evidence, making any modification or substitution detectable and causing verification to fail.

Replay Attack. Another potential attack is the replay of stale evidence, where an attacker may present old IMA measurements and container information to bypass the attestation process. To ensure that the current container state is accurately reflected in the attestation report, CARVE includes a nonce provided by the remote verifier in the attestation report. If the nonce does not match the expected value, the attestation report is considered invalid, resulting in the container being stopped.

Boot-chain and Attestation-path Tampering. The malicious cloud operator may attempt to replace the guest kernel, IMA implementation, or attestation daemon during boot time. CARVE includes these components in the verified boot chain and aborts the boot or initial attestation procedure (fail-closed) if their expected measurements cannot be validated.

C. Use Cases

While existing CVM attestation mechanisms can establish trust at launch time, they do not comprehensively address the runtime integrity of container workloads (cf. Section II-C). In contrast, CARVE applies to any containerized workload that requires runtime integrity verification, such as machine learning inference, privacy-preserving federated learning, and other security-sensitive applications. In such scenarios, CARVE extends the trust established at CVM boot to individual containers, even in the presence of a malicious cloud operator or container-level adversary (cf. Section III-B), by binding container-specific runtime integrity evidence to the attestation mechanism. Hence, workload owners obtain dependable claims that their containers execute the expected software stack without trusting the cloud provider or colocated tenants, replacing organizational trust with technical guarantees [23].

V. IMPLEMENTATION

We implemented a prototype of CARVE on an AMD SEV-SNP CVM that fulfills G1–G4 and follows the system workflow in Section IV-A. CARVE leverages measured direct boot, cgroup-aware IMA measurements, and the attestation mechanism of SEV-SNP to provide end-to-end security for container workloads.

A. Local Verification

Local verification is performed by an initramfs script during early boot, before rootfs mounting. During early boot, the initramfs configures dm-verity [24] using the expected root hash passed through the measured kernel command line and mounts the rootfs only if dm-verity verification succeeds. As the expected root hash is included in the measured kernel command line, the initial remote attestation binds the attestation daemon to the measured CVM configuration. The daemon is executed only after the dm-verity verification succeeds and is therefore included in the guest TCB. If the verification fails, the boot process is aborted, preventing an untrusted VM from running.

Our prototype utilizes IMA's built-in `tcb` policy, which measures loaded kernel modules, executed files, memory-mapped files (e.g., shared libraries), and files accessed by root [13]. Custom IMA policies may instead define which file operations are measured. CARVE can bind the selected policy to the measured boot configuration by including its expected digest in the kernel command line and verifying it during early boot.

B. Customized IMA Template

To enable container-specific runtime integrity measurements, we modify the guest VM kernel by adding a new template field (`cgroup`) that retrieves the cgroup ID of the process triggering the measurement event. For further indexing and retrieval of the container-specific measurements, we also add a new field (`seq`), which is a monotonically increasing per-cgroup sequence number assigned to each entry. Our customized template is denoted by `d-ng|n-ng|cgroup|seq`, preserving the standard IMA template fields (`d-ng` and `n-ng`) for the file digest and path, respectively.

When IMA processes a measurement event, the field handler obtains the cgroup context from the triggering task and serializes its ID into the measurement entry. In our Docker-based prototype, processes belonging to the same container are mapped to the same container cgroup, allowing their entries to be associated with that container instance. The identifier therefore allows the attestation daemon to distinguish measurements generated by multiple containers, even when the containers execute the same binary or access files with identical paths. Our customization does not alter the existing file digest computation or the measurement list structure but extends the measurement entries with container-specific metadata, maintaining compatibility with the original IMA implementation. Based on the additional cgroup and sequence metadata, container-specific indices are maintained, enabling

efficient retrieval of the container-specific evidence during runtime attestation (Section V-C1).

C. Attestation Daemon

The attestation daemon is implemented as a system service that is automatically started after the CVM is launched. The attestation daemon is responsible for collecting the container IMA measurements (Section V-C1) and generating the container-specific evidence for attestation (Section V-C2).

1) IMA Measurement Collection: The original IMA implementation does not provide a query mechanism to retrieve entries associated with specific identifiers, so the attestation daemon would have to read the entire list and filter each entry whenever it collects evidence for a container. This can cause significant delays in the attestation process, where the collection time grows linearly with the number N of measurements as the CVM continues to run ($O(N)$).

We thus add an indexing mechanism to the IMA measurement collection process, where the attestation daemon maintains an in-memory index over the IMA measurement list. Upon startup, the attestation daemon runs a background thread that continuously reads the IMA measurement list, parses the cgroup identifier and sequence number from each entry, and updates the index accordingly. For each runtime attestation request, the attestation daemon uses the target container's cgroup identifier to locate the corresponding entries and performs a binary search over their sequence numbers to retrieve only the measurements generated after the last successful attestation. Therefore, with indexing, the measurement collection cost is reduced from $O(N)$ to $O(\log N_c + k)$, where N_c is the number of indexed entries associated with a target cgroup c, and k is the number of new entries returned for the current attestation.

2) Container-specific Evidence Generation: When evidence generation is triggered, the attestation daemon first requests a $32\,\mathrm{B}$ nonce from the remote verifier to ensure freshness of the attestation. The attestation daemon computes `REPORT_DATA` field by hashing the concatenation of the evidence digest and the nonce from the remote verifier. In detail, the evidence digest is computed as:

$$\texttt{REPORT_DATA} = H\big(H(\texttt{IMA}_c) \parallel H(\texttt{Info}_c)\big) \parallel N.$$

Here, $\texttt{IMA}_c$ is the ordered set of newly collected IMA measurements and $\texttt{Info}_c$ represents the target container information (container identity, image digest, cgroup ID, and attested sequence range). The evidence is serialized using a deterministic JSON field order before hashing. After generating the attestation report, the attestation daemon sends it and the container-specific evidence to the remote verifier. Then, the remote verifier validates the report signature and launch state, recomputes the evidence digest, and compares the IMA measurements against the expected manifest for the target container. If the verification fails, the remote verifier returns a failure response to the attestation daemon, which then stops the target container. Otherwise, the attestation daemon updates the last attested sequence number for the target container and continues monitoring new measurements.

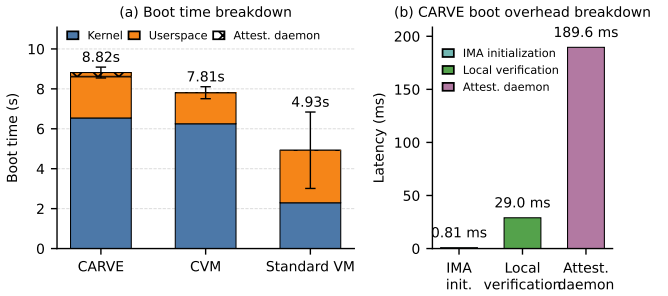


Fig. 2. Boot time of the CVM with CARVE architecture compared to two baseline configurations (cf. Section VI-A). Our custom components and local verification do not introduce significant overhead to the CVM boot time, with an average increase of $1.01\,\mathrm{s}$ compared to the plain CVM.

Altogether, local verification, the customized IMA template, and the attestation daemon realize the attestation flow (cf. Section IV-A) within the measured software stack of the CVM, fulfilling our design goals (G1–G4).

VI. Evaluation

Our prototype is evaluated on a host machine with two AMD EPYC 9135 processors. The CVM is launched with QEMU v.10.0.0 with SEV-SNP support, OVMF UEFI v.2.7, 48 vCPUs, $128\,\mathrm{GB}$ of RAM, and Linux kernel v.6.16 with SNP features and a customized IMA template. For evaluation, we use three Docker container workloads (C_A, C_B, and C_C) to represent a multi-tenant cloud environment and our use cases (cf. Section IV-C). C_A is a lightweight workload that performs simple feature-based prediction using a minimal Python runtime and a small application-specific model. The image size of C_A is $150\,\mathrm{MB}$, with 49 measured files. C_B is a moderate ML workload, including additional data processing and model execution logic, resulting in a larger runtime footprint than C_A, with an image size of $557\,\mathrm{MB}$ and 548 measured files. Lastly, C_C is a heavyweight model-serving container that relies on a more complex software stack, including an `nginx`-based inference server and ML-related dependencies, with an image size of $1.2\,\mathrm{GB}$ and 650 measured files.

A. Bootstrapping

We first evaluate the boot-time latency of the CVM, including the loading of the customized IMA template and the initialization of the attestation daemon during early boot. Our measurements are collected from 50 CVM boot iterations, and we report the average boot time for each process. We compare the boot time of CARVE with two baseline configurations: (i) a standard VM without SEV-SNP support or CVM attestation, and (ii) a plain CVM with SEV-SNP support but without the CARVE architecture. Both baselines are launched without IMA support and thus do not include IMA-related initialization overhead. All other parameters, including the host hardware, VM boot parameters, VM image, and kernel version, remain the same as the CARVE implementation (cf. Section V).

Fig. 2 shows the breakdown of the VM launch time for CARVE and the two baseline configurations. Compared to the standard VM with no security features, the plain CVM

introduces additional overhead of approximately 2.88 s. This addition is attributed to the SEV-SNP initialization, which has also been reported in prior work [25]. CARVE adds only 1.01 s to the total boot time compared to the plain CVM, with increases of 0.29 s in the kernel and 0.72 s in userspace initialization. As CARVE requires a customized kernel for cgroup-aware IMA measurements and an attestation daemon for runtime attestation, the additional overhead is due to the initialization of these components during early boot. Among CARVE's components, the startup of the attestation daemon takes the longest, averaging 189.6 ms during userspace initialization. Hence, the boot-time overhead of CARVE constitutes a one-time cost of 1.01 s per CVM launch, which does not affect the subsequent container attestation latency.

B. Container-Specific IMA Measurement Processing

Our customized IMA template additionally records the cgroup information and sequence number of the measured files (cf. Section V-B). When compared to the standard IMA template, this customization introduces additional overhead for every measurement event covered by the defined IMA policy. We thus evaluate the additional kernel-side measurement-processing cost for file execution, root file access, and kernel-module loading events. Specifically, we compare the overhead of the cgroup-aware IMA template (cf. Section V-B) against the default `ima-ng` template as a baseline. Other configurations (e.g., kernel version, IMA policy, and VM image) remain the same, and we report results from 50 iterations of each operation.

We trace the IMA initialization functions to explicitly measure the processing cost incurred by IMA. For both configurations, we trace `ima_alloc_init_template`, which is called when a new measurement entry is created. Without the cgroup-aware IMA template, processing a newly generated measurement takes an average of 366 µs. Further, for the cgroup-aware IMA template, we also trace `ima_cgroupid_init` to isolate the cost of the customization. To simulate realistic workloads, we stress the IMA measurement processing by concurrently running C_A–C_C, with a total of 32 containers, and measure the IMA processing time for each process that triggers a measurement event. As a result, we observe that the cgroup-aware IMA template introduces an average overhead of 224 µs per measurement event, increasing the total IMA processing time from 366 µs to 590 µs. We note that this overhead does not represent a constant increase in the execution time of every file operation, as IMA may reuse cached measurements depending on its policy and file state.

C. Attestation Evidence Generation

During container runtime, the attestation daemon collects the container-specific IMA measurements and generates the attestation evidence to be sent to the remote verifier (cf. Section V-C2). We now evaluate the evidence generation latency for C_A–C_C, analyzing its process in detail, including the collection of IMA measurements and evidence packaging for attestation (Fig. 3) from 50 iterations of each experiment.

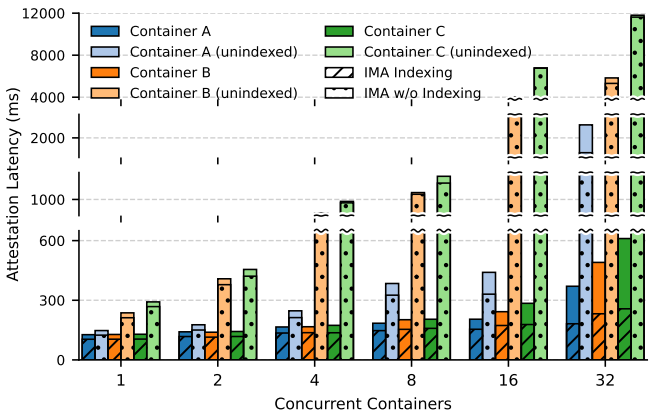


Fig. 3. Attestation latency for C_A–C_C. With sequential filtering, the attestation latency increases significantly with larger container workloads, as the IMA measurement list is append-only. CARVE solves this issue by indexing, resulting in a median attestation latency of 610 ms for C_C.

Although attestation evidence generation includes packaging the IMA measurements and generating the SNP report, these operations add only 183 ms of average latency even with 32 concurrently running containers (C_C), while most of the overall attestation latency is attributable to IMA measurement collection. Hence, we focus on the performance of the collection process and compare the sequential filtering of the measurement list against our indexing mechanism. As the indexing reduces the number of measurements to be checked, the collection process remains stable and does not increase significantly with larger container workloads. For instance, C_C's IMA measurement indexing only increases from 182 ms (single container) to 257 ms (32 concurrent containers). Meanwhile, we observe a significant increase without indexing, where the sequential filtering for C_C increases from 268 ms (single container) to 11.6 s (32 concurrent containers).

D. Time-to-Detection and Attestation Interval

To fulfill G1 and G4, CARVE needs to correctly detect a runtime integrity violation and enforce the verification result without disrupting other containers running in the same CVM. We therefore evaluate the end-to-end behavior of CARVE, from the execution of an unauthorized binary to its detection by the remote verifier and the subsequent removal of the corresponding container. In each run, we execute C_A–C_C concurrently and inject a runtime integrity violation into C_B by executing an unauthorized binary, while C_A and C_C continue to run normally. The results are collected from 100 experimental runs, and we report the median detection and removal latency. Detection latency is measured from unauthorized binary execution until receipt of the verifier's failure decision, while removal latency ends when the target container is stopped.

CARVE successfully detected the unauthorized execution and selectively removed C_B in all 100 runs, while C_A and C_C continued to pass runtime verification and remained operational. When evaluating the detection and removal latency, we configure the attestation interval to 0.5 s, 1 s, 2 s and 5 s to

analyze the impact of the attestation frequency on the end-to-end response time. Longer attestation intervals lead to longer detection and removal latencies. With an attestation interval of $0.5\,\mathrm{s}$, the detection latency is $0.487\,\mathrm{s}$, and the removal latency is $0.639\,\mathrm{s}$. With an interval of $5\,\mathrm{s}$, the detection latency increases to $2.256\,\mathrm{s}$ and the removal latency to $2.399\,\mathrm{s}$. As we inject the violation at random points within each attestation period, the median waiting time to the next verification remains lower than the configured interval.

In summary, our evaluation shows that CARVE confines its overhead to $1.01\,\mathrm{s}$ of additional boot time, $224\,\mu\mathrm{s}$ per IMA measurement event, and sub-second attestation latency, while reliably detecting and selectively removing compromised containers even under 32 concurrent workloads.

VII. Discussion

CARVE provides hardware-bound, container-scoped evidence of policy-covered file events in CVMs, extending launch-time attestation with runtime verification. We now discuss the limitations and trade-offs of our approach, as well as its compatibility with other use cases and technologies.

Limitations. Although CARVE provides file-mediated runtime integrity evidence, it does not attest to the control flow or mutable memory of a process after the measured code has been loaded. Consequently, attacks that do not involve unauthorized file executions, such as return-oriented programming (ROP) attacks, may not be detected by CARVE. Additionally, periodic attestation introduces a trade-off between detection latency (cf. Section VI-D) and attestation overhead (cf. Section VI-C). As our architecture is measurement-based, an unauthorized file may execute before the next attestation decision and container removal; CARVE therefore provides detection and response rather than pre-execution prevention. Thus, future work could combine CARVE with Secure Boot [26], signed kernel modules, and IMA appraisal [22] to strengthen the guest TCB and prevent unauthorized files from being executed, rather than detecting them only during subsequent attestation.

Compatibility. Our design is compatible with other container runtimes and orchestration platforms, as it builds on the cgroup mechanism that is widely supported across other container runtimes. The evidence-binding mechanism can also be extended to other CVM technologies such as Intel TDX [7] and ARM CCA [27], which provide a field in the attestation report for binding user-provided evidence.

VIII. Conclusion

CARVE extends SEV-SNP launch-time attestation with container-scoped evidence of runtime file events. Our prototype on AMD SEV-SNP CVMs demonstrates the feasibility and effectiveness of our approach, with boot time overhead of $1.01\,\mathrm{s}$ compared to a plain CVM, and a median attestation latency of $610\,\mathrm{ms}$ for a heavyweight container workload, even with 32 concurrently running containers (cf. Sections VI-A and VI-C). Regarding security, CARVE successfully detects unauthorized file executions and removes the compromised container without affecting others running in the same CVM (cf. Section VI-D). Our work can also be applied to other container runtimes and CVM technologies, providing a practical solution for end-to-end security in diverse deployment scenarios. In future work, CARVE can be further combined with preventive mechanisms such as Secure Boot and IMA appraisal to detect and block unauthorized executions.

Acknowledgments

This work was funded by the German Federal Ministry of Research, Technology and Space (BMFTR) under funding reference number 02J24A030.

References

[1] S. Sultan *et al.*, "Container Security: Issues, Challenges, and the Road Ahead," *IEEE Access*, vol. 7, pp. 52 976–52 996, 2019.
[2] M. S. Haq *et al.*, "SoK: A Comprehensive Analysis and Evaluation of Docker Container Attack and Defense Mechanisms," in *IEEE S&P*, 2024, pp. 4573–4590.
[3] A. Moncada-Torres *et al.*, "VANTAGE6: An open source priVAcy preserviNg federaTed leArninG infrastructurE for Secure Insight eXchange," *AMIA Annu. Symp. Proc.*, vol. 2020, pp. 870–877, 2021.
[4] N. Hynes *et al.*, "Efficient Deep Learning on Multi-Source Private Data," arXiv:1807.06689, 2018.
[5] F. Zaritto *et al.*, "Extending Kubernetes for Pods Integrity Verification," *J. Netw. Syst. Manag.*, vol. 34, p. 14, 2025.
[6] AMD, "AMD SEV-SNP: Strengthening VM Isolation with Integrity Protection and More," AMD, White Paper, 2020.
[7] Intel, "Intel Trust Domain Extensions," Intel, White Paper, 2025.
[8] M. Li *et al.*, "SoK: Understanding Design Choices and Pitfalls of Trusted Execution Environments," in *ASIACCS*. ACM, 2024, pp. 1600–1616.
[9] A. S. M. Asadujjaman *et al.*, "Securing Confidential VMs in Public Clouds," in *CloudCom*, 2024, pp. 55–62.
[10] J. Lohmöller *et al.*, "Between Promise and Practice: Challenges and Misperceptions of Applying Privacy Enhancing Technologies in Business Contexts," in *HICSS*, 2026, pp. 6294–6303.
[11] M. A. Johnson *et al.*, "Parma: Confidential Containers via Attested Execution Policies," arXiv:2302.03976, 2023.
[12] Confidential Containers Community, "Confidential Containers," https://confidentialcontainers.org, 2026.
[13] W. Luo *et al.*, "Container-IMA: A Privacy-Preserving Integrity Measurement Architecture for Containers," in *RAID*, 2019, pp. 487–500.
[14] Y. Sun *et al.*, "Security Namespace: Making Linux Security Frameworks Available to Containers," in *USENIX Security*, 2018, pp. 1423–1439.
[15] R. Sailer *et al.*, "Design and Implementation of a TCG-based Integrity Measurement Architecture," in *USENIX Security*, 2004, pp. 223–238.
[16] L. Wilke and G. Scopelliti, "SNPGuard: Remote Attestation of SEV-SNP VMs Using Open Source Tools," in *EuroS&PW*, 2024, pp. 193–198.
[17] A. Galanou *et al.*, "Full Trust Alchemist: Reforging Attestation for Cloud-based Confidential Workloads," in *ACM Middleware*, 2025.
[18] V. Narayanan *et al.*, "Remote Attestation of Confidential VMs Using Ephemeral vTPMs," in *ACSAC*. ACM, 2023, pp. 732–743.
[19] J. Mao *et al.*, "Towards Verifiable Trust Proof for Trusted Confidential Virtual Machines," *IEEE Trans. Netw. Sci. Eng.*, vol. 13, pp. 552–567, 2026.
[20] F. Brasser *et al.*, "Trusted Container Extensions for Container-based Confidential Computing," arXiv:2205.05747, 2022.
[21] L. Ferro *et al.*, "Privacy-Preserving Container Attestation," *J. Netw. Syst. Manag.*, vol. 34, p. 5, 2026.
[22] S. H. Lee, "Poster: RICe: Runtime Integrity for Containers Leveraging eBPF and IMA Appraisal," in *IEEE S&P*, 2026.
[23] H. Jeon *et al.*, "Replacing Organizational Trust with Technical Guarantees," *Industry 4.0 Science*, vol. 42, no. 4, pp. 72–77, 2026.
[24] The Linux Kernel Development Community, "dm-verity," Linux Kernel Documentation.
[25] M. Misono *et al.*, "Confidential VMs Explained: An Empirical Analysis of AMD SEV-SNP and Intel TDX," *POMACS*, vol. 8, no. 3, pp. 1–42, 2024.
[26] R. Haas and M. Pirker, "The State of Boot Integrity on Linux – A Brief Review," in *ARES*. ACM, 2024, pp. 1–6.
[27] Arm, "Arm Confidential Compute Architecture."