Securing Confidential VMs in Public Clouds

A S M Asadujjaman*, Davi Pontes*, Eduardo Falcão†, Andrey Brito*
*Federal University of Campina Grande, Campina Grande, Paraíba, Brazil
Email: {a.asadujjaman, davi.pontes}@lsd.ufcg.edu.br, andrey@computacao.ufcg.edu.br
†Federal University of Rio Grande do Norte, Natal, Rio Grande do Norte, Brazil
Email: eduardo@dca.ufrn.br

***Abstract*—Cloud providers and CPU vendors are working together to deliver the promise of Confidential Computing through the Confidential Virtual Machine (CVM) offerings. However, the steps required to verify the authenticity of CVMs are still unclear. Technical specifications are difficult to follow, and the implementations by the cloud providers are incomplete. Currently, there is no serviceable procedure to ensure desired security properties for CVM users. In this work, our goal is to secure confidential VMs by facilitating their authenticity verification and addressing security issues arising from their incomplete implementations. To that end, firstly, we identify a set of necessary attestation properties and formulate guidelines to verify them. Secondly, we show why the current offerings are insufficient to guarantee security and what else is needed. In that regard, as Intel has released its CVM technology recently, we focus our attention on the Intel TDX CVM offerings. After analyzing and evaluating Intel TDX services from major cloud providers, we have identified the possibility of a malware injection attack and designed a solution to detect it. Through experiments, we show that our solution does not add any significant overhead.**

I. Introduction

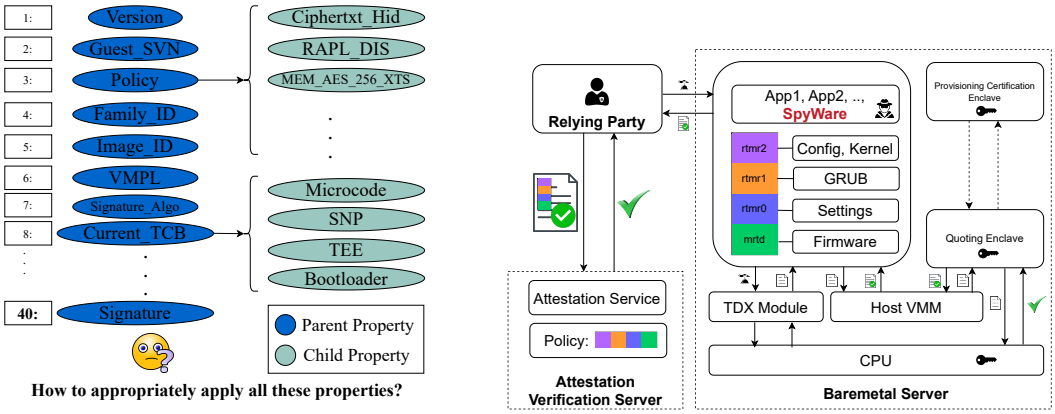
Confidential computing is a promising technology to enable businesses to continue harvesting the benefits of cloud computing while providing them with the ability to ensure the confidentiality and integrity of their data. Confidential computing offers hardware-based encryption and integrity protection mechanisms to protect data "in use". Thus, businesses now hope that they can be fully protected. The latest innovation in the confidential computing space is the Confidential Virtual Machine (CVM), where the main memory of the VM is protected through encryption and integrity preservation mechanisms. Users can use a process called *remote attestation* in which various *attestaion properties* are matched against their expected values to ensure their VM is securely running on an authentic hardware platform [9].

However, CVM is a complex new technology that is implemented differently by individual CPU vendors. To compound the complexity, various cloud providers have adopted the technology differently. Moreover, crucial features to ensure the integrity of the CVMs are still missing in the current cloud offerings. Without proper verification, the CVMs may not provide the expected protection. As reviewed in Section VI, no existing work addresses these problems. In the following, we illustrate these problems with two motivating examples.

Motivating Example 1. As shown in Figure 1a, AMD SEV-SNP specifies *forty attestation properties* as part of its attestation report. To further increase the number of fields to verify, many of these properties are a combination of multiple fields (e.g., the *Policy* property is a collection of twelve individual fields). Not only are there many fields to understand and verify but also, information about how to obtain an expected value to match these fields against is not easy to find.

Motivating Example 2. As shown in Figure 1b, Intel TDX is designed to allow a Relying Party to perform remote attestation of CVMs with the help of attestation quotes. The attestation quotes carry a structure known as *attestation report* that includes measurements of various stages of the boot process that Intel TDX stores at its registers: `MRTD`, `RTMR0`, `RTMR1`, `RTMR2`, and `RTMR3`. The CVM was initialized from an image that contains a malicious application named "SpyWare". However, as per our evaluation of the latest CVMs deployed on major cloud providers, applications are not measured into any of these measurement registers. Thus, when the Relying Party forwards the attestation report to the Attestation Verification Server, the CVM passes the attestation even though it contains a malicious program. To make matters more challenging, application measurement is a dynamic property that continuously changes as new applications are launched. This makes remote attestation difficult to achieve by comparing the measurement against a pre-calculated value.

In this paper, our goal is twofold. Firstly, we aim to provide the CVM users with a set of attestation prop-



(a) **(b)**

Fig. 1: Motivating examples: a) SEV-SNP CVMs' attestation reports have 40 properties where many of them have additional child properties, b) Remote attestation in Intel TDX uses an attestation report that includes measurement registers. Applications are not currently measured into these registers in the CVM offerings from major cloud providers allowing the possibility of malicious application injection.

erties and a guideline on how to verify those properties. We present the properties in a platform-agnostic way to simplify the complexities of individual CPU manufacturers' designs. Secondly, we aim to secure CVMs against any potential attack due to implementation deficiencies in the current CVM offerings. Providing security against these implementation issues requires considering vendor-specific design and due to space limitations, we cannot address both Intel and AMD in this paper. To that end, as the majority of the processors worldwide are from Intel [7] and they released their CVM technology just recently in 2023, we direct our focus on their CVM technology, Intel TDX. In particular, we provide a solution to the malicious application injection problem as depicted in Motivating Example 2 above. As part of the solution, we enhanced the Integrity Measurement Architecture (IMA) subsystem of the Linux kernel and developed a kernel-mode driver. Experimental results show that our solution does not add any significant overhead to the startup time of the kernel. In addition, we point out the gaps in existing Intel TDX offerings by the major cloud providers. In summary, our major contributions are as follows:

1) We identify a set of necessary attestation properties that can authenticate confidential VMs and formulate guidelines on verifying them (Section III).
2) We propose a solution to secure confidential VMs against malicious application injection attacks (Section IV).
3) We investigate CVM offerings from the major cloud providers and report missing security features necessary to secure the CVMs (Section IV).
4) We develop software (modification of the Linux kernel and a *Python* script) to defend against malicious application injection attacks. We show how users can develop their customized kernel to interface with the Intel Application Binary Interface (ABI), and thus, gain control of the remote attestation process (Section V).
5) We experimentally evaluate our proposed solutions to show they do not add significant overhead (Section V).

II. Background

A. Overview of Confidential Computing

Confidential computing secures "data in use" for cloud tenants while considering that the cloud platform may be under the control of adversaries. This assumption allows the cloud tenant to exclude most parts of the cloud platform from having to be trusted (e.g., hypervisors and cloud administrators). In confidential computing, there are two types of isolation [11]: 1) process-level isolation (e.g., Intel SGX) and 2) VM-level isolation (e.g., Intel TDX and AMD SEV-SNP). In the former, only the application that runs inside a so-called Enclave is trusted, while the operating system remains outside the Trusted Computing Base (TCB). The process-level isolation requires developing applications separately to be aware of the Enclave. To overcome this limitation, VM-level

isolation techniques have recently emerged where existing applications can run without modification. In VM-level isolation (i.e., CVM), the entire VM is inside the TCB. Besides the VM, only the hardware or firmware from the CPU vendor (e.g., Intel, AMD, ARM, etc.) is trusted [3], [5]. In this paper, we only cover VM-level isolation technologies.

B. Intel TDX

Intel TDX is the CVM technology from Intel [5]. It uses a dedicated CPU mode, called SEAM mode, to host a piece of software called the TDX module, responsible for serving the role of a trusted micro-hypervisor for CVMs. This design allows it to keep the traditional fully fledged third-party hypervisors out of the trusted computing base while still allowing them to manage CVMs.

C. AMD SEV-SNP

The CVM technology from AMD is known as SEV-SNP [6]. It uses a dedicated sub-system called secure-processor (AMD-SP), which hosts firmware that assumes the role of a micro-hypervisor (similar to the Intel TDX module).

D. Virtual Trusted Platform Module (vTPM)

A TPM is traditionally a hardware device that aids in securing a machine in several ways, such as providing secure storage for cryptographic keys and integrity measurements [2]. In the context of CVMs, TPMs are usually implemented in software and, are called virtual TPMs or vTPMs. One of the most useful features of a vTPM is its Platform Configuration Registers (PCRs).

E. Integrity Measurement Architecture (IMA)

One of the technologies that enable runtime integrity measurement in Linux is the Integrity Measurement Architecture (IMA) [18]. IMA is responsible for calculating the hash of certain files as soon as they are read. The exact files for which IMA calculates hashes are determined by its policy. Besides calculating the hashes, IMA also stores them to a TPM (or vTPM, depending on the platform).

III. Verifying Attestation Properties

To help users identify and verify the necessary attestation properties, our approach is as follows. Firstly, we design a set of generic attestation properties and map them to CPU vendor-specific counterparts. Secondly, we provide guidelines on how to verify each category of attestation properties.

A. Attestation Properties Design

In Table I, we list our designed property names (column 1) along with the technology-specific counterparts (Intel TDX-specific terms in column 2 and AMD SEV-SNP-specific terms in column 4). These properties are designed to form a *chain of trust* that starts with a component whose authenticity is guaranteed by hardware mechanisms (e.g., a secured chip) or digital signatures. This chain of trust is designed to continue up to the final layer of the software stack. Our identified properties are as follows:

Initial Measurement. We define initial measurement as a set of hashes of corresponding firmware components where: 1) the first component is loaded by an unmeasured component whose credibility is ensured by the CVM vendor (e.g., code that is protected by hardware mechanisms and/or signed by the vendor), 2) every component except the first one is measured by the previous component in the chain, and 3) the measurements are performed only once during the boot process of the VM.

$I = \{i_k : i_k \leftarrow i_{k-1}\}$

In the above equation, the left arrow ($\leftarrow$) indicates that the measurement of the k$^{\text{th}}$ component i_k is measured by software whose measurement is i_{k-1}.

In light of current CVM offerings, it mainly includes the virtual BIOS firmware. In the case of Intel TDX, the hash of the TDX module (MRSEAM) is also included.

Runtime Measurement. We define runtime measurement as a set of hashes of corresponding software components where: 1) the first component is measured by the last component of the initial measurement, and 2) it may yield different values depending on the software components executed. Notice that, in this case, the last component represents the set of processes running in the OS.

$R = \{i_k : r_k \leftarrow r_{k-1} \text{ and } r_1 \leftarrow i_N\}$

Intel TDX includes four runtime measurement registers labeled `RTMR[0]` to `RTMR[3]`. On the other hand, AMD SEV-SNP does not include a dedicated runtime measurement register, and users must rely on vTPM PCRs.

Nonce. It is designed to prevent replay attacks by ensuring the freshness of the report. In Intel TDX and AMD SEV-SNP, nonce is supported by allowing a user-provided data field to be included in the attestation report.

Security Version. It is a set of properties in the attestation report that provides a mechanism allowing users to verify that the firmware version of the

TABLE I: Generic properties to verify in attestation reports and their technology-specific counterparts (Intel TDX and AMD SEV-SNP).

Generic Property	TDX		SEV-SNP	
	Property	**Example**	**Property**	**Example**
Initial Measurement	MRSEAM	64 bytes	Not applicable	-
	MRTD	64 bytes	MEASUREMENT	64 bytes
Runtime Measurement	RTMR0..3	64 bytes	Not available	-
Nonce	REPORTDATA	64 bytes	REPORT_DATA	64 bytes
Security Version	TEE_TCB_SVN	3.0.6	LAUNCH_TCB	3.209.20.0
	TEE_TCB_SVN2	3.0.7	CURRENT_TCB	3.209.20.0
	CPUSVN	6.6.2.2.3.1.0.3	Not applicable	-
Security Settings	ATTRIBUTES	{DEBUG: 0, ...}	POLICY	{DEBUG: 0, ...}
Custom settings	XFAM	{..., AVX512: 1, ...}	PLATFORM	{..., SIMD: 1, ...}

platform is up-to-date. Each update of the security version is associated with a new encryption key.

In the case of Intel TDX, the security version comprises the version of the Intel TDX module and the microcode version. In the case of AMD SEV-SNP, it comprises the version of the secure processor's firmware version.

Security Settings. Security settings affect the security of the CVM as they may allow the hypervisor to read the private memory of the CVM. Therefore, it must be ensured that these settings match the desired values. In Intel TDX, `ATTRIBUTES.DEBUG` is set to 1 when the VM is in debug mode. For AMD SEV-SNP, this is indicated in `POLICY.DEBUG` flag.

Custom Settings. We group some vendor-specific hardware features as custom settings. In Intel, the flags in the `XFAM` property, and in AMD SEV-SNP, the flags of the `PLATFORM` property are considered part of this generic property.

B. Attestation Properties Verification

Existing documentation from CPU vendors and cloud providers does not clarify how to verify the attestation properties. Through our empirical evaluation and analysis of the CVM services, we have formulated the following guidelines to perform such verification.

Verifying Measurements. To verify measurements (i.e., *initial measurement* and *runtime measurement*), an expected value for each property should be pre-calculated. To pre-calculate a measurement, the following steps can be followed:

1) List the exact software and non-software components that contributed to computing a particular measurement with the help of an *event-log* (e.g., Confidential Computing Eventlog or CCEL).
2) Obtain source code and build parameters of each software component.
3) Build the software components to obtain their binaries.
4) Replay all the components in the same order as in the *eventlog* to compute the expected measurement.

Example III.1. Suppose, the CCEL contains 2 events with the `IMR` field set to 1. The first event corresponds to the plaintext file `grub.cfg` and the second event corresponds to the GRUB binary, `core.img`. Then, the user can compute the expected value for `RTMR1` as `H(H(grub.cfg)||H(core.img))`. Here, `H()` represents the `SHA-384` hash function and || represents a string concatenation.

It is to be noted that users may need to contact their cloud providers to obtain some of these settings. For example, in building the Intel TDX module (software corresponding to the `MRSEAM` value) three parameters must be known. These are `BUILD_DATE`, `BUILD_NUM`, and `UPDATE_VER`. According to our evaluations, these parameters are currently not made available and access to obtain them using the Intel ABI is denied for the guest CVM.

Verifying Security Versions. Security version information can be collected from the CPU vendors. For example, Intel provides an API [8] to retrieve the status of software versions through HTTPS requests. These APIs can be utilized to ensure that the security versions are up-to-date. On the other hand, AMD publishes a list containing all the firmware updates [6]. However, the list from AMD does not contain the complete TCB information.

Verifying Security Settings. Security settings can be verified by ensuring that any feature that negatively affects security (e.g., Debug) is not enabled.

Verifying Custom Settings. These properties can be verified against pre-computed values based on the presence or absence of reported vulnerabilities and performance trade-offs. For example, if `AVX2`

or `AVX512` is enabled on a platform, it may be vulnerable to Gather Data Sampling (GDS) [4] if the relevant microcode update is not applied. The user must carefully decide if the mitigation is necessary as it may affect performance.

IV. Missing Security Aspects

This section focuses on security aspects missing from existing Intel TDX CVM services.

A. Malicious Applications Injection

Currently, no cloud provider is providing a CVM image that covers the verification of all the processes running on a VM. This missing security aspect opens the door to the injection of malicious applications. For example, an attacker with privileged access to the bare-metal server may replace an application scheduled to be launched at startup with a malicious version.

To solve this issue, a naïve solution could be modifying the Linux IMA to extend a runtime measurement register with application measurements. However, simply extending a runtime measurement register with application measurements may lead to the failure of the remote attestation even when there are no malicious applications (i.e., it may lead to false positives). This may be due to a temporary file created by an application at a late stage of the boot process or during runtime. To solve this problem, we propose an algorithm (Algorithm 1) that searches for a match against a pre-calculated measurement within the IMA measurement list. Thus a failure to find a match can confirm the presence of an injected application.

Algorithm 1: Pre-caluclated measurement search and match algorithm

Data: Pre-calculated measurement (M), IMA measurement list (L)
Result: Confirmation of whether M can be formulated from the application template hashes within L
Initialize m to SHA-384 hash of 48 bytes of zeros;
for *each record* r *in* L **do**
  $t \leftarrow$ template hash in r;
  if *it is the first record* **then**
    $m \leftarrow m||t$;
  else
    $m \leftarrow$ SHA-384 hash of $m||t$;
  end
  if $M == m$ **then**
    return match;
  end
end
return failure;

B. Issues in Cloud TDX CVM Services

Among the cloud providers offering CVM services, we have performed hands-on evaluations of TDX offerings by Google, Azure, and Alibaba. We exclude Amazon Web Services (AWS) from our evaluation as they do not provide any TDX CVM service. CVM services are usually made available in three stages: private preview, public preview, and commercial. We started our evaluation with a private preview of TDX on GCP (Google has now made a public preview of TDX available). We have also evaluated TDX on Azure in its public preview. TDX on Alibaba Cloud is available commercially (i.e., no longer in a preview stage). In our evaluation, we have found several critical aspects missing. We list them in Table II. In the following, we detail these missing features:

1) *TDCALL Allowed.* TDCALL is the instruction to interface with the Intel TDX ABI. While it is allowed in GCP and Alibaba, Azure blocks the calls.
2) *Reproducible Measurements.* Necessary information required for reproducing the measurements is not made available, leaving the user unable to verify the integrity of the software.
3) *Custom Image.* The cloud providers currently support only built-in images for CVM creation. Therefore, users are unable to use their own custom images.
4) *Open-source Firmware.* As part of the process to launch a VM, the BIOS firmware is provided by the cloud. Currently, there is no documentation on which firmware is deployed and how to access the source code.
5) *Open-source vTPM.* The process for remote attestation of the vTPM is not documented, and the vTPM source code is not made public.

TABLE II: Missing features in public cloud TDX confidential VM offering. The symbols (-) and (•) mean unavailable and available, respectively.

Feature	Azure	Alibaba	GCP
TDCALL Allowed	-	•	•
Reproducible Measurements	-	-	-
Custom Image	-	•	-
Open-source Firmware	-	-	-
vTPM Open-source	-	-	-
Opensource Attestation	-	-	•
Firmware Security Version	-	-	-

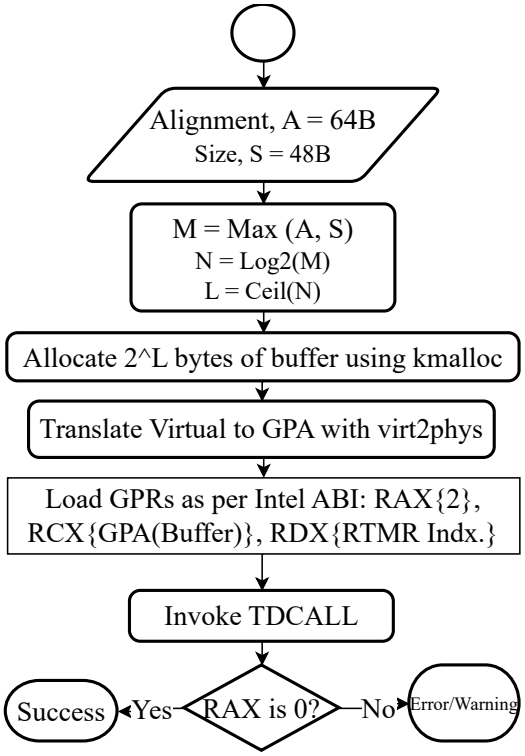


Fig. 2: The flowchart of the procedure for interacting with TDX hardware and firmware using the `TDCALL` instruction (updating RTMR as an example).

6) *Firmware Security Version.* Microcode and source code of the firmware are not released by specific security versions.

V. Implementation and Evaluation

We implemented our solution to detect malicious applications. Our implementation is divided into two steps as follows:

A. Interfacing with Intel ABI

Figure 2 illustrates our formulated steps for interacting with Intel TDX components. Intel specifies the TDCALL instruction for this purpose. The TDCALL instruction requires memory allocation before its invocation. The allocated memory has a minimum size requirement and must be aligned to a specified number. Moreover, users must specify the guest physical address (GPA) instead of the virtual address. The need for GPA led us to adopt a kernel-module-based approach instead of a user-space program. However, even with a kernel module, a challenge remains: to guarantee aligned memory. In the Linux kernel development library, no function explicitly allows memory alignment by a given number. To this end, we exploit

```
int64_t extend_rtmr(char*
    runtime_measurement){
  void *buf = kzalloc(64, GFP_KERNEL);
  int64_t execresult = 0;

  for(int i = 0; i < 48; i++)
    *((char*)buf+i) = *(runtime_measurement+
    i);

  phys_addr_t gpa_buf;
  gpa_buf = virt_to_phys(buf);
  asm("movq $2, %%rax;"      //Extend RTMR
  "movq %[pbuf], %%rcx;"     //Measurement
    buffer
  "movq $2, %%rdx;"          //RTMR[2]
  "tdcall;"
  "movq %%rax, %[eresult];" //Success?
  : [eresult] "=r"(execresult) : [pbuf] "r"
    (gpa_buf) : );
  kfree(buf);
  return execresult;
}
```

Listing 1: Prototype implementation of the function to extend a runtime measurement register from IMA.

the fact that the `kmalloc` function is guaranteed to allocate memory aligned to the allocation bytes when the allocation bytes is an integer power of 2. Therefore, we choose a number greater between the alignment size and allocation size and then choose the smallest power of 2 greater than or equal to that number as the allocation bytes. This way, we guarantee that the buffer will always satisfy both the allocation size and alignment requirements. We then use the `virt2phys` function to obtain the GPA. Finally, as mentioned in Intel ABI, the relevant registers are loaded with the addresses and constants before the TDCALL instruction is invoked. We have implemented the process to extend RTMRs in two ways: 1) a kernel module and 2) modified IMA code in Linux kernel.

B. Modifying the Linux IMA

Extending RTMR. To extend RTMR with application measurements from IMA, we added a function named `extend_rtmr` (Listing 1) in `security/integrity/ima/ima_queue.c` and issued a call to that function from `ima_add_template_entry` function to our `extend_rtmr` function by passing the `digest` as a parameter. The `extend_rtmr` function copies the `digest` to its buffer to satisfy the memory alignment requirements of Intel ABI.

We tested our software to extend RTMR on GCP and Alibaba. Alibaba's default CVM comes with *binutils* version 2.35, which does not support TD-

CALL instruction. After updating binutils to version 2.42, our driver containing the new TDCALL instruction could be compiled. It is to be noted that the minimum binutils version supporting TDCALL is 2.36.

Producing Appropriate Template. The Linux kernel currently in use by the CVM services (v6.5) is hardcoded to output template hash in `SHA-1` in the measurement list. This hindered our ability to verify application measurements. To overcome this limitation, we modified the IMA to output the template hash in the algorithm chosen by the user. So, for TDX, our modified kernel can appropriately produce the template hash in `SHA-384`. All our modifications to the Linux IMA will be made available on our public GitHub repository.

C. Experimental Evaluation

We experimentally evaluated our modification of the Linux kernel and the application measurement verification algorithm.

Experimental Settings. These experiments were carried out on GCP with a machine type of `c3-standard-4`. Experiments were performed for different settings of the `ima_policy` kernel parameter. This is because IMA policies determine which files or components will be included in the measurement. Thus, the number of times our added code gets executed for different IMA policies will vary. We repeated each experiment 30 times.

IMA Modification. To evaluate whether the IMA modification adds any significant overhead in terms of startup completion time, we performed a series of experiments. As our kernel modification is on the IMA subsystem, we used the IMA startup time as our metric. Specifically, we considered the timestamp of the last message from IMA in the `dmesg` ring buffer as the IMA startup completion time. We performed these experiments for both the unmodified and modified kernel. Our experimental findings are illustrated in Figure 3, where it is clear that the modified kernel does not have significantly more overhead than the unmodified kernel.

Application Measurement Verification. We evaluate our application measurement verification algorithm at runtime of the CVM after boot is completed. Table III illustrates the results. Here, `Num_entry` means the number of applications measured, and `Exec_time` means the amount of time taken to match the pre-calculated measurement with measurement computed from the application measurements in the `ascii_runtime_measurement` list.

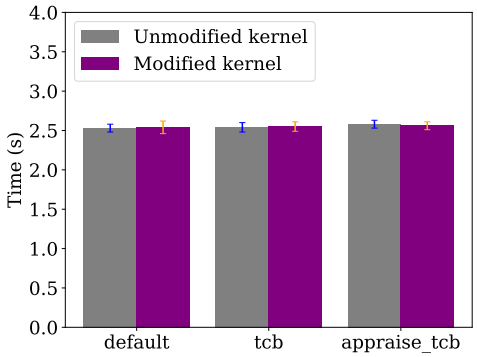


Fig. 3: Comparison of startup completion time between the unmodified and our modified Linux kernel for different IMA policies shows that our modification does not add any significant overhead.

IMA Policy	Num_entry	Exec_time (s)
default	1	0.017
tcb	1978	0.022
appraise tcb	2005	0.022

TABLE III: Application measurement verification times

VI. Related Work

In this section, we review the literature related to confidential computing.

A. Confidential Computing Concepts

An overview of TEE technologies is given by Lacoste et al. [15]. They briefly compare Intel SGX, Intel TDX, and AMD SEV-SNP, followed by a discussion of various aspects, such as benefits and attacks/vulnerabilities. Cheng et al. [10] provides a detailed discussion of the Intel TDX technology. A discussion of various architectural aspects of both SEV-SNP and TDX can be found in [14]. Guanciale et al. [12] compare confidential computing technologies from four silicon providers: Intel, AMD, ARM, and IBM. They provide an overview of each confidential computing platform followed by comparison, use case, and open research problems. However, none of the aforementioned works focus on the difficulty of configuring and verifying attestation properties essential for the security of CVMs as we do.

B. Remote Attestation

Intel Trust Authority [1] is a quote verification service from Intel that works in two modes: passport mode and background check mode. Using their service, users still need to supply policies that leave them with the problems our paper solves. Therefore, our work is complementary to that of Intel Trust

Authority. Moreover, it works only for Intel TDX and cannot be used as a verifier for other TEE solutions. Additionally, although the research community agrees to trust standardized hardware from a vendor with open specifications (e.g., Intel or AMD CPUs supporting hardware TEE features), they find it less reasonable to trust a remotely hosted internet-facing service to do a complete verification [9]. Our work can help users develop their own verifier.

Haidong et al. [13] describe trust authority-client as a tool to perform runtime measurements. Opera [9] proposes a remote attestation mechanism where users of TEEs can verify attestation quotes by themselves. Keylime [17] utilizes trusted computing to scale trust services that would otherwise be provided by slow hardware TPMs. It combines TPM and IMA to facilitate remote attestation focusing on runtime integrity. TRIGLAV [16] proposes remote attestation of virtual machines intending to ensure runtime integrity. It extends the threat model of confidential computing by considering the possibility of CVMs being compromised at runtime.

VII. Conclusion

In this work, we have investigated the latest innovation in the confidential computing ecosystem, namely, Confidential Virtual Machines (CVM) from a security perspective. Our investigation shows that the CVM technology is complex, insufficiently documented, and incompletely adopted. The results are complex and incomplete services that are not yet ready to deliver the desired protection to the users. To address these issues, our approach is twofold: firstly, we develop a generic set of attestation properties and provide guidelines on their usage to verify the authenticity of CVMs. Secondly, we identify and report gaps in the current services. Specifically, we identify a potential feasible attack on current Intel TDX CVM services. We provide an algorithm to detect this attack and through experimental evaluation of our implementations, we show that our solution does not add any significant overhead. Our future goal will be to use AI techniques to mine vulnerabilities from CVE databases and warn the users if any of those vulnerabilities affect their *custom settings*.

ACKNOWLEDGMENT

This work has been financed through the Secure and Scalable Identity Provisioning (SSIP) project, a collaboration between Hewlett Packard Enterprise Brazil and the EMBRAPII unit UFCG-CEEI (Universidade Federal de Campina Grande) with the incentive of the Informatics Law (Law 8.248 from October 23rd, 1991).

References

[1] Intel® trust authority. https://docs.trustauthority.intel.com/main/articles/introduction.html. [Online; accessed 05-Sep-2024].

[2] Trusted platform module (TPM). https://trustedcomputinggroup.org/resource/trusted-platform-module-tpm-summary/, 2008. [Online; accessed 05-Sep-2024].

[3] Strengthening VM isolation with integrity protection and more. https://www.amd.com/system/files/TechDocs/SEV-SNP-strengthening-vm-isolation-with-integrity-protection-and-more.pdf, 2020. [Online; accessed 24-Sep-2024].

[4] Gather data sampling (GDS) vulnerability. https://www.intel.com/content/www/us/en/developer/articles/technical/software-security-guidance/technical-documentation/gather-data-sampling.html, 2023. [Online; accessed 01-Sep-2024].

[5] Intel trust domain extensions. https://www.intel.com/content/dam/develop/external/us/en/documents/tdx-whitepaper-final9-17.pdf, 2023. [Online; accessed 24-Sep-2024].

[6] Secure Encrypted Virtualization (SEV). https://www.amd.com/en/developer/sev.html, 2023. [Online; accessed 3-Sep-2024].

[7] Distribution of intel and amd x86 cpus. https://www.statista.com/statistics/735904/worldwide-x86-intel-amd-market-share/, 2024. [Online; accessed 24-Sep-2024].

[8] Get TDX TCB info. https://api.portal.trustedservices.intel.com/content/documentation.html, 2024. [Online; accessed 25-Sep-2024].

[9] Guoxing Chen, Yinqian Zhang, and Ten-Hwang Lai. Opera: Open remote attestation for intel's secure enclaves. In *ACM SIGSAC*, pages 2317–2331, 2019.

[10] Pau-Chen Cheng, Wojciech Ozga, Enriquillo Valdez, Salman Ahmed, Zhongshu Gu, Hani Jamjoom, Hubertus Franke, and James Bottomley. Intel TDX demystified: A top-down approach. *arXiv preprint arXiv:2303.15540*, 2023.

[11] Christophe de Dinechin. Confidential computing platform-specific details. https://www.redhat.com/en/blog/confidential-computing-platform-specific-details, 2023. [Online; accessed 24-Sep-2024].

[12] Roberto Guanciale, Nicolae Paladi, and Arash Vahidi. Sok: Confidential quartet-comparison of platforms for virtualization-based confidential computing. In *SEED*, pages 109–120. IEEE, 2022.

[13] Xia Haidong, Lu Ken, Ying Ruoyu, Dong Xiaocheng, and Zhao Yanhui. Runtime integrity measurement and attestation in a trust domain. https://www.intel.com/content/www/us/en/developer/articles/community/runtime-integrity-measure-and-attest-trust-domain.html, 2023. [Online; accessed 29-Aug-2024].

[14] Kevin Kollenda. General overview of amd sev-snp and intel tdx.

[15] Marc Lacoste and Vincent Lefebvre. Trusted execution environments for telecoms: Strengths, weaknesses, opportunities, and threats. *IEEE Security & Privacy*, 2023.

[16] Wojciech Ozga, Christof Fetzer, et al. Triglav: Remote attestation of the virtual machine's runtime integrity in public clouds. In *CLOUD*, pages 1–12. IEEE, 2021.

[17] Nabil Schear, Patrick T Cable, Thomas M Moyer, Bryan Richard, and Robert Rudd. Bootstrapping and maintaining trust in the cloud. In *ACSAC*, pages 65–77, 2016.

[18] Huzaifa Sidhpurwala. How to use the linux kernel's integrity measurement architecture. https://www.redhat.com/en/blog/how-use-linux-kernels-integrity-measurement-architecture, 2020. [Online; accessed 24-Sep-2024].