Secure Distributed Computing on Untrusted Fog Infrastructures Using Trusted Linux Containers

Mohammad-Mahdi Bazm, Marc Lacoste, Mario Südholt, Jean-Marc Menaud

Secure Distributed Computing on Untrusted Fog Infrastructures Using Trusted Linux Containers

Mohammad-Mahdi Bazm*, Marc Lacoste*, Mario Südholt†, Jean-Marc Menaud†

*Orange Labs, 44 Avenue de la République, 92320 Châtillon, France

†IMT Atlantique, 4 Rue Alfred Kastler, 44307 Nantes, France

***Abstract*—Fog and Edge computing provide a large pool of resources at the edge of the network that may be used for distributed computing. Fog infrastructure heterogeneity also results in complex configuration of distributed applications on computing nodes. Linux containers are a mainstream technique allowing to run packaged applications and micro services. However, running applications on remote hosts owned by third parties is challenging because of untrusted operating systems and hardware maintained by third parties. To meet such challenges, we may leverage trusted execution mechanisms. In this work, we propose a model for distributed computing on Fog infrastructures using Linux containers secured by Intel's Software Guard Extensions (SGX) technology. We implement our model on a Docker and OpenSGX platform. The result is a secure and flexible approach for distributed computing on Fog infrastructures.**

***Index Terms*—Fog computing security, distributed computing, Intel Software Guard Extensions (SGX), trusted computing, secure computing, Linux containers.**

I. Introduction

Fog computing extends computing resources and services to the edge of the network to provide QoS by supporting low latency and location awareness to meet end-user requirements. This paradigm is also well positioned for Big data and distributed computing applications due to the large amount of computing resources accessible through virtualization technology. Recently, Linux containers have received more attention in Fog and IoT clouds due to their flexibility and performance benefits compared to VMs. Distributed computing based on VMs and Linux containers offers more security (*e.g.*, isolation) than traditional computing models *e.g.*, user-based. However, integrity and confidentiality remain key concerns because of *untrusted* operating systems and hosts maintained by third-parties in the cloud infrastructure. The primary challenge is how to execute a distributed application securely inside VMs or in Linux containers running on remote hosts (*i.e.*, guaranteeing isolated execution, data integrity and confidentiality). Hardware-assisted trusted execution mechanisms such as *Intel's Software Guard Extensions (SGX)* and *ARM TrustZone*, are highly promising to overcome such challenges. A secure component like the TPM (Trusted Platform Module) in a computing platform plays the role of *Root of Trust (RoT)* through specific embedded cryptographic mechanisms. To make the platform trusted, the RoT extends its trust to other components of the

platform (*e.g.*, OS) by building a *Chain of Trust (CoT)* in which each element extends its trust to the next element in the chain until launching applications. This mechanism allows identifying any modification in the host before launching a container [1]. However, such mechanism is not necessarily implemented by remote hosts especially in heterogeneous infrastructures such as Fog, thus requiring new solutions. Intel SGX allows executing securely sensitive part/data of an application inside isolated zones (called *enclaves*) through the processor, by encryption mechanisms to protect data from operating system and hardware memory attacks. Consequently, it allows secure execution of the application on untrusted remote hosts. This technology can be leveraged in distributed computing where a distributed application should be run on remote hosts of the cloud infrastructure. It is also integrated into Linux containers to protect packaged applications [2].

In this paper, we discuss the challenge of trusted computing on remote hosts. We review different scenarios to secure execution of Linux containers on remote hosts in the cloud infrastructure through trusted execution mechanisms, and finally we propose a model for secure distributed computing through using Linux containers hardened by Intel SGX, to perform trusted execution on untrusted Fog computing infrastructures.

The paper is organized as follows: Section II gives an overview of existing methods of distributed execution of applications in Fog computing infrastructures. Section III discusses security challenges of computing on remote hosts, and Section IV reviews two corresponding trusted execution scenarios. Section V gives an outlook on the model and implementation of our prototype and Section VI evaluates it. Section VII reviews related works. Section VIII concludes.

II. Distributed Computing on Fog Infrastructures

In *traditional* models, specific OS mechanisms allow multiplexing processor and memory between different processes and user accounts. A distributed application runs as a process on a remote machine through an operating system user. However, in highly heterogeneous infrastructures, legacy compatibility is an issue for the execution of distributed applications. Integrity and confidentiality of data and isolated execution cannot be guaranteed because security mechanisms are implemented in the operating system of remote hosts and managed by untrusted host administrators. In *VM-based* computing, a distributed application runs on a set of consolidated VMs. All VMs are managed by the administrator of the distributed computing platform, although VMs run on remote physical machines owned by third parties. This computing model offers more security for the execution of the distributed application because of the virtualization abstraction layer. It also provides stronger isolation and enables to implement and enforce security mechanisms inside VMs independently of the host, to guarantee data processing. However, the VM-based model suffers from flexibility limitations (*e.g.*, VM migration) intrinsic to virtualization techniques [3]. In *container-based* computing, the distributed application is packaged in a set of containers. It then runs on hosts managed by third parties on the Fog infrastructure in the form of a set of containers to perform computations. This model provides more flexibility than the VM-based model regarding cluster building, because of fast provisioning and easy management of Linux containers.

III. Problem Statement

We address two main challenges related to secure and flexible distributed execution of applications on remote

hosts. ***Remote Execution:*** In remote computing, if an application executes directly on hosts, inside VMs, or in Linux containers, there is always a security concern with such computing models: how to secure (*i.e.*, guaranteeing isolated execution of application) remote computations? Running an application inside VMs/Linux containers may appear more secure than running it on hosts for isolation and confidentiality. For instance, all containers/applications in a host share the same kernel instance. Thus, any kernel compromise results in security vulnerabilities. The same scenario applies for VMs although isolation is stronger between VMs than containers due to enforced isolation by the virtualization layer. An adversary may employ Virtual Machine Introspection (VMI) [4] to retrieve information of the VM. We may also run a container inside a VM, to enforce isolation [5]. However, the real challenge is the execution of the application (either inside VMs or Linux containers) on remote hosts maintained by untrusted third-parties (the hypervisor/OS of hosts thus remain untrusted).

Hardware-assisted primitives provide an isolated execution environment between running applications/containers in the host OS, or between running VMs. These mechanisms provide integrity and confidentiality for the execution of applications/ containers/ VMs: the memory of applications/containers/VMs is hardware-encrypted, and therefore protected from any unauthorized access from untrusted OS/hypervisor. Memory encryption techniques create strong isolation between applications, application-OS, VMs, VM-hypervisor, containers, and container-OS. However, we cannot always trust the application running inside containers/VMs because the application may be modified or compromised by an adversary, thus resulting in information leakage or underlying system compromise. To meet such challenge, *Remote Attestation* allows remote parties to detect any change in the application. It enables a trusted device (*i.e.*, a device with RoT) to present reliable evidence to remote parties about the state of application executing on remote hosts. This evidence may be for instance a cryptographic signature of a hash of application.

Performance and Flexibility: For distributed computing, a large set of computing resources is needed to run a distributed application on Fog infrastructures. How to provision such a resource pool in an efficient way with greater security and flexibility than the traditional approachs? Even when using trusted computing mechanisms [6] [7], challenges remain regarding fast provisioning and configuration on remote hosts. Linux containers provide a reliable way of encapsulating an application and its configuration with the flexibility of image-based deployment methods. Therefore, distributed computing using containers which are secured by trusted execution mechanisms has been getting more attention for Fog computing [3].

IV. Trusted Execution Over Fog Infrastructures

We distinguish two scenarios for trusted execution of Linux containers on remote hosts in the Fog infrastructure (Fig. 1).

a) Trusted host: Software CoT process: Containers are run on a cluster of **trusted** hosts in the infrastructure. Before launching a container image, a host proves itself as trusted host by following the CoT mechanism. The CoT starts from boot to launching the container engine that starts with an implicitly trusted component (*i.e.*, RoT) and every other component trusted before being executed (Fig. 1a). If the host is trusted, it is added to the pool of trusted hosts which are maintained by third parties. Consequently, the host can launch container images. The container engine then

verifies run-time trustworthy of the container image. If the proof is valid, the container is authorized to be executed on the host and the master node starts sending data to the container for processing. *Trusted Docker* containers [1], and *CoreOS* [8] provide such security solutions by extending the CoT to the Linux container image.

b) Untrusted host: hardware trusted execution: Containers are directly launched on a cluster of **untrusted** hosts. Trusted execution mechanisms are leveraged in the container image encapsulating the application, through hardware mechanisms. The underlying layers (*i.e.*, container engine, OS) are considered untrusted (Fig. 1b). The application executes in the trusted execution environement provided by hardware. If the application is proved trusted through remote attestation, it means it has not been tampered with. The master node then starts sending data to the application. This scenario is an optimized solution not based on a CoT which is a complex process to implement on hosts especially in heterogeneous infrastructures like Fog ones. This solution is preferred when third parties do not want to implement the CoT in their systems. SCONE [2] provides such capability by securing Docker containers leveraging Intel SGX technology.

V. Secure Distributed Computing Model

We consider $\mathcal{H} = \{h_1, h_2, ..., h_n\}$ a set of physical machines in the Fog infrastructure with a hardware-backed trusted execution technology $te \in \mathcal{TE}$– set of trusted computing technologies. We also consider $\mathcal{C} = \{c_1, c_2, ..., c_k\}$ a set of containers to run on $\mathcal{H}$. We introduce a software component $sc \in SC$– set of software components, to be integrated in the container image c. c is composed of sc and $u \in U$– set of base images used to provide some libraries to the software. Any modification in sc and u results in new container image $c^{'} \in C^{'}$– set of tampered images. Thus the *Tamper* operation on c denoted as follows:

$$Tamper: \ \mathcal{SC} \times \mathcal{U} \rightarrow \mathcal{C}^{'}, \ (sc, u) \mapsto c^{'} \qquad (1)$$

We also consider dataset $D = \{d_1, d_2, ..., d_k\}$ to be processed by containers and $R = \{r_1, r_2, ..., r_k\}$ the result of data processing. The *Process* operation of d_i by container c_i through te on h_j is defined as follows:

$$Process : \mathcal{D} \times \mathcal{C} \times \mathcal{H} \times \mathcal{TE} \rightarrow \mathcal{R}, \ (d_i, c_i, h_j, te) \mapsto r \qquad (2)$$

Definition. *c_i is a trusted container if its whole image has not been tampered with and proved as such through a hardware-assisted remote attestation mechanism provided by the trusted execution technology te available on hosts in $\mathcal{H}$.*

Trust Model: Our model leverages Intel SGX as trusted execution technology, and employs **scenario b**. Thus, critical parts of the application deployed in the container are protected inside enclaves. Other parts outside enclaves, are unprotected. Although all network links are encrypted between hosts by network security protocols such as *IPSec*, the network is not trusted.

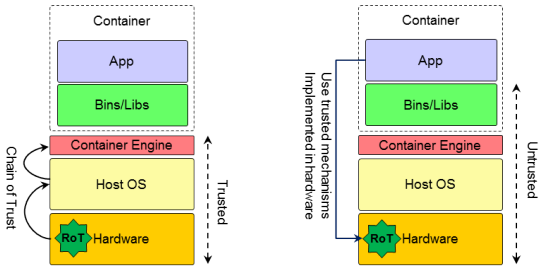


Figure 1: Different scenarios of trusted execution of containers.

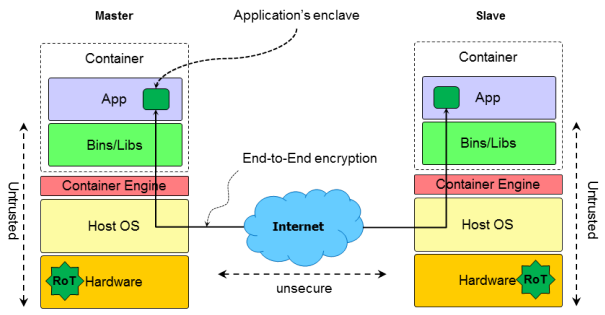


Figure 2: End-to-end encryption between two enclaves.

Thus security between the application in the container and the master application is enforced through encryption mechanisms to guarantee end-to-end confidentiality. Intel SGX provides some mechanisms to encrypt communications between two enclaves. Keys used by enclaves are also stored inside the enclave protected by hardware. Thus, only the application has access to keys, and communications between enclaves of slave and master applications are encrypted (Fig. 2).

Container Deployment: The first step to build the distributed computing environment is the deployment of containers on physical machines. Different orchestration frameworks enable to manage a set of containers *e.g.*, Kubernetes, on hosts.

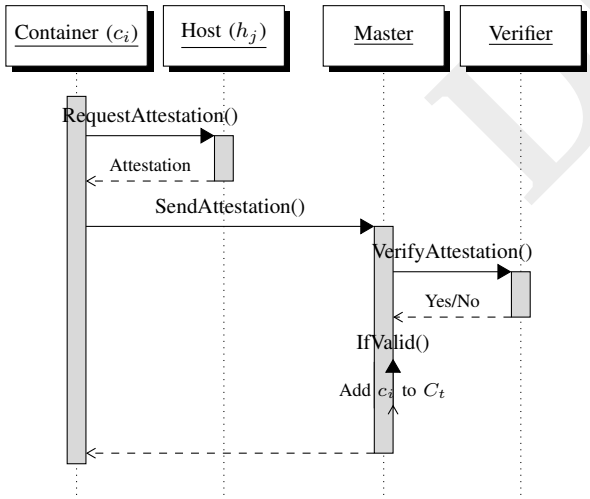


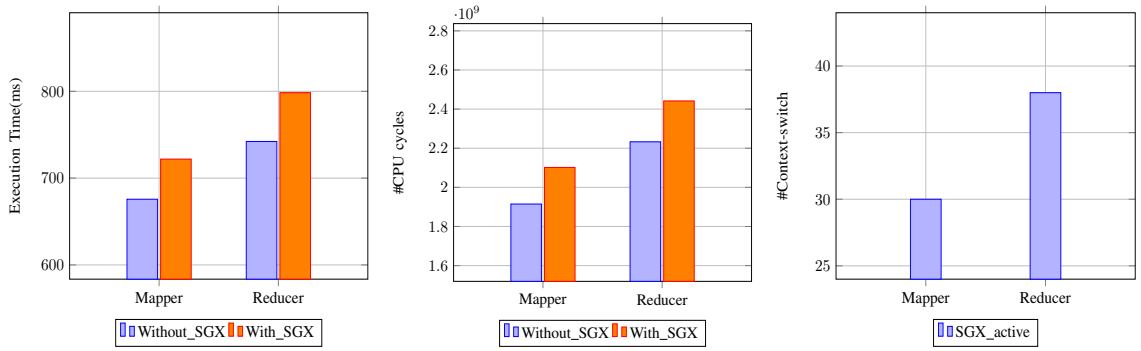
Figure 3: Join protocol: adding a container to a trusted cluster.

Trusted Cluster Creation: To build a trusted cluster composed of several trusted nodes, when a new container c_i wishes to be integrated to the cluster C_t, it initiates the *join* protocol (Fig. 3). It consists of a remote attestation mechanism to prove the container as a trusted node, by communicating with the master node. If the container is approved as a trusted node (*i.e.*, the application deployed in the container has not been tampered with), the node is integrated to the cluster and the master starts to send data to the approved container for processing. Otherwise, the node is considered untrusted because the packaged application has been tampered with.

VI. Implementation and Evaluation

To implement our prototype, we need a container image (u) that provides Intel SGX mechanisms to the application running within container. We used OpenSGX [9], a research platform for developing SGX-based applications. It implements and emulates Intel SGX ecosystem instructions, and hardware components leveraging the QEMU emulator. OpenSGX provides certain mechanisms available in SGX supported processors.

To evaluate the model, we implemented a *MapReduce* distributed computing framework based on Linux containers, to count number of words in a text file. MapReduce being well-recognized for its scalability and flexibility, decomposing an application into microservices. In the prototype, we have three types of nodes: *Mapper*, *Reducer*, and *Master*. The Master splits jobs/-data and distributes them to Mapper nodes. Then, Mappers perform their computing jobs and return the results back to Reducers for merging. Finally, Reducers send results to the Master node. Our experimental prototype is composed of one Master and 10 Slaves nodes. In our



(a) Node execution time (b) Number of CPU cycles (c) Number of context switches

Figure 4: Performance of Mapper and Reducer nodes with and w/o Intel SGX.

model, we consider that data to be processed by nodes are sensitive. Thus, they must be protected inside Intel SGX enclaves. Lesser critical sections of the application such as network functions are thus placed outside of the enclave and not protected by Intel SGX.

To evaluate the performance overhead of executing SGX instructions on computing nodes (*i.e.*, Linux containers) running Mapper and Reducer programs, we measure *execution time*, *CPU cycles*, and *Context switching* when SGX is enabled/not-enabled. We use a *Dell i5-3340M @2.70GHz* with 8GB of RAM, running Ubuntu 14.04.

Node execution time: As shown in Fig. 4a, the execution time of a complete workflow of a client node is higher than normal execution when using Intel SGX. We verified this difference through Linux `perf`. This additional execution time is caused by SGX special instructions, SGX library calls, enclave initialization, and memory encryption mechanisms.

Number of CPU cycles: We measured the number of CPU cycles when SGX was enabled (Fig. 4b). Using SGX results execution of a higher number of instructions because of instructions used by SGX to process data and code in enclaves.

Context switching: Another phenomenon that has impact on the performance of SGX applications is *context switching* between the enclave and the rest of application. During the execution of the application, enclave data and code processing leads to CPU mode changes. Therefore, each enclave enter/exit requires to store/restore CPU states, operations which are costly in terms of performance. Fig. 4c shows the number of context-switches incurred in the execution of Mapper and Reducer nodes. Overall, Intel SGX instructions impose $\backsimeq$11.2% performance overhead to Mapper and Reducer nodes. This is essentially the overhead of using OpenSGX in our prototype (*i.e.*, performance emulation of Intel SGX).

Our prototype leverages Intel SGX enclaves, thus provides the same security guarantees which are provided by Intel SGX. Intel SGX (like other existing trusted computing approaches), protects the software against memory attacks such as malicious OS, malicious hypervisor, and malicious firmware. However, it does not protect against cache-based side-channel attacks [10].

VII. Related Works

Zhang et al. [6] proposed an approach for secure computing on cloud infrastructures based on trusted virtual

machines. Their approach implements trusted computing mechanisms using TPM to guarantee data privacy and security. *CoreOS* [8] is a minimal Linux OS to run Linux containers especially those that are Docker-based. It provides a secure and reliable cluster for running distributed applications. It leverages Kubernetes for cluster management and Docker for application management. It also offers a trusted execution environment leveraging the CoT mechanism. CoreOS extends the CoT to container image by implementing specific mechanisms in *rkt* [11] container engine to provide the remote attestation.

VIII. Conclusion

We have seen that containers hardened by a trusted execution technology like Intel SGX bring more flexibility and security to distributed computing. We proposed a container-based model for secure distributed computing over Fog infrastructures. To illustrate the model, we implemented a container-based MapReduce prototype. Evaluation results show a reasonable performance overhead of using Intel SGX. As future work, we intend to explore AMD-SEV for Linux containers.

References

[1] "Trusted docker containers and trusted vms in openstack." https://01.org/sites/default/files/openstacksummit_vancouver_trusteddockercontainers.pdf.

[2] S. Arnautov *et al.*, "Scone: Secure linux containers with intel sgx.," in *OSDI*, pp. 689–703, 2016.

[3] V. Korkhov *et al.*, "Distributed computing infrastructure based on dynamic container clusters," in *International Conference on Computational Science and Its Applications*, pp. 263–275, Springer, 2016.

[4] T. Garfinkel *et al.*, "A virtual machine introspection based architecture for intrusion detection.," in *"NDSS"*, vol. 3, pp. 191–206, 2003.

[5] Intel, "Intel clear containers: Building a virtualization continuum," white paper.

[6] N. Zhang *et al.*, "Enabling trusted data-intensive execution in cloud computing," in *Communications and Network Security (CNS), 2014 IEEE Conference on*, pp. 355–363, IEEE, 2014.

[7] R. Pires *et al.*, "A lightweight mapreduce framework for secure processing with SGX," *CoRR*, vol. abs/1705.05684, 2017.

[8] "Coreos container operating system." https://coreos.com/.

[9] P. Jain *et al.*, "Opensgx: An open platform for sgx research.," in *NDSS*, 2016.

[10] M.-M. Bazm *et al.*, "Side-Channels Beyond the Cloud Edge : New Isolation Threats and Solutions," in *CSNet*, (Rio de Janeiro, Brazil), Oct. 2017.

[11] "The security-minded container engine by coreos: rkt hits 1.0." https://coreos.com/blog/rkt-hits-1-0.html.