Horizon 2020

Project Acronym and Title: 5GhOSTS – 5th Generation Security for Telecom Services

THE TRUST MODEL FOR MULTI-TENANT 5G TELECOM SYSTEMS RUNNING VIRTUALIZED MULTI-COMPONENT SERVICES

Deliverable number: D1.3

Version 0.9/1.0

Funded by the European Union's Horizon 2020 research and innovation programme under the Marie Sklodowska-Curie Grant Agreement No. 814035

Project Acronym: 5GhOSTS
Project Full Title: 5th Generation Security for Telecom Services
Call: H2020-MSCA-ITN-2018
Grant Number: 814035
Project URL: https://5ghosts.eu/

Editor:	Christoph Baumann, Ericsson
Deliverable nature:	Report (R)
Dissemination level:	Public (PU)
Contractual Delivery Date:	2021-02-28
Actual Delivery Date	2021-07-19
Number of pages:	54
Keywords:	5G, Trust Model, Security, Privacy, Virtual Network Function, Container, Kubernetes, Confidential Computing, Trusted Execution
Authors:	Merve Turhan, merve.turhan@ericsson.com, Ericsson Gianluca Scopelliti, gianluca.scopelliti@ericsson.com, Ericsson Christoph Baumann, christoph.baumann@ericsson.com, Ericsson Jan Tobias Mühlberg, jantobias.muehlberg@cs.kuleuven.be, KU Leuven Mykyta Petik, mykyta.petik@kuleuven.be, KU Leuven Eddy Truyen, Eddy.Truyen@cs.kuleuven.be, KU Leuven
Peer review:	Jan Tobias Mühlberg, jantobias.muehlberg@cs.kuleuven.be, KU Leuven Hans Eriksson, hans.eriksson@ericsson.com, Ericsson

Abstract

In the 5GhOSTS project we analyze and improve the security of service-based implementations of 5G networks, with a special focus on security and privacy aspects of the development and deployment of containerized Virtual Network Functions (VNFs). Research and development activities in 5GhOSTS center around three novel use cases, Vehicle to Infrastructure (V2I), Smart Office, and Remote Surgery, all of which impose challenging requirements regarding low-latency, privacy, and service resilience on our work. We address these requirements by developing a Kubernetes-based edge platforms for 4/5G, which is informed by comprehensive security and privacy threat modelling by both researchers from the domains of telecommunications law and computer security.

In this deliverable we focus on the trust model for our 4/5G edge platform, and survey security and privacy preserving technologies that will enable us to implement this trust model with Kubernetes-based VNF deployments, ensuring the integrity and overall security of the deployed VNFs in the presence of strong attackers and even untrusted infrastructure, and thereby protecting the data and privacy of users of the service.

The survey leads us to four specific research tracks that the 5GhOSTS project will follow-up on in the near future. In particular, we will further investigate the security of orchestration, isolation and attestation mechanisms, focusing on advanced and efficient approaches to runtime attestation, extend isolation primitives to intra-container isolation, incident detection and recovery, work explicitly towards securing communications interfaces between VNFs, and high-performance and low-latency approaches to integrate these mechanisms. Our work will be backed up by formal research into verifying, e.g., specific isolation approaches, interfaces, as well as security policies. Furthermore, we aim to work towards building a technology stack that enables privacy impact assessment of user-facing services.

Executive Report Summary

In the 5GhOSTS project we analyze and improve the security of service-based implementations of 5G networks, with a special focus on security and privacy aspects of the development and deployment of containerized Virtual Network Functions (VNFs). Research and development activities in 5GhOSTS center around three novel use cases, Vehicle to Infrastructure (V2I), Smart Office, and Remote Surgery, all of which impose challenging requirements regarding low-latency, privacy, and service resilience on our work. We address these requirements by developing a Kubernetes-based edge platforms for 4/5G, which is informed by comprehensive security and privacy threat modelling by both researchers from the domains of telecommunications law and computer security. These efforts have been elaborated on in previous project output (cf. [1, 2, 3]). In this deliverable we focus on the trust model for our 4/5G edge platform, and survey security and privacy preserving technologies that will enable us to implement this trust model with Kubernetes-based VNF deployments, ensuring the integrity and overall security of the deployed VNFs in the presence of strong attackers and even untrusted infrastructure, and thereby protecting the data and privacy of users of the service.

In Chapter 2 we present a threat and trust modelling effort for our V2I use case, presenting the different actors and their trust relations. For example, the user side covers driver, pedestrian, and the road sensor infrastructure such as traffic lights or cameras. As we illustrate, attackers must be considered as distrusted and all systems must take a precaution against attackers. Attackers can easily reach the user side, attempt to spoof other users' identity, or compromise them as a step stone to attack the edge platform. That is why the user side actors do not trust each other. However, the users must semi-trust the edge service provider and mobile network operator, because they depend on a lot of information and functionality from them. For example, in the traffic management service use case, they should share data, and get optimized directions with lower traffic or real-time information about their immediate environment. Conversely, Mobile Network Operator (MNO) distrust the user side. However they semi-trust the edge manager as it is one of a few well-identified customers of the MNO's edge and mobile network infrastructure. Nevertheless, if protections against a compromise of either side are dropped, this may pose a risk if such a compromise actually occurs. Overall, there are no completely trusted actors in the V2I edge cloud scenario. Explicating and understanding this is important for our choices of technology. To satisfy the security and privacy needs of all actors we need to aim for a notion of a Zero Trust Architecture, where active security controls give novel guarantees about authenticity and integrity to the stakeholders involved in each interaction, establishing an immediate notion of trustworthiness.

In Chapter 3 we then review technology and architectural choices to harden and to securely deploy and use Kubernetes containers that implement VNFs. While our focus there is on Trusted Execution Technology (TEEs) and similar approaches that enable a notion of confidential computing, where VNFs can execute securely with a minimal Trusted Computing Base, we present these approaches in the context of established means of reducing the attack surface of containers, static vulnerability detection, and secure image deployment. Our survey identifies a number of weaknesses and shortcomings in established approaches to use TEEs in combination with Kubernetes containers in general and VNF scenarios in particular. These relate to overall system performance, the need for support for highly dynamic and heterogeneous for our specific use cases, and unclear security objectives and attacker models, which need to be reworked and put in context to enable use cases in 5G edge.

The survey leads us to four specific research tracks that the 5GhOSTS project will follow-up on in the near future. In particular, we will further investigate the security of orchestration, isolation and attestation mechanisms, focusing on advanced and efficient approaches to runtime attestation, extend isolation primitives to intra-container isolation, incident detection and recovery, work explicitly towards securing communications interfaces between VNFs, and high-performance and low-latency approaches to integrate these mechanisms. Our work will be backed up by formal research into verifying, e.g., specific isolation approaches, interfaces, as well as security policies. Furthermore, we aim to work towards building a technology stack that enables privacy impact assessment of user-facing services.

Contents

List of Figures

List of Tables

List of Acronyms

AESM	Application Enclave Services Manager
AWS	Amazon Web Services
CFA	Control-Flow Attestation
CFG	Control-Flow Graph
CFI	Control-Flow Integrity
CNCF	Cloud Native Computing Foundation
CoT	Chain-of-Trust
CRD	Custom Resource Definition
CRI	Container Runtime Interface
CSP	Cloud Service Provider
CVE	Common Vulnerabilities and Exposures
DAC	Discretionary Access Control
DCAP	Data Center Attestation Primitives
DCT	Docker Content Trust
EDL	Enclave Definition Language
EDP	Enclave Development Platform
EPC	Enclave Page Cache
EPID	Enhanced Privacy ID
GID	Group ID
GKE	Google Kubernetes Engine
IMA	Integrity Measurement Architecture
MAC	Mandatory Access Control
OCI	Open Container Initiative
OPA	Open Policy Agent
OS	Operating System
RoT	Root-of-Trust
PSP	Pod Security Policy
RBAC	Role-Based Access Control
ROP	Return-Oriented Programming
SDK	Software Development Kit
SecL-DC	Security Libraries for Data Center
SEV	Secure Encrypted Virtualization
SGX	Software Guard Extensions
TEE	Trusted Execution Environment
TCB	Trusted Computing Base
TDX	Trust Domain Extensions
TPM	Trusted Platform Module
TXT	Trusted Execution Technology
UID	User ID
VM	Virtual Machine
WAMR	WebAssembly Micro Runtime

1. Introduction

The 5GhOSTS project analyzes and improves the security of service-based implementations of 5G networks, relevant to protect the EU's critical communication infrastructure. Starting from the emerging 3GPP's service-based architecture for 5G networks, which includes virtualization of mobile and core network functions, the project aims to improve the security of virtualization technologies: containers, lightweight Virtual Machines and orchestration frameworks. Unlike previous evolutions in the telecommunications sector, the 5th Generation of Telecommunication Systems (5G) presents diverse and novel requirements for technologies such as heterogeneous air interfaces, Software Defined Networking, Network Functions Virtualization, Mobile Edge Computing and Fog Computing, as well as algorithms to optimize the management of such complex networks. As a result, the 5G evolution will mainly be built on layers of software services. The telecom industry is migrating to virtualized and orchestrated environments, allowing the deployment of Virtual Network Functions (VNFs) on cloud infrastructure enabling the concept of network slicing. The main motivation driver for this move is sustainability through cost and energy reduction as well as full automation of telecom systems operations facilitating dynamic and scalable adaptation to service demands. State-of-the-art light-weight virtualization and container orchestration frameworks clearly contribute to this driver, but do not meet the stringent security requirements of telecommunication systems.

Together with 5GhOSTS D1.4, "Privacy Requirements for 5G Telecom Systems Running Virtualized Multi-Component Services" [3], this deliverable develops as threat model and trust model for the deployment of Virtual Network Functions (VNFs) in 5G networks. We then review the state-of-the-art regarding secure and privacy-preserving techniques to implement VNFs, involving the 5GhOSTS use cases from 5GhOSTS D1.2 [1], and taking our base orchestration framework from 5GhOSTS D3.1 [2] as an example. This work is thus based on the main results of the 5GhOSTS requirements analysis, which revealed three crucial requirements to be considered: Firstly, we emphasize on low-latency and resilience requirements the necessitate distributed deployment of container-based VNFs across multiple data networks in the edge, hence a study of existing Kubernetes-based edge platforms for 4/5G has been commenced. Secondly to facilitate security and privacy threat modelling by both researchers in law and computer science, a system model has been created at two levels of abstraction. At the highest level, existing standards and relevant grey literature have been incarnated into an integrated model consisting of the application, platform and network layers; at the second level, a Kubernetes-specific model of the application and platform layers has been created. Thirdly, a categorization of security threats according to three security challenges (i.e., life-cycle-management of container images, inter-container communication and container storage) identifies which threats are the most relevant and the most pervasive.

1.1. Use Cases

In 5GhOSTS D1.2 [1] we define the scope and focus of our research to be on 5G Edge Computing. 5G Edge refers to an approach to distributed computing that emphasizes on locating applications and the general-purpose compute, storage, and associated switching and control functions needed to run them relatively close to end users and/or IoT endpoints. In the context of ETSI, the term Multi-edge computing (MEC) is also used. Edge computing offers a service environment with ultra-low latency and high-bandwidth as well as direct access to real-time network information enabling applications to be context enriched and hence embellished to offer context-related services for the user or for the network provider. This allows local content, services and applications to be accelerated, increasing their responsiveness and performance. As we will study the balance between low latency communication and security operations, we focus on three applications: Vehicle to Infrastructure (V2I), Smart Office (SO), including augmented and virtual reality (AR/VR) applications, and Remote Surgery (RS).

1.1.1. Vehicle to Infrastructure (V2I)

In this use case, V2I applications transmits messages containing V2I application information that relate to a traffic situation to a roadside unit. Ultra-low end-to-end latency will be a challenge for functionalities like warning signals and traffic flow optimization, whereas higher data rates will be required to share video information between cars and infrastructure. Edge computing infrastructure, located at the edge of the network – and typically at network aggregation points – allows direct communication from vehicle to vehicle, infrastructure and pedestrians, by using the mobile operators' networks

to complement centralized clouds with distributed edge clouds. This provides support for low-latency or location based V2I services possible such as hyper-local HD roadmaps, optimized side-link congestion control, low-latency V2V relays, location-based analytics, and more.

1.1.2. Smart Office (SO)

This application focuses on services that need high-speed execution of bandwidth-intensive applications, processing of a vast amount of data in a cloud, and instant communication by video. Ultra-high traffic volume, and for some applications latency, are the main challenges applicable for this application in addition to instant video communication requiring a secure enterprise environment/localized content. SO targets a clear and usually well-controlled set of subscribers inherently localized within the premises of the enterprise and/or where the core operations are carried out.

1.1.3. Remote Surgery (RS)

Remote surgery applications have a traffic behavior characterized by irregular bursts of data where request response delays are very critical. Edge computing seeks to optimize the response time to have remote surgery services transferred with as short a delay as possible. The main challenges for this application is to guarantee high service reliability and a low latency in service response time for delay-critical applications.

1.2. Privacy and data protection in 5G

In [3], we provide a comprehensive analysis of potential legal challenges to privacy, data protection, and security in the 5G value chain running virtualized multi-component services based on three use cases, and develop privacy requirements for such services. Specifically, we describe and analyze technological novelties relevant to the challenges to privacy, security, and data protection in the 5G core network. Major privacy-disrupting technological novelties in 5G are assessed and the applicable legal framework is outlined, providing a legal qualification of roles and responsibilities of stakeholders, and discussing the influence of novel technologies on compliance with EU law. In [3]. we further examine the scope of applicable laws at the EU level, namely that of the ePrivacy Directive, European Electronic Communications Code, and GDPR, as well as the potential impact of the draft ePrivacy Regulation. The deliverable identifies and explains the potential risks to user's rights to security, privacy, and data protection in 5G networks based on the use cases, and analyzes and suggests possible techniques, strategies, and best practices to mitigate the legal risks regarding privacy and data protection in 5G. Potential legal challenges arise from the implementation of SDN and network function virtualization (NFV), introducing new stakeholders in 5G networks, usage of cloud-based solutions, the status of electronic communication service providers, and the problem of identifying processors and controllers of personal databases on GDPR requirements. A comprehensive summary of the legal requirements and challenges regarding privacy and data protection is provided in section 2.1.

Summary and Outline Based on our security and privacy requirements, this deliverable explicates the threat model and trust model for 5G deployments, and outlines the technical implications and scope of 5GhOSTS. As we illustrate, 5G edge computing solutions face a multitude of challenges due to security threats and data protection regulations. Consequently a need arises for technology to, on the one hand thwarts prevalent threats to security and privacy, but on the other hand simplifies the enforcement and audit of compliance to privacy requirements. Any such solution additionally must have low impact on performance, so as to not hinder low-latency edge applications.

We review state-of-the-art techniques to guarantee secure distribution and integrity protection of VNFs, based on a refined thread model for multi-tenancy in Kubernetes clusters. Our survey emphasizes on techniques that employ modern concepts in the context of Confidential Computing, Trusted Computing, and Trusted Execution, which are capable of providing extended security and privacy protection in future highly dynamic 5G scenarios.

Our analysis opens up four exiting and interconnected directions of research, involving the integration of novel techniques for intra-container isolation and attestation, performance and network security, and approaches to privacy impact assessment, which will be researched by the 5GhOSTS consortium.

2. Privacy, Security, and Trust in 5G Networks

In this section we provide a summary of our Deliverable D1.4 [3], outlining the results of our analysis of potential legal challenges to privacy, data protection, and security in the 5G value chain running virtualized multi-component services. Please refer to that deliverable [3] for a comprehensive discussion of our methodology and further references. We then develop a trust model and specific threat models that reflect our security and privacy requirements.

2.1. Privacy Requirements

The implementation of virtualization, SDN, NFV, and wider reliance on IP and cloud technologies as the primary means of data transportation creates a plethora of privacy challenges in 5G networks. 3GPP, an international standardization body, defines the 5G system architecture as the one consisting of a core network and a radio access network (RAN). RAN enables radio communication between the network and user's devices, such as mobile phones. The core network handles the transportation of the data and contains network functions (NF). Important virtualization-related novelties lie in the core network, including the application of network slicing technology which contributes to network optimization. In 5GhOSTS we concentrate on the core network since the majority of disrupting technologies are contained there. 5G may become the encompassing channel of communication for nearly all data and create a situation where almost all individuals will be using 5G network through one of their devices.

Related research, as we elaborate in [3], has identified following classification of privacy categories in 5G networks:

Data privacy: incorporates all data generated and transferred by the consumers through 5G mobile networks. Special attention is paid to the protection of sensitive personal data, such as health or biometric data;

Location privacy: the above-mentioned omnipresence of 5G-enabled sensors and devices allows the precise geolocation and tracking of users. Such tracking can be performed by various actors in 5G networks, such as third-party application developers, cloud and edge computing service providers, and telecommunication providers themselves, among others;

Identity privacy: since 5G networks involve an enormous number of interconnected devices and their users, it is essential to ensure the protection of subscribers' identities. Subscriber's identity includes the usage of International Mobile Subscriber Identity (IMSI – used for identification and location of subscribers) by user's device and the risk of exposing the IMSI to potential attackers. Identity also includes the profile data used in various third-party apps, most importantly healthcare, fitness, banking and shopping apps, which gather and store large amounts of sensitive data.

2.1.1. EU legal and policy framework applicable to 5G

With the General Data Protection Regulation, European Electronic Communications Code and ePrivacy Directive, the rules on data protection and electronic communications in Member States have been significantly harmonized. These laws serve as a lex specialis for the processing of personal data and regulation of electronic communications. The decisions of the Court of Justice of the European Union, opinions and guidelines of Article 29 Working Party European Data Protection Supervisor, and BEREC (Body of European Regulators for Electronic Communications) affect the implementation and application of privacy, security and data protection legislation and thus are crucial for developing privacy requirements for 5G networks.

When developing privacy requirements for 5G Telecom Systems running virtualized multi-component services the legal provisions related to protecting user's privacy and personal data protection must be certainly paid exceptional attention. Considering various types and categories of data to be transferred through 5G networks and the technical challenges to distinguish between the types and categories of data the privacy requirements must be developed with keeping in mind the possibility that any piece of data transferred via 5G networks may be considered personal data. The application of EU data protection and privacy rules in mobile communications is a broad topic. Data protection and privacy in the EU is regulated primarily by the GDPR as lex generalis. However, when it comes to 5G networks, there is a number of other relevant legal acts to be assessed, such as the European Electronic Communications Code.

In 5G networks various third parties will be involved in processing personal data such as CSPs and third-party application developers. How to coordinate between and balance the competing interests of these actors is one of the most important legal questions to be addressed in the context of data protection and privacy of individuals, especially in light of the data protection by design obligation in Article 25 GDPR. Different laws – both national and European – may apply depending on the exact legal qualification of stakeholders, thus there is a need for legal certainty, which includes stability, consistency, and clarity of the law. Insufficient legal safeguards to activities of new stakeholders in mobile networks also raise legal issues related to loss of control of data, loss of data ownership, and ambiguity of liability of processors and controllers of personal data. These challenges may greatly affect the protection of user's rights with regard to the privacy and personal data.

Transferring personal data via cloud-based means and virtualized networks results in a complicated technical environment with multiple stakeholders who pursue different interests and apply different security standards. SDN providers, together with Cloud Service Providers (CSPs) become important stakeholders on the 5G Telecom market and will process personal data of end-users, as well as other data – such as machine-to-machine communications – which may contain personal data of other individuals.

We discuss the implications of the EU legal framework, specifically, privacy as a human and fundamental right, initial European efforts in harmonization of privacy legislation, the GDPR, and the ePrivacy Directive, in the 5GhOSTS Deliverable D1.3 [3].

2.1.2. Implementing privacy, security and data protection risk mitigating measures in 5G networks

Following our analysis, privacy, security and data protection in virtualized 5G networks largely depends on actors operating in 5G value chain. Their decisions, management practices, procedures applied will have a decisive impact on the functioning of 5G networks. Since some of the security and privacy requirements issued by 3GPP are not applied by default or are optional, the overall effectiveness of security depends on to what extent the operators of 5G networks will enforce these standards. Anonymization, pseudonymization and implementation of data protection by design contribute to the GDPR-compliant security and privacy in 5G networks. The personal data processed via 5G networks may prove to have a high commercial value. The new cloud, edge computing, virtualization and software-reliant generation of mobile networks is vulnerable to software bugs, data leaks, lack of coordination between different parties involved in the processing of personal data, and cyber attacks.

Implementation of effective privacy preserving technologies, data protection measures and security standards is not just a requirement of law but a necessity enabling the operation of 5G network as a whole. As was noted by S. Rizou, E. Alexandropoulou-Egyptiadou and K. E. Psannis "the scope of privacy protection would be not only the effort to avoid the administrative fines of millions of euro, but to establish from the beginning of 5G technology, a fair integrated treatment for data protection rights" [6]. Adherence to GDPR requirements namely to principles relating to processing personal data, principle of data protection by design and by default is crucial for ensuring the privacy and data protection. Users must be able to easily obtain full and clear information on the processing of personal data, the data controllers, purposes of the processing, profiling and automated decision-making, and the ways to exercise their rights to rectification and erasure of data. Considering the use of edge computing and cloud services and the option of distributed processing of personal data in 5G networks, traceability mechanisms have to be implemented, which may prove to be a substantial challenge. The fact that 5G contributes to the generation of an immense amount of personal data highlights the importance of effective data minimization measures.

There is no exhaustive list of actions or privacy-preserving technologies necessary to achieve compliance with legal requirements. The privacy-disruptive novelties introduced by 5G networks require the development of new and efficient privacy enhancing and privacy preserving technologies to cope with the growing among of data and interoperability issues. Researchers have proposed a number of technical and organizational measures as well as privacy risk mitigation techniques to enhance privacy, protection of data and security in 5G networks.

It has been noted that existing trust models may not be fully suitable for 5G networks, and the development of new models is required, which consider the presence of many novel actors and vendors providing various services in 5G. Some of the network actors may use external cloud services and it is important to prevent unlawful cross-border transfer of data. Identity management in 5G would also change due to reliance on many small IoT devices that are not capable of bearing a standard identity module. Identity management is important to protect the security of the network as well as personal data stored on the devices themselves. Compliance with principles of lawfulness, fairness and transparency will ensure that users' data is not processed and is not transferred to third parties without the user's informed consent. Researchers note the potential conflict between privacy and computing trust.

2.1.3. Conclusions

In summary, 5G networks highlight the importance of the development of authorized and regulated certification mechanisms envisaged by GDPR. The success of 5G networks depends on its ability to guarantee security, privacy and protection of personal data. While there is a need to balance compliance with legislative provisions and over-protection of the network, it is essential to apply rigorous assessment of any privacy-preserving technologies and techniques to ensure the protection of users' rights. The GDPR was designed as a risk-based future-proof legislation to tackle not-yet-known threats to privacy and data protection. 5G however, poses a substantial privacy-disruption threat if not implemented in the correct way, namely by applying the principle of data protection by design and by default in all steps, including research, standardization, development, deployment and operations of the 5G network. Some researchers argue that GDPR may prove inadequate to address the threats to privacy and users' right posed by the Internet of Everything and an 'always connected' society led by the commercialization of personal data. The deployment of 5G infrastructure will likely require a re-assessment of trust relationships within the-network, which then leads to the development of new, comprehensive, and use-case-sensitive trust models which would be the technical basis for Data Protection Impact Assessment (DPIA). In the following section of this deliverable D1.3, we develop the trust model and define specific threat models that such impact assessments can be based on.

2.2. Threat Model

The privacy requirements outlined above are founded on a strong security posture of the 5G network in general and edge computing platforms in particular. The goal of this section is to analyze security threats for a specific 5G use case and thus lay the ground work for the trust model underlying the 5GhOSTS project.

In 1.1, we summarize the identified three 5G Ultra-Reliable Low-Latency Communications (URLLC) applications and use cases, i.e., Virtual/Augmented Reality (VR/AR), Vehicle to Infrastructure (V2I), and Remote Surgery (RS), poised to induce a significant surge in demand for both computational and communication resources.

The new use cases contain new relationships and reveal new risks that need to be understood and controlled. It refers to the need for a trust model that can describe the system, identify the assets, perform threat analysis. This deliverable begins by briefly describing the Intelligent Transportation System and its components. It focuses on the V2I scenario and explains how the actors interact with each other. Then a STRIDE analysis is performed to identify possible threats and possible mitigations against them. Based on this analysis we highlight threats of particular relevance for the privacy requirements and define a trust model as a basis for future work.

2.2.1. Describing the Scenario and Trust Relations Analysis

To simplify the threat modeling activity, this work assumes a smart road environment as a practical edge computing use case. All the involved actors are listed below. We assume here that a Mobile Network Operator (MNO) supplies the physical and virtual infrastructure for both 5G mobile network communication and edge computing capabilities. The edge platform itself is controlled by an edge manager and it hosts a number of services to end users developed and deployed by edge application providers. The user side communicates with these applications on the MEC platform via the mobile network. Attacker are assumed to be able to interact with the all of the other components in a direct or indirect fashion.

Actor	Description / Examples
Mobile Network Operator (Infrastructure Provider)	Verizon, AT&T, T-Mobile, British Telecom, Vodafone, etc., potentially in cooperation with cloud providers (MS Azure, Amazon AWS, Google Cloud, IBM)
Edge Manager	Management and Orchestration of Service Providers
Edge Application Provider	*Traffic Management:* collecting of all information from car, traffic sign, pedestrian and giving the instruction to them *Insurance Company:* store all the driver information health driver license, collecting from driver preference and analysis the who is guilty in case of damage *Localization Service:* provide real-time map and traffic data, as well as localized information and services

Actor	Description / Examples
Edge Application Users	*Smart Cars:* GPS, LIDAR, Video Camera, Radar. the smart cars have a lot of sensor radar and camera environment. They give a lot of information to the MEC in real-time, and they need to output the information continuously in order to prevent any incidents on the road. *Pedestrians:* collecting traffic information from the service provider, e.g., for augmented reality application. *Roadside IoT Devices:* Traffic lights and signs, cameras, sensors (e.g., electromagnetic, optical, acoustic, air quality, weather)
Attacker	They may target all the actors in the scenario, their typical motivations and goals include: • Stealing sensitive end user information (PII) • Disrupting critical infrastructure and services • Gaining control of the system to ransom it • Deploying illicit software on the computing infrastructure (DDoS clients, Crypto currency miners)

2.2.2. Identifying Assets

An asset is anything that has value to an individual or organization and therefore requires protection. A good security posture should focus on protecting valuable data and infrastructure, as it is no use to waste effort protecting non-valuable things. In order to determine all relevant assets to protect, it is not enough though to just focus on valuable targets of the attacker.Typically, successful attacks consist of stages where certain information, e.g., credentials, serve as stepping step stones to compromise the desired system assets. Hence one of the main goals of thread modeling is to identify all the possible avenues for attackers to achieve their objectives. The process process can then be applied recursively to all step stone assets discovered earlier.

For the V2I Edge Computing Platform we identified a large number of assets in this way. We categorize them into six categories as shown in Fig. 2.1: Edge Platform Provider, Edge Application Provider, Management and Orchestration (MANO), MNO, Edge Computing and Communication Infrastructure, as well as assets out of scope of this threat modeling process. We list all identified assets below.

Asset Group	Examples	Explanation
Application Software	• Container Image • Machine Code running in container (Execution context) – code – files – stack – registers (incl. instruction pointer, stack pointer) – static variables – heap memory • Container Image distribution infrastructure	Application Software covers what is required for an application to run on the Edge platform. The software itself could be a traditional program or a machine learning model, e.g., used in an intelligent transport system. Container images are pulled from the Container Image distribution infrastructure and run in the container. The term "Execution context" refers to the environment for running machine code on a computer's instruction set architecture. It includes code, files, stack, heap, static variables, and registers.
Life Cycle Management Proxy (LCM)	LCM Proxy for user applications	The user application life-cycle management proxy allows device applications to request on-boarding, instantiation, termination of user applications and when supported, relocation of user applications in and out of the MEC system.
Edge Cloud Services	• Edge Service API • Service Registry • Traffic Rules Control • DNS handling	Edge Cloud Services are responsible for the communication within the edge cloud, managing the edge services, and traffic rules control. Service registry monitors the edge application life-cycle.

Asset Group	Examples	Explanation
5G Core Functions	• AMF (Access and Mobility Management Function) • SMF (Session Management Function)	5G core functions manage the edge platform and user plane communication requirements in synchronization with the mobile network core component. AMF is responsible for handling connection, mobility management, access authorization and authentication. SMF is responsible for User Equipment IP address allocation and management.
Edge Computing and Communication Infrastructure	• Virtualization Infrastructure • VIM (Virtualization Infrastructure Manager) • gNB (gNodeB, 5G base station) • UPF (User Plane Function) • Mobile Network	This asset group comprises computing and communication infrastructure, e.g. software system of gNB, virtualization. The User Plane Function is responsible for packet routing and forwarding to the Edge Cloud and Mobile network. It allows users to communicate with edge applications.
Physical System	• Computing, Network, Storage hardware of edge platform • Antennas / Base station hardware • Cooling system • Energy / Power supply	This is the physical hardware and its support systems that make up the edge computing infrastructure. Antennas / base stations are part of the 5G mobile network and allow end users connect to the edge platform.
Stored Data	Application data: • Keys, credentials • Sensor data • Machine learning models • End User Data (Personally Identifiable information, PII): – User identifiers and addresses – User payment information – User geo-location – User health information, e.g., sleepiness, medical problems – User preferences, e.g., frequent routes, home, work address Control plane data: • `etcd` • Edge Service data: – Service registry – Connection data – Traffic rules and DNS configuration 5G service data: • Connection and session data • IP allocation	An Edge platform stores a multitude of different data. For applications, e.g., IoT devices send sensor data to the edge platform to be analyzed immediately or be used for training later on, e.g., in machine learning models. Such models and their weights must also be stored. For the ITS, users can share private information (PII) to edge platforms such as geolocation, driver health information, their current route. These should be stored securely and are subject to data protection regulations. Many apps require secrets to enable secure communication between components such as connection strings, SSH private keys, and X.509 private keys and stored in edge platform. Besides such application data, the edge platform must also provide storage for control plane, edge service, and 5G core function data. `etcd` is the distributed key-value store used to hold and manage the critical information of cluster.

Asset Group	Examples	Explanation
Log Information	Application logs: • Monitoring policy violations • Performance monitoring (Response Time, Processing Time) • Authentication successes and failures • Session management failures • Application errors • Application code, file, and/or memory changes • Suspicious behavior Kubernetes Logs: • Cluster Logs • Events Logs • Audit Logs	There are two main categories of logs in the edge platform, application logs and Kubernetes logs. The application logs records what happens in the container runtime. There are many policies in place such as network policy or image policy. If third any policy violations occur, those should be recorded. Suspicious behavior includes events such as application runtime errors, connectivity problems, performance issues, third party service error messages, file system errors, file upload virus detection, and configuration changes, all of which which should be logged. Kubernetes logs are more related to the control plane of the edge platform. Kubernetes Cluster Logs are responsible for `kube-apiserver`, `kube-scheduler`, `etcd`, `kube-proxy`, Kubelet logs. Kubernetes Events Logs monitor what happens in the cluster such as a container being created or starting or errors such as the exhaustion of resources. Kubernetes Audit Log record who or what issued API requests, what the request was for, and the result.
System Resources (provided by host OS or HW)	• System Counter and timers • Network Date and Time • Random Number Generators • Virtualization support • HW crypto support • Enclaves / TEEs • HW accelerators (GPU, FPGA, NPU, TPU, NIC, etc.) • System calls and drivers	System Resources include the hardware functionality that needs to be configured correctly by the edge platform provider to provide virtualized cloud computing capabilities. Moreover, they include hardware resources offered to edge applications. For example, a trusted execution environment, such as enclaves, can be offered by the Edge platform. Also, for machine learning applications hardware acceleration may be provided using GPUs, FPGAs, or similar.
Management and Orchestration	• Admin credentials • Orchestration scripts, e.g., helm charts • Pod specifications • Image policies • Network policies • RBAC roles and policies • Admission controllers • Pod security policies • ResourceQuota and LimitRange • Service Level Agreements • Service Billing System	Management and Orchestration services manage the edge platform by configuring network and system resources, and determine different policies for the operation of the cloud platform. Image policies specify which images are allowed to be run on our cluster. Network policies specify how groups of pods can communicate with each other and with other network endpoints. RBAC roles and policies are used to assign access to a computer or network resources in the Kubernetes Cluster. ResourceQuota is a object provides constraints that limit aggregate resource consumption per namespace. It covers Compute Resource Quata Storage Resource Quota, Object Count Quota. Service level agreement is an agreement between the service provider and the service user that describes how services should be executed and its execution terms. SLA should state the amount of memory consumption (storage), estimated execution time, maintenance duration and processing cost. A Service Billing System is used to record the usage of edge resources by the application providers in order to charge them accordingly.
Network	Inter-Container Network (Data Plane)	It covers all communication between edge applications and with end users.
	Control Plane Network	It covers to all communication necessary to run the k8s cluster, e.g., between kubelets and the API server, or for fetching images. Also MANO communication and communication between 5G core functions and the core network are considered part of the control plane.
Out of Scope	• External Configuration Data • Software Developers for the Edge Service Application • Edge platform admins and operators • Cluster Administrator • Application Manager	We do not consider assets for our threat model that are external to the edge platform itself, as we cannot take precautions within the platform to protect their security. For example the current date and time or position of the edge platform are usually provided by external sources (e.g., GPS, internet time) or set manually by the infrastructure provider. While this information can be protected within the platform, we have to trust the external information and do not consider how to validate its authenticity. Also we do not consider the personnel operating the edge platform as assets here, hence threats like social engineering attacks are out of scope as well.

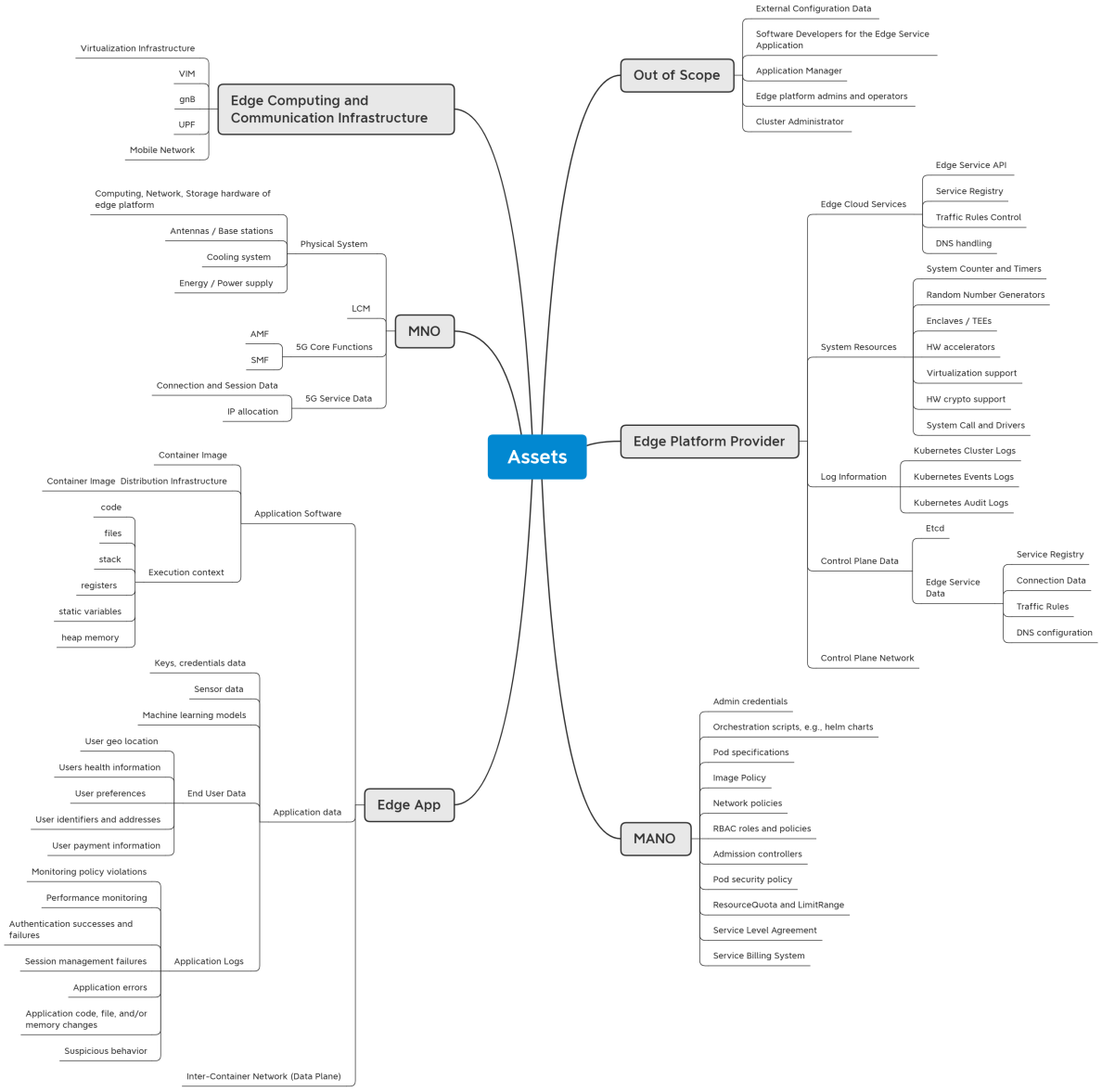

Figure 2.1: Assets of the V2I Edge Computing Platform

2.2.3. Data Flow Diagram

Data-flow diagrams (DFDs) are graphical representations of the systems and should specify each element, their interactions and helpful context. DFDs consist of elements (data stores and processes) connected by data flows, interacting with external entities (those outside the developer's or the organization's control). We considered a Kubernetes-based edge platform that is hosted on virtualization and communication infrastructure provided by the MNO. The corresponding data flow diagram is shown in Figure 2.2. It shows the assets listed above and their relations with each other as follows:

- Data stores represent files, databases, registry keys, and the like.
- Processes are computations or programs run by a computer.
- Managed Entities are separate collection of processes and data, managed by an actor in their entirety
- External entity represents user and operator side.
- A trust boundary is a location in the DFD where data changes its level of trust.

While the DFD should be largely self-explanatory given the description of the assets above, we still want to highlight some facts about the V2I edge computing platform DFD:

- Shared Services and 5G Core Services communicate with all other components, as symbolized by the four connected data flow arrows.
- We assume a zero-trust policy for all edge applications and services hence all pods and services are within separate trust boundaries.
- General purpose storage and system resources are depicted as shared services outside of the trust boundaries of the services and applications that use them.
- It is an important question how edge applications communicate with each other. While the platform offers the service registry as a broker of communication between different services, proxied communication may break low-latency requirements. It is therefore conceivable that pods of different edge applications are permitted to communicate with each other directly, if only temporarily.
- The Kubernetes Control plane is not explicitly shown in the DFD but is part of the MANO process. Its configuration artifacts and control state are stored in the corresponding storage node which covers `etcd` and other databases used by management and orchestration.
- The virtualization infrastructure needs to trust its manager VIM and the k8s cluster trusts MANO commands, hence in the DFD they are within the same trust boundary, respectively.
- Since the Edge Platform Provider may not fully trust the Infrastructure Provider, there is a trust boundary between the edge platform and the infrastructure layer. However, the Edge Platform Admin still needs access to both VIM and MANO to efficiently operate the edge platform.
- It could be conceivable that different actors are responsible for providing the virtualization infrastructure for the edge cloud on one hand and the 5G mobile network on the other. For simplification, we assume here that the MNO provides both.

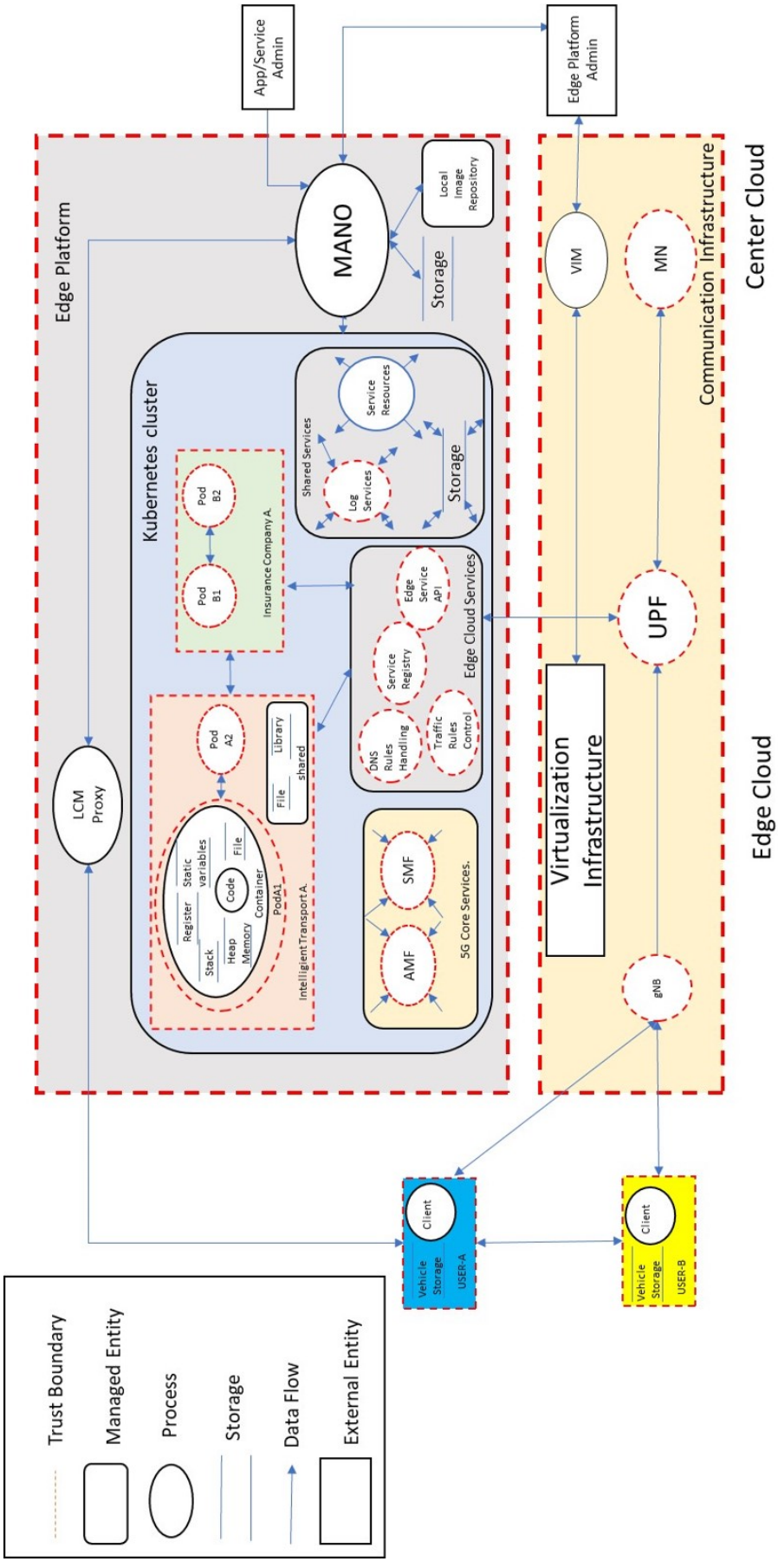

Figure 2.2: Data Flow Diagram

2.2.4. Identifying Threats

Threat modeling is a structured approach of identifying and prioritizing potential threats. We used the STRIDE method to perform a threat analysis for the V2I scenario. STRIDE is an acronym that stands for Spoofing, Tampering, Repudiation, InformationDisclosure, Denial of Service, and Elevation of Privilege. The STRIDE threats are the opposite of some of the properties we would like our system to have:authenticity, integrity, non-repudiation, confidentiality, availability, and authorization respectively.

The full result of the STRIDE analysis can be found in the Appendix. Below we highlight threats to the Edge Application and the Edge Platform that we deem most relevant in the context of 5GhOSTS. Possible technical solutions to counter these threats are discussed subsequently and listed in the appendix as well.

2.2.4.1. Selected Threats to Edge Application

We first highlight threats to applications running on the edge. As these edge services handle end user data, their security is of utmost importance for data protection and privacy. It has to be ensured that data is being processed only in ways that have been consented to. To this end it is crucial to ensure that only approved applications running in trustworthy execution contexts get access to the user data. This notion of application and system integrity and authenticity is being threatened by the following issues.

SpoofingImage The identity of a service image fetched from the local image repository can be spoofed, e.g., by presenting fake tags, replacing the image in repository or image cache, or by intercepting it during transmission. This could occur due to a privileged attacker with control plane access, or by compromising the image repository, e.g., due to lack of proper authentications. Consequently, a compromised image may be pulled instead of the desired one, containing vulnerabilities or even malware.

SpoofingSysResource The compromised edge cloud provider can provide broken or fake enclaves to the app by obtaining attestation keys, e.g., via Plundervolt [7] or transient execution attacks like Foreshadow or SGXpectre [8].

TamperingImage A container image is maliciously modified, customized, or replaced in the time between creation and deployment. This could occur due to an external attacker intercepting transmission of the image or compromising the image repository. A compromised image repository may deploy outdated versions of a given image, potentially undoing patches and fixes of past vulnerabilities, thus re-enabling the exploitation of previously discovered software security flaws in the application. Another possibility is a compromised edge cloud orchestrater, purposefully deploying the wrong, potentially malicious images that may strive to modify sensitive information or attempt to attack its users. Attackers with sufficient admin privileges may configure malicious admission controllers that infect deployed images via mutating web-hooks.

TamperingExecContext The majority of software security vulnerabilities are memory safety violations, i.e., programming errors that break the memory abstraction of unsafe high-level programming languages such as C. Examples for such errors are buffer overflows, use after free, memory race conditions, uninitialized variables and null-pointer references. Memory safety violations may crash a program or lead to unexpected changes in program behavior. In the worst case they may enable remote code execution and thus break system integrity. Such vulnerabilities allow attackers to manipulate the control flow of applications by providing it with specially crafted input values. Usually, affecting the control flow requires a memory safety violation first, i.e., overwriting executable code or function pointers, smashing the stack to overwrite return addresses (return-oriented programming, ROP), or corrupting application data to change the outcome of conditional program branches (data-oriented programming, DOP). Rowhammer and code injection attacks can also allow manipulation of the execution context. Achieving remote code execution within a container of an application allows an attacker to send requests to other services or the k8s API server on behalf of it.

TamperingSysResource On a host computation node the edge cloud provider or an application with elevated privilege may sneakily modify the virtual environment provided to the application, subverting its assumptions that underlie security-related functionality. For instance, address space and file management could be manipulated to slightly change an application's behavior. The OS could provide a faulty system time or counters, compromising functionality based on having a precise notion of time and date. Finally, a privileged attacker could provide fake crypto primitives, fake enclaves, or weak entropy random numbers to weaken cryptography and thus downgrade application security.

Instead of attacking the data processing architecture attackers may also target user data directly. The following threats

have the potential to violate data integrity which is mandated by data protection regulations like GDPR.

TamperingAppDataUser This threat has two threat vectors: data in use and data at rest.

- *Data in use:* Software vulnerabilities may allow attackers to tamper with an application's execution contexts or even take control of its control flow and privileges by malicious parties. Subsequently the attacker may obtain a write primitive and modify mission-critical or sensitive data application data including personal data that is being processed. Through its exposed API an application may also be targeted by confused deputy attacks, attempting to trick it into using its privileges to tamper with the desired data. Also, Rowhammer attacks can corrupt application data in RAM on the same node.
- *Data at rest:* If there are applications that require root privileges on the node or if an application can get privileges due to vulnerabilities on the host, these apps may replace stored data of other applications with former versions of it as well as tamper with the data. Depending on the application this may enable security exploits if the authenticity (freshness) of the data is not checked.

TamperingAppDataAdmin The edge cloud provider has write privileges on all storage nodes for the edge application and can thus manipulate app data at rest. Integrity protection helps to discover such manipulations but cannot guarantee the availability of data. Nevertheless, a cloud provider has usually a self-interest to provide reliable and robust data storage.

Besides data integrity, also the confidentiality of personal information and other application data must be ensured. Attackers may attempt to leak end user data directly or via the stepping stone of credentials to gain unauthorized access data stored in the edge cloud, in direct violation of data protection requirements.

InfoDisclExecContextUser Any software security vulnerability (such as control-flow hijacking, remote code execution, memory safety violation) may lead to the disclosure of sensitive application data such as cryptographic keys, if exploited by an attacker. Also Memory Reuse, Side-channel, and Confused Deputy attacks, among others, may cause the leak of information to co-located malicious applications or external attackers. Using third-party libraries in the software may contribute to such threats of the execution context.

InfoDisclExecContextAdmin As the Edge cloud provider holds a privileged access in the node, it can dump application data in use from RAM and registers. Memory dump analysis can then be used to extract important information from these dumps.

InfoDisclAppDataUser Many apps require secrets to enable secure communication between components such as include connection strings and SSH private keys. If these secrets are stored in image or configuration files, anyone with access to these files can easily parse it to learn these secrets. Also, if a comprised app obtained a memory read primitive on a host or storage node, it could be able to access confidential data of other apps.

InfoDisclAppDataAdmin An edge cloud provider may typically get access to any unencrypted information stored on disks and simply copy such data. Even if disks are encrypted, backups and system logs of running applications may contain decrypted information which may thus be available to honest but curious infrastructure providers.

ElevPrivAppData Because of the weak config of authorization (RBAC exploited by app admin, application admin can add some roles/tag etc), the app can get privileged access to application data in storage nodes.

Finally, we point out two more threats posed by weaknesses in application design.

RepudationApplog For different reasons, such as privacy regulations and intrusion detection, it is necessary to keep logs of all accesses to a storage node. A compromised application may try to falsify such logs in order to hide any previous transgressions.

TamperingDataPlane Any compromised edge orchestrater may strive to weaken network configurations and isolation, e.g., redirect traffic to unauthorized recipients. Also, by default, inter-pod communication is not authenticated, encrypted, or integrity-protected, hence a privileged attacker could stage man-in-the-middle attacks. The design of network services, functions, and other edge cloud applications should follow a rigorous security design, featuring principles of compartmentalization, defense in depth, least privilege, and zero trust. If a design violates these principles, vulnerabilities in one part of an application may allow an attacker to spread to other parts of the edge platform.

The repudiation threat interferes with data protection regulations that mandate that all data processing should be auditable. The tampering threat highlights that different edge applications must not blindly trust one another and should be isolated from each other to avoid the compromise of data or data processing.

2.2.4.2. Selected Threats to Edge Platform

While our main focus is on the protection of edge applications themselves, the security of the edge platform is the base on which application security is built. In the appendix we provide the full threat model, here we just present a small selection of threats.

From the previous section it should be obvious that the edge cloud provider holds powerful privileges that might threaten privacy and user data protection if they were to be abused. Therefore it is crucial to prevent attackers from obtaining any of these privileges as illustrated by the following threat vectors.

ElevPrivExecContext Excessive privileges given to application containers, unsafe configurations like writable hostPath mounts, as well as vulnerabilities in the virtualization infrastructure, operating system, and the container runtime may allow a malicious application to escape its sandbox and acquire root privileges on the node. Such privileges may allow the attacker to spread to other parts of the edge cloud system. Subsequently also data and service integrity, confidentiality, and availability may be threatened.

ElevPrivMANO Compromised apps or external attackers may be able to assign privileged roles to themselves in the cluster due to weak design, e.g., having permissions to create arbitrary role bindings, or in collusion with a tenant admin. Also, getting access to a pod's service account may allow attackers to perform actions using that pod's privileges.

Edge cloud providers also have an accountability for the processing happening on their platform. Therefore logging and auditing capabilities must be in place. However their integrity is threatened by the following issues.

RepudiationLogs Compromised edge applications or external attackers can try to take actions to prevent logs from being useful, including filling up the log to make it hard to find an attack or forcing logs to "roll over". They may also do things to set off so many alarms that the real attack is lost in a deluge of noise. Sending logs over a network exposes them integrity threats as well. The tampering of logs is a step stone for repudiation of (formerly) logged events.

RepudiationMANO A privileged attacker with admin access may reconfigure the MANO and disable or clear the event logs.

Having outlined threats to the edge application and edge cloud provider, we can now define our trust model for the V2I edge cloud platform.

2.3. Trust Model

Below we consider the notion of trust between different actors in the V2I system. However, we do not want to use binary system trust relations like trusted and untrusted between our actors, as this notion seems too simplistic and does not take into account expected behaviors and acceptance of risks. Instead, we define four levels of trust as follows:

- Distrusted: Not trusted at all, considered malicious or with harmful intentions
- Untrusted: Unclear if malicious or benign, one must not rely on any of their behavior and only grant restricted access to services and data. This is usually how service providers view their users.
- Semi-trusted: This is similar to the honest-but-curious attacker model, i.e., we do not expect active attacks from such an actor but passive attacks like eavesdropping, dumping or inferring data, etc., may be expected. At the same time there is some risk of the actor becoming compromised or being hacked, i.e., they become untrusted. However they are trusted to at least provide availability of the system according to their best efforts.
- Trusted: Completely reliable and honest, hence allowed to receive confidential information. With a trusted actor one deems the risk of compromise very low. Thus no countermeasures against such eventualities need to be taken, often because they might also not be very practical or feasible.

Figure 2.3 shows the different actors in the V2I system and their trust relations. The user side covers driver, pedestrian, and the road sensor infrastructure such as traffic lights or cameras. As shown in the diagram, attackers must be considered as distrusted and all systems must take a precaution against attackers. Attackers can easily reach the user side, attempt to spoof other users' identity, or compromise them as a step stone to attack the edge platform. That is why the user side actors do not trust each other. However, the users must semi-trust the edge service provider and mobile network operator, because they depend on a lot of information and functionality from them. For example, in the traffic management service use case, they should share data, and get optimized directions with lower traffic or real-time information about their immediate environment.

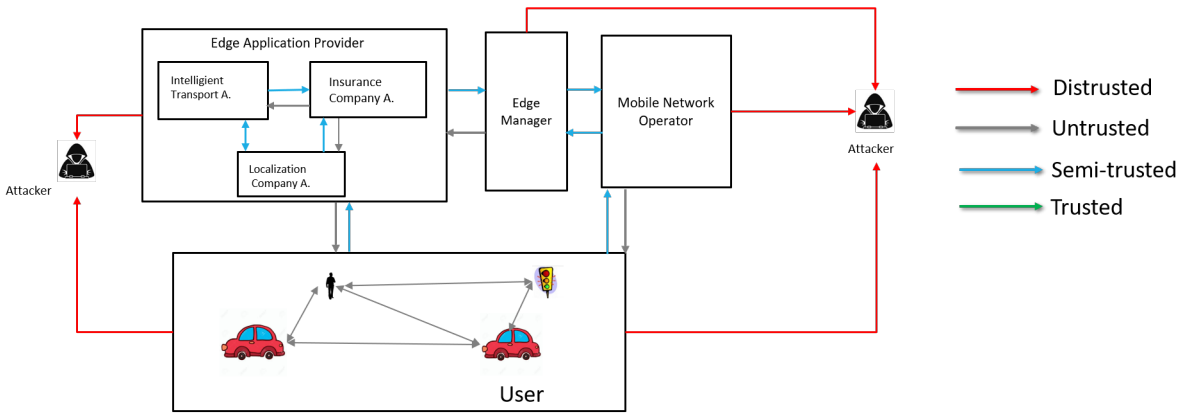

Figure 2.3: Trust model for the V2I edge cloud scenario

Edge application providers store private user information, and they must protect it according to privacy regulations like GDPR. They should not fully trust anyone. As a result, the user side is untrusted by them. However, the provider should semi-trust edge manager as the edge manager is responsible for the configuration and management of the platform and edge services consumed by the applications.

Each edge application decides on a case by case basis whether to trust other applications. The general principle is that every application only trusts its own code. They should follow the zero-trust policy and only trust other applications if they are authenticated and deemed trustworthy after an audit by the application developers.

Mobile network operators distrust the user side. However they semi-trust the edge manager as it is one of a few well-identified customers of the MNO's edge and mobile network infrastructure. If the Edge cloud provider and MNO are the same entity, the trust relation between them could become trusted. Nevertheless, if protections against a compromise of either side are dropped, it may pose a risk if such a compromise actually occurs. Hence there are no completely trusted actors in the V2I edge cloud scenario.

2.4. Technical Implications and Scope of 5GhOSTS

As we have seen in the preceding sections, 5G edge computing solutions face a multitude of challenges due to security threats and data protection regulations. Consequently a need arises for technology to, on the one hand thwarts prevalent threats to security and privacy, but on the other hand simplifies the enforcement and audit of compliance to privacy requirements. Any such solution additionally must have low impact on performance, so as to not hinder low-latency edge applications. Within the 5GhOSTS project we have started to evaluate potential technologies and while many solutions to the security threats exist, the issues of privacy and low latency do not seem to be fully addressed by them yet, opening the possibility for improvements by our research.

To give an outline of the security solutions we again distinguish threats to the application and the edge platform separately. For the former, one of the most prominent technologies are Trusted Execution Environments (TEEs), e.g., Intel SGX enclaves. On one hand this technology supports data protection, in particular privacy, because all data processing occurs in a "black box" that is only accessible to the application handling the data itself. Hence TEEs thwart information disclosure threats both from other applications and the edge platform provider. On the other hand, attestation capabilities allow the application provider to assert that the correct application has been loaded on a trustworthy computing platform, thus thwarting tampering threats to the image or execution platform. We are planning to expand on these benefits by studying runtime attestation schemes that allow to assert dynamically that applications are still in a good state, not having succumbed to runtime attacks abusing memory safety vulnerabilities and the like.

Besides TEEs, also other container runtimes that employ Micro-VMs may serve to limit the exposure to a potentially compromised cloud provider. Firstly, returning to hardware-based virtualization reduces the exposure to the host OS, hence offering less opportunity for tampering or information disclosure. Secondly, novel VM encryption and attestation technologies like AMD SEV or Intel TDX can turn Micro-VMs into TEEs themselves creating an alternative to process-based SGX enclaves. So far it is unclear which of the technologies is the best fit for edge computing applications in terms of security and performance.

Another application security trend that we are following is the increasing support for intra-process resource isolation, a

technique that allows to further compartmentalize an application into mutually distrusting components. Limiting the memory access capability of these components according to the least privilege principle, the impact of security vulnerabilities can be reduced along with the size of the trusted computing base of an application.

One of the main concerns for the edge cloud provider is to prevent privilege escalation of applications, as such could lead to attackers seizing control of the platform. Hence, computing nodes and container runtimes need to be hardened according to best practices. Besides that, secure Micro-VM-based container runtimes like Kata Containers give better protection against privilege escalation due to an additional layer of hardware-based isolation, but may introduce performance overheads compared to bare-metal solutions. Also runtime attestation techniques may help the edge cloud provider to assess the health of applications. However, such approaches must take data protection into account and not allow the cloud provider to learn details about the data processing occurring within the application's execution environment.

Another headache for edge cloud providers is the correct configuration of the Kubernetes cluster to ensure robust tenant isolation. Separation mechanisms like network policies need to be configured in such a way that only legitimate communication between application pods is possible. Formal methods may verify the correct design and implementation of such policies. Furthermore, correct configurations need to be enforced dynamically and protected against tampering by compromised tenant admins. While AI-based techniques for anomaly detection allow to monitor operation and expose transgressions in the cluster, such approaches can only detect ongoing attacks but not prevent them from occurring. Instead, custom admission controllers, e.g., deployed by the Open Policy Agent (OPA), have the potential to restrict the actions of applications and admins according to the least privilege principle. Again, formal methods may help to guide the rule design for such admission controllers.

Having outlined the scope of relevant technologies for the 5GhOSTS project, we will next dive deeper into some of the mentioned solutions. Subsequently we will explore the area of container integrity, focusing on image authenticity, attestation, trusted execution environments and secure container runtimes.

3. Secure Distribution and Integrity Protection of Virtualized Network Functions

To avoid having to re-write and re-deploying the entire application every time updates are made to the Multi-Access Edge Computing (MEC, [9]) platform or any program component, MEC platform functions could be implemented as microservices with each microservice as a separate entity with no dependency on other microservices forming the MEC application [10, 11]. The microservices interact with each other to implement the logic of the application and communicate via network calls to enforce separation and avoid tight coupling. Consequently, each microservice exposes an application programming interface (API) for other microservices to communicate in collaborative way. This permits scaling of only those microservices that need scaling while keeping the rest of the application untouched. In addition, a microservice based approach provides better fault isolation such that the failure of a specific application instance only affects those services for which it is required.

Since microservice-based applications can contain hundreds of microservices, an efficient Execution environment that microservice applications use is of paramount importance to ensure that the advantages associated with microservice are accrued. Running multiple services on a single OS instance may risk having conflicting library versions and application components, not mentioning the fact that one microservice failure could affect the availability of others. Using VMs could solve such problems but using individual VMs for each microservice will exact a heavy cost, since each requires its own OS. Furthermore, running multiple application components within a single VM may lead to application problems due to component conflicts. With containers on the other hand, a single OS instance running either on a physical machine or a VM can support multiple containers, each running within its own separate execution environment. Container orchestration frameworks of which Kubernetes has become the de-facto standard, eliminate infrastructure complexities associated with deploying, scaling, and managing containerized microservice applications by automating processes involved in running containers, and providing networking support for inter-connecting microservices. This makes containers more suitable for microservice architecture than VMs. However, kernel-specific libraries are still shared among containers which may still lead to minimal conflicts. This problem is graver in telecommunication deployments as the kernel needs to be customized to support dedicated performance acceleration hardware.

In D3.1 [2] we discussed the Kubernetes container orchestration framework, with more focus on the worker node where the workload containers are deployed. The various components of the worker node that play an important role with regards to container networking performance and security are then discussed in detail. These notably include container runtimes and Container Networking Interface (CNI) plugins. Network policies, a Kubernetes native resource for configurable network isolation is then evaluated to assess its impact on the performance of inter-container communication. We further discussed Data Plane Development Kit (DPDK), a set of data-plane libraries that achieve fast packet processing by bypassing the heavy layers of the Linux kernel networking stack are as well discussed, followed by a context-aware policy enforcement engine, Open Policy Agent (OPA). D3.1 [2] concluded with a reference architecture and a data flow diagram to illustrate how the various components highlighted in the document interact.

In this deliverable we build upon D3.1 [2] and discuss approaches to harden Kubernetes deployments against attacks. In this Chapter we first survey established mitigations against attacks that harm the integrity and confidentiality of container deployments. We then discuss these mitigations in the context of our trust model and use-case requirements, and elaborate on extended security and privacy preserving technologies and their applicability for 5G edge VNF deployments.

3.1. Overview

In our architecture (Figure 3.1), we model a multi-tenant Kubernetes cluster in the cloud. We assume that all the control plane components run on a separate, secure environment directly managed by the cluster administrator. The worker nodes, instead, are Virtual Machines (VMs) that might be located in many different places and administrated by many different providers. Each VM runs the operating system plus the Kubernetes software and the container runtimes at the very least. Furthermore, container images are stored in one or more image registries. A registry can be either public or private, and it can be situated inside the cluster (e.g., as a microservice) or outside (e.g., as an external repository managed directly by tenants or by a third party).

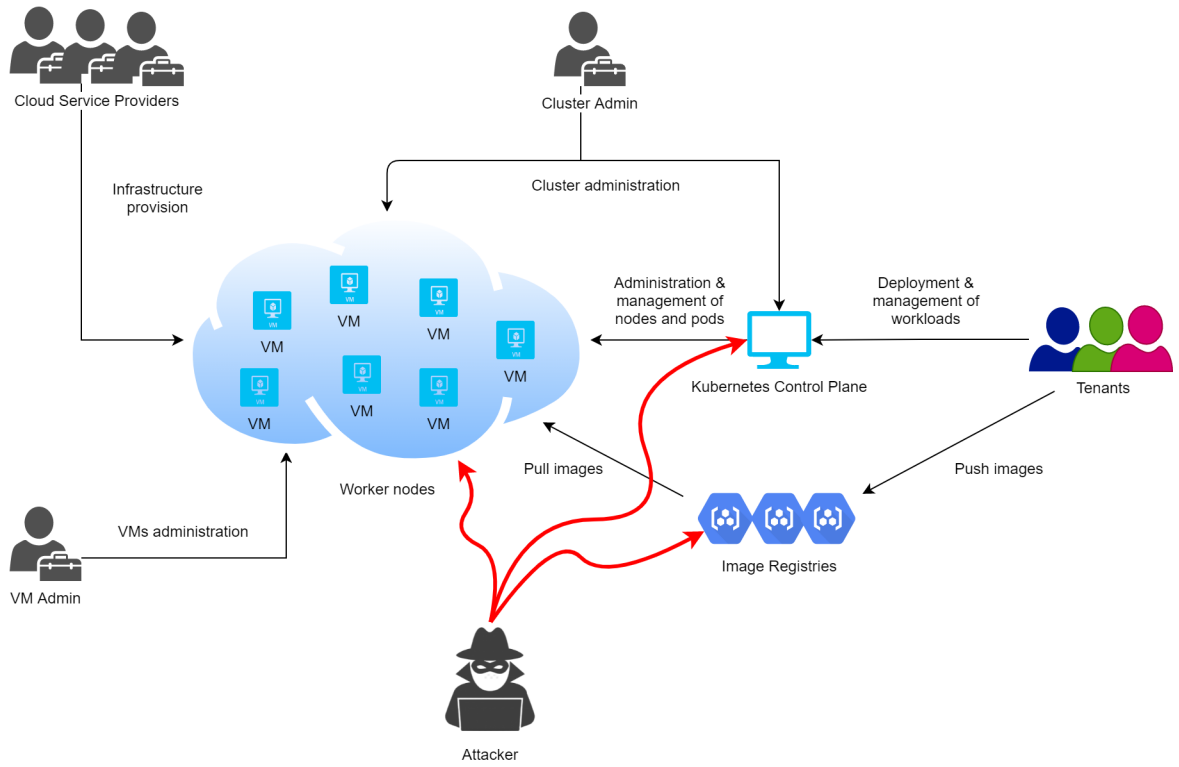

Figure 3.1: An overview of a multi-tenant Kubernetes cluster in the cloud and the different actors that interact with it.

We identify four logical actors that are responsible for the installation and configuration of the cluster, as well as the deployment of workloads:

- **Cloud Service Provider (CSP)**. A CSP is an entity that manages the cloud infrastructure where the worker nodes are running. As the name indicates, they provide the infrastructure (hardware, virtualization layer, networking) to the other actors of the system. We consider a variable number of CSPs in our general architecture, i.e., the VMs in the cluster do not necessarily belong to the same CSP.
- **VM Admin**. The VM administrator is responsible for setting up and managing the VMs that are provided by the CSPs and used as worker nodes. Some of the tasks done by this actor are the following: installing and configuring the OS, the Kubernetes software and container runtimes, setting up the users with their permissions. The VM admin has no privileges on the Kubernetes control plane.
- **Cluster Admin**. The cluster administrator is responsible for setting up and managing the cluster. Some of the tasks done by this actor are the following: initializing the cluster, joining the worker nodes, installing the necessary plugins and/or admission controllers, configuring the tenants' privileges, managing the security aspects of the cluster such as tenant isolation, authentication, authorization. The cluster admin has limited privileges on the VMs, which are mostly used for the initial setup of the cluster.
- **Tenant**. A tenant is someone able to deploy workloads (i.e., applications) to the cluster. We consider a variable number of tenants. Each tenant must be isolated from the others, for example by assigning each a different namespace. Additionally, tenants have limited privileges on the cluster (e.g., to deploy pods), but no direct access on any of the VMs.

3.1.1. Worker nodes

A worker node is a VM where containers can be scheduled for execution. We identify the following layers on each node:

- **Hardware layer**, i.e., the hardware infrastructure that provides the resources used by the VM.
- **Hypervisor layer**, i.e., the abstraction layer that allows to run multiple VMs on the same hardware infrastructure.
- **Operating system layer**, i.e., the Operating System (OS) and other software installed on the VM. This also includes the high-level and low-level container runtimes, as described below.

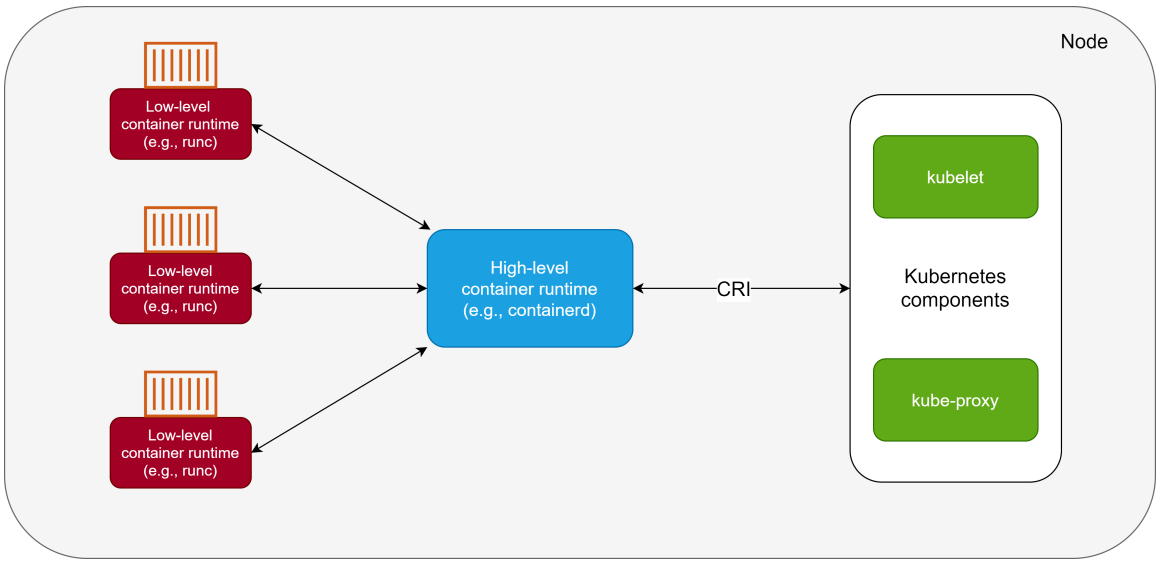

Figure 3.2: High-level view of the essential software components needed to run containers on a Kubernetes worker node

- **Orchestrator layer**, i.e., the Kubernetes components: *kubelet* and *kube-proxy*.
- **Application layer**, i.e., the containers running on the node.

Figure 3.2 shows how the various software components interact to manage the execution of containers. The Kubernetes components, more precisely the kubelet, are connected to the high-level container runtime through the Container Runtime Interface (CRI), which consists of protocol buffers, gRPC API, and libraries used for communication. This allows the kubelet to be compatible with several different runtimes, provided that they comply with the CRI. The high-level container runtime receives commands from the kubelet to manage the images (e.g., `pull`, `list`, `inspect`) and the container lifecycle (e.g., `create`, `start`, `stop`, `delete`, `list`, `inspect`). The most popular high-level container runtimes are *containerd* (the default runtime), *CRI-O*, and *Docker* (now deprecated).

Containers are executed in isolated environments provided by low-level container runtimes. Each container runs a container image, a lightweight and standalone package that includes everything needed to run an application: code, runtime, system tools, system libraries and settings. The Open Container Initiative (OCI) project defines standards for images and container environments. First, the *image-spec* standard defines the structure of a container image. Tools such as Docker allow to create an OCI-compliant image by taking as input the source code and a configuration file (called *Dockerfile*). Second, the *runtime-spec* standard outlines how to spawn and run OCI images. Therefore, any low-level container runtime must comply with the runtime-spec standard, and is able to manage any image that comply with the image-spec standard. The most popular is *runc*, which is also the default runtime used in Kubernetes. However, there are many different low-level runtimes that provide different levels of security and performance. We discuss low-level container runtimes more in detail in section 3.4.2.3.

3.1.2. Image/container lifecycle

A container that is executed in a Kubernetes cluster follows multiple stages (Figure 3.3). We distinguish between two main phases, *Development* and *Operations*. The former includes all the steps involved in the creation and distribution of container images, starting from writing the source code up to publishing the image to a registry. The latter concerns the execution of a container, from its creation to its disposal.

In the Development phase, we categorize three distinct steps:

- **Develop**. This phase concerns the development of the application's source code.
- **Build**. In this step, the container image is created. This image must comply to the OCI standard, and it includes the code developed in the previous phase plus any other dependencies and settings.
- **Release**. This phase involves the release and distribution of the image. This consists of pushing the image to a specific image repository. Optionally, some additional operations are done before or after the image is pushed. For example, the image can be signed for integrity protection (section 3.3).

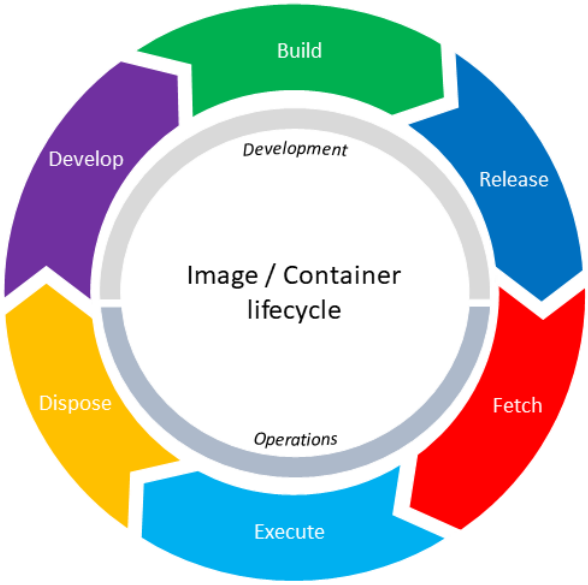

Figure 3.3: Image/container lifecycle

Regarding the Operation phase, we identify the following stages:

- **Fetch**. In this phase, a new container is created and set up, the corresponding image is fetched from the registry (or from a local cache) and loaded inside the container.
- **Execute**. This step involves the execution of the container. This includes all the different states that a container might have over time, e.g., `running`, `paused`, etc.
- **Dispose**. In this phase, the container is stopped and deleted. This happens, for example, if the container has completed its job, it needs to be migrated to another node, or a during a rolling update.

3.2. Specialized Threat Model

An attack aimed to disrupt the integrity of an image/container can be targeted at different components of our architecture (Figure 3.1). We distinguish between attacks targeted at the cluster's control plane, at the nodes, and at the image registries. Attacks directly targeted to the actors' workplaces (e.g., the tenants' development platform) are considered out of scope. Network attacks such as man-in-the-middle attacks or data tampering are possible, but they are not the main focus, as we assume that each connection is protected with respect to confidentiality, integrity, and authenticity (e.g., using the encrypted TLS communication protocol).

For each kind of target (i.e., cluster, node, registry), we classify the users that interact with it, describing their privileges. The principle of *least privilege* is applied: every user must have nothing more than the minimal set of privileges and permissions that is needed to perform their tasks. By having a clear separation of users and privileges, we can build more easily a trust model between the actors of the system.

This section is structured as follows: subsections 3.2.1, 3.2.2 and 3.2.3 discuss about the privileges of users interacting with the cluster, nodes, and image registries respectively. Subsection 3.2.4, instead, presents our refined trust model.

3.2.1. Cluster: users and privileges

The Kubernetes control plane can be configured using role-based access control to assign roles (and privileges) to each user (or group of users). In our architecture, only the cluster administrator and the tenants directly communicate with the control plane. Assuming a proper configuration of the cluster, other external users have no privileges, i.e., every attempt to query the API server results with a *401 Unauthorized* or a *403 Forbidden* response.

More in detail, we distinguish between the following users and privileges:

A. **Cluster Admin**: The cluster administrator has full privileges over the whole cluster. A malicious cluster admin (or any other user that has somehow gained such privileges) can completely take over the entire cluster. Possible attacks on integrity are: joining malicious nodes to the cluster and scheduling one or more pods to run on such nodes, intercepting and altering any resource objects provided by a tenant, giving administrative privileges to a tenant, tampering with sensitive data (e.g., secrets).

B. **Tenant**: A tenant can deploy workloads on the cluster. Normally, tenants are assigned to different namespaces and they cannot interfere with each other. However, containers belonging to different tenants can still run on the same node. Thus, if a malicious container can somehow "escape" the isolation of the container runtime, it can affect the integrity of both the host OS and other containers on the same node. Moreover, a malicious container might affect others by exploiting their public or private interfaces (e.g., a microservice that is used by other tenants via inter-namespace communication). If the targeted container has some additional privileges on the cluster, the tenant could also escalate their privileges to gain additional powers and tamper with other tenants.

C. **End User**: End users cannot directly communicate with the control plane, as explained above. Yet, these users can still interact with the cluster via the public interfaces offered by certain containers (e.g., a web server). Attacks can exploit certain vulnerabilities in such containers to disrupt their integrity. Similarly, from a container that has been successfully exploited by the attacker, it is possible to perform other attacks on the cluster, as described in point B.

3.2.2. Node: users and privileges

A node is a VM that is used by the control plane to schedule the execution of containers. We assume that all the VMs of the cluster only serve this purpose, i.e., no other software is installed or running except for the Kubernetes software (kubelet, kube-proxy), the container runtimes (both high and low level) and the containers themselves. Additionally, no one except for the cluster admin and VM admin has a user account on the VMs.

We identify the following users of a node:

A. **Cloud Service Providers**: The CSP provides the hardware resources, virtualization, and communication infrastructure, on top of which VMs are executed. They should not have any direct access nor privileges on a VM.

B. **VM Admin**: The VM admin has full privileges over the whole node/VM. A malicious VM admin (or any other user that has somehow gained such privileges) can completely take over the node, potentially affecting the cluster as well. Possible integrity attacks are: tampering with the memory, tampering with the local container image's cache, installing a custom malicious kubelet or container runtime, giving root access to other users.

C. **Cluster Admin**: The cluster admin is configured to have root access only on a subset of commands needed to configure the node in the cluster (e.g., *kubeadm*). Nevertheless, as described in the previous section, the cluster admin can join an arbitrary number of external nodes to the cluster, in which they have full admin privileges. Therefore, on such nodes, the same attacks as explained in point B. are possible.

D. **Tenant**: A tenant is not a "classic" user of a node. They do not have user accounts, but they can run their containers on the node, through the Kubernetes control plane. Thus, there is no direct interaction between a tenant and a node, since the control plane acts as a sort of man-in-the-middle between the two. Normally, the control plane decides by itself on which node the container should be scheduled for execution, basing on the resources available on the nodes at a given time. However, the tenant could force the execution on a specific node by properly setting some constraints in the Pod spec, such as *nodeName*, *nodeSelector*, *nodeAffinity* or *podAffinity*. A malicious container could then try to "escape" the isolation of the container runtime to perform attacks on the node itself and on other containers.

E. **End User**: The end user of a node is anyone that wants to directly access the node via a public interface, without any privileges. Besides the containers running on the node, the end user can only interact with the kubelet, and (optionally) an SSH server that is typically used by the VM admin and cluster admin to configure the VM. However, assuming that proper configuration is in place and no escalation of privileges is possible, every attempt to connect with such services would fail, if not properly authenticated.

3.2.3. Image registry: users and privileges

An image registry is a database for storing container images along with their metadata. Images can be pushed (write) or pulled (read). In both cases, a specific image repository and an image reference (tag or digest) must be provided, e.g., `rustlang/rust:latest`, where `rustlang/rust` is the name of the image repository and `latest` is the tag.

Push operations always require write privileges on that specific image repository. Instead, pull operations can be done without any privileges on public repositories, but they require read privileges on private ones.

We classify the following users of an image registry:

A. **Registry Admin**: The registry administrator has full access over the registry. Therefore, they can perform both read and write operations on all the image repositories in the registry, regardless of whether they are public or private. A malicious registry admin (or any other user that has somehow gained such privileges) can violate the integrity of all the stored images.

B. **User**: Authenticated users of a registry have full control (r/w) over their own images. Additionally, they can also pull images from public repositories, but they do not have write privileges on them. If the registry resides within the cluster and is private, tenants are the only users. If, instead, the registry is publicly accessible (e.g., *Docker Hub*), anyone can create an account and become a user.

C. **Guest**: We define *guest* an unauthenticated user who accesses the registry. This works only if the registry is public, since on private registries authentication is always required. By not having any privileges, guests are only allowed to pull from public repositories.

3.2.4. Trust model

As showed in the previous sections, there are many actors that are in some way involved in the instantiation and operation of a multi-tenant, cloud-based Kubernetes cluster. In order to better understand which are the most critical security aspects (especially in terms of integrity) and which solutions need to be implemented to build a more robust system, this section presents a trust model between the most relevant actors of the system.

We consider three different levels of trust:

- **Trusted**. *"A trusts B"* means that B is completely reliable and honest to A, hence no particular security mechanisms are required or necessary.
- **Semi-trusted**. *"A semi-trusts B"* means that A expects a good behavior from B in general, but at the same time B cannot be considered fully trusted as they can be compromised by external attackers, or simply they are not able to provide guarantees about certain aspects. In our case, we consider a sort of *"honest but curious"* attacker model: we trust B for availability, we expect a good behavior in terms of integrity (unless a compromise happens), but confidentiality is not provided or required.
- **Untrusted**. *"A does not trust B"* means that it is unclear for A if B is malicious or benign, therefore A should take the appropriate countermeasures to ensure that B cannot cause any harm.

A fourth trust level would be **distrusted** or malicious, however none of the relations between the actors we are considering can be distrusted. For example, if a tenant considered the cluster administrator malicious, they simply would not use the cluster at all. External attackers, instead, are obviously distrusted by anyone in the system.

Figure 3.4 shows the trust relations between the relevant actors of our system. Some arrows are not displayed, meaning that those relations are not relevant in our context (e.g., the VM admin to the cluster admin, or a CSP to any other actors). This trust model considers each actor as a distinct physical entity; however, the color of some arrows would be different if the same person or organization covered multiple roles. For example, if a tenant manages their own private image registry, the arrow between tenant and registry administrator would be green on both sides, i.e., there is full trust as they are the same person.

The CSPs provide the infrastructure on top of which the cluster is running. Generally, a sort of "contract" between a CSP and a VM admin is established, in which the latter pays for the services offered by the former. Therefore, the VM admin semi-trusts the CSPs for the availability of such infrastructure. Conversely, the CSPs does not rely on the functionality of the VMs or their services, so no trust relation is required.

There is a tight connection between the cluster administrator and the VM admin, since the VMs configured by the latter are used by the former in the Kubernetes cluster. Hence, there should be a semi-trust relation from the cluster admin to the VM admin, because it is expected that the latter configures the VMs properly and does not install some malware.

The cluster admin should not trust tenants, as a tenant might be malicious or compromised by an external attacker. For the same reason, tenants do not trust each other. A tenant, instead, has to semi-trust the cluster admin and the VM admin, as the operation of their workloads depends on the Kubernetes control plane and the nodes where the containers run.

Concerning the container images stored on a registry, the tenant needs to semi-trust the registry administrator. On the other hand, a registry administrator does not trust their users, and the same considerations made for the cluster admin-tenant

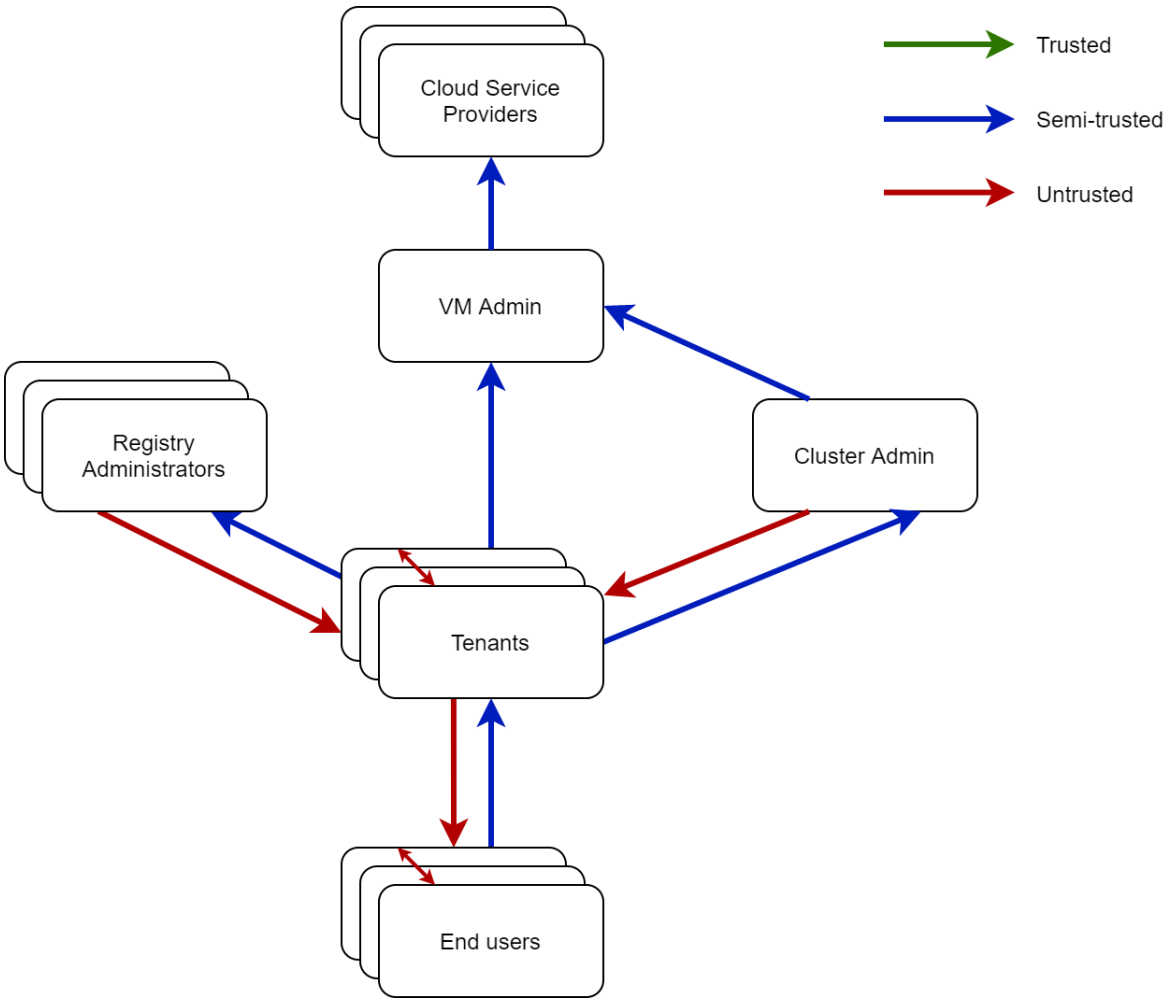

Figure 3.4: Trust relations between the most relevant actors of the Kubernetes cluster

relation apply here.

Finally, tenants must not trust end users, as an attacker can easily reach the end-user side. Similarly, end users do not trust each other. Instead, an end user semi-trusts a tenant (and, from them, the whole infrastructure).

3.3. Container Image Security

Existing approaches to secure containerized services focus on the *Build* phase or the *Release* and *Fetch* phases of the image/container lifecycle. Techniques applied in the first phase aim at hardening containers by reducing their attack surface and by statically detecting and eliminating vulnerabilities. The *Release* and *Fetch* phases, methods to secure transmission and storage, and to verify the integrity of deployed containers are being used. In this section we review a number of these approaches.

3.3.1. Reducing the attack surface

To reduce the attack surface as much as possible, it is important to build a container image that contains only the necessary components needed to run the main application. Moreover, the application should be run with the minimum set of privileges that is required for its correct execution (*least privilege* principle).

Some of the best practices that should be applied in this direction are:

- **Use a minimal base image**. There are plenty of images that can be chosen as base image, from general-purpose (e.g., `ubuntu`) to more specialized ones (e.g., `python`). Usually, such images also have multiple tags, used

not only to identify a specific version (e.g., `ubuntu:18.04`) but also to provide different variants in terms of installed packages and features. The base image should be selected according to the application's requirements, but at the same time it should be kept as minimal as possible in order to decrease the size of the container image and consequently reduce the attack surface. It is also highly recommended not to use base images released by unknown (and therefore untrusted) authors.

- **Install only the essential packages**. The container image should only contain the minimal set of packages required by the application. As an example, when installing a package with `apt-get install` it is useful to specify the `--no-install-recommends` flag, which prevents the installation of all the packages that are recommended but not necessary to have along with the one being installed.
- **Remove temporary files and packages**. Sometimes, certain packages or files are only required during the building phase of the image, but not when the container is running (e.g., a language tool chain). To reduce the size of the image, as well as the attack surface, it is recommended to remove such components when they are no longer needed. A convenient feature of a Dockerfile is called *multi-stage builds*, which allows to build intermediate images used for different tasks. For instance, it is common to build an intermediate image to compile an application from its source code. That is, the intermediate image is used as the "building environment" and includes all the dependencies (tool chains, compilers, etc.) needed to build the application; the compiled executable is then simply copied from the intermediate to the final image.
- **Change default user**. The default user of a container image is typically `root`. This might be a huge security concern, not only for the container itself but also for the host OS. A good practice when defining a Dockerfile is to create a new user with known User ID (UID) and Group ID (GID), and then switch user (using the `USER` instruction) when the root privileges are no longer required.
- **Perform regular updates**. For increased security, it is a good practice to keep the image up to date, not only regarding the application itself but also the base image and all the installed packages. More recent versions of a package might patch some vulnerabilities discovered over time. Thus, keeping the image up to date would increase its overall security.

3.3.2. Detecting vulnerabilities statically

Once the image is built, *scanners* can be used to check if it contains some known Common Vulnerabilities and Exposures (CVEs). This procedure can be performed at different stages of the image/container lifecycle: before the distribution of the image (*Release* phase), possibly integrated with the CI/CD pipeline; in the image registry, after the distribution; during the admission in the Kubernetes cluster (*Fetch* phase). Different stages concern different actors: tenant, registry administrator and cluster administrator respectively.

There are several image scanners available in the market, all of them sharing similar functionalities. As an example, Docker provides a built-in tool that can be invoked via the `docker scan` command, which could be seamlessly integrated in the CI/CD pipeline. For the *Fetch* phase, instead, some of the scanners can be used in conjunction with admission controllers that accept or reject an image basing on the scan results (see section 3.4.2.2). Popular image registries such as Docker Hub contain a built-in image scanner, which is typically offered to the users as a premium service.

3.3.3. Image selection, secure transmission and storage

Security in transit and at rest. When an image is sent over the network (i.e., during the *Release* and *Fetch* phases of the container/image lifecycle) it is important to protect the communication channel to avoid common attacks such as data tampering, man-in-the-middle attacks, eavesdropping, etc. The mainstream solution nowadays consists of using authenticated TLS channels to preserve the confidentiality, integrity and authenticity of data in transit.

Security of images at rest is equally important. An image stored in a registry should be protected with proper mechanisms of access control, in order to give read/write access to an image repository only to authorized users. This behavior should be implemented and enforced by the registry administrator. For increased security, tenants might want to store their own images in a private registry, in which they have exclusive access. A private registry could also be installed in the cluster, in this case access is restricted to the tenants and cluster administrator.

Normally, when an image is fetched from a registry, it is stored in a local cache inside the node. This way, subsequent requests of the same image can use the local copy instead of pulling it again from the registry, reducing the start-up time of a container and the traffic load on the network. Unfortunately, local caches do not implement any access control mechanism, allowing a tenant to potentially use other tenants' images. Privileged users in the node might also corrupt

the local cache, injecting malicious images and/or tampering with existing ones. An effective defense mechanism against these problems is to enable the *AlwaysPullImages* admission controller in the cluster, which forces the kubelet to ignore the local cache and always fetch the images from the registries. However, this solution might bring additional start-up latencies, which might not be acceptable in some critical scenarios; therefore, custom solutions might involve a local cache deployed on the cluster, complemented with an admission controller for access control enforcement.

To add another layer of security both in transit and at rest, images could also be encrypted (to preserve confidentiality) and/or authenticated (to preserve integrity and authenticity). This chapter mainly focuses on image integrity, hence we will not discuss image encryption. Instead, we examine image signing in section 3.3.4.

Image identifiers. An image repository is uniquely identified by the following tuple: `<registry_url,owner_name,repository_name>`, where `registry_url` is the URL of the registry where the image is stored, `owner_name` is the name of the user who owns the image, and `repository_name` is the name of the image repository. A repository might contain several images; each image in a repository is uniquely identified by a label, either a *tag* or a *digest*. When a tenant deploys a container to a cluster, they select the image to use by specifying the image repository and label.

An image is labeled with a tag during its creation. Tags are not uniquely assigned, i.e., the same tag can be reassigned to another image in the same repository. For this reason, tags are defined *mutable*. The mutability of tags might create some inconsistencies when the same tag is used more than once. If a tag is reused, then it is possible that different storage locations (registries, local caches, etc.) contain different versions of the same image. This may cause some undesired consequences in a cluster, e.g., that different replicas of the same container run different images. Solutions to this problem are: use semantic version tags such as `v1.2.3`, to assign a specific version to each image and avoid tag reuse; set the *ImagePullPolicy* of a container to *Always* and/or enable the *AlwaysPullImages* admission controller, to bypass the local caches and always fetch an image from the registry; use digests instead of tags.

A digest is an SHA-256 hash over the content of a container image, used both as a reference and an integrity check (see section 3.3.4). Every time a new version of the image is built, the resulting digest would consequently be different, as the content of the image would change. Therefore, in contrast with tags, digests are *immutable*. Digests, however, are not human-friendly as they are not easily readable, and it is easy to make mistakes in the resource specification. Therefore, it is recommended to use digests when automated deployment tools are in place (e.g., a CI/CD pipeline) to reduce risk of misconfiguration, while semantic tags could be a better choice when manual configuration or monitoring are required.

3.3.4. Integrity verification and remote attestation

The integrity of a container image can be violated in different ways and different moments of time. When a new container is scheduled for execution, the corresponding image is fetched by the kubelet and loaded inside the container, ready for execution. In this section, we look at techniques to ensure that (1) the fetched image is authentic, and (2) the image is not tampered with before the *Execute* phase.

3.3.4.1. Integrity verification

We rely on cryptographic techniques to ensure the integrity of images. Hash functions are used to calculate the digest of an image, while asymmetric encryption is used to apply a digital signature over the image's metadata.

Digests. Digests are used as integrity measurement and verification of an image. As shown in Figure 3.5, the image manifest contains hash values of the configuration file (`config`) and each layer that composes the image (`layers`), as well as a global digest of the whole image that can be also used as a reference, as explained above. These values are used as integrity checks: all the digests are recalculated after the pulling phase, and if a certain layer or the configuration file are corrupted the hash would not match the one included in the manifest, causing an integrity violation. Clearly, an attacker that has control over the manifest file can modify the reference digests accordingly, and a corrupted image would still bypass the integrity verification. However, any attempt of tampering would fail if the tenant references the image by digest in the Pod spec, because the recalculated global digest would not match the reference provided by the tenant. The attacker now would need additional powers to also modify this digest, either targeting the tenant's workstation, the Kubernetes control plane, the network, or the node where the image is being pulled.

```
$ docker manifest inspect --verbose hello-world
{
        "Ref": "docker.io/library/hello-world:latest",
        "Digest": "sha256:f3b3b28a45160805bb16542c9531888519430e9e6d6ffc09d72261b0d26ff74f",
        "SchemaV2Manifest": {
                "schemaVersion": 2,
                "mediaType": "application/vnd.docker.distribution.manifest.v2+json",
                "config": {
                        "mediaType": "application/vnd.docker.container.image.v1+json",
                        "size": 1520,
                        "digest": "sha256:1815c82652c03bfd8644afda26fb184f2ed891d921b20a0703b46768f9755c57"
                },
                "layers": [
                        {
                                "mediaType": "application/vnd.docker.image.rootfs.diff.tar.gzip",
                                "size": 972,
                                "digest": "sha256:b04784fba78d739b526e27edc02a5a8cd07b1052e9283f5fc155828f4b614c28"
                        }
                ]
        },
        "Platform": {
                "architecture": "amd64",
                "os": "linux"
        }
}
```


Figure 3.5: Manifest file of the hello-world image, obtained using the Docker command line interface. Source: https://docs.docker.com/engine/reference/commandline/manifest/

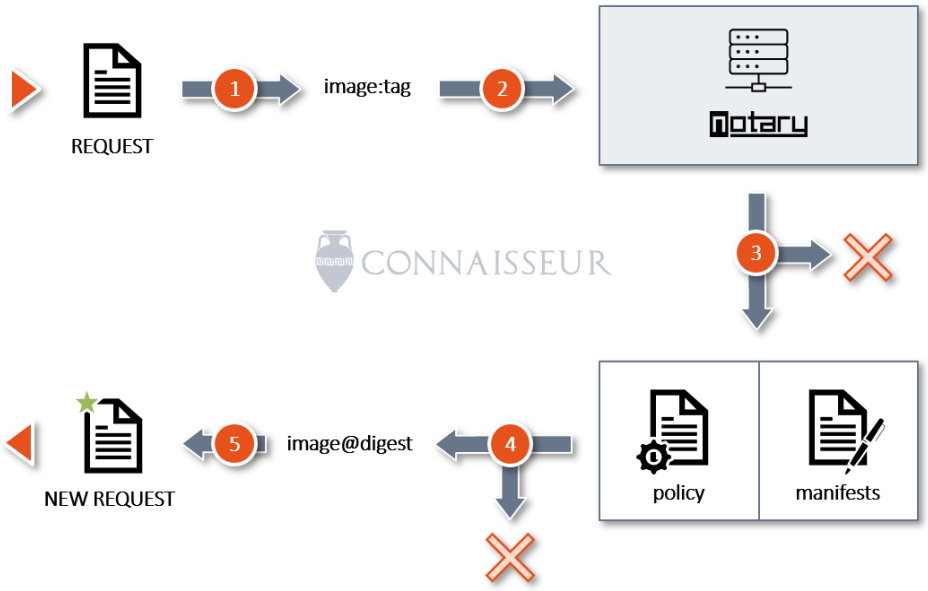

Figure 3.6: Overview of Connaisseur's process flow. Source: [4]

Image signing. Integrity verification using digests is a very simple and effective mechanism, but if the image is referenced by tag the checks are not enough to protect its integrity. Hence, another method is to apply a digital signature on an image, and thereafter verify this signature when the image is being deployed on the cluster. This is a more powerful integrity mechanism as images are now linked to a specific identity, which can be the identity of the tenant that is deploying the workload or some trusted 3rd party. In this way, the control plane could also decide to accept or reject images according to the identity of the owner, for example rejecting an image that has a unknown or no signature (see section 3.4.2.2).

The key aspect of image signing is to cryptographically link an image tag with its digest. This is done by applying a digital signature over some trust data, which is then stored along with the image and retrieved during the *Fetch* phase. This technique allows tenants to reference images by tag, yet retaining the same integrity guarantees as with referencing images by digest, as explained above.

At the time of writing, Kubernetes does not support any signature-based integrity verification natively. However, as k8s is highly customizable, several projects exist to enable this functionality. One of the most popular is Connaisseur [4], a mutating admission controller that exploits a feature called Docker Content Trust (DCT), used for signature verification. DCT allows a user to digitally sign an image with a specific tag before pushing it to a registry. It is built on top of Notary, a tool for publishing and managing trusted collections of content. As shown in Figure 3.6, Connaisseur intercepts all the deployment requests to the cluster and checks the images included in the request. For each image, it queries the Notary server to retrieve the image's trust data. If such data is not present or is invalid, the whole deployment is rejected. Otherwise, the tag is transformed in a trusted and cryptographically verified digest, which is later used to fetch the image. Connaisseur also provides a configurable image policy, used for example to allow the deployment of certain unsigned images, or to accept signatures only by specific signers.

A similar project is Portieris [12], which shares the same mechanism as Connaisseur but also supports simple signatures, a scheme that does not rely on DCT and Notary but simply performs signature verification given a known public key.

A basic verification mechanism is also implemented in the CRI-O container runtime. However, this is not a cluster-wide solution since it has to be enabled on each node separately. Moreover, the verification is not performed on cached images, but only on those fetched from a registry.

3.3.4.2. Remote attestation

Signature-based integrity verification techniques are very effective to ensure that only authorized and authentic images enter the cluster during the fetching phase. Yet, this does not protect the loading process, i.e., the container creation and start. Assuming that the image is correctly downloaded on a node, there is a small window before the container is executed in which the image can be altered or replaced. A privileged user on the node might have write access to the local image cache, thus being able to tamper with an image after the integrity check. Alternatively, a malicious container runtime might run a different image and/or skip the integrity checks.

To ensure that a process is running an authentic software, *Remote Attestation* can be leveraged. As the name suggests, it is a technique used to verify the correctness of an application running on a remote node. It consists of obtaining a cryptographic proof of the identity of a process, which generally includes the hash of the application's binary plus some metadata. This proof is generated and signed by a specialized hardware component, defined the Root-of-Trust (RoT) of a node; as such, no software can forge, or tamper with, attestation messages. The verifier (i.e., whoever wants to verify the application) receives the proof and (1) checks if the message is authentic by verifying the signature, (2) compares the identity of the application with the expected (reference) value, ensuring that they match. If both steps have positive result, the whole attestation succeeds.

Remote Attestation can ensure the integrity of an image during all the steps up to the loading phase included. Hence, if used on the whole cluster, it can replace (and provide better security compared to) the techniques described so far. Yet, to do this, nodes must provide the hardware capabilities to perform attestations, and the verification mechanism must be properly integrated with Kubernetes. In addition, there are also performance concerns that must be considered. It is worth noting that Remote Attestation is static, i.e., the integrity is verified only over a binary/package file and not on runtime's code and data. Hence, runtime inconsistencies cannot be detected by this technique.

Currently, there is no native support for the attestation of containers in Kubernetes. However, some external projects implement this functionality; for example, [13] and [14] provide a framework to attest Docker containers using a TPM-based approach (see section 3.4.1.2). Other projects, instead, use Trusted Execution Environments (TEEs) to deploy confidential workloads, in which remote attestation is integrated to provide a tenant and/or end users the guarantee that their code and data are protected at runtime. We discuss in detail about TEEs and containers in section 3.5.

3.4. Attack Surface and TCB Assessment

During its execution time, the integrity of a container should be enforced to counter the possible presence of malicious activity in the cluster. We focus on identify threats coming from the same node where the container, defined as *target container*, is executing. Here, the attack surface includes:

- The node, i.e., the host OS plus running software/modules such as the kubelet, container runtimes, etc. Attacks might come from inside (e.g., malicious software, privileged user) or outside the node (e.g., an attacker that exploits a certain service publicly exposed).
- Other containers running on the same node. A container might be able to break the isolation mechanisms provided by the low-level runtime and affect the target container directly or through the host OS.
- The target container itself, which is susceptible to attacks if it contains vulnerabilities.

Essentially, attacks performed anywhere on the node can potentially disrupt the integrity of the target container. Therefore, security mechanisms must be implemented in order to reduce the attack surface, to enforce a defense-in-depth approach, as well as to detect ongoing attacks. In each of the following subsections, we explore solutions that focus on specific components, respectively: the host OS; other containers running on the same node as the target container; the target container itself.

3.4.1. Prevention and detection of malicious activity on the node

As already mentioned in section 3.2.2, a worker node should only serve the purpose to execute the containers that are part of the Kubernetes cluster. Therefore, the VM admin should take care of the configuration of the VMs in order to reduce the attack surface as much as possible.

Such prevention techniques should be complemented by detection approaches to ensure at runtime that a node is not compromised. We will discuss about node attestation, i.e., obtaining a cryptographic proof that a remote node is working as expected. For example, this proof might demonstrate that the node at a given time is running a genuine version of the OS, and no unknown software is running. This proof can then be used by the control plane to increase the trustworthiness of the node, which can therefore be used to schedule the execution of critical workloads that require high security guarantees. Conversely, if the attestation of a node fails, the control plane might migrate all the containers running on that node elsewhere, preventing at the same time the execution of new containers until a new, successful attestation response is received.

3.4.1.1. Reducing the attack surface

To reduce the attack surface, it is essential to remove all the unnecessary software and packages from the OS. The VM only needs to run the Kubernetes software with its dependencies (e.g., the container runtimes), along with a few optional components such as logging tools.

To this purpose, several OS distributions specifically optimized for running containers are available for use. Some of the most popular ones are the following:

- **Google Container-Optimized OS** [15]. This minimal OS can be used in the Google Cloud Platform environment, and it is the default OS image in Google Kubernetes Engine (GKE). It comes with the Docker runtime, and it provides some interesting security features such as small footprint and automatic updates. It does not include a package manager, nor it supports the execution of non-containerized applications or the installation of third-party kernel modules. Unfortunately, this OS distribution cannot be used outside the Google's cloud environment.
- **Amazon Bottlerocket** [16]. Bottlerocket is a Linux-based open-source operating system used in Amazon Web Services (AWS). It is similar to Google's OS, and it can be used with no additional cost within AWS. Since it is open source, it is possible to produce custom builds of the OS to fit the customers' needs, for example by installing specific container runtimes.
- **Rancher k3OS** [17]. This is a Linux distribution specifically designed to run *k3s*, a lightweight version of Kubernetes [18]. This OS is designed to be managed by `kubectl` once the node is bootstrapped, meaning that the node itself only needs to join a cluster, and after that all the aspects of the OS can be managed from the Kubernetes' control plane. The k3OS operator is a component used for automatic upgrades and maintenance of k3OS and k3s.
- **Red Hat Enterprise Linux CoreOS** [19]. This lightweight OS, initially developed as a stand-alone product, is now part of OpenShift, Red Hat's hybrid cloud platform. It uses CRI-O instead of Docker as high-level container runtime.

To ease the configuration of multiple nodes at the same time, it is possible to use automated tools that adopt an Infrastructure-As-Code approach, such as Terraform [20] and Ansible [21]. In this scenario, the VM admin can specify the desired configuration of a VM in a declarative way using configuration files, which are taken as input by these

tools to automatically set up the VM. This is particularly useful in a Kubernetes cluster, where all the worker nodes need the same configuration. Additionally, it is easier to create new nodes or replace faulty ones.

3.4.1.2. Node attestation

A Trusted Platform Module (TPM) [22] is a small hardware component that contains cryptographic keys, small storage, a crypto unit, and a random number generator. Its primary purpose is to verify the integrity of software in a platform, computing cryptographic *measurements* over software configurations that are used as integrity evidences. If a measurement matches a known expected value, then the software can be considered trusted. Otherwise, the software might be corrupted and some countermeasures should be taken to avoid possible attacks, e.g., block its execution.

Using a TPM, we can create a Chain-of-Trust (CoT) that starts from the TPM itself (the Root-of-Trust) and continues to the BIOS, GRUB, up to the OS kernel. Moreover, the CoT can be extended to the application layer thanks to kernel components such as Integrity Measurement Architecture (IMA) [23], allowing to verify the integrity of user processes as well.

Intel Trusted Execution Technology (TXT) [24] leverages the TPM and other hardware extensions to provide verified launch, secret protection, and attestation of a platform. That is, this technology ensures high protection against hypervisor attacks, BIOS and firmware attacks, malicious root-kits, and so on. Hardware-based enforcement mechanisms can block the launch of a software component whose measurement does not match the expected value. Besides, measurements can be also provided at runtime, so that a local or remote user can attest the platform and ensure that each software component has been verified during startup.

Solutions such as Intel Security Libraries for Data Center (SecL-DC) [25] and Azure Attestation [26] create an infrastructure for the remote attestation of platforms. Clients can then verify at runtime the state of a cloud environment and decide to deploy security-sensitive tasks only on verified nodes.

Node attestation can be integrated with Kubernetes to build trust over the nodes of a cluster. In short, trusted nodes whose attestation has succeeded can be distinguished from untrusted ones by applying a label to them, so that the control plane can be aware of which nodes are trusted and which ones not. Then, scheduling decisions can be made to deploy pods that require special care only to trusted nodes. Identifying such pods can be done in several ways, for example by adding annotations in the Pod specification. Note that the nodes not marked as trusted are not necessarily malicious; for example, some nodes might simply not provide the hardware features required for the attestation (e.g., missing TPM).

Intel SecL-DC provides some extensions to integrate its attestation framework with Kubernetes [27], implementing a custom controller and scheduler, as well as declaring new Custom Resource Definitions (CRDs), to enforce the behavior illustrated above.

3.4.2. Prevention and detection of malicious activity from other containers

In a multi-tenant cluster, containers of different tenants might run on the same node. In this scenario, a malicious or compromised container might potentially affect other tenants' containers in the same node either directly or indirectly (i.e., through the host OS). For this reason, it is important to enforce the isolation between containers, and between a container and the host.

Particularly, we look at three prevention techniques: section 3.4.2.1 discusses about how to limit the capabilities of a container in a node; section 3.4.2.2 explains how a cluster administrator can control which container images are used on the cluster; section 3.4.2.3 shows how to increase container isolation using specific low-level container runtimes. In section 3.4.2.4, instead, we introduce a tool to monitor the execution of containers and detect possible malicious activity.

3.4.2.1. Limit the capabilities of containers

Host resources. Container isolation is achieved using Linux namespaces, a feature used to partition kernel resources so that processes can only see (and consume) resources that belong to their own namespace, without being able to access the resources of other namespaces.

There are different kinds of namespaces, such as PID, network, user, and IPC namespaces. Normally, each container has its own namespaces to ensure isolation from other containers and the host itself. However, some containers can share one or more namespaces; for example, all the containers belonging to the same pod share the same network namespace, allowing them to communicate through the loopback interface. Beside namespaces, a container has its own filesystem, which is isolated from the host using `chroot`.

It is possible for a user to break this isolation, by allowing a container to share the same resources as the host for legitimate reasons. For instance:

- A container can be executed within some of the host's namespaces, namely network, PID and IPC. In Kubernetes, this is realized by setting `hostNetwork`, `hostPID` and `hostIPC` in the Pod specification.
- A container can expose its services on the same network as the host, even if they do not share the same network namespace. In Kubernetes, this is achieved by setting `hostIP`, `hostPort` and `protocol` in the Pod specification (`spec.containers.ports`).
- A container can access the host filesystem through mounted volumes. This is done in Kubernetes by setting `hostPath` in the Pod specification (`spec.volumes`).

Resource sharing can be a security threat if not used carefully. As an example, a container running as root has full access to all its filesystem, including mounted volumes as well. Hence, if a volume containing critical data is shared between multiple containers, a malicious container can overwrite such data to violate the integrity of other containers.

Security context. In Kubernetes, a security context is used to define privilege and access control settings of pods and containers. This is achieved by specifying a `securityContext` field in the Pod and/or Container specification. At Pod level, the `securityContext` field is a `PodSecurityContext` object, while at Container level it is a `SecurityContext` object. Although they are different objects, they share some common fields; the rule is that the same field specified at Container level overrides the one specified at Pod level.

Some of the settings that can be specified in `PodSecurityContext` and `SecurityContext` objects are the following (between brackets, we specify if they can be set at Pod level, Container level or both):

- [both] `runAsUser` and `runAsGroup`. Specify the UID and GID used for running the entry point of the container process. Can be leveraged to enforce Discretionary Access Control (DAC).
- [both] `runAsNonRoot`. If true, the kubelet will validate the image at runtime to ensure that it does not run with UID 0 (root), failing to start the container if it does.
- [pod] `fsGroup`. This special group is used to set the owning GID of volumes owned by the pod. If this field is set, the kubelet will change the ownership of such volumes, and new files created in the volume will be owned by this group as well. Note that not all volume types allow the kubelet to change the ownership of a volume.
- [container] `privileged`. If true, runs the container in privileged mode. All processes running on a privileged container are equivalent to root on the host.
- [container] `allowPrivilegeEscalation`. This flag controls whether a process can gain more privileges than its parent or not. This can happen by invoking the `execve` system call in Linux. With this flag set as false, the kubelet will enforce the use of the `no_new_privs` option when the new container starts.
- [container] `readOnlyRootFilesystem`. If true, the container's root filesystem will be mounted as read-only.
- [container] `capabilities`. This field allows to specify the container's Linux capabilities, used to grant or remove certain privileges to a process. That is, an unprivileged process might gain a set of additional capabilities that are typically only assigned to a privileged process. Conversely, a privileged process might be deprived of some capabilities.
- [both] `seLinuxOptions`. SELinux [28] is a kernel module used to enable Mandatory Access Control (MAC). This field specifies the SELinux context to be applied to the containers. Note that the kernel module must be enabled in the host, otherwise this field will not have effect.
- [both] `seccompProfile`. This field sets the seccomp profile of a container. It is used to restrict the container's system calls by allowing only the ones that are needed for its execution. Alternatively, seccomp profiles can also be used to observe which system calls are used by a container without blocking them.

Additionally, AppArmor [29] profiles can also be used to further restrict a container's access to resources. However, since this feature in Kubernetes is still in beta, it is not included in the `securityContext` but can be defined as an annotation in the Pod's metadata. As with SELinux, to use AppArmor profiles the kernel module must be enabled in the host OS.

Enforcing pod's security settings. Namespace and filesystem isolation, as well as a proper configuration of security contexts, are a powerful way to enforce container isolation. However, it is a tenant's responsibility to configure the specification of their own pods. In a multi-tenant cluster, this means that a malicious (or sloppy) tenant can deploy a pod with additional capabilities, e.g., to run as root in privileged mode. Obviously, this can be a huge threat for other tenants.

Fortunately, there are tools to automatically enforce pods' security settings in Kubernetes. The core essence of such tools is defining *policies* to specify some constraints that every pod should satisfy, e.g., *"containers are not allowed to run as root"*. An admission controller is then responsible to verify if the request sent by a tenant complies with the policies. If it does not, the admission controller can either reject or mutate the request.

The built-in solution in Kubernetes are Pod Security Policies (PSPs), consisting of:

- `PodSecurityPolicy` resource objects, a cluster-level resource that controls security aspects of the pod specification, defining a set of conditions that a pod must satisfy in order to be accepted into the system. Essentially, these conditions concern the security context and host resources as described above. A `PodSecurityPolicy` resource can target specific pods using Role-Based Access Control (RBAC).
- The PSP admission controller, which validates requests to create and update pods on the cluster basing on the `PodSecurityPolicy` resources.

It is important mentioning that PSP has been deprecated from Kubernetes 1.21 due to some serious usability problems and will be removed in a future release. However, several external admission controllers for policy enforcement exist nowadays, such as K-rail [30], Kyverno [31], and Open Policy Agent (OPA) Gatekeeper [32]. Besides, a new built-in feature temporarily called PSP Replacement Policy is already under development, and will be available in the future releases of Kubernetes.

3.4.2.2. Restrict image usage

For increased security, the cluster administrator can control what images are used by tenants, to avoid unknown, vulnerable, or malicious images running in the cluster. Policies, together with admission controllers and/or image scanners, can be defined to restrict image usage, for example rejecting images that:

- Have some high or critical CVEs
- Come from certain registries or repositories
- Are owned by certain vendors/users
- Are not signed
- Have a certain tag (e.g., `latest`)
- Contain certain software (e.g., blacklisted base images or packages)

It is essential that the control plane checks the security properties of the images during the admission phase, before they enter the cluster. Policy-enforcing admission controllers such as OPA [32] and Kritis [33] can enforce this behavior, rejecting images that do not meet the policy requirements. However, they need to be manually integrated with an image scanner to actively check for vulnerabilities. Other admission controllers also provide this functionality, such as Anchore [34], Sysdig OPA Image Scanner [35], and Clair [36].

3.4.2.3. Increase container isolation

Instead of rejecting untrusted images, another approach would be to run them on a highly isolated environment in which defense in depth is enforced so that even if an image is malicious, it cannot affect the host system. For containers, this can be achieved by using specific low-level container runtimes that aims to increase container isolation.

Categories of low-level container runtimes. In Kubernetes, the default low-level container runtime is *runC*, which is optimized for performance and thus does not provide any additional security features other than namespace and filesystem isolation, as explained in 3.4.2.1. In a similar way, *crun* is a C implementation of runC that achieves higher performance, but provides the same level of isolation.

Some runtimes propose a micro-VM approach, consisting of running the container in a lightweight virtual machine along with a guest kernel to provide all the features required by the application. Micro-VMs do not need to include all the devices and functionalities of traditional VMs; hence, they have a smaller memory footprint and a reduced startup time, allowing to spawn new VMs on demand whenever it is needed. Hardware virtualization offers a strong isolation between the host and the container environment, logically separating memory, I/O, and network. However, a communication channel between host and micro-VM is still needed, mainly used to control the container's lifecycle and privileges (e.g., start/stop, cgroups/capabilities). The two most popular micro-VM runtimes are Kata [37] (Figure 3.7) and Firecracker [38].

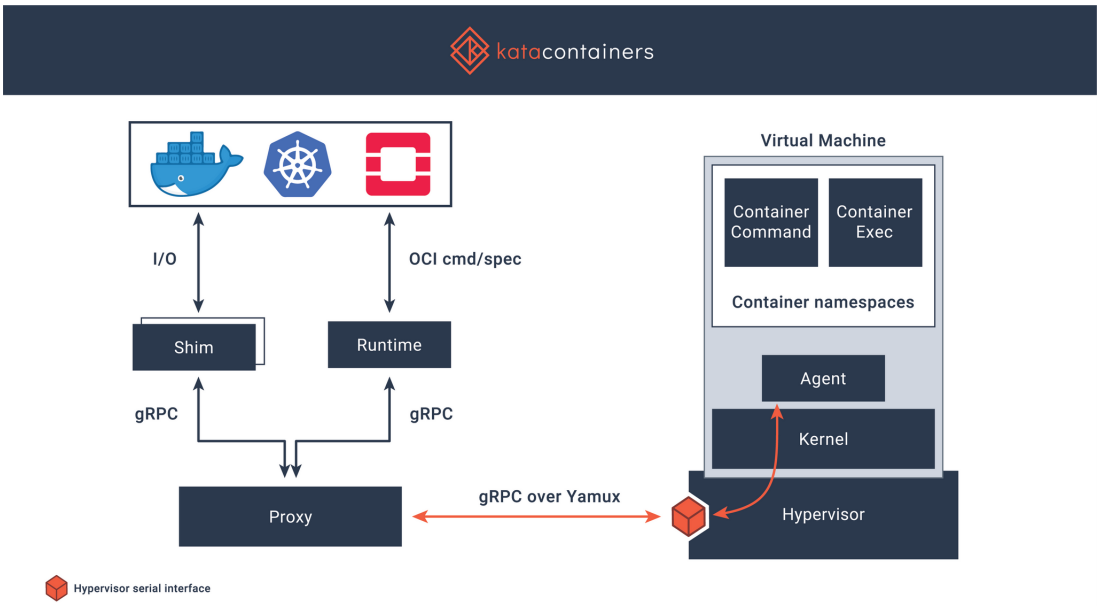

Figure 3.7: Kata's architecture. Source: https://katacontainers.io

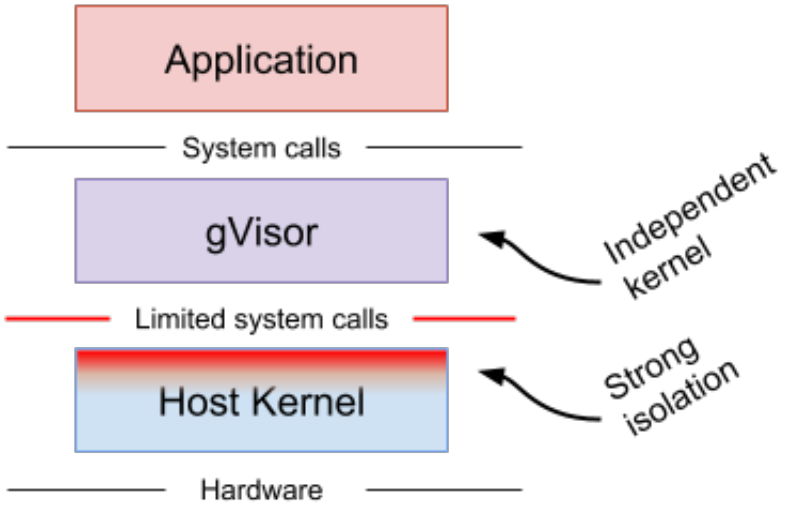

Figure 3.8: gVisor's approach. Source: https://gvisor.dev/docs/

A different technique is called *sandboxing*. It consists of adding one layer of abstraction between the containerized application and the host kernel, using a library OS running in the container environment. All the system calls invoked by the application are therefore intercepted by the guest OS and executed in user space; as a result, the application cannot directly interact with the host. Since the guest OS can provide many functionalities on its own without the need to access the host kernel, several system calls can be blocked from the container runtime by using a seccomp profile, thus reducing the attack surface. This approach is used by gVisor [39] (Figure 3.8).

Micro-VM and sandboxing approaches increase the isolation of a containerized application, protecting the host system from malicious or compromised workloads. While a micro-VM exploits strong hardware isolation, sandboxing adds a software layer between the application and the host, as well as a reduced attack surface. However, better security entails worse performance: it has been tested experimentally that both approaches have a longer startup time and worse runtime performance compared to runC and crun [40, 41, 42].

Integration with Kubernetes. As all the runtimes we have seen so far comply with the OCI specification, they can seamlessly replace the default runC. Yet, to use a specific low-level container runtime, it first needs to be installed on the nodes, and the high-level container runtime must be properly configured. Additionally, the Kubernetes' control plane needs to be aware of which runtime must be used for each container, according to the Pod specification.

Starting from Kubernetes v1.20, *RuntimeClass* has become a stable feature, used to deal with the selection of low-level container runtimes. Assuming that the VM admin has installed the runtimes on the nodes and configured the high-level container runtimes accordingly, the cluster administrator can create a `RuntimeClass` resource object for each new low-level runtime, indicating a custom name (e.g., *gVisor*) and the corresponding handler that will be used to spawn containers. Once RuntimeClasses are configured in the cluster, they can be used by specifying the `runtimeClassName` field in the Pod spec. If the RuntimeClass does not exist, or the high-level runtime cannot run the handler for some reason, the pod will fail to start.

By default, the control plane assumes that a certain RuntimeClass is supported by all the nodes in the cluster. If this is not the case, two additional things are required: (1) ensure that the nodes supporting the RuntimeClass have a common label, and (2) reference this label in the `RuntimeClass` resource specification by setting the `scheduling.nodeSelector` field. This way, pods running with that RuntimeClass will not be scheduled on nodes that do not support it. Besides, the `overhead` field can be defined for each RuntimeClass, used to take into account the additional resources (e.g., CPU and memory) needed to run a pod on the corresponding low-level runtime, ensuring that such resources are accounted for when making scheduling decisions.

3.4.2.4. Detect malicious activity

So far, we looked at prevention techniques that the cluster administrator can enable to have a better control of what is running in the cluster. Yet this is not enough if a malicious container is somehow able to circumvent the restrictions it is subject to, or if a container that was considered trusted suddenly becomes compromised by an external attacker. As such, runtime monitoring tools should be used to detect malicious activity in the cluster.

Falco [43] can be used for this purpose. It is an open source runtime security tool part of the Cloud Native Computing Foundation (CNCF). Essentially, Falco monitors the usage of system calls by userspace processes, generating alerts if the behavior of a process does not comply with a set of rules declared by the user (i.e., the cluster administrator in our case). It uses a driver to monitor the system call information, which can be installed as a kernel module, an eBPF probe, or as userspace instrumentation. This information is then collected and parsed by a software component that runs in user space, which sends alerts if anomalies are detected. A configuration file is used to define how Falco is run, the rules to assert, and how to perform alerts.

Falco is a very powerful tool to detect unusual activity at runtime. For example, it can be used to detect privilege escalation, execution of shell and SSH binaries, namespace changes, spawning of new processes, read/writes to certain directories, etc. Alerts can be as simple as logs written to standard output or a file, or they can be calls to remote HTTP/RPC endpoints. Therefore, even though Falco is only a detection tool per se, it can be used in cooperation with other tools to actively take countermeasures as soon as an alert is generated.

Starting from version 0.13.0, Falco can additionally manage Kubernetes audit events as event sources. Kubernetes provides a log of all the requests sent to the API server, hence the audit log can track all the changes made to the cluster. This allows Falco to actively monitor the cluster's activity, including the creation and update of pods. Consequently, Falco can be used to enforce pod's security settings as well.

3.4.3. Prevention and detection of anomalies during the execution of the target container

The previous sections discussed cluster-wide solutions that either the VM admin or the cluster admin (or both) can implement to increase the overall security of the Kubernetes cluster. This section, instead, focuses on the tenants' perspective, describing solutions and countermeasures that they can adopt in order to both prevent and detect possible attacks.

3.4.3.1. Deploy bug-free code

A comprehensive discussion about developing bug-free code is far beyond the scope of this deliverable. In general, this issue concerns all the software and is therefore not strictly related to containers and Kubernetes. Over the years, several solutions have been proposed and implemented, such as memory-safe programming languages [44, 45, 46], C/C++ extensions [47, 48, 49], formal techniques [50, 51, 52], and testing [53, 54]. Each approach has its pros and cons, and

thus none of them can be adopted as the universal solution. When prevention is insufficient, detection techniques can be exploited to reveal anomalies or detect attacks during the execution of an application (see 3.4.3.3 and 3.4.3.4). In this case, vulnerabilities are discovered only after the application is released, and can be fixed in future releases.

3.4.3.2. Create robust container images

We refer to section 3.3 for a broad discussion about container image security.

3.4.3.3. Logging and monitoring

The simplest detection mechanism is logging, useful to understand what is happening inside an application. Some options are available in Kubernetes, from basic logging (i.e., printing some text to standard output, and then checking the logs with `kubectl logs`), to collecting the logs at node or cluster level. However, checking the logs requires manual work and does not scale with big deployments.

Monitoring resources such as memory and CPU usage can be another way to detect anomalies in a container. An unexpectedly high amount of resources consumed might be an indicator that something is not working properly. The detection, again, can be done manually using web-UI dashboards such as Kubernetes Dashboard or Grafana, or using alerts that are automatically triggered when a certain event occurs, e.g., when the usage of a certain resource crosses a specified threshold. Compared to logging, this method does not require to inspect each container separately, as the whole workload can be monitored at the same time with the help of visual components such as charts, plots, diagrams. It is difficult to detect whether the integrity of a container has been violated or not just by looking at the resource usage; in many cases, an attack may not cause significant deviations from the normal utilization, keeping it undetected.

3.4.3.4. Control-flow attestation

Another approach to detect runtime attacks is by monitoring the control flow of an application. Control-Flow Integrity (CFI) has been proposed since several years [55]. It is a technique used to ensure that a program follows a predetermined Control-Flow Graph (CFG) [56], calculated at compile time or retrieved statically from the program's binary. CFI techniques usually consist of adding checks to the program's binary, which are executed at runtime to verify whether the destination address of an indirect branch is valid or not, with respect to the CFG.

Using a similar mechanism, new techniques for Control-Flow Attestation (CFA) (also known as Runtime Attestation) are emerging in recent years. CFA aims to provide to a remote verifier a trace of the execution of a program, which is then checked against the CFG to verify that the program is following a valid path. Deviations from the CFG might be an indicator that an attack is in progress, therefore CFA can be used as a powerful detection tool to promptly stop any attack attempts as soon as they are discovered.

Analogously to Remote Attestation, CFA aims to provide a remote party a proof of authenticity of a program; the difference is that CFA performs checks and measurements on the behavior of the program at runtime, thus providing stronger integrity guarantees than Remote Attestation, which can only guarantee the authenticity of a program during boot time.

Several projects implement CFA for small devices such as microcontrollers [57, 58, 59, 60, 61, 62, 63, 64], some of them proposing variations to the CFG to also detect data-flow alterations, others introducing hardware modifications instead of using software checks. Concerning high-end systems, ScaRR [65] is a CFA implementation based on a novel model for representing the execution path of a program, which guarantees fast verification of attestation evidences. Another approach [66] uses a different mechanism to detect Return-Oriented Programming (ROP) attacks, using taint tracking and TPM-based attestation for remote verification.

Currently, there are no solutions for integrating CFA in Kubernetes; as such, future work might be carried out in this direction. This technique can provide strong integrity guarantees of containerized applications, even in the presence of critical vulnerabilities. However, it is worth noting that CFA may cause non-negligible performance penalties that must be considered when deploying a performance-critical workload.

3.5. Trusted Computing Architectures / Confidential Computing for 5G

A technology that emerged in recent years are TEEs [67], where code and data of a process are isolated inside a dedicated area of the memory to provide runtime protection in terms of confidentiality and integrity.

Often, memory isolation is combined with runtime memory encryption in order to defend not only against software attackers but also against attacker with access to physical memory. The size of the protected data and programs may vary wildly, from key protection facilities for trusted cryptographic operations to whole VMs running in a TEE. Commonly, trusted execution environments also have attestation capabilities, i.e., they allow application deployers to obtain cryptographic proofs that their software was deployed as intended in an authentic TEE.

We provide a broad overview of different TEE technologies in 3.5.1; in the remainder of this section, instead, we focus on Intel Software Guard Extensions (SGX) [68], the most popular TEE in use nowadays.

3.5.1. Trusted Execution Environments

Code and data of a running application protected by a TEE are encrypted by the CPU and stored on protected memory pages. Only the CPU can decrypt the content of the TEE instance, using dedicated keys that cannot be directly accessed in software. This means that the host system, or anything that is located outside the TEE instance, cannot access what is running inside. This causes a reduction of the Trusted Computing Base (TCB): an application running in a TEE instance does not need to trust the underlying system, but only the hardware and the code within the TEE instance. In other words, an application can run securely even in compromised/untrusted nodes, if the protection mechanisms of the TEE are in place. In addition, Remote Attestation is performed to ensure that a remote workload is indeed running in a protected environment. It consists of obtaining a cryptographic proof attesting that the workload is running untampered in a TEE instance; this proof is generated and digitally signed by the hardware, so that the remote verifier can attest its authenticity. The attestation response not only attests that the workload is correctly running in a TEE instance, but also includes information about the identity of the workload. Thus, the verifier can then validate the authenticity of the running application. Some TEE technologies such as SGX also attest the platform's (TCB) state, to ensure that it is up-to-date.

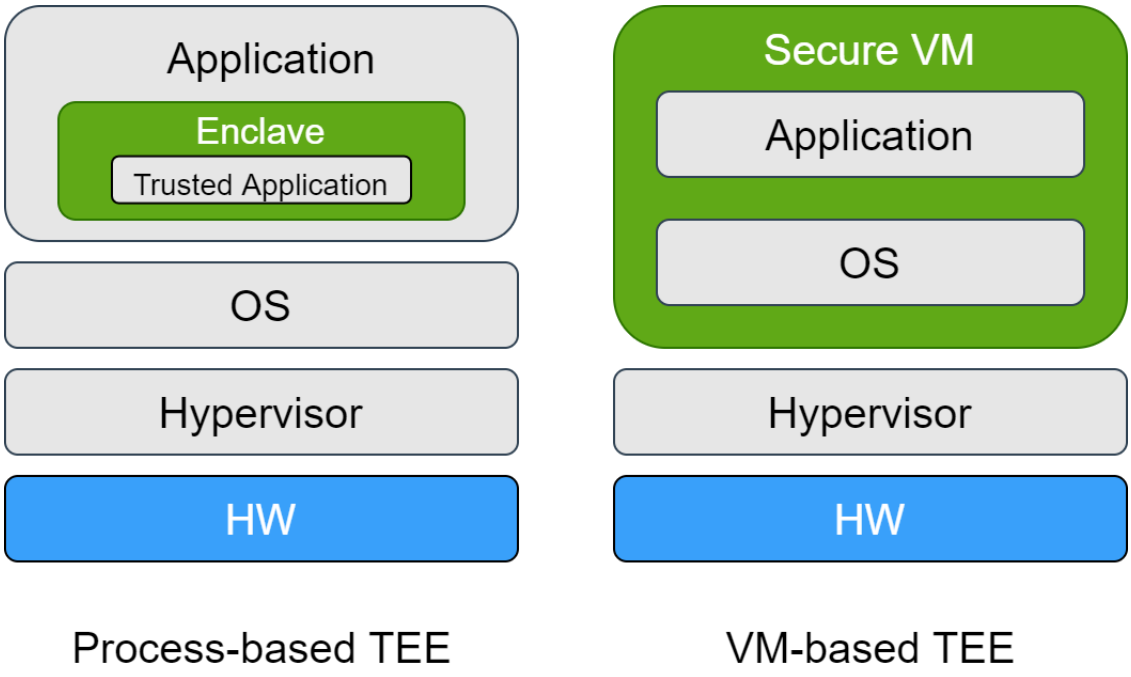

Figure 3.9: Differences between a Process-based TEE and a VM-based TEE. In a process-based TEE, only a small portion of code and data (the "trusted application") is protected in an enclave. In a VM-based TEE, instead, a whole virtual machine (the "secure VM") is protected by the hardware, including the OS and all the processes running on it.

Different TEE solutions come with very different architectures. We distinguish between two main families of TEEs, as shown in Figure 3.9:

- **Process-based**. Process-based TEEs allow running a process (or part of it) in a TEE instance, called *enclave*. Other processes, the OS and the hypervisor are all outside of the enclave, thus making the TCB small. The hardware isolates and protects the enclave from the host, as a sort of reverse sandbox; crossing the boundary between enclave and host (e.g., to execute system calls) is possible through a well-defined interface, though a context switch from

protected to unprotected code (or vice-versa) is generally costly in terms of performance. A legacy application cannot run in a process-based TEE as-is, but it needs some refactoring, either at source code level or at binary level, depending on the solution used. The most popular process-based TEEs is Intel SGX.

- **VM-based**. VM-based TEEs use a completely different approach, that is running an entire VM in a TEE instance. This solution is clearly less lightweight than process-based TEEs, as the TCB now includes an entire OS. However, VM-based TEEs have some advantages: there is no enclave boundary between a process and the OS, which means less performance penalties to e.g., execute system calls; besides, an application does not need any redesign or refactoring to run in a secure VM. AMD Secure Encrypted Virtualization (SEV) [69] and the new Intel Trust Domain Extensions (TDX) [70] use an VM-based TEE approach.

	Memory encryption	Memory integrity	Memory size limitation	Enclave number	Enclave granularity	Fast boot	Efficient interaction	Remote attestation
SGX	Yes	Yes	All 256MB EPC	Unlimited	Fine-grained	No	No	Pre/post-boot
SEV	Yes	No	All phy-mem	Limited (15)	Coarse-grained	No	No	Pre-boot only
Enclavisor	Yes	Partial [*]	All phy-mem	Unlimited	Multiple	Yes	Yes	Pre/post-boot

Figure 3.10: A comparison on Intel SGX, AMD SEV and Enclavisor from different dimensions. Source: [5]

Custom solutions can be implemented combining process-based and VM-based technologies to get the advantages of both approaches, while at the same time trying to overcome their limitations. This is the case of Enclavisor [5], a research project built on top of AMD SEV to run single processes in enclaves. In short, it consists of a small, trusted software layer that runs inside a secure VM to manage an arbitrary number of enclaves within it, each of them isolated from the others. In addition, Enclavisor removes the OS from the TCB by running it on a separate (normal) VM, sharing a portion of memory with the secure VM for the execution of system calls. Figure 3.10, extracted from Enclavisor's paper, highlights the limitations of SGX and SEV and shows how Enclavisor tries to mix their approaches to get the benefits of both.

3.5.1.1. Developing an SGX application

Essentially, there are two ways to develop an SGX application:

1. Use an Software Development Kit (SDK) or an abstraction framework to develop a new enclaved application from scratch, or to re-factor existing applications.
2. Use a middleware such as a library OS to run legacy applications in an enclave with little to no modifications, depending on the middleware used.

The first approach requires more manual work, but it allows to move to the enclave only the critical part of the application, leaving outside of the TCB most of the code. The second approach, instead, has a much bigger TCB since the whole application would run in the enclave, as well as the middleware component. Hence, any vulnerability in the code would potentially compromise the application. However, a middleware might provide additional features such as an encrypted filesystem, and it might also introduce some performance improvements such as reducing the number of context switches between the enclave and the untrusted world.

SDKs and abstraction frameworks. Only a few SDKs are available for writing SGX applications. The most popular are:

- **Intel SGX SDK** [71]. The official SDK provided by Intel, written in C/C++. The application is divided in two parts: the untrusted component and the enclave. The former is responsible for creating and initializing the latter. Communication between the two components is made using special function calls called ECALLs (from the untrusted world to the enclave) and OCALLs (the other way around). These functions are declared in a special Enclave Definition Language (EDL) file. Moreover, the SDK comes with special tools such as `sgx_edger8r` to parse the EDL file and generate glue code, and `sgx_sign` to sign enclaves.
- **Rust SGX SDK** [72]. This SDK enables the development of enclave application using Rust, a modern and efficient programming language that provides memory safety by design. It runs on top of the Intel SGX SDK, meaning that the development of an SGX application is analogous as before, but it comes with extra security guarantees thanks to Rust's memory safety.
- **Fortanix Enclave Development Platform (EDP)** [73]. This SDK is entirely written in Rust and it is fully integrated with the Rust compiler, which means that we can build an SGX enclave using the usual Rust tools, e.g., cargo, and

most of Rust's standard library. Writing an enclaved application is straightforward: the code looks exactly like a normal application, while the SDK makes the SGX bindings at compile time. An untrusted component that creates and initializes the enclave at runtime is still required, however EDP provides a default component called `ftxsgx-runner` that can be used for this purpose. If needed, a custom runner can be easily implemented as well. The usercall interface is used by the enclave to communicate with the external world (a mechanism similar to ECALLs/OCALLs in the SGX SDK), e.g., to perform system calls.

On top of these SDKs, high-level frameworks can be built to ease the development of enclaved applications across different TEEs, as well as to provide APIs for some operations such as Remote Attestation. One of the most popular projects in this regard is Open Enclave [74], which is an open source project that aims to generalize the development of enclaved applications across TEEs from different vendors. Currently, it supports Intel SGX, while support for other TEEs is under development. Another project is Google Asylo [75], with analogous goals.

Middleware approaches. We distinguish between the following middleware approaches to port legacy applications in SGX:

- **Library OSes**. This approach consists of including an entire library OS in the application's binary. It brings the largest TCB but requires zero changes on the legacy application's code. Furthermore, a library OS can bring several performance improvements, and can provide an API for some common tasks such as attestation. Nowadays, the most popular projects that use this model are Graphene [76], Occlum [77], Mystikos [78], Anjuna [79], and Fortanix [80].
- **Runtimes**. This method is based on running a runtime inside the enclave, which is essentially a software that allows the execution of other applications. Examples of runtimes are: EGo [81], targeted to applications written using Go; interpreters for languages such as Python [82] and Javascript [83]; WebAssembly runtimes such as WebAssembly Micro Runtime (WAMR) [84]. A runtime can run a legacy application inside the enclave without requiring any code modifications. Compared to a library OS, this approach may cause more performance penalties, however a runtime running inside the enclave might as well come with additional features. For example, projects such as Enarx [85], TWINE [86], Veracruz [87] and Oak [88] use a WebAssembly runtime to run an application inside the enclave, but they also provide new functionalities such as deployment of encrypted workloads and automatic attestation.
- **Wrappers**. This model, adopted by Panoply [89] and SCONE [90], provides wrappers to functions that cannot be executed inside the enclave. The idea is that the application does not call those functions directly; instead, it calls the wrappers, which perform a context switch to successfully execute low-level system calls and any other instructions that need to be executed outside of the enclave. This approach requires some modifications in the legacy application, either at source code level or at binary level, to link the API calls to the wrappers instead of the real functions.

3.5.1.2. Run an SGX enclave from a container

There are essentially two ways to run an SGX enclave from within a Docker container:

- **Manually building the container image to include the enclave and any other dependencies**. This method is not different from running the SGX enclave in the host machine. The steps necessary to create and run the enclave are the same, which depend on the approach used as explained in the previous section (e.g., using a framework such as Intel SGX SDK).
- **Using special tools to automatically convert a normal container image to an SGX-enabled one**. These tools exploit the middleware approach (see previous section), making modifications in the container image to run the application in an SGX enclave. For example, such tool is provided by SCONE, Fortanix and Graphene, each of them using their own middleware solution.

In both cases, the container must be able to access the SGX device in the host machine. To do so, it is sufficient to link the device to the container at startup time: for example, using Docker one would use the flag `--device=/dev/sgx_enclave` in the `docker run` command. Additionally, the Application Enclave Services Manager (AESM) service (a component used to perform some common tasks such as attestation) should be made available to the enclave if needed. It is possible to run the AESM service inside the container itself or use the instance running in the host by mounting the `/var/run/aesmd` volume in the container's filesystem.

3.5.1.3. Integration of TEEs in Kubernetes

To integrate an TEE in Kubernetes, two fundamental points must be addressed:

- **Management of TEE resources**. The control plane must be aware of the TEE. First, TEE-enabled nodes should be identified and marked with a special label. Second, a plugin should monitor the (limited) TEE resources in a node such as the amount of encrypted memory available. Third, a scheduler should assign TEE-enabled pods to TEE-enabled nodes that have enough resources.
- **Management of TEE features such as attestation**. Each TEE comes with different features, and each feature might also provide different alternatives (for example, attestation in SGX has two modes, Enhanced Privacy ID (EPID) [91] and Data Center Attestation Primitives (DCAP) [92]). The control plane should be aware of all these features. Concerning attestation, for instance, a tenant could attest their own enclaves themselves, but it would be preferred if the control plane managed the attestation of all the TEE-enabled pods directly, in order to make the TEE completely transparent to the tenant, as well as to automatically take countermeasures on enclaves whose attestation failed. The tenant in this case could simply query the control plane to verify that the attestation has succeeded. Other features might require special care and actions by the control plane as well, such as sealing of sensitive data.

Next, we focus on SGX: we explain how an orchestrater such as Kubernetes can be made aware of SGX enclaves and nodes, and we introduce existing solutions that also enable other functionalities.

Management of SGX resources. Previous work [93] gave a detailed explanation on how to make a Kubernetes cluster aware of SGX. In summary, the paper points out the following aspects:

- **Identify SGX-enabled nodes**. *Device plugins* can be used in Kubernetes to detect hardware resources in a node (e.g., GPU). Hence, a device plugin for SGX can be installed in a cluster to mark nodes as able to run SGX enclaves by exposing their Enclave Page Cache (EPC) resources (i.e., the encrypted memory available on a node) to the cluster.
- **Provide metrics on resource usage**. It is essential to provide metrics to monitor the EPC usage on each node, to be able to schedule pods in a node without exceeding the maximum encrypted memory available. Although SGX supports over-commitment of the EPC, the performance impact to move enclaves in and out of the encrypted memory would be enormous, therefore it should be avoided. To provide these metrics, the paper suggests modifying the SGX driver to expose the total number of EPC pages available and the number of EPC pages used at a certain time. After that, a probe deployed as a DaemonSet in the cluster can be used to fetch the metrics and make them available to the control plane.
- **Schedule the execution of SGX pods on SGX nodes**. To do this, an SGX-aware scheduler must be installed on the cluster. This scheduler would check the SGX metrics to identify SGX-enabled nodes available for the execution of new SGX-enabled pods. Kubernetes supports multiple schedulers to run concurrently on the same cluster, therefore it is easy to deploy custom schedulers.
- **Enforce limits on EPC usage**. It is important to prevent pods from abusing the EPC, to avoid over-commitment and allow all pods to use a fair amount of EPC resources. The paper proposes making slight modifications in the SGX driver and the kubelet to monitor the EPC usage of pods, identifying the ones that are not respecting the usage declared in the pod specification (under `spec.containers.resources.limits`) and allowing the control plane to take countermeasures such as preemption. A proper EPC usage enforcement would require the execution of a dedicated cgroup controller in the kernel, however the paper argues that this solution would require a huge engineering and development effort.

Existing solutions. Several SGX plugins are available for Kubernetes, developed in the context of research [93], by organizations that wanted to support SGX in their cloud infrastructure [94, 95], or for commercial use [96]. Recently, Intel released its own official SGX plugin too [97]. All of these plugins implement similar functionalities, as explained in the paragraph above. However, none of them currently implement a per-container enforcement of EPC memory usage, allowing a malicious tenant to potentially saturate all the EPC in a node. This issue is particularly critical on multi-tenant clusters, and therefore needs special attention. Fortunately, work is in progress to implement a proper enforcement of EPC usage on Intel's plugin [98].

SGX plugins give the ability to run and manage SGX enclaves in a Kubernetes cluster. Yet, some manual effort from the tenants and the cluster admin is still required, as there is no support for additional functionalities such as automatic

conversion of a container image to enable SGX, remote attestation, establishment of secure channels between enclaves, data sealing, etc.

Other solutions try to go beyond these limitations, aiming to provide Kubernetes the security guarantees of Intel SGX in a more transparent way. The most relevant projects are:

- **Inclavare containers** [99]. Developed by Alibaba Cloud and Ant Group and cooperated with Intel, Inclavare containers is an open source, low-level container runtime that enables confidential computing with Intel SGX. It is fully OCI compliant, meaning that it can be seamlessly used within Kubernetes to run any OCI container images. It can run an unmodified application in the enclave, currently supporting Occlum, Graphene and WAMR as middleware solutions. Additionally, it constructs a general infrastructure for the attestation of workloads.
- **Marblerun** [100]. Marblerun is an open source framework developed by Edgeless Systems for creating distributed confidential applications using a service mesh approach. A tenant who wants to deploy a workload can submit a Manifest file declaring the topology of the distributed application, then Marblerun takes care of all the rest, namely the initialization and attestation of all the microservices, and the setup of encrypted connections between microservices using mutual TLS terminated within the enclaves. After the initialization phase, the tenant can, at any time, get an attestation statement of the whole application. That is, the attestation provides the guarantee that every microservice is correctly running inside an SGX enclave and attested, and all the connections between microservices are encrypted and authenticated. The same attestation statement can be also obtained by end users directly. The Marblerun's control plane also offers persistence of data using virtual sealing keys that allow microservices to be migrated on other nodes while still being able to unseal their data. Additionally, it manages certificate provision, secrets management and recovery.

3.5.1.4. Limitations of TEEs

The adoption of TEEs in Kubernetes gives a tenant strong confidentiality and integrity guarantees of their workloads, especially in a multi-tenant cluster in which tenants are mutually distrusting. Besides, an TEE would remove from the TCB the cloud service providers, the VM admin, and to some extent even the cluster administrator. Nevertheless, some aspects need to be considered:

- **Heterogeneity**. As previously described, each TEE targets a specific processor, e.g. SGX for Intel and SEV for AMD. In a cloud infrastructure, nodes from different hardware vendors might be used, and some nodes might not even provide the hardware requirements for a specific TEE. As such, some tasks would be challenging, e.g., supporting multiple TEEs, running critical pods on specific zones, migrating pods, and so forth.
- **Limited resources**. The hardware resources of a TEE are not unlimited. For example, the maximum EPC available in SGX is currently only about 128 to 256 MB. Additionally, AMD SEV only allows at most 15 secure VMs to run concurrently, as each running VM is identified by a 4-bit ID. In both cases, over-using the TEE resources would lead to severe performance degradation. These limitations might be removed in the future; for instance, the upcoming Intel Ice Lake processors will support up to 1 TB of EPC.
- **Performance degradation**. Running a workload in a TEE would unavoidably reduce its performance, compared to running it natively [101, 102, 103]. A VM-based TEE is less susceptible to performance degradation than a process-based TEE, as the OS runs in the TEE instance together with the application, and therefore there are no additional penalties for e.g., executing system calls. In general, crossing the boundary with the TEE instance has a high cost, which considerably impacts on the performance of specific operations such as I/O. CPU-intensive applications such as machine learning algorithms are less affected instead.
- **Lack of functionalities**. In process-based TEEs, some system functionalities might not be available inside the enclave for security reasons, as the kernel is outside of the TCB and therefore it cannot be trusted. For example, in Fortanix EDP features such as environment variables, time and timeouts, filesystem operations, and multi-processing are not supported. Nevertheless, a middleware approach such as a library OS might provide some of them in a secure way.
- **Security**. Some threats are not part of the attacker model of a TEE, such as physical/hardware attacks and denial-of-service attacks. Certain TEEs also suffer from serious vulnerabilities: Intel SGX has been the target of numerous attacks [104] such as cache attacks, branch prediction attacks, and speculative execution attacks; AMD SEV, instead, does not ensure memory integrity, which might lead to tampering with the encrypted memory of a secure VM [105] or even extracting secrets from it [106].

4. Summary and Outlook

In the 5GhOSTS project we analyze and improve the security of service-based implementations of 5G networks, with a special focus on security and privacy aspects of the development and deployment of containerized Virtual Network Functions (VNFs). Research and development activities in 5GhOSTS center around three novel use cases, Vehicle to Infrastructure (V2I), Smart Office, and Remote Surgery, all of which impose challenging requirements regarding low-latency, privacy, and service resilience on our work. We address these requirements by developing a Kubernetes-based edge platforms for 4/5G, which is informed by comprehensive security and privacy threat modelling by both researchers from the domains of telecommunications law and computer security. These efforts have been elaborated on in previous project output (cf. [1, 2, 3]). In this deliverable we focus on the trust model for our 4/5G edge platform, and survey security and privacy preserving technologies that will enable us to implement this trust model with Kubernetes-based VNF deployments, ensuring the integrity and overall security of the deployed VNFs in the presence of strong attackers and even untrusted infrastructure, and thereby protecting the data and privacy of users of the service.

In Chapter 2 we present a threat and trust modelling effort for our V2I use case, presenting the different actors and their trust relations. For example, the user side covers driver, pedestrian, and the road sensor infrastructure such as traffic lights or cameras. As we illustrate, attackers must be considered as distrusted and all systems must take a precaution against attackers. Attackers can easily reach the user side, attempt to spoof other users' identity, or compromise them as a step stone to attack the edge platform. That is why the user side actors do not trust each other. However, the users must semi-trust the edge service provider and mobile network operator, because they depend on a lot of information and functionality from them. For example, in the traffic management service use case, they should share data, and get optimized directions with lower traffic or real-time information about their immediate environment. Conversely, Mobile Network Operator (MNO) distrust the user side. However they semi-trust the edge manager as it is one of a few well-identified customers of the MNO's edge and mobile network infrastructure. Nevertheless, if protections against a compromise of either side are dropped, this may pose a risk if such a compromise actually occurs. Overall, there are no completely trusted actors in the V2I edge cloud scenario. Explicating and understanding this is important for our choices of technology. To satisfy the security and privacy needs of all actors we need to aim for a notion of a Zero Trust Architecture, where active security controls give novel guarantees about authenticity and integrity to the stakeholders involved in each interaction, establishing an immediate notion of trustworthiness.

In Chapter 3 we then review technology and architectural choices to harden and to securely deploy and use Kubernetes containers that implement VNFs. While our focus there is on Trusted Execution Technology (TEEs) and similar approaches that enable a notion of confidential computing, where VNFs can execute securely with a minimal Trusted Computing Base, we present these approaches in the context of established means of reducing the attack surface of containers, static vulnerability detection, and secure image deployment. Our survey identifies a number of weaknesses and shortcomings in established approaches to use TEEs in combination with Kubernetes containers in general and VNF scenarios in particular. These relate to overall system performance, the need for support for highly dynamic and heterogeneous for our specific use cases, and unclear security objectives and attacker models, which need to be reworked and put in context to enable use cases in 5G edge.

The survey leads us to four specific research tracks that the 5GhOSTS project will follow-up on in the near future. In particular, we will further investigate the security of orchestration, isolation and attestation mechanisms, focusing on advanced and efficient approaches to runtime attestation, extend isolation primitives to intra-container isolation, incident detection and recovery, work explicitly towards securing communications interfaces between VNFs, and high-performance and low-latency approaches to integrate these mechanisms. Our work will be backed up by formal research into verifying, e.g., specific isolation approaches, interfaces, as well as security policies. Furthermore, we aim to work towards building a technology stack that enables privacy impact assessment of user-facing services.

Bibliography

[1] 5GhOSTS, "D1.2: Use cases and initial conceptualization of the 4 ESRs' research projects, inter-project relations, and interdisciplinary common approach," Confidential Project Deliverable, 2020.

[2] ——, "D3.1: Design and configuration of a base orchestration framework," Confidential Project Deliverable, 2021.

[3] ——, "D1.4: Privacy requirements for 5G telecom systems running virtualized multi-component services," Public Project Deliverable, 2021.

[4] "Connaisseur: An admission controller that integrates Image Signature Verification into a Kubernetes cluster," https://github.com/sse-secure-systems/connaisseur, accessed: Jul 13th, 2021.

[5] J. Gu, X. Wu, B. Zhu, Y. Xia, B. Zang, H. Guan, and H. Chen, "Enclavisor: A hardware-software co-design for enclaves on untrusted cloud," *IEEE Transactions on Computers*, 2020.

[6] S. Rizou, E. Alexandropoulou-Egyptiadou, and K. E. Psannis, "Gdpr interference with next generation 5g and iot networks," *IEEE Access*, vol. 8, pp. 108 052–108 061, 2020.

[7] K. Murdock, D. Oswald, F. D. Garcia, J. Van Bulck, F. Piessens, and D. Gruss, "Plundervolt: How a little bit of undervolting can create a lot of trouble," *IEEE Security Privacy*, vol. 18, no. 5, pp. 28–37, 2020.

[8] G. Chen, S. Chen, Y. Xiao, Y. Zhang, Z. Lin, and T. H. Lai, "Sgxpectre: Stealing intel secrets from sgx enclaves via speculative execution," in *2019 IEEE European Symposium on Security and Privacy (EuroS P)*, 2019, pp. 142–157.

[9] M. ETSI, "Multi-access edge computing (mec) framework and reference architecture," *ETSI GS MEC*, vol. 3, p. V2, 2019.

[10] D. Sabella, V. Sukhomlinov, L. Trang, S. Kekki, P. Paglierani, R. Rossbach, X. Li, Y. Fang, D. Druta, F. Giust *et al.*, "Developing software for multi-access edge computing," *ETSI white paper*, vol. 20, pp. 1–38, 2019.

[11] D. Sabella, A. Reznik, and R. Frazao, *Multi-access edge computing in action*. CRC Press, 2019.

[12] "Portieris: A Kubernetes Admission Controller for verifying image trust with Notary," https://github.com/IBM/portieris, accessed: Jul 13th, 2021.

[13] M. De Benedictis and A. Lioy, "Integrity verification of docker containers for a lightweight cloud environment," *Future Generation Computer Systems*, vol. 97, pp. 236–246, 2019.

[14] W. Luo, Q. Shen, Y. Xia, and Z. Wu, "Container-ima: a privacy-preserving integrity measurement architecture for containers," in *22nd International Symposium on Research in Attacks, Intrusions and Defenses ({RAID} 2019)*, 2019, pp. 487–500.

[15] "Google Container-Optimized OS Overview," https://cloud.google.com/container-optimized-os/docs/concepts/features-and-benefits, accessed: Jul 13th, 2021.

[16] "Amazon Bottlerocket: Linux-based operating system purpose-built to run containers," https://aws.amazon.com/bottlerocket/, accessed: Jul 13th, 2021.

[17] "Rancher k3OS: Purpose-built OS for Kubernetes, fully managed by Kubernetes." https://github.com/rancher/k3os, accessed: Jul 13th, 2021.

[18] "K3s: The certified Kubernetes distribution built for IoT & Edge computing ," https://k3s.io, accessed: Jul 13th, 2021.

[19] "Red Hat Enterprise Linux CoreOS," https://access.redhat.com/documentation/en-us/openshift_container_platform/4.1/html/architecture/architecture-rhcos, accessed: Jul 13th, 2021.

[20] "Terraform: Deliver Infrastructure as Code," https://www.terraform.io/, accessed: Jul 13th, 2021.

[21] "Red Hat Ansible Automation Platform," https://www.ansible.com/, accessed: Jul 13th, 2021.

[22] J. D. Osborn and D. C. Challener, "Trusted platform module evolution," *Johns Hopkins APL Technical Digest (Applied Physics Laboratory)*, vol. 32, no. 2, pp. 536–543, 2013.

[23] R. Sailer, X. Zhang, T. Jaeger, and L. Van Doorn, "Design and implementation of a tcg-based integrity measurement architecture." in *USENIX Security symposium*, vol. 13, no. 2004, 2004, pp. 223–238.

[24] W. Futral and J. Greene, "Fundamental principles of intel® txt," in *Intel® Trusted Execution Technology for Server Platforms*. Springer, 2013, pp. 15–36.

[25] "Intel Security Libraries for Data Center (SecL-DC)," https://01.org/intel-secl, accessed: Jul 13th, 2021.

[26] "Microsoft Azure Attestation," https://docs.microsoft.com/en-us/azure/attestation/overview, accessed: Jul 13th, 2021.

[27] "Kubernetes extensions for Intel Secl-DC," https://github.com/intel-secl/k8s-extensions, accessed: Jul 13th, 2021.

[28] B. McCarty, *SELinux*. O'Reilly Japan, 2005.

[29] "AppArmor: Linux kernel security module," https://apparmor.net, accessed: Jul 13th, 2021.

[30] "K-Rail: Kubernetes security tool for policy enforcement," https://github.com/cruise-automation/k-rail, accessed: Jul 13th, 2021.

[31] "Kyverno: Kubernetes Native Policy Management," https://github.com/kyverno/kyverno, accessed: Jul 13th, 2021.

[32] "Gatekeeper: Policy Controller for Kubernetes," https://github.com/open-policy-agent/gatekeeper, accessed: Jul 13th, 2021.

[33] "Kritis: Deploy-time Policy Enforcer for Kubernetes applications," https://github.com/grafeas/kritis, accessed: Jul 13th, 2021.

[34] "Kubernetes Dynamic Webhook controller for interacting with Anchore Engine," https://github.com/anchore/kubernetes-admission-controller, accessed: Jul 13th, 2021.

[35] "Kubernetes Admission Controller for Image Scanning using OPA," https://github.com/sysdiglabs/opa-image-scanner, accessed: Jul 13th, 2021.

[36] "Clair: Vulnerability Static Analysis for Containers," https://github.com/quay/clair, accessed: Jul 13th, 2021.

[37] A. Randazzo and I. Tinnirello, "Kata containers: An emerging architecture for enabling mec services in fast and secure way," in *2019 Sixth International Conference on Internet of Things: Systems, Management and Security (IOTSMS)*. IEEE, 2019, pp. 209–214.

[38] A. Agache, M. Brooker, A. Iordache, A. Liguori, R. Neugebauer, P. Piwonka, and D.-M. Popa, "Firecracker: Lightweight virtualization for serverless applications," in *17th {usenix} symposium on networked systems design and implementation ({nsdi} 20)*, 2020, pp. 419–434.

[39] "gVisor: application kernel for containers that provides efficient defense-in-depth," https://gvisor.dev, accessed: Jul 13th, 2021.

[40] W. Viktorsson, C. Klein, and J. Tordsson, "Security-performance trade-offs of kubernetes container runtimes," in *2020 28th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*. IEEE, 2020, pp. 1–4.

[41] G. E. de Velp, E. Rivière, and R. Sadre, "Understanding the performance of container execution environments," in *Proceedings of the 2020 6th International Workshop on Container Technologies and Container Clouds*, 2020, pp. 37–42.

[42] E. G. Young, P. Zhu, T. Caraza-Harter, A. C. Arpaci-Dusseau, and R. H. Arpaci-Dusseau, "The true cost of containing: A gvisor case study," in *11th {USENIX} Workshop on Hot Topics in Cloud Computing (HotCloud 19)*, 2019.

[43] "Falco: Cloud Native Runtime Security," https://falco.org, accessed: Jul 13th, 2021.

[44] J. Gosling, B. Joy, G. Steele, and G. Bracha, *The Java language specification*. Addison-Wesley Professional, 2000.

[45] A. A. Donovan and B. W. Kernighan, *The Go programming language*. Addison-Wesley Professional, 2015.

[46] S. Klabnik and C. Nichols, *The Rust Programming Language (Covers Rust 2018)*. No Starch Press, 2019.

[47] D. Dhurjati, S. Kowshik, V. Adve, and C. Lattner, "Memory safety without runtime checks or garbage collection," in *Proceedings of the 2003 ACM SIGPLAN conference on Language, compiler, and tool for embedded systems*, 2003, pp. 69–80.

[48] G. C. Necula, S. McPeak, and W. Weimer, "Ccured: Type-safe retrofitting of legacy code," in *Proceedings of the 29th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, 2002, pp. 128–139.

[49] S. Nagarakatte, J. Zhao, M. M. Martin, and S. Zdancewic, "Softbound: Highly compatible and complete spatial memory safety for c," in *Proceedings of the 30th ACM SIGPLAN Conference on Programming Language Design and Implementation*, 2009, pp. 245–258.

[50] B. Jacobs, J. Smans, P. Philippaerts, F. Vogels, W. Penninckx, and F. Piessens, "Verifast: A powerful, sound, predictable, fast verifier for c and java," in *NASA formal methods symposium*. Springer, 2011, pp. 41–55.

[51] M. Ouimet and K. Lundqvist, "Formal software verification: Model checking and theorem proving," *Embedded Systems Laboratory Technical Report ESL-TIK-00214*, 2007.

[52] J. Woodcock, P. G. Larsen, J. Bicarregui, and J. Fitzgerald, "Formal methods: Practice and experience," *ACM computing surveys (CSUR)*, vol. 41, no. 4, pp. 1–36, 2009.

[53] C. Kaner, J. Falk, and H. Q. Nguyen, *Testing computer software*. John Wiley & Sons, 1999.

[54] A. Takanen, J. D. Demott, C. Miller, and A. Kettunen, *Fuzzing for software security testing and quality assurance*. Artech House, 2018.

[55] M. Abadi, M. Budiu, U. Erlingsson, and J. Ligatti, "Control-flow integrity principles, implementations, and applications," *ACM Transactions on Information and System Security (TISSEC)*, vol. 13, no. 1, pp. 1–40, 2009.

[56] F. E. Allen, "Control flow analysis," *ACM Sigplan Notices*, vol. 5, no. 7, pp. 1–19, 1970.

[57] S. Zeitouni, G. Dessouky, O. Arias, D. Sullivan, A. Ibrahim, Y. Jin, and A.-R. Sadeghi, "Atrium: Runtime attestation resilient under memory attacks," in *2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 2017, pp. 384–391.

[58] T. Abera, N. Asokan, L. Davi, J.-E. Ekberg, T. Nyman, A. Paverd, A.-R. Sadeghi, and G. Tsudik, "C-flat: control-flow attestation for embedded systems software," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, 2016, pp. 743–754.

[59] B. Kuang, A. Fu, L. Zhou, W. Susilo, and Y. Zhang, "Do-ra: data-oriented runtime attestation for iot devices," *Computers & Security*, vol. 97, p. 101945, 2020.

[60] G. Dessouky, T. Abera, A. Ibrahim, and A.-R. Sadeghi, "Litehax: lightweight hardware-assisted attestation of program execution," in *2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 2018, pp. 1–8.

[61] J. Hu, D. Huo, M. Wang, Y. Wang, Y. Zhang, and Y. Li, "A probability prediction based mutable control-flow attestation scheme on embedded platforms," in *2019 18th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/13th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE)*. IEEE, 2019, pp. 530–537.

[62] G. Dessouky, S. Zeitouni, T. Nyman, A. Paverd, L. Davi, P. Koeberl, N. Asokan, and A.-R. Sadeghi, "Lo-fat: Low-overhead control flow attestation in hardware," in *Proceedings of the 54th Annual Design Automation Conference 2017*, 2017, pp. 1–6.

[63] N. Koutroumpouchos, C. Ntantogian, S.-A. Menesidou, K. Liang, P. Gouvas, C. Xenakis, and T. Giannetsos, "Secure edge computing with lightweight control-flow property-based attestation," in *2019 IEEE Conference on Network Softwarization (NetSoft)*. IEEE, 2019, pp. 84–92.

[64] I. D. O. Nunes, S. Jakkamsetti, and G. Tsudik, "Tiny-cfa: A minimalistic approach for control-flow attestation using verified proofs of execution," *arXiv preprint arXiv:2011.07400*, 2020.

[65] F. Toffalini, E. Losiouk, A. Biondo, J. Zhou, and M. Conti, "Scarr: Scalable runtime remote attestation for complex systems," in *22nd International Symposium on Research in Attacks, Intrusions and Defenses ({RAID} 2019)*, 2019, pp. 121–134.

[66] L. Davi, A.-R. Sadeghi, and M. Winandy, "Dynamic integrity measurement and attestation: towards defense against return-oriented programming attacks," in *Proceedings of the 2009 ACM workshop on Scalable trusted computing*, 2009, pp. 49–54.

[67] M. Sabt, M. Achemlal, and A. Bouabdallah, "Trusted execution environment: what it is, and what it is not," in *2015 IEEE Trustcom/BigDataSE/ISPA*, vol. 1. IEEE, 2015, pp. 57–64.

[68] V. Costan and S. Devadas, "Intel sgx explained." *IACR Cryptol. ePrint Arch.*, vol. 2016, no. 86, pp. 1–118, 2016.

[69] D. Kaplan, J. Powell, and T. Woller, "Amd memory encryption," *White paper*, 2016.

[70] "Intel Trust Domain Extensions (TDX)," https://software.intel.com/content/www/us/en/develop/articles/intel-trust-domain-extensions.html, accessed: Jul 13th, 2021.

[71] "Intel SGX SDK," https://software.intel.com/content/www/us/en/develop/topics/software-guard-extensions/sdk.html, accessed: Jul 13th, 2021.

[72] H. Wang, P. Wang, Y. Ding, M. Sun, Y. Jing, R. Duan, L. Li, Y. Zhang, T. Wei, and Z. Lin, "Towards memory safe enclave programming with rust-sgx," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, 2019, pp. 2333–2350.

[73] "Fortanix Enclave Development Platform (EDP)," https://edp.fortanix.com/, accessed: Jul 13th, 2021.

[74] "Open Enclave SDK," https://openenclave.io/sdk/, accessed: Jul 13th, 2021.

[75] "Asylo: An open and flexible framework for enclave applications," https://asylo.dev, accessed: Jul 13th, 2021.

[76] C.-C. Tsai, D. E. Porter, and M. Vij, "Graphene-sgx: A practical library {OS} for unmodified applications on {SGX}," in *2017 {USENIX} Annual Technical Conference ({USENIX}{ATC} 17)*, 2017, pp. 645–658.

[77] Y. Shen, H. Tian, Y. Chen, K. Chen, R. Wang, Y. Xu, Y. Xia, and S. Yan, "Occlum: Secure and efficient multitasking inside a single enclave of intel sgx," in *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, 2020, pp. 955–970.

[78] "Mystikos: Tools and runtime for launching unmodified container images in Trusted Execution Environments," https://github.com/deislabs/mystikos, accessed: Jul 13th, 2021.

[79] "Anjuna confidential cloud platform," https://www.anjuna.io/, accessed: Jul 13th, 2021.

[80] "Fortanix Runtime Encryption," https://fortanix.com/products/runtime-encryption, accessed: Jul 13th, 2021.

[81] "EGo: Build confidential Go apps with ease," https://www.ego.dev/, accessed: Jul 13th, 2021.

[82] H. Wang, M. Sun, Q. Feng, P. Wang, T. Li, and Y. Ding, "Towards memory safe python enclave for security sensitive computation," *arXiv preprint arXiv:2005.05996*, 2020.

[83] D. Goltzsche, C. Wulf, D. Muthukumaran, K. Rieck, P. Pietzuch, and R. Kapitza, "Trustjs: Trusted client-side execution of javascript," in *Proceedings of the 10th European Workshop on Systems Security*, 2017, pp. 1–6.

[84] "WebAssembly Micro Runtime (WAMR)," https://github.com/bytecodealliance/wasm-micro-runtime, accessed: Jul 13th, 2021.

[85] "Enarx," https://enarx.dev/, accessed: Jul 13th, 2021.

[86] J. Ménétrey, M. Pasin, P. Felber, and V. Schiavoni, "Twine: An embedded trusted runtime for webassembly," *arXiv preprint arXiv:2103.15860*, 2021.

[87] "Veracruz: privacy-preserving compute," https://github.com/veracruz-project/veracruz, accessed: Jul 13th, 2021.

[88] "Project Oak: Meaningful control of data in distributed systems," https://github.com/project-oak/oak, accessed: Jul 13th, 2021.

[89] S. Shinde, D. Le Tien, S. Tople, and P. Saxena, "Panoply: Low-tcb linux applications with sgx enclaves." in *NDSS*, 2017.

[90] S. Arnautov, B. Trach, F. Gregor, T. Knauth, A. Martin, C. Priebe, J. Lind, D. Muthukumaran, D. O'keeffe, M. L. Stillwell *et al.*, "{SCONE}: Secure linux containers with intel {SGX}," in *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*, 2016, pp. 689–703.

[91] S. Johnson, V. Scarlata, C. Rozas, E. Brickell, and F. Mckeen, "Intel® software guard extensions: Epid provisioning and attestation services," *White Paper*, vol. 1, no. 1-10, p. 119, 2016.

[92] V. Scarlata, S. Johnson, J. Beaney, and P. Zmijewski, "Supporting third party attestation for intel® sgx with intel® data center attestation primitives," *White paper*, 2018.

[93] S. Vaucher, R. Pires, P. Felber, M. Pasin, V. Schiavoni, and C. Fetzer, "Sgx-aware container orchestration for heterogeneous clusters," in *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2018, pp. 730–741.

[94] "Confidential computing nodes on Azure Kubernetes Service," https://docs.microsoft.com/en-us/azure/confidential-computing/confidential-nodes-aks-overview, accessed: Jul 13th, 2021.

[95] "Alibaba's SGX plugin for Kubernetes," https://partners-intl.aliyun.com/help/doc-detail/177706.htm, accessed: Jul 13th, 2021.

[96] "SCONE's SGX plugin for Kubernetes," https://sconedocs.github.io/helm_sgxdevplugin, accessed: Jul 13th, 2021.

[97] "Intel's SGX plugin for Kubernetes," https://intel.github.io/intel-device-plugins-for-kubernetes/cmd/sgx_plugin/README.html, accessed: Jul 13th, 2021.

[98] "Intel SGX plugin for Kubernetes: implementing EPC usage enforcement ," https://github.com/intel/intel-device-plugins-for-kubernetes/pull/654, accessed: Jul 13th, 2021.

[99] "Inclavare Containers: An open source enclave container runtime and security architecture for confidential computing scenarios," https://inclavare-containers.io/en, accessed: Jul 13th, 2021.

[100] "Marblerun: The control plane for confidential computing," https://www.marblerun.sh/, accessed: Jul 13th, 2021.

[101] A. T. Gjerdrum, R. Pettersen, H. D. Johansen, and D. Johansen, "Performance of trusted computing in cloud infrastructures with intel sgx." in *CLOSER*, 2017, pp. 668–675.

[102] S. Mofrad, F. Zhang, S. Lu, and W. Shi, "A comparison study of intel sgx and amd memory encryption technology," in *Proceedings of the 7th International Workshop on Hardware and Architectural Support for Security and Privacy*, 2018, pp. 1–8.

[103] T. Dinh Ngoc, B. Bui, S. Bitchebe, A. Tchana, V. Schiavoni, P. Felber, and D. Hagimont, "Everything you should know about intel sgx performance on virtualized systems," *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, vol. 3, no. 1, pp. 1–21, 2019.

[104] A. Nilsson, P. N. Bideh, and J. Brorsson, "A survey of published attacks on intel sgx," *arXiv preprint arXiv:2006.13598*, 2020.

[105] L. Wilke, J. Wichelmann, M. Morbitzer, and T. Eisenbarth, "Sevurity: No security without integrity: Breaking integrity-free memory encryption with minimal assumptions," in *2020 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2020, pp. 1483–1496.

[106] M. Morbitzer, M. Huber, J. Horsch, and S. Wessel, "Severed: Subverting amd's virtual machine encryption," in *Proceedings of the 11th European Workshop on Systems Security*, 2018, pp. 1–6.

[107] Y. Jang, J. Lee, S. Lee, and T. Kim, "Sgx-bomb: Locking down the processor via rowhammer attack," in *Proceedings of the 2nd Workshop on System Software for Trusted Execution*, ser. SysTEX'17. New York, NY, USA: Association for Computing Machinery, 2017. [Online]. Available: https://doi.org/10.1145/3152701.3152709

[108] M. Schwarz, S. Weiser, and D. Gruss, "Practical enclave malware with intel SGX," *CoRR*, vol. abs/1902.03256, 2019. [Online]. Available: http://arxiv.org/abs/1902.03256

[109] P. Stewin and I. Bystrov, "Understanding dma malware," in *Detection of Intrusions and Malware, and Vulnerability Assessment*, U. Flegel, E. Markatos, and W. Robertson, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 21–41.

A. Appendix

A.1. Threats to Edge Application

Category	Threats	Affected Assets	Possible Countermeasures
Spoofing	**SpoofingImage:** The identity of a service image fetched from the local image repository can be spoofed, e.g., by presenting fake tags, replacing the image in repository or image cache, or by intercepting it during transmission. This could occur due to a privileged attacker with control plane access, or by compromising the image repository, e.g., due to lack of proper authentications. Consequently, a compromised image may be pulled instead of the desired one, containing vulnerabilities or even malware.	Container Image, Container Image Distribution Infrastructure, Execution Context, Application Data	• Sign the images according to Trusted Image Policy • Use digests to identify images • Admission controllers to validate images / check signatures • Authentication and Access Control • Vulnerability Scan Images by image registry • Monitor containers across their life cycle • Attestation of images in TEEs
	SpoofingAppData: A compromised application or external attacker can use stolen other leaked credentials (API keys, tokens, crypto keys) to pretend to be another application.	Application Data	• Authentication and Access Control • Least privilege application design • Network Policies • Rotate Credentials regularly
	SpoofingSysResource: The compromised edge cloud provider can provide broken or fake enclaves to the app by obtaining attestation keys, e.g., via Plundervolt [7] or transient execution attacks like Foreshadow or SGXpectre [8].	System Resources, Application Data, Execution Context	• Attest that system software version up-to-date, e.g., microcode update and BIOS update • Restricted access to power control for applications • Software countermeasures against transient execution attacks
	SpoofingDataPlane: IP/ARP/DNS spoofing in the cluster can be achieved via privileged networking, e.g., by pods having improper permission, by a malicious or compromised container/pod achieving container-to-host break-out, etc. In some cases, e.g., ARP spoofing, it is enough for an external attacker to obtain remote code execution within a pod in the cluster, to be able to spoof the traffic of other pods.	Inter-Container Network (Data Plane)	• Authentication • Limit pod permissions • Monitor and restrict traffic between pods • Mutually authenticated TLS connections between containers
Tampering	**TamperingImage:** A container image is maliciously modified, customized, or replaced in the time between creation and deployment. This could occur due to an external attacker intercepting transmission of the image or compromising the image repository. A compromised image repository may deploy outdated versions of a given image, potentially undoing patches and fixes of past vulnerabilities, thus re-enabling the exploitation of previously discovered software security flaws in the application. Another possibility is a compromised edge cloud orchestrater, purposefully deploying the wrong, potentially malicious images that may strive to modify sensitive information or attempt to attack its users. Attackers with sufficient admin privileges may configure malicious admission controllers that infect deployed images via mutating webhooks.	Container Image, Container Image Distribution Infrastructure, Application Data	• Authentication and Access Control • Vulnerability scanning of images • Least privilege design principle • Monitor containers across their life cycle • Policies for admission controllers

Category	Threats	Affected Assets	Possible Countermeasures
	TamperingExecContext: The majority of software security vulnerabilities are memory safety violations, i.e., programming errors that break the memory abstraction of unsafe high-level programming languages such as C. Examples for such errors are buffer overflows, use after free, memory race conditions, uninitialized variables and null-pointer references. Memory safety violations may crash a program or lead to unexpected changes in program behavior. In the worst case they may enable remote code execution and thus break system integrity. Such vulnerabilities allow attackers to manipulate the control flow of applications by providing it with specially crafted input values. Usually, affecting the control flow requires a memory safety violation first, i.e., overwriting executable code or function pointers, smashing the stack to overwrite return addresses (return-oriented programming, ROP), or corrupting application data to change the outcome of conditional program branches (data-oriented programming, DOP). Rowhammer and code injection attacks can also allow manipulation of the execution context. Achieving remote code execution within a container of an application allows an attacker to send requests to other services or the k8s API server on behalf of it.	Execution Context, Application Data	• Bounds Checking • Assign NULL to pointer after freeing memory • Compiler Hardening Options • Memory-Safe Languages • Memory and Address Sanitizer tools • Stack Smashing Protection • Canary Values • Shadow Stack • Hardware support for Memory Safety and Control Flow Integrity • Static code analysis • Data Execution Protection • Memory Layout Randomization • Program Diversification • Reduce container attack surface, limit impact: – no unused code – no bash/CLI – no SSH access
	TamperingAppDataUser: This threat has two threat vectors: data in use and data at rest. • *Data in use:* Software vulnerabilities may allow attackers to tamper with an application's execution contexts or even take control of its control flow and privileges by malicious parties. Subsequently the attacker may obtain a write primitive and modify mission-critical or sensitive data application data including personal data that is being processed. Through its exposed API an application may also be targeted by confused deputy attacks, attempting to trick it into using its privileges to tamper with the desired data. Also, Rowhammer attacks can corrupt application data in RAM on the same node. • *Data at rest:* If there are applications that require root privileges on the node or if an application can get privileges due to vulnerabilities on the host, these apps may replace stored data of other applications with former versions of it as well as tamper with the data. Depending on the application this may enable security exploits if the authenticity (freshness) of the data is not checked.	Application data	Data in Use: • Memory Integrity Protection • In-Process Memory Isolation • Software security measures (see above) Data at Rest: • Secure Storage with proper Authentication and Authorization • Storage node hardening • Distributed / redundant storage systems, e.g., Ceph • Cryptographic integrity / freshness protection
	TamperingAppDataAdmin: The edge cloud provider has write privileges on all storage nodes for the edge application and can thus manipulate app data at rest. Integrity protection helps to discover such manipulations but cannot guarantee the availability of data. Nevertheless, a cloud provider has usually a self-interest to provide reliable and robust data storage.	Application data	• External trusted storage • Secure Storage with proper Authentication and Authorization, using TEEs to protect credentials • Tamper-proof logging and audit, e.g., distributed ledger • Cryptographic Integrity / Freshness protection

Category	Threats	Affected Assets	Possible Countermeasures
	TamperingSysResource: On a host computation node the edge cloud provider or an application with elevated privilege may sneakily modify the virtual environment provided to the application, subverting its assumptions that underlie security-related functionality. For instance, address space and file management could be manipulated to slightly change an application's behavior. The OS could provide a faulty system time or counters, compromising functionality based on having a precise notion of time and date. Finally, a privileged attacker could provide fake crypto primitives, fake enclaves, or weak entropy random numbers to weaken cryptography and thus downgrade application security.	System Resources, Container Image, Application Data, Execution Context	• Attestation • Trusted Execution Environments and Trusted Platform Modules • Micro-VMs or Unikernels to reduce hosts OS dependencies
	TamperingDataPlane: Any compromised edge orchestrater may strive to weaken network configurations and isolation, e.g., redirect traffic to unauthorized recipients. Also, by default, inter-pod communication is not authenticated, encrypted, or integrity-protected, hence a privileged attacker could stage man-in-the-middle attacks. The design of network services, functions, and other edge cloud applications should follow a rigorous security design, featuring principles of compartmentalization, defense in depth, least privilege, and zero trust. If a design violates these principles, vulnerabilities in one part of an application may allow an attacker to spread to other parts of the edge platform.	Inter-Container Network (Data Plane), Application Data	• Mutually authenticated TLS connections between containers • Network Policies • Secure Application and Network Design
Repudiation	**RepudationAppLog:** For different reasons, such as privacy regulations and intrusion detection, it is necessary to keep logs of all accesses to a storage node. A compromised application may try to falsify such logs in order to hide any previous transgressions	Application Log	• Secure Log Design
Information Disclosure	**InfoDisclImage:** Private images may be leaked from storage nodes or images caches with improper authentication or access control. Privileged attackers may dump the image from memory or leak it by monitoring control plane traffic.	Container Image, Container Image Distribution Infrastructure	• Access Control • Authentication • Encrypted storage and transmission • Trusted Execution Environments
	InfoDisclExecContextUser: Any software security vulnerability (such as control-flow hijacking, remote code execution, memory safety violation) may lead to the disclosure of sensitive application data such as cryptographic keys, if exploited by an attacker. Also Memory Reuse, Side-channel, and Confused Deputy attacks, among others, may cause the leak of information to co-located malicious applications or external attackers. Using third-party libraries in the software may contribute to such threats of the execution context.	Execution Context, Application Data	• In-Process Memory Isolation • Software security measures (see *TamperingExecContext* above)
	InfoDisclExecContextAdmin: As the Edge cloud provider holds a privileged access in the node, it can dump application data in use from RAM and registers. Memory dump analysis can then be used to extract important information from these dumps.	Execution Context, Application Data	• Trusted Execution Environments • Trusted Platform Module and Key Protection Technology • Secure Multiparty Computation • Homomorphic Encryption

Category	Threats	Affected Assets	Possible Countermeasures
	InfoDisclAppDataUser: Many apps require secrets to enable secure communication between components such as include connection strings and SSH private keys. If these secrets are stored in image or configuration files, anyone with access to these files can easily parse it to learn these secrets. Also, if a comprised app obtained a memory read primitive on a host or storage node, it could be able to access confidential data of other apps.	Application data	• Secure image design & secret handling • Trusted Execution Environments • Encrypted storage
	InfoDisclAppDataAdmin: An edge cloud provider may typically get access to any unencrypted information stored on disks and simply copy such data. Even if disks are encrypted, backups and system logs of running applications may contain decrypted information which may thus be available to honest but curious infrastructure providers.	Application Data	• Trusted Execution Environments • Encrypted storage • Micro-VMs or Unikernels to reduce host OS interactions
	InfoDisclSysResource: Side channel attacks may compromise TEEs and steal protected application data. Advanced attackers may be able to subvert the highest-privilege system software to install stealthy persistent threats, e.g., root-kits that monitor the system and exfiltrate application data.	System Resources, Container Image, Application Data, Execution Context	• Secure Boot and Attestation • TEEs with protective measures against side channels
	InfoDisclDataPlane: If an attacker can monitor or intercept data plane traffic (e.g., via privileged network capabilities or having access to networking hardware), they might leak sensitive application information, e.g., via man-in-the-middle attacks or network analysis on metadata. Having access to the eBPF facility of the host system may also allow to exploit potential vulnerabilities in that system and learn about the network activities or message contents of co-hosted applications.	Inter-Container Network (Data Plane), Application Data	• Encrypted data plane traffic • Mutually authenticated communication • Restricted networking capabilities • Disable user mode eBPF access
Denial of Service	**DoSImage:** A compromised application with access to a local image repository can try to fetch big images or a lot of images at the same time, polluting the image caches and thus delaying the image delivery for other application in the same node. Also, without proper access control, the external or privileged app can delete or overwrite the images, preventing other applications from loading (quickly). Furthermore, naming or tag conflicts may prevent the accessing the right images. If an attacker manages to take control of an image repository, it may refuse an app to fetch any images.	Container Image, Container Image Distribution Infrastructure	• Dynamic scaling of image distribution infrastructure • Monitoring and Anomaly Detection • Limit resource consumption of a container • Resource isolation, i.e., no sharing of repositories and caches • Resource Quotas and Limit Ranges
	DoSExecContext: Malicious applications may try to crash a node, stopping co-hosted applications and causing delays by the need to restart them. For instance, the SGX-Bomb attack [107] uses the Rowhammer method against enclave memory to trigger a processor lockdown. A noisy neighbor application may try to monopolies CPU, network, and memory system capacity to adversely affect performance of network services.	Execution Context	• Attest latest updates are installed • Intel Resource Director Technology • Resource Quotas and Limit Ranges • Monitoring and Anomaly Detection
	DoSAppData: A compromised app can create a lot of traffic to slow down the storage service so that other apps cannot access their data in time. Attackers may attempt to delete application data.	Application data	• Resource Quotas and Limit Ranges • Intel Resource Director Technology • Monitoring and Anomaly Detection • Backup strategy

Category	Threats	Affected Assets	Possible Countermeasures
	DoSSysResource: The edge cloud provider can prevent the app to use enclaves (downgrading). A noisy neighbor application may try to monopolize low-level system resources like the SGX enclave page cache, or hardware accelerators to adversely affect performance of network services.	System Resources	• Separate pod placement via affinity rules • Resource Quotas and Limit Ranges
	DoSDataPlane: Edge applications may contain libraries that support a multitude of different secure communication protocols and versions, some of them outdated and vulnerable. Network attackers may then downgrade the security of connections by forcing a fallback to vulnerable protocols or versions. Even if communication is secure and networks are logically isolated, malicious "noisy neighbors" may intentionally degrade the quality of service of co-hosted edge applications by making excessive use of network resources. An adversary with privileged network access may also deny a service network connectivity by dropping its packets.	Inter-Container Network (Data Plane)	• Minimal Application Design • Resource Quotas and Limit Ranges • Intel Resource Director Technology • Monitoring and Anomaly Detection • Restrict networking capabilities
Elevation of Privilege	**ElevPrivAppData:** Because of a weak configuration of authorization (RBAC exploited by app admin, application admin can add some roles/tag etc, insecure storage of credentials), a compromised application could get privileged access to application data in storage nodes.	Application data	• Configuration Hardening • Least Privilege principle • OPA (control what roles and tags can be assigned)
	ElevPrivMANO: A Kubernetes cluster offers a lot of configuration options, many of them relevant for system integrity and isolation of different workloads. If there are vulnerabilities or insufficient hardening in the cluster, an attacker, e.g., a neighboring tenant admin, may elevate their management and orchestration privileges to interfere with an application, e.g., by reconfiguring its security posture, leaking k8s secrets, or impeding its performance. Similar threats are posed by compromised cluster admins.	Management and Orchestration, Edge App Provider assets	• Authentication • Authorization • OPA (limit admin actions) • Monitoring and Anomaly Detection
	ElevPrivDataPlane: Applications may give attackers inadvertent access to services and components by exposing insufficiently protected APIs to other tenants or even public networks. By default, all pods within a cluster may communicate with each other. Subsequently, attackers may leak sensitive information or abuse further vulnerabilities to achieve remote code execution.	Inter-Container Network (Data Plane), Application Data, Execution Context	• Restrict and audit API exposure • Network policies • Monitor network traffic • Anomaly Detection • Authentication and Access Control

A.2. Threats to Edge Platform

Category	Threats	Affected Assets	Possible Countermeasures
Spoofing	**SpoofingPhysical:** A Fake Base Station (FBS) transmitting synchronization signals with sufficient transmission power to surpass neighboring legitimate cells may attract end users to connect to it. If the FBS also provides a fake edge computing platform with corresponding fake applications, it may lead to leakage of private information targeted for the legitimate edge platform.	Application Data	• Authentication of Edge platform and services towards end user • Integrity protection and authentication of broadcasting messages • False Base Station detection and take-down
	SpoofingControlData: Any compromised application or the external attacker can use leaked control plane credentials to pretend to be a control plane process. This may give full control of the edge cloud to an attacker with access to, e.g., the API server or etcd database.	Control Data, Edge App Provider assets	• Restricted connections only from the Kubernetes API • Enable RBAC with Least Privilege • Monitoring and Anomaly Detection • Integrity monitoring • Rotating credentials regularly
	SpoofingMANO: If a compromised application admin can create an admin role that is not restricted (weak authentication and access control), they can thus access MANO functionality. Also, if the authentication of accesses to MANO is not enabled, access credentials can easily be spoofed by malicious applications. Similar threats are posed due to stolen MANO credentials.	Management and Orchestration, Edge Cloud Provider assets, Edge App Provider assets	• Authentication • Enable RBAC with Least Privilege • OPA (restrict what application admins can do) • Monitoring and Anomaly Detection
	SpoofingDataPlane: A Kubernetes cluster typically performs domain name resolution via a CoreDNS service that is controlled by rules stored as a ConfigMap object. If an attacker is able to modify this file, they may poison it and take over the identity of other, legitimate services in the cluster.	Inter-Container Network (Data Plane)	• Authentication • Access Control • Monitoring and Anomaly Detection
Tampering	**TamperingPhysical:** The system integrity of edge platforms deployed in public spaces may be tampered with by attackers accessing the hardware in the real world. In order to perform maintenance activities and provide technical support, edge platforms cannot be completely sealed off. Possible avenues for attackers are then accessing the system through debug ports or other physical interfaces. Another option are bit fault attacks on memory.	Physical System, Edge Cloud Provider assets, Edge App Provider assets, Management and Orchestration	• Ruggedized Hardware • Physical Security and Surveillance • Disable unused ports • Authentication • Memory integrity protection • Enclaves with memory encryption
	TamperingMANO: If a compromised app or external attacker can obtain a privileged access to MANO due to vulnerable configurations or colluding tenant admins, they can reconfigure the management scripts or policies of MANO. Moreover, they can use Kubernetes controllers to schedule or persist malicious software on the cluster. Also, confused deputy attacks, memory safety vulnerabilities, and remote code execution against the MANO implementation can corrupt the management scripts or policies.	Management and Orchestration, Edge Cloud Provider assets, Edge App Provider assets	• Authentication and Access Control • Least Privilege • Monitoring and Anomaly Detection • OPA (restrict what admins can do)
	TamperingControlPlane: An adversary with sufficient networking privileges on a node or the cluster, potentially having obtained suitable communications keys, may attempt to interfere with control messages, modifying, adding, or removing packets over the control plane network.	Control Plane Network, Management and Orchestration, Edge Cloud Provider assets	• Enable Control Plane encryption and integrity protection • Best practices, e.g., credential rotation • Monitoring and Anomaly detection

Category	Threats	Affected Assets	Possible Countermeasures
	TamperingControlData: Due to weak configuration of the edge cloud, the control data can be changeable without authentication by compromised apps, tenant admins, or external attackers.	Control Data, Management and Orchestration	• Authentication and Access Control • OPA (restrict what admins can do) • Monitoring and Anomaly detection
Repudiation	**RepudiationLogs:** Compromised edge applications or external attackers can try to take actions to prevent logs from being useful, including filling up the log to make it hard to find an attack or forcing logs to "roll over". They may also do things to set off so many alarms that the real attack is lost in a deluge of noise. Sending logs over a network exposes them integrity threats as well. The tampering of logs is a step stone for repudiation of (formerly) logged events.	Kubernetes Audit Logs	• Secure Log Design • Monitoring and Anomaly detection
	RepudiationMANO: A privileged attacker with admin access may reconfigure the MANO and disable or clear the event logs.	Management and Orchestration, Kubernetes Audit Logs	• Monitoring and Anomaly detection • Access control (RBAC) • OPA (restrict what admins can do)
Information Disclosure	**InfoDisclAppData:** Kubernetes allows applications to store secrets like credentials and tokens in the etcd database. Attackers with the privilege to retrieve secrets, e.g., via access to an application's service account, may retrieve corresponding k8s secrets from the API service. Moreover, attackers with sufficient admin privileges may be able to install malicious admission controllers as validating webhooks to leak secrets, e.g., from API server requests.	Kubernetes Secrets, Application Data	• Access control (RBAC) • Monitoring and Anomaly Detection • OPA (restrict what admins can do)
	InfoDisclPhysical: Third-party personnel or insiders can steal or access storage media physically. Using cold boot techniques, they can access clear-text memory in RAM even if disks are encrypted. Physical side channels information may be used to extract sensitive information by statistical analysis. External attacker can leak information with power / EM / sound analysis in close proximity to the edge hardware.	Physical System, Edge Cloud Provider assets, Edge App Provider assets	• Disk encryption • Physical security • Disabling ports / physical hardening • Monitoring / alarms • Sound / EM Shielding
	InfoDisclImage: Private images can be leaked from repositories or registries with improper authentication or access control. Attackers controlling a compromised pod may use its credentials to pull private images from the repository.	Container Image	• Authentication and Access Control (RBAC) • OPA (restrict which images can be pulled) • Anomaly Detection
	InfoDisclMANO: A compromised app or an external attacker which obtain privileged access can easily leak information from MANO, e.g., user information, logs, credentials, configuration information, container images, and orchestration scripts. Memory safety vulnerabilities, as well as Confused Deputy and Side Channel attacks can lead the leakages of the configuration information, management scripts, or policies of MANO. Such information may serve as a stepping stone for further attacks.	Management and Orchestration	• Authentication and Access Control (RBAC) • Monitoring and Anomaly Detection • OPA (restrict what admins can do) • Software security hardening
	InfoDisclControlData: Attackers with access to the API server, the K8s dashboard pod, or the Kubelet API may retrieve control data and cluster status information, accessible even without authentication. This discovery effort could expose the state of the edge cloud to the reader and serve as a stepping stone for further attacks. Also network probing and mapping could be attempted by malicious applications to this end.	Control Data, Management and Orchestration	• Authentication • Monitoring and Anomaly Detection • Network Policies

Category	Threats	Affected Assets	Possible Countermeasures
	InfoDisclControlPlane: An adversary with sufficient admin privileges on a node or the cluster, potentially having obtained suitable communications keys, may attempt to monitor the control plane traffic in order to leak sensitive information, e.g., k8s secrets or access credentials.	Control Data, Application Data	• Enable Control Plane encryption and authentication • Rotate credentials regularly
	InfoDisclExecContext: Edge platform providers must harden the configuration of container runtimes to ensure the desired information flow separation between different tenants that share the same compute nodes. Any shortcomings in this area may lead to leakages of sensitive host or tenant information. For instance, an insecure runtime configuration of containers may allow to mount sensitive directories on the host.	Control Data, Management and Orchestration, Application Data	• Hardened host configuration • Secure container runtimes
	InfoDiscl5G: Malicious edge applications may attempt to obtain data belonging to the 5G core functions (AMF, SMF) deployed alongside applications on the edge platform. A successful attack may leak sensitive user information or credentials to access the core network itself.	5G Core Functions	• Network policies • Authentication and Access Control • Monitoring and Anomaly Detection
Denial of Service	**DoSMANO:** Attackers with sufficient admin privileges may make the cluster unavailable, e.g., by shutting down nodes and crucial services.	Management and Orchestration	• Authentication and Authorization • OPA (restrict what admins can do)
	DoSControlPlane: Compromised applications with sufficient access permissions could overload the Edge API server by sending a large number of requests. The control plane network operation can be delayed with noisy neighbor attacks on the data plane, if there are no separate network interfaces.	Control Plane Network, Management and Orchestration	• Configure encryption and authentication to secure control plane components • Quotas and Rate limiting • Separate network interfaces • DoS Detection and Response
	DoSControlData: Compromised apps can create a lot of traffic to slow down the storage service so that control data cannot be accessible. Attackers with sufficient privileges may attempt to destroy deployments or configuration data.	Control Data	• Quotas and Rate limiting • Anomaly Detection • OPA (restrict what admins can do) • Backup and redundancy routines
	DoSPhysical: An external attacker may try to sabotage the edge system by physically tampering with the edge hardware or its support components, e.g., power supply and cooling.	Physical System, Edge Computing and Communication Infrastructure	• Physical Security and Monitoring • Ruggedized Hardware • Fallback Power Supply
	DoS5G: Targeting the 5G core functions scheduled on the edge cloud, attackers may attempt to disrupt end user connections to the edge or even the mobile network in general. For example, the hand-over procedure implemented by AMF could be affected.	5G Core Functions	• Network policies • Authentication and Access Control • Monitoring and Anomaly Detection
Elevation of Privileges	**ElevPrivControlPlane:** If there is privileged networking in absence of authentication, man-in-the-middle attacks could intercept control network traffic and steal developer or administrator credentials within that traffic. Misconfiguration of the communication infrastructure, e.g., UPF, may allow unauthorized external access to edge platform.	Control Plane Network	• Configure encryption and authentication to secure control plane components • Monitor network utilization • Intrusion detection
	ElevPrivDataPlane: Applications that have privileged networking capabilities or work in collusion with a compromised tenant admin may circumvent network policies to reach restricted parts of the application communication network.	Data Plane Network, Application Data	• Monitoring and Anomaly Detection • Restrict network capabilities • Authentication and Access Control • OPA (restrict what admins can do)

Category	Threats	Affected Assets	Possible Countermeasures
	ElevPrivExecContext: Excessive privileges given to application containers, unsafe configurations like writable hostPath mounts, as well as vulnerabilities in the virtualization infrastructure, operating system, and the container runtime may allow a malicious application to escape its sandbox and acquire root privileges on the node. Such privileges may allow the attacker to spread to other parts of the edge cloud system. Subsequently also data and service integrity, confidentiality, and availability may be threatened.	Virtualization Infrastructure, Management and Orchestration	• Container Hardening • Restricting privileged containers • Secure runtimes (Kata, etc.) • Anomaly Detection
	ElevPrivImage: Attackers who have obtained certain admin privileges may deliberately run compromised images on the cluster that may in turn attack other applications or the cluster itself. Moreover, the privilege to deploy pods or controllers on the cluster may allow to allocate resources for running malicious software such as bot nets or crypto miners. Such undesired containers may also be injected stealthily as side cars into existing legitimate pods or be disguised as legitimate pods with familiar names in the kube-system namespace.	Container Images, Execution context, MANO, Virtualization Infrastructure	• Admission Controllers • Anomaly Detection
	ElevPrivSysResource: An application with access to special system resources may abuse such resource to attack the host system and elevate its privileges. For example, malicious applications running in enclaves may attempt to stealthily exploit memory safety violations in the host [108]. Another example is the use of peripheral DMA capabilities to circumvent memory isolation [109].	System Resources, Edge Cloud Provider assets	• Admission control to system resources • SR-IOV and IOMMU memory access restrictions on DMA
	ElevPrivControlData: If a comprised edge application obtains control of kubelet, it can manipulate control data and may send privileged requests to the API server.	Control Data	• Authentication and Access Control • Monitoring and Anomaly Detection
	ElevPrivPhysical: If an external attacker can reach the physical edge system, they may access the system through USB and debug ports or other physical interfaces. By tampering with the boot process they may get full control of the system.	Physical System, Edge Computing and Communication Infrastructure, Edge Cloud Provider assets, Edge Application Provider assets	• Secure Boot • Physical Security and Surveillance • Disable unused ports • Ruggedized Hardware
	ElevPrivMANO: Compromised apps or external attackers may be able to assign privileged roles to themselves in the cluster due to weak design, e.g., having permissions to create arbitrary role bindings, or in collusion with a tenant admin. Also, getting access to a pod's service account may allow attackers to perform actions using that pod's privileges.	Management and Orchestration, Edge Cloud Provider assets, Edge Application Provider assets	• Authentication and Access control • OPA (restrict what admins can do) • Anomaly detection
	ElevPriv5G: If an attacker manages to compromise a pod running the 5G core services (AMF and SMF) they get full access to all user data handled by these functions. Moreover the attacker may obtain credentials allowing lateral movement into the 5G core network or user devices.	5G core functions, User data	• Network Policies • Authentication and Access control • Software security hardening • Monitoring and Anomaly Detection

The 5GhOSTS project

July 19, 2021

5GhOSTS-D1.3-0.9/1.0

Horizon 2020