5G ExPerimentation Infrastructure hosting Cloud-nativE Netapps for public proTection and disaster RElief

Innovation Action – ICT-41-2020 - 5G PPP – 5G

Innovations for verticals with third party services

D2.7: Cloud-native security intermediate version

Delivery date: April 2023

Dissemination level: Public

Project Title:	**5G-EPICENTRE - 5G ExPerimentation Infrastructure hosting Cloud-nativE Netapps for public proTection and disaster RElief**
Duration:	**1 January 2021 – 31 December 2023**
Project URL	https://www.5gepicentre.eu/

This project has received funding from the European Union's Horizon 2020 Innovation Action programme under Grant Agreement No 101016521.

Document Information

Deliverable	D2.7: Cloud-native security intermediate version
Work Package	WP2: Cloud-native 5G NFV
Task(s)	T2.6: Attack surface decrease and network edge access control
Type	Report
Dissemination Level	Public
Due Date	M28, April 30, 2023
Submission Date	M28, April 30, 2023
Document Lead	Luis Cordeiro (ONE)
Contributors	André Gomes (ONE) Luís Cordeiro (ONE) Pedro Tomás (ONE)
Internal Review	Christos Skoufis (EBOS) Daniele Ronzani (ATH)

Document history

Version	Date	Changes	Contributor(s)
V0.1	13/03/2023	Initial deliverable structure	Pedro Tomás (ONE)
V1.0	11/04/2023	Deliverable ready for internal review	André Gomes (ONE) Luís Cordeiro (ONE) Pedro Tomás (ONE)
V1.1	14/04/2023	First internal review comments and feedback	Christos Skoufis (EBOS)
V1.2	18/04/2023	Second internal review comments and feedback	Daniele Ronzani (ATH)
V1.5	19/04/2023	Final version for Quality Review	André Gomes (ONE) Luís Cordeiro (ONE) Pedro Tomás (ONE)
V1.6	21/04/2023	Quality review (copyediting; proofreading; formatting).	Konstantinos Apostolakis (FORTH)
V2.0	28/04/2023	Final version for submission, after quality review	André Gomes (ONE) Luís Cordeiro (ONE) Pedro Tomás (ONE)

Project Partners

Logo	Partner	Country	Short name
	AIRBUS DS SLC	France	**ADS**
	NOVA TELECOMMUNICATIONS SINGLE MEMBER S.A.	Greece	**NOVA**
	Altice Labs SA	Portugal	**ALB**
	Fraunhofer-Gesellschaft zur Förderung der angewandten Forschung e.V.	Germany	**HHI**
	Foundation for Research and Technology Hellas	Greece	**FORTH**
	Universidad de Malaga	Spain	**UMA**
	Centre Tecnològic de Telecomunicacions de Catalunya	Spain	**CTTC**
	Istella SpA	Italy	**IST**
	One Source Consultoria Informatica LDA	Portugal	**ONE**
	Iquadrat Informatica SL	Spain	**IQU**
	Nemergent Solutions S.L.	Spain	**NEM**
	EBOS Technologies Limited	Cyprus	**EBOS**
	Athonet SRL	Italy	**ATH**
	RedZinc Services Limited	Ireland	**RZ**
	OptoPrecision GmbH	Germany	**OPTO**
	Youbiquo SRL	Italy	**YBQ**
	ORamaVR SA	Switzerland	**ORAMA**

List of abbreviations

Abbreviation	Definition
5GC	5G Core
ACL	Access Control Lists
AF	Application Function
AI	Artificial Intelligence
AICO	Analytics, Intelligence, Control and Orchestration
AMF	Access and Mobility Management Function
API	Application Programming Interface
APT	Advanced Persistent Threat
ARPF	Authentication Credential Repository and Processing Function
AUSF	Authentication Server Function
CI/CD	Continuous Integration / Continuous Deployment
CNF	Containerized Network Function
CNI	Container Network Interface
CNN	Convolutional Neural Network
CPU	Central Processing Unit
CUPS	Control and User Plane Separation
DDoS	Distributed Denial of Service
DevOps	Development and Operations
DNN	Data Networks Name
DoS	Denial of Service
eMBB	enhanced Mobile Broadband

FL	Federated Learning
gNB	gNodeB
HSPF	Holistic Security and Privacy Framework
HTTP	Hypertext Transfer Protocol
IAM	Identity and Access Management
IDS	Intrusion Detection System
IE	Information Element
IoT	Internet of Things
IP(sec)	Internet Protocol (Security)
IT	Information and Technology
IPX	Interconnection Network
JSON	JavaScript Object Notation
JWS	JSON Web Signature
MAC	Media Access Control
MCx	Mission Critical Services
ML	Machine Learning
mMTC	massive Machine Type Communications
MQTT	Message Queue Telemetry Transport
mTLS	mutual Transport Layer Security
MNO	Mobile Network Operator
MTTR	Mean Time To Repair
MVNO	Mobile Virtual Network Operator
NEF	Network Exposure Function
NF	Network Function

NFV	Network Function Virtualisation (Orchestrator)
NIDS	Network Intrusion and Detection System
NRF	Network Repository Function
NS	Network Service
OS	Operating System
OPA	Open-Policy-Agent
OSI	Open Systems Interconnection
PLMN	Public Land Mobile Network
PDR	Public Protection and Disaster Relief
PS	Port Scan
QoS	Quality of Service
RBAC	Role-based Access Control
REST	Representational State Transfer
S-NSSAI	Single Network Slice Selection Assistance Information
SA	Service Agent
SAST	Static Application Security Testing
SBA	Service-Based Architecture
SBD	Security By Design
SBI	Service-Based Interface
SBP	Service Based Provider
SCP	Service Communication Proxy
SEAF	Security Anchor Function
SELinux	Security-Enhanced Linux
SEPP	Secure Edge Protection Proxy

SIDF	Subscription Identifier De-Concealing Function
SME	Small AND Medium-sized Enterprise
SMF	Session Management Function
SoA	State of the Art
SSH	Secure Shell
SUCI	Subscriber Concealed Identifier
SUPI	Subscription Permanent Identifier
SVM	Support Vector Machine
TLS	Transport Layer Security
UDM	Unified Data Management
UE	User Equipment
UML	Unified Modelling Language
UPF	User Plane Function
URLLC	Ultra-reliable low-latency communication
VIM	Virtualized Infrastructure Manager
VLAN	Virtual Local Area Network
VM	Virtual Machine
VMM	VM Monitor
VNF	Virtualisation Network Function
VSA	Virtual Security Appliance
WP	Work Package
ZTA	Zero Trust Architecture

Executive summary

This document presents the 5G-EPICENTRE deliverable D2.7 “Cloud-native security intermediate version”, corresponding to Task T2.6 “Attack surface decrease and network edge access control" under Work Package 2 “Cloud-native 5G NFV”. The main focus of T2.6 activities (and consequently, of the present deliverable) has been to produce a security framework able to provide security for any Network Application executing in a Kubernetes environment. Therefore, this document presents the evolution and details of the latest version of the Holistic Security and Privacy Framework (HSPF), which corresponds to the security framework being developed under the scope of 5G-EPICENTRE. As part of the framework design, security standards and guidelines are presented, with special focus on the ones related to 5G technologies.

As mentioned, the evolution of the HSPF as of M28 is presented in this document, as well as the foreseen major milestones. An additional highlight is given to the latest HSPF implementation architecture, with the presence of the most relevant aspects related to its development, validation and deployment.

Table of Contents

List of Figures

List of Tables

1 Introduction

The objective of this document is to specify the Holistic Security and Privacy Framework (HSPF), an oriented cloud-native security framework designed to bring protection for any Network Application deployed in a Kubernetes environment. In addition, as part of the design of this framework, several security concepts and standards have been identified, which are also reported in this deliverable.

The challenges posed by present-day Information and Technology (IT) markets demand the implementation of distributed and composable systems from organisations, which are being moved outside of their physical boundaries. To that extent, the cloud is contributing to the simplification of their operations and removing much of the burden efforts involved in managing and deploying traditional server infrastructure. Moreover, organizations are leveraging automation capabilities from software-driven infrastructure models, which has resulted in a cloud-native approach.

The focus on a cloud-native approach for applications has led to an increased attack surface that requires the adoption of security measures throughout the software development lifecycle, from the moment the applications are designed, until the moment they are deployed and operated in production environments. Such an approach is referred to as Security-By-Design (SBD) and is one of the many security concepts considered while developing any application within the 5G-EPICENTRE paradigm.

This deliverable is structured as follows: Section 1.3 presents 5G security-related aspects collected during the design phase of the HSPF, including an overview of the 5G security challenges, and how the 5G architecture considers those aspects. Section 3 presents the HSPF, with special emphasis on its evolution throughout the project timeline; its latest implementation and respective validation activities; how it evolved to a Network Application; and what are the major activities foreseen for the remaining duration of Task 2.6. Finally, in Section 4, the overall conclusions of the deliverable are presented.

The document is complemented with four Annexes that cover different aspects gathered throughout the design and early stages of the HSPF, as well as some preliminary validation activities conducted with the first version of the framework. In detail:

(i) **Annex I: Cloud-native security** contains a set of concepts and standards about cloud-native security, in addition to specific cloud-native security specifications.
(ii) **Annex II: Proposed Security and Privacy Framework – Background** details the tools and security concepts behind the security framework.
(iii) **Annex III: Security Framework – Initial Experiments** describes a set of preliminary experiments conducted to explore different mechanisms offered by the technologies in use, and to conduct a first validation of the initial version of the HSPF.
(iv) **Annex IV: HSPF Evolution** provides complementary information regarding the evolution of the HSPF.
(v) **Annex V: Latest HSPF implementation architecture** presents additional information about the latest deployment and validation activities conducted with the latest HSPF implementation architecture.

1.1 5G-EPICENTRE Holistic Security and Privacy Framework

The overall 5G-EPICENTRE architecture is segmented into a multi-layered approach (Frontend, Backend, Federation, and Infrastructure layers), as defined in Deliverable D1.4 “Experimentation requirements and architecture specification final version”. The Frontend layer includes the processes supporting the interaction between the platform and Public Protection and Disaster Relief (PPDR) solution providers. The Backend layer comprises the functional components of the platform, while the Federation layer comprehends the cross-testbed orchestration of network services and resources to ensure an optimal experiment environment. Finally, the 5G testbed infrastructural elements of each of the federated testbeds compose the infrastructure layer. Orthogonal to those

layers, and to address the security challenges of the envisioned (and highly complex) 5G scenarios, an HSPF was also proposed to complement the architecture's security viewpoint.

As part of the scope of the 5G-EPICENTRE project, a security framework will provide the capabilities addressing the enforcement policies, managing the access control, and assuring the proper security practices across the 5G-EPICENTRE architecture. The proposed solution is the HSPF, a framework that is transversal to all conceptual layers of the 5G-EPICENTRE architecture, addressing different security requirements for each of them.

1.2 Mapping of project's outputs

The purpose of this section is to map 5G-EPICENTRE Grant Agreement (GA) commitments within the formal Task description, against the project's respective outputs and work performed.

Table 1: Adherence to 5G-EPICENTRE's GA Task's Description

5G-EPICENTRE Task	Respective Document Chapters	Justification
T2.6: Attack surface decrease and network edge access control *"This Task aims to configure secure network policies to deal with the increased attack surface resulting from the shift toward edge VNF containerization, and the inevitably larger network size. This will be achieved through minimal host Operating System (OS) distribution, ensuring that host management tools are executed in isolated management containers. Furthermore, OS-level security measures, along with per-program restricted access profiles will be deployed".*	Annex I: Cloud-native security	During the design phase of the framework, it was necessary to collect several concepts and standards. As a result, the *Standards and best practices* Section presents the security standards and best practices in cloud-native environments, while the *Cloud-native security specifications* Section describes the specifications that a cloud-native security system must contain. Further, Section *Cloud-native security specifications* presents the specifications needed into a cloud-native security system. (*e.g.*, the concept of namespaces and network policies).
T2.6: Attack surface decrease and network edge access control *"[…] Concerning Access Control, the 5G-EPICENTRE architecture (T1.3) should be flexible enough to enable the network edge to be autonomous – to some extent, depending on the security requirements - in terms of decisions to grant access or not. Therefore, specifications shall be put into place to enable the Policy Enforcement Points deployed at the edge to carry out the decision request to an always reachable Policy Decision Point".*	2 – 5G security and attack surface	Section 2.3 addresses the 5G Security Architecture, where an overview of the authentication mechanisms of 5G networks is given.
	Annex II: Proposed Security and Privacy Framework – Background	The *Technologies and tools* Section presents the technologies that may be used to configure a distributed cloud-native network system, namely the ones needed to enforce the required security policies.

T2.6: Attack surface decrease and network edge access control *"[…] If pure IoT authentication and authorization is concerned, alternative patterns might be deployed to ensure a fair-enough security of the remote autonomous devices deployed on the field, which still satisfies critical communications constraints".*	Annex I: Cloud-native security	The *Standards and best practices* Section presents the security standards and best practices in cloud-native environments. The authentication and authorization mechanisms include this diversity of standards.
	2 – 5G security and attack surface	Section 2.3 addresses the 5G Security Architecture, where the requirements of Quality of Service for PPDR applications are one of the major concerns.
T2.6: Attack surface decrease and network edge access control *"[…] Finally, this Task will define proper resource guarantees while deploying co-located instances of containerized VNFs (CNFs), so as to reap benefits in terms of VNF performance and agility gained due to the weaker-than-traditional-VM isolation, while at the same time neutralizing any chances of there being interference between co-located VNFs".*	3 – Holistic Security and Privacy Framework	Section 3.1 presents the security framework evolution from the conceptual architecture up to the current implementation, which is further detailed in Section 3.2. Section 3.5 presents the validation activities conducted so far, while Section 3.6 presents a Roadmap highlighting the major activities for the remaining period of T2.6.

1.3 Updates since the initial deliverable version

The present document is the second version of deliverable D2.1. Table 2 lists all updates introduced in this latest version of the deliverable, describing what content is new or updated when comparing to the previous version.

Table 2: Deliverable updates since D2.1

Chapter	Updates since D2.1
Section 1	This Section has been updated to reflect the recent document structure.
Section 2	There are no significant updates to this Section since D2.1. It has been kept, considering its importance to this deliverable.
Section 3	Section 3 presents a new structure when compared to D2.1. This Section starts by summarizing the evolution of the HSPF throughout the lifetime of Task 2.6. Afterwards, a complete description of the latest implementation work is provided, followed by an explanation on how this framework may be deployed, as well as the evolution of the HSPF up to Network Application approach. Finally, the HSPF validation activities are described, concluded by a Roadmap containing the key future milestones. The content initially present in Section 4 of D2.1 is now part of this Section.

Section 4	Highlights the conclusions achieved from the updated work in the present deliverable.
Annex I - III	All three Annexes present the same content as in D2.1.
Annex IV: HSPF Evolution	Presents additional information about the evolution of the HSPF, briefly described in Section 3.1.
Annex V: Latest HSPF implementation architecture	This Annex provides further information on different topics related with the latest HSPF implementation, namely the specification of the Report Interface, validation activities and the first HSPF deployment into a 5G-EPICENTRE testbed as a Network Application.

2 5G security and attack surface

This Section provides an overview of the 5G security challenges, and a review of the methodologies discussed in the literature to decrease the attack surface in 5G scenarios.

This section starts by introducing key information regarding the 5G architecture and related to the solutions proposed in this document. Important security challenges brought by 5G are then overviewed and finally, the 5G security architecture is analysed from the 5G-EPICENTRE Use Cases (UCs)' perspective.

2.1 5G architecture

In the 5G Core (5GC), key features such as Control and User Plane Separation (CUPS); Service-Based Architecture (SBA); Network Slicing; and Access Agnostic are included.

The SBA represents an architectural concept defined by the 5G standards. The SBA enables quick and easy integration of new network services and puts the telecommunication network design approach closer to the IT networks. Network Functions (NFs) can be virtualised and provide their services to other NFs, or third-party "verticals", using the standard HTTP/2 Internet protocol and Representational State Transfer (REST) API-based Service-Based Interfaces (SBIs).

The Network Repository Function (NRF) is one of the key components of the SBA, maintaining an updated repository of all the 5G elements available in the operator's network, along with their services provided by elements in the 5GC, which can be instantiated, scaled, and terminated. Network function management allows NF instances in the same network to register, update and de-register their profile in the NRF. Network function discovery allows an NF to discover the services of other NF instances in the same network. It also supports one NRF querying an NRF in another network on behalf of another NF. Finally, the OAuth2 authentication service is where the NRF issues a token to the requesting NF, which the latter one can use to prove that it is authorised to consume the service of another NF.

2.2 5G security challenges

The 5G SBA offers many security features which include lessons learned from previous generations of network technologies. On the other hand, the 5G SBA is a completely new network concept that opens up towards new customers and services. These new players lead to new security challenges.

Network migration requires several security considerations, considering that mobile core network attackers use whichever protocol and network generation gives them the result they desire. In a mixed architecture, where some elements are 4G and others 5G, interworking functions will enable communications across generations, which will require specific approaches in terms of security. With all the new NFs and services in 5G, roaming and legacy interactions will become quite complex.

Slicing opens the door for new customers directly into the core network, increasing the attack surface over the previous generations. From a security standpoint, it should be noticed that NF profile information may also include dynamic load information.

2.3 5G security architecture

An extremely high volume of connections is expected during the next five years, namely, somewhere near 21 billion Internet of Things (IoT) devices [1]. Therefore, 5G networks will need to support an extremely high number of simultaneous connections. Enabling use cases, like high bandwidth video streaming and mission critical IoT solutions, will require ultra-low latency and Quality of Service (QoS), among others. To support such a wide variety of use cases, network signalling complexity and traffic overload will have to be handled by the 5G networks.

To address the 5G signalling complexity, a signalling strategy should ensure that networks are robust enough to come in line with the needs of businesses. Operators are required to support 2G, 3G, 4G, and 5G for at least the next decade, adding more signalling complexity, interworking, and interoperability issues [2].

Moreover, with 5G touching every aspect of life with its broad set of use cases, the potential security threat is expected to increase. Operators need to invest heavily in securing their 5G networks before they can touch upon use cases in support of business and mission-critical industry vertical applications, such as healthcare and banking. Deploying a robust signalling and routing framework can help operators tackle the most serious issues.

One very important stage is the inter-Public Land Mobile Network (PLMN) communication, which typically contains highly confidential user information. The Secure Edge Protection Proxy (SEPP) sits at the perimeter of the PLMN, enabling secured inter-network function communication across the PLMN network. It maintains the confidentiality and integrity of the 5GC.

NF and NRF mutually authenticate each other using Transport Layer Security (TLS) or Internet Protocol Security (IPsec) at the lower layer of the Open Systems Interconnection (OSI) stack model, and the NRF may provide authentication and authorisation to NFs to establish secure communication between each other.

Authorisation to access services is an important security control, contributing to avoiding attacks coming from the Interconnection Networks (IPX), so authorisation works across PLMN boundaries should also be considered, including the ones between different 5G networks.

2.3.1 5G security features

The increasing adoption of 5G services will move considerable new business workflows that, in turn, will create new opportunities for fraudulent activity. As a result, new and traditional threats and vulnerabilities will need to be managed in the 5G ecosystem.

5G has introduced several features in order to improve its security, such as the following ones:

- The implementation of a SEPP, which takes the responsibility to protect the network from attacks arriving from the roaming network.
- The unification of different 5G access protocols and devices into a framework of authentication mechanisms.
- Introduction of the NRF authorisation function into the network architecture.
- Protection of user privacy on the air interface.
- Extended home network control for roaming users.

The important security connections to be protected are the following ones:

- Network and the interconnection network.
- Network interconnection between slices.
- Shared and non-shared NFs.
- Dedicated NFs and the shared infrastructure.
- Elements of legacy generations such as 2G, 3G and 4G NFs.

Security controls need to be applied at multiple points in the network and across multiple layers. Enabling packet capture and the ability to implement security at container ingress is critical to ensure that bad traffic stays out of a service provider's network. Enabling encryption is also fundamental in a 5G network security offering.

2.3.2 Security entities in the 5G Core network

The 5G system architecture introduces the following security entities in the 5GC network: Authentication Server Function (AUSF), Authentication Credential Repository and Processing Function (ARPF), Subscription Identifier

De-Concealing Function (SIDF) and Security Anchor Function forms (SEAF). The role of these entities is presented below [3]:

- **AUSF** supports authentication for the 3GPP access and untrusted non-3GPP access. It terminates requests from the SEAF and interacts with the ARPF.
- **ARPF** stores permanent secrets (K) as the base of short tier, keys, executes cryptographic algorithms and creates authentication vectors.
- **SIDF** de-conceals the Subscription Permanent Identifier (SUPI) from the Subscriber Concealed Identifier (SUCI).
- **SEAF** is an outcome of the primary authentication, the unified, common anchor key (KSEAF) for all the access scenarios. It provides authentication functionalities through the Access and Mobility Management Function (AMF) in the serving network. The SEAF shall support primary authentication using the SUCI.

5GC provides security requirements on the User Equipment (UE), gNodeB (gNB), AMF, Unified Data Management (UDM), SEAF and AUSF. The core network security comprises trust boundaries, aligned with divisions defined by network operators in order to divide their networks into trust zones. The messages that transpose those boundaries should follow NF service-based discovery, and registration shall support confidentiality, integrity, and replay protection. NRF shall be able to ensure that NF discovery and registration requests are authorised. The NRF and NFs that are requesting service shall be mutually authenticated.

The security domains are described next:

- **Network access security** is the domain providing a set of security characteristics that enable a UE to securely authenticate and access network services (including 3GPP and non-3GPP access), as well as guarding against attacks on radio interfaces. It also comprises the transfer of security context from serving network to access network for access security.
- **Network domain security** provides a set of security features that allow network nodes to exchange signalling and user plane data securely.
- **User domain security** provides a collection of security elements that protect a user's access to mobile devices.
- **Application domain security** includes a set of security features that allow user and provider domain apps to securely exchange messages.
- **SBA architecture** comprises a set of security capabilities that allow the SBA architecture's NFs to securely connect both within the serving network domain and with other network domains. NF registration, discovery, and authorisation security aspects as well as protection for service-based interfaces are examples of such capabilities.
- **Visibility and configuration** comprise a set of features that allow the user to know whether a security feature is active or not**.**

2.3.3 Authentication and authorisation in 5G systems

The 5G system shall satisfy the following authentication and authorisation requirements [3]:

- **Subscription authentication:** In the process of UE and network authentication and key agreement, the serving network must authenticate the SUPI.
- **Serving network authentication:** The UE must use implicit key authentication to verify the serving network identification.
- **UE authorisation:** The UE will be authorised by the serving network using the subscription profile provided by the home network. The authenticated SUPI is used for UE authorisation.

- **Serving network authorisation by the home network:** The UE must be certain that it is connected to a serving network that has been permitted by the home network to offer services to the UE. This authorisation is "implicit" in the sense that a successful authentication and key agreement run implies it.
- **Access network authorisation:** It must be assured that the UE is connected to an access network that has been permitted by the serving network to offer services to the UE. This authorisation is "implicit" in the sense that it is indicated by successful access network security establishment. This authorisation for access networks applies to all forms of access networks.
- **Unauthenticated Emergency Services:** 5G technology offers unauthenticated access for emergency services to meet legal requirements in some places. This feature will not be supported by serving networks in areas where unauthenticated emergency services are prohibited.

Impersonating devices can be used to launch Denial of Service (DoS) attacks, but that's not all they can do. They can also utilise their trusted network node status to launch "man in the middle" attacks, in which they deliver malicious commands to connected devices.

One type of attack drives devices to "bid down" to lower-quality network protocols, resulting in a drop in service quality. This might be a low-key, yet devastating attack on business networks. An attacker could attempt a bidding down attack by making the UE and the network entities respectively believe that the other side does not support a security feature, even when both sides support that security feature. It shall be ensured that a bidding down attack, in the above sense, can be prevented.

2.3.4 Service Communications Proxy

Despite the unprecedented benefits for operators, the 5G architecture is not yet fully equipped to deal with some of the major challenges that come with raised signalling traffic such as the following:

- Routing and optimisation.
- Traffic management.
- Robustness scalability and security.
- Network visibility.
- Core security with authorisation and authentication.

Furthermore, classical in-depth network security approaches, such as perimeter firewalls cannot be easily applied to Cloud-native scenarios, such as the ones envisioned by 5G-EPICENTRE container-based approaches. These require finer control regarding the network communications between all the different containers. **Service mesh** is a Cloud-native approach to support different security capabilities, including logging of API traffic, observability tagging, network traffic encryption, authentication, and authorization. Beyond the centralized management of the policies, it can also be used to support policy enforcement between different edge/cloud network traffic in Cloud-native 5G scenarios. The service mesh manages the communication between the services from the individual context to an infrastructure layer. Therefore, service mesh is introduced into the application as a set of network proxies. The requests are routed between microservices using proxies in their own infrastructure layer. The proxies implementing the service mesh are called **sidecars**, because they are run decoupled in parallel from each service. The service mesh does not introduce functionalities to the application's execution environment.

The adoption of a service mesh in the 5GC could represent a solution to the aforementioned challenges, but service meshes are not aware of 5G. The **Service Communication Proxy (SCP)**, recently included in the 5G architecture as an optional component to help the SBA, addresses this problem, by incorporating 5G awareness into the service mesh and by developing a secure 5GC signalling architecture that enables 5GC network routing control, robustness, and observability. The SCP focuses on network internal communication to facilitate the NF installation process.

The SCP performs delegated discovery in the form of an internal register and controller service, and it also implements an individual Service Agent (SA) for each NF. Such an approach enables indirect communications between each 5GC component in the SBA. This SA acts as a sidecar in service mesh implementations and supports its application to many use cases. The SA performs critical tasks that are peripheral to the primary role that an NF was designed to perform.

All top-tier service providers can use a cloud-native infrastructure solution to provide visibility, control, security, and scale for 5G network deployments. From both the core and edge computing standpoints, this approach can help reduce costs and complexity while constructing and operating a 5G network.

2.3.5 Mutual authentication

The authentication mechanisms between the Network Exposure Function (NEF) and an Application Function (AF), located outside the 3GPP operator domain, rely on mutual authentication based on client and server certificates performed between the NEF and AF using TLS. TLS provides integrity, replay and confidentiality protection for the interface between the NEF and the AF. The support of TLS is mandatory.

After the authentication, the NEF determines whether the AF is authorised to send requests for the 3GPP network entity. The NEF shall authorise the requests from the AF using an OAuth-based authorisation mechanism.

2.3.6 NEF security requirements

The NEF supports external exposure of capabilities of NFs to AFs. The interface between the NEF and the AF shall fulfil the following requirements:

- Integrity, replay and confidentiality protection for communication between the NEF and AF shall be supported.
- Mutual authentication between the NEF and AF shall be supported.
- Internal 5GC information, such as Data Networks Name (DNN), Single Network Slice Selection Assistance Information (S-NSSAI), *etc.*, shall not be sent outside the 3GPP operator domain.
- The SUPI shall not be sent outside the 3GPP operator domain by the NEF.

The NEF shall be able to determine whether the AF is authorised to interact with the relevant NFs.

As specified in the latest 5G-PPP Software Network Working Group White Paper [4], the development of Network Applications (such as those contemplated in the 5G-EPICENTRE project's UCs) aims at simplifying both the implementation and the deployment of vertical systems over the 5GC network. For businesses to create and deploy such applications, operators need to securely expose their 5G network services to their developers and third-party developers. The NEF acts as a centralised point for service exposure and plays a key role in authorising all access requests originating from outside the 3GPP network to enable cellular IoT, non-IoT, edge computing and API gateway use cases for operators.

2.3.7 Authentication and authorisation between NFs

To prevent injection, or spoofing of UP traffic over N9, it is recommended to use a common firewall that can correlate HTTP/2 methods and GTP-U ones to bind and filter out any malicious traffic on N9. The use of a common firewall may place other implementation restrictions (*e.g.*, co-location of the Session Management Function [SMF], SEPP and User Plane Function [UPF]) aiming to achieve a higher security level in the interactions among different NFs.

2.3.8 Authorisation of NF service access

The authorisation framework uses the OAuth 2.0 framework. Access tokens are JavaScript Object Notation (JSON) Web Tokens and are secured with digital signatures, or Message Authentication Codes based on the JSON

Web Signature (JWS). Recently, a service mesh has started to be attained by different approaches aiming to address a different set of security mechanisms for the 5GC [5]. The service mesh makes this interesting because the 5G components of SBI generate HTTP2 traffic, and thus they can be treated like any other HTTP application, including service routing, circuit-breaking, *etc*. The only exception to this is the UPF, which only uses GPRS Tunnelling Protocol User Plane (GTP-U) and Packet Forwarding Control Protocol (PFCP).

As defined in IETF RFC 6749 [6], all NFs and the NRF must support the OAuth 2.0 authorisation mechanism with the "Client Credentials" grant type.

The NRF will serve as the authorisation server, issuing access tokens to NF service consumers so they can use the NF service providers' services. If an NF gets an OAuth 2.0 authorisation token in the "Authorisation" HTTP request header field, the NF must validate the access token, its validity, and access scope before granting access to the requested resource.

2.4 Attack surface in 5G

The latest advances of 5G technologies and concepts such as the "softwarisation" and virtual network functions (VNFs); and openness to their part development, present a more challenging scenario from an orchestration standpoint. Inevitably, from a security standpoint, this also means an increased attack surface, and new challenges on how to secure the entire infrastructure. As new 5G capabilities are introduced, new types of threats emerge, demanding new approaches to security [7]. For instance, for network services (NSs), one of the key characteristics discussed in 5G is the additional complexity and security concerns on how to properly ensure their isolation, due to the number of involved components, legacy interworking, and configuration risks.

Three major attack scenarios for 5G network slicing were uncovered by AdaptiveMobile [8], including the following: i) user data extraction; ii) DoS against another network function; and iii) access to an NF and related information of another vertical. These attack scenarios, specifically focused on network slicing, describe how to gain access to resources of another slice, and how to perform a DoS attack on another slice. They also explain how to extract user-specific information (such as a location) from another slice. Current approaches and technologies are not mitigating such attacks. The lack of in-built observability in SOL011 and SOL005 5G architecture interfaces is highlighted in [9], which makes the corresponding Network Function Virtualisation Orchestrator (NFVO) exposed interfaces sensitive points in terms of security. Moreover, the attack surface is not only limited to the NFs and to a single deployment host, but also extends to many nodes. Multiple and heterogeneous domains bring additional complexity and a wider attack surface [10].

On the other hand, the increasing adoption of cloud-native architectures and the microservice paradigm in the telecommunication sector has allowed decoupling classical monolithic NFV, previously deployed in purpose-built hardware, into multiple smaller services running on top of VMs and containers. Such a cloud-native oriented approach allows the better fulfilment of the requirements of different 5G service types, such as enhanced Mobile Broadband (eMBB), massive Machine Type Communications (mMTC), or Ultra-Reliable Low-Latency Communication (URLLC). Nevertheless, despite the numerous benefits (*e.g.*, increased modularity and flexibility), a Cloud-native and a microservice approach results in larger and more complex infrastructures, which inevitably increases the attack surface at multiple levels.

The increased number of connections between microservices raises new risks of man-in-the-middle attacks spread over the infrastructure, so traffic authentication and authorization between services are vital concepts. The increased number of components might also lead to misconfigured, and thus, vulnerable assets. For instance, a recently disclosed backdoor [11] leverages misconfigured Docker API ports to infiltrate Docker servers and later execute malware on the target's infrastructure. Nowadays, those services might extend far from the traditional on-premises deployments. As we continue to scale up the number of microservices, it is paramount to build strategies to cope with increasingly Cloud-native environments and have the means to monitor the complex mesh resulting from all the microservice communications.

Breaking up those traditional monolithic network functions into microservices, often deployed in different nodes and composed of multiple operating systems, programming languages and third-party libraries, is by itself, an open challenge, not only from an architectural standpoint, but also from a security point of view. For instance, how to know what is running, how to roll out new service versions and how to monitor and secure all those microservices [12] become quite challenging tasks. In addition, the application of security patches to containers presents additional challenges because they are usually considered as immutable, which means that any reconfiguration or update involves rebuilding and redeploying the container.

5G-based scenarios composed of several heterogeneous Cloud-native domains, such as the ones considered in the 5G-EPICENTRE project, also create new security challenges, due to the underlying increase in size and complexity.

The heterogeneity of sets of containers increases the attack surface and raises important security concerns. Thus, it will be important to carefully configure them in terms of authentication and authorization processes [1].

Customer's misconfigurations of cloud resources are being presented as the leading cause of data loss in the cloud environment [13] and due to the human factor. Many examples related to cloud storage buckets and blob breaches exist. For instance, a misconfigured cloud storage bucket exposed Pfizer drug safety reports, 0x00sec reported that an S3 bucket was publicly accessible for 63 days, and more than 54,000 scanned New South Wales driver's licenses were found in open cloud storage [14].

Security can be applied along with all the different steps including defining stack, securing the cloud, securing Infrastructure-as-Code (IaC), monitoring workloads, alert at runtime (it is important to have the context and identify the source for the security threats).

Special attention should be attained due to the short lifespan of containers: 49% are alive less than 5 minutes [15]. This is caused by the uniqueness of their purposes since most of them only need to go live long enough to execute a specific function. These extremely short lifespans have security implications, requiring a DevOps approach. Such approaches may follow MITRE ATT&CK type of frameworks, allowing to codify into tools such as sysdig [16].

Moreover, an increased number of this kind of containers is expected, because they add isolation and execution capabilities for individualised functions or processes to run autonomously and independently.

2.5 5G-EPICENTRE Use Cases' security considerations

In the previous sections we have described the important aspects of 5G security and means to address them through the HSPF. In this Section, security considerations for each UC in the project are addressed. Table 3 presents security measures that are implemented by the project's UCs at an application level.

Table 3: Security measures applied by the UCs (at the application level)

Related protocol, service or other	HSPF Proposed Security Measures
MQTT	Use of WSS (encrypted WebSocket channels). Use of Access Control Lists (ACLs) for different connections.
Application Front-End	Use of secure and encrypted channels (*e.g.*, HTTPS). Two-factor authentication for user authentication on the application web/mobile application.

	Role-Based Access Control (RBAC) for access to the web/mobile application.
Internal Communications	Encrypted connections for internal communications (Through the use of HTTPS). Token based authentication for access to critical services (*e.g.*, database).
External Communications	Encrypted connections for all external communications.
Application API(s)	Token based authentication for API requests. RBAC for API requests.

Considering the yet ongoing integration between UCs and Testbeds, as well as Cloud-native security standards and best practices, it is assumed that several UCs may still have to conduct some changes to their existent solutions aiming to assure the correct implementation of the needed security mechanisms, including those needed to interact with the 5G Core. An example is that all the UCs that provide MCx services are already complying or in the process to do so, with the security requirements defined in TS 33.501 [1].

As such, a detailed list of the implemented security mechanisms by each UC will be provided in the final version of this deliverable (D2.8), already reflecting the eventual changes needed to ensure the correct implementation of the mentioned mechanisms. This decision is supported on the Roadmap defined in this document (see Section 3.6), which foresees the increase of interaction among the partners responsible for the HSPF and the UC owners, testbed owners (and even third-party experimenters), in order to deploy and validate the HSPF next to their respective Network Applications.

In addition, with the deployment of the UCs using Kubernetes and considering the HSPF, it is foreseen that the less frequent types of attacks to the application layer (*e.g.*, DDoS attacks, HTTP floods, SQL Injections, Cross-site scripting, Port Scan, among others) will be properly detected and blocked by the Network Application derived from this framework (see Section 3.4).

3 Holistic Security and Privacy Framework

This Section aims to present a set of relevant aspects related with the HSPF, starting with the framework architecture evolution present in Section 3.1; followed by the description of the latest HSPF implementation architecture in Section 3.2; the deployment process in Section 3.3; and the path of HSPF until it could be classified as Network Application in Section 3.4. The validation activities conducted with the latest implementation architecture are detailed in Section 3.5, and a Roadmap containing the foreseen future activities is provided in Section 3.6.

3.1 Framework Evolution

The first HSPF reference architecture was the result of an extensive analysis to the existent SoA in the area and the collection of requirements stemming from the 5G-EPICENTRE security challenges. This reference architecture is briefly presented in Section 3.1.1, while Section 3.1.2 briefly presents the first stable HSPF implementation architecture. In addition, *Annex IV: HSPF Evolution* provides additional information to both these Sections, delivering a detailed description of each respective version available in *D2.1: Cloud-native security specifications.* Finally, Section 3.1.3 reports on the lessons learnt so far, which impacted the latest implementation architecture.

3.1.1 Reference Architecture

Figure 1 depicts the initial reference architecture of the proposed Security and Privacy Framework, gathering three key elements: a security engine, a policy engine, and an Artificial Intelligence (AI) engine. This architecture was the result of an initial research on the state of the art at that time, and it follows the logical components of a conceptual framework model, such as the one specified in ZTA [17].

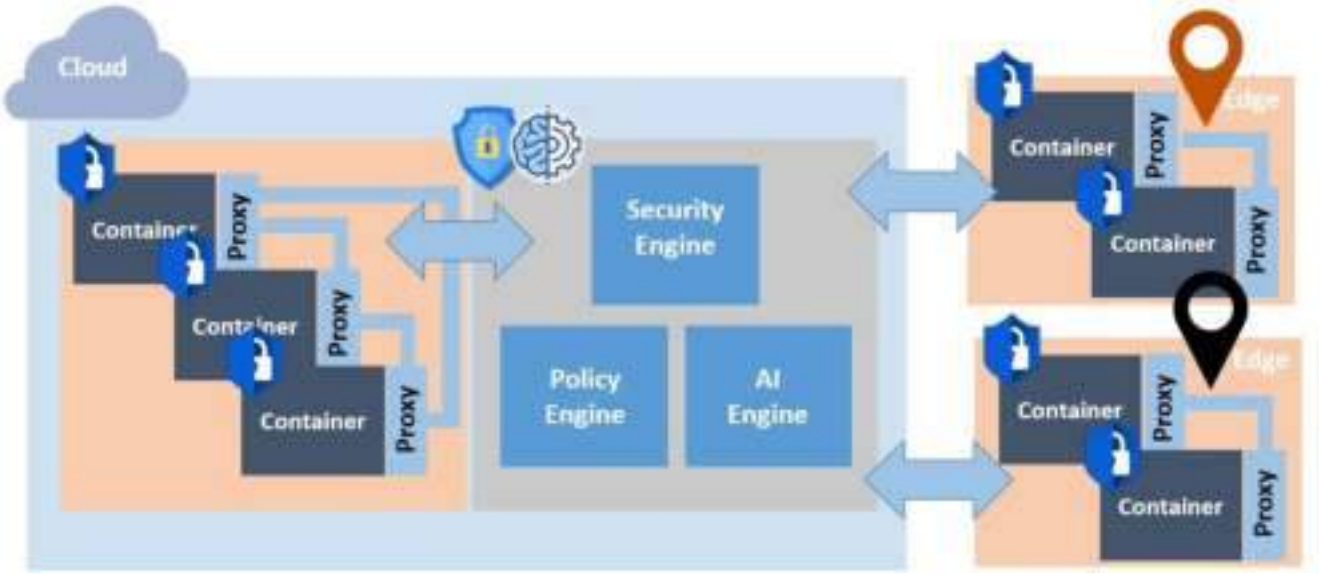


Figure 1: Holistic Security and Privacy Framework – reference architecture

3.1.2 HSPF Implementation Architecture

The first stable version of the HSPF implementation (depicted in Figure 2) has two major components: the set of *collection agents* and the Analytics, Intelligence, Control and Orchestration (AICO) component. Each *collection agent* is responsible to capture traffic data and send it to the AICO component. The AICO is composed of four components: *Data Collection*, *Analytics*, *Intelligence* and the *Control & Orchestration*.

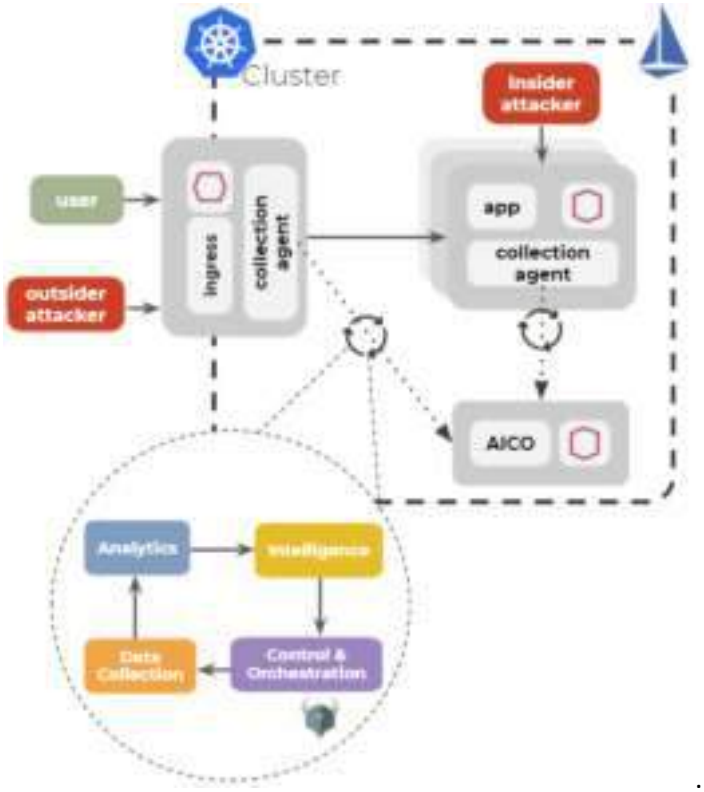


Figure 2: Initial stable architecture of the HSPF implementation

3.1.3 Lessons Learnt and Current Implementation

As a result of the previous validation activities and performance analysis conducted to the former HSPF implementation architecture, several drawbacks and weak points arose:

- A centralized algorithm for the traffic of all the micro-services revealed not to be the best option (considering that different micro-services do present different types of traffic).
- Supervised approaches require not only normal traffic but also malicious traffic for every training execution. This resulted in several issues, as explained in Section 3.2.3.
- There was no user-friendly monitoring tool that could be used to provide observability over the system.
- Upon the detection of an attack, the HSPF should provide more flexibility to the UC owner, or the third-party experimenter using it (*e.g.*, directly blocking the IP might not be desired behaviour).

Considering the previous issues, several changes were applied to the latest HSPF implementation architecture. The biggest change is the evolution towards a Federated Learning (FL) approach. This evolution is supported on the need of not only providing security to a single deployment of a Network Application executing into a specific location (*e.g.*, testbed), but, instead, to be able to simultaneously monitor and manage different deployments of the same Network Application in different locations. This approach also enables the continuous training of the AI models in different locations, leveraging the usage of data collected from other locations, albeit without risking to compromise the security and privacy of the original data.

Another important change, regarding the low-level architecture, is the presence of the collector and the classifier next to each micro-service, within each Network Application, instead of having a centralized component in charge of performing the classification of all the inbound and outbound traffic. This change is believed to be crucial to achieve a fine-tuned traffic classification for each unique micro-service communications profile. Also, a key addition is the presence of a Dashboard, where statistical and telemetric information is presented. Finally, upon the detection of an anomaly, the HSPF now has a reporting interface, thought to provide real-time warnings

and statistical information to third-party applications via different methods, thus aligning with the requirements of a Network Application itself. A detailed description of the latest reference architecture may be found below in Section 3.2.

3.2 Latest Implementation

This Section aims to detail the relevant aspects related with the latest HSPF implementation. Section 3.2.1 starts by providing its latest architecture from a macro-level point of view. Section 3.2.2 details how the traffic is collected and processed, so it may be then classified by the ML model, which is described in Section 3.2.3. Section 3.2.4 details the different stages present in the classification process, while Section 3.2.5 presents and details the flow taken by the collected traffic, including their classification and later use for ML training purposes.

3.2.1 Architecture

Figure 3 depicts a macro-level overview of the latest HSPF architecture.

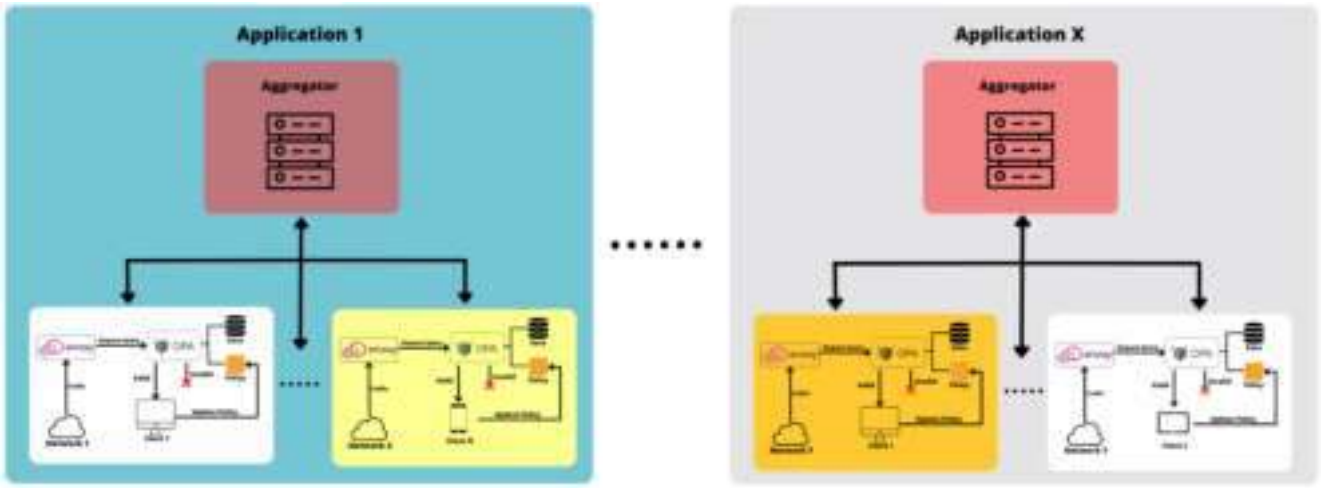


Figure 3: Latest architecture of the HSPF implementation

The latest architecture considers the major characteristics of a FL-based approach, including the ability to train AI models on decentralized data, thus avoiding potential data breaches that could compromise the data security and privacy.

Taking a closer look to an HSPF deployment (see Figure 4), it is possible to find three major components: (i) Collector and Agent (which are co-located); (ii) Dashboard; and (iii) Report Interface. In addition, the Agent also communicates with the Aggregator, as explained below.

The Collector and Agent component has several responsibilities. The Collector part is responsible for collecting the network inbound and outbound traffic passing through the main interface of the application it is next to. This information is then coupled into flows and stored until analysed by the ML model. The Agent, on the other hand, is responsible for inferencing stored data, resulting in the identification of the potential malicious flows that are then reported to the Dashboard and to the Report Interface. Periodically, there is also a routine in this component, that is in charge of training a new instance of the ML model, a process that is managed by the Aggregator (further details on the training and evaluation of the algorithm may be found in section 3.2.5).

Acknowledging the importance of strategies for assuring data security and privacy, a strategy has been developed to avoid the replication of flows among different components of the HSPF. Bearing this in mind, but also recognizing the need of maintaining the characteristics of the ML models in use for history and replication

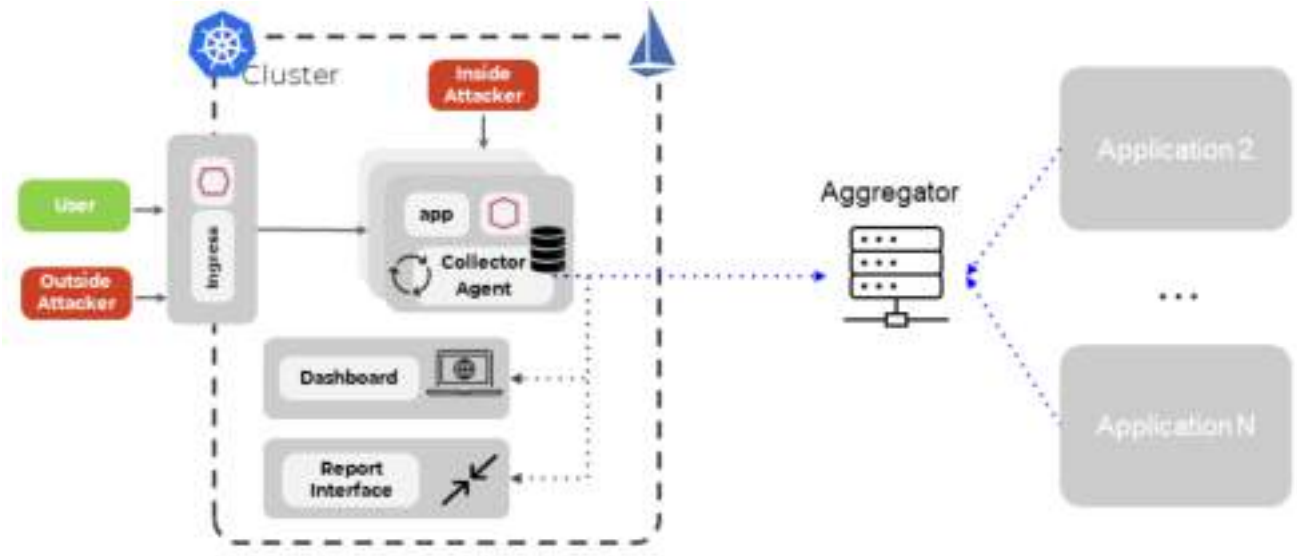


Figure 4: Latest architecture of the HSPF implementation (detailed)

purposes, an Aggregator has been introduced in the architecture. Due to the characteristics of the ML models in use, it is possible to keep only a diminished set of information about these models (*e.g.*, weights between layers), thus, reducing the space needed to store a traditional AI model and achieving the goal of not sharing information among components.

The Dashboard consists of a GUI that provides observability over the inbound and outbound traffic of the protected network application, namely the partial identified as malicious. The graphical representation of the data is achieved using custom graphical elements, partially enabled by Grafana [18] and data collected/stored by Prometheus. Figure 5 depicts the preliminary version of this dashboard.

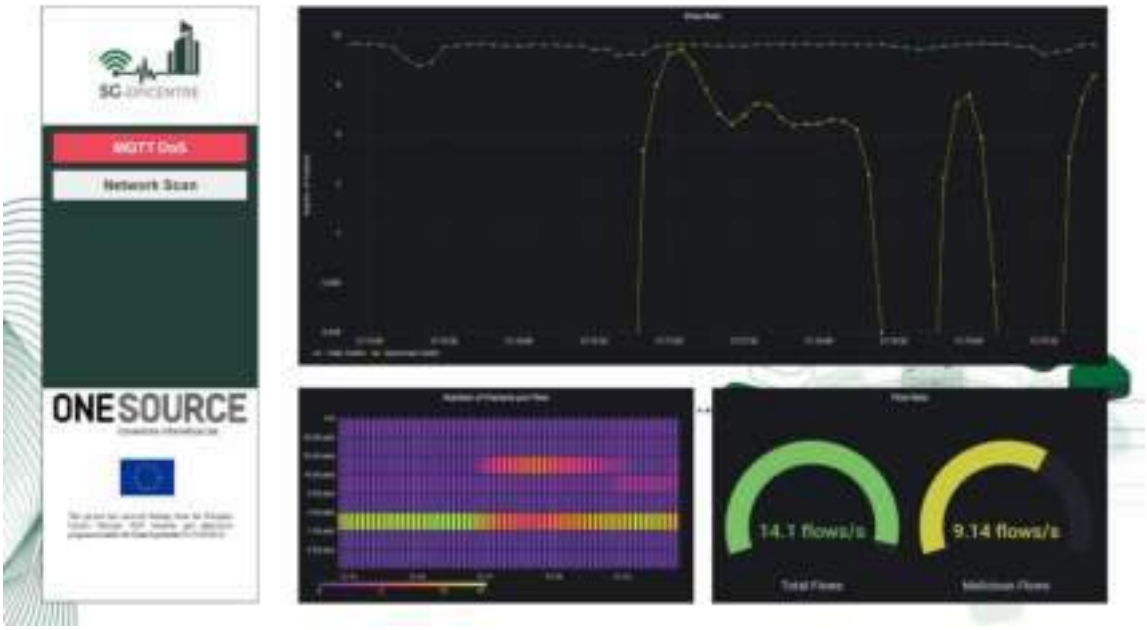


Figure 5: HSPF Preliminary Dashboard

The Report Interface arises from the need of expanding the set of options of a UC owner or third-party experimenter when reacting to the detection of malicious flow. Besides keeping the possibility of applying the default

behaviour (*e.g.*, block the origin IP upon the detection of a malicious flow), this interface now allows the communication with an MQTT broker. Such possibility was developed, bearing in mind that the application owner may not want the system to act on its own upon the detection of malicious flows, but rather might want to implement a custom method/action to handle attacks. It must still be noted that with the possibility of sending messages to a, MQTT message broker, the HSPF is now able to directly report to the 5G-EPICENTRE Publisher component every time an attack occurs, targeting a Network Application deployed in one of the project testbeds. Additional information on the Report Interface is present in *Annex V: Latest HSPF implementation architecture*.

Beyond the Report Interface, the HSPF continues to offer a way for taking actions immediately after the detection of malicious traffic via the application of traffic policies on the Ingress level, which is achieved using the Open Policy Agent (OPA) [19]. Naturally, the UC owner or third-party experimenter will have the ability to decide which are the desired actions to be taken upon the identification of malicious traffic when deploying this tool.

3.2.2 Data and network traffic collection

The first step towards the detection of traffic anomalies is to perform the traffic capture itself. As such, collectors are injected into all the micro-services of the application, with the sole purpose of registering the incoming and outgoing traffic for the respective service. Also called as “side-cars”, their final output is a *pcap* file, containing all the network traffic previously obtained using the *tcpdump* [20] tool.

Considering all the *pcap* files produced by the side-cars, for each one, a csv formatted file is generated, which is used later on to train the AI approaches. Such conversion is made using the NFStream [21] tool, aiming to produce datasets with a set of features similar to the ones present in CIC-IDS2017 [22] , which was initially used during the preliminary set of experiments as a baseline dataset.

Bearing in mind that there is a set of features that are not relevant for the identification of traffic anomalies (since their value is not related with the traffic characteristics itself and rather represent the origin, endpoint and other metadata of the flow), those have been removed. The entire list of removed features is the following: *Source IP*, *Source MAC*, *Source OUI*, *Source Port*, *Destination IP*, *Destination MAC*, *Destination OUI*, *Bidirectional First Seen ms*, *Bidirectional Last Seen ms*, *Forward First Seen ms*, *Forward Last Seen ms*, *Backward First Seen ms*, *Backward Last Seen ms*.

Assuming the high range of values for some of the features, and assuming it is not possible to always define the range of values accurately for each feature, it was decided to apply a standardization process instead of a min-max normalization, as before. The applied method was the Standard Scaler, which subtracts from the value of each instance the mean of the set of values (for each feature), and divides it with the respective standard deviation.

3.2.3 Classification Model

With a focus on unsupervised approaches, the implemented one is based on an Autoencoder. An Autoencoder is a ML algorithm based on Unsupervised Deep Learning, which is known to present a great balance between classification performances and fast classification times [23].

The reasoning behind the pursuit of unsupervised approaches is mainly supported on the previous experiments conducted with supervised ones. For supervised approaches, it was concluded that it is necessary to previously train the algorithms with a set of attacks (*e.g.*, anomalous flows), which translates into an inability of the algorithm to identify unseen attacks. Also, considering that every time the algorithm needed to be trained in runtime, it was necessary to have the dataset of known attacks locally, or send the collected data (from the micro-service communications) to an external place to allow this training. Both options had some limitations. The first presented an issue with storage space availability, and the second a major privacy (and potential security) issue. As an alternative, unsupervised approaches should only require normal data to be trained with, thus removing the need of having a dataset with known attacks every time the algorithm needed to be trained.

The implemented ML architecture is depicted in Figure 6.

Figure 6: Model architecture

3.2.4 Classification Stages

Due to the uncertainty associated with the ML model classifications, it is necessary to minimize the potential effects that a misclassification might have. As such, a process has been developed, to allow a way to handle how the detected traffic anomalies evolve, and are then classified as attacks, which is carried out by the Agent and is depicted in Figure 7.

The Agent internally contains two lists: a *greyed* list, which is used as a "grey zone" for IPs that have already been in the origin of malicious traffic, but have not yet been blocked, and a *blocked* list, where the blocked IPs are registered.

Upon the identification of an anomaly, which happens when a flow (or set of flows) is classified as malicious traffic, the Agent will verify if the *Source IP* of the identified flow is already marked as *greyed* in the internal list. In cases where the IP is not in the *greyed* list, it is added, and the respective counter of occurrences is set to 1. On other hand, if the IP is already in the list, the respective entry is incremented by 1 and the counter value is

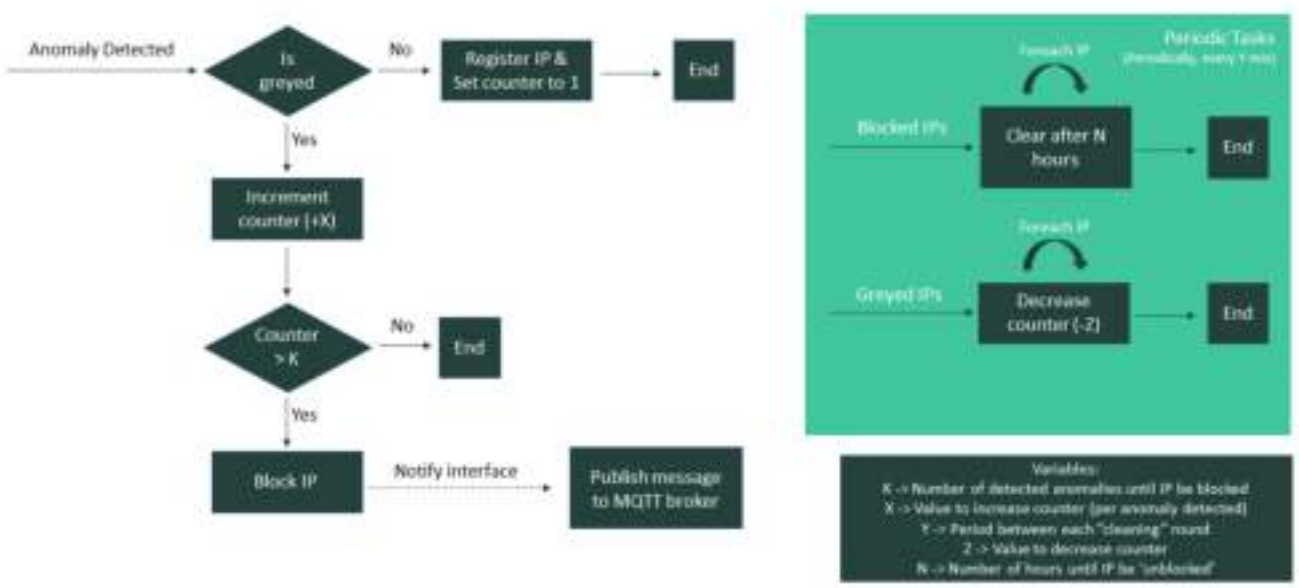


Figure 7: From anomalies to attacks - custom process

compared with a threshold. If the respective counter is larger than the threshold, the IP in question will be blocked by means of the application of a security policy defined by OPA, and a message will be sent to the Publisher component reporting this incident.

There is also a periodic task in charge of cleaning both internal lists. IPs are removed from the *blocked* list if the last detected attack happened already N hours ago (configurable value). However, for the *greyed* list, counters associated with each IP are decreased on each execution of this task.

It must be noted that all thresholds, periodicity intervals and increase/decrease values, may be defined while deploying the HSPF, thus allowing this process to be tailored to the needs of 5G-EPICNETRE UCs and of third-party experimenters.

3.2.5 Classification Flow

Figure 8 depicts the classification process from the traffic collection stage up to the periodic ML model training.

The first step towards traffic anomaly detection is precisely to capture the network traffic. This is achieved using the *NFStream* plugin [21], which collects the network packets and aggregates them in flows with a set of specific features.

Upon the writing of the flow to a temporary file, a co-located component, the 'Agent', extracts the flow(s) from the file and performs inference over them. This file is intended to act as a buffer, but with the advantage of being persistent. As such, in case of a temporary down time of the system (which may be caused by the most different of reasons), a new instance of the Agent will be automatically instantiated by Kubernetes, and this new instance will have access to all the previous and unprocessed flows. In addition, since the Agent will not only be able to process flows individually but also in windows, the existence of a file solves the need of having the flows history available to the system.

The Agent has two main processes: perform inference from the collected data (as mentioned before) and train a new instance of the ML algorithm. These processes are executed periodically, except for the inference that also considers the number of flows collected until a certain period. Moreover, both processes are only executed if

during the previous elapsed time new information has been collected. Otherwise, the system remains idle until the next iteration.

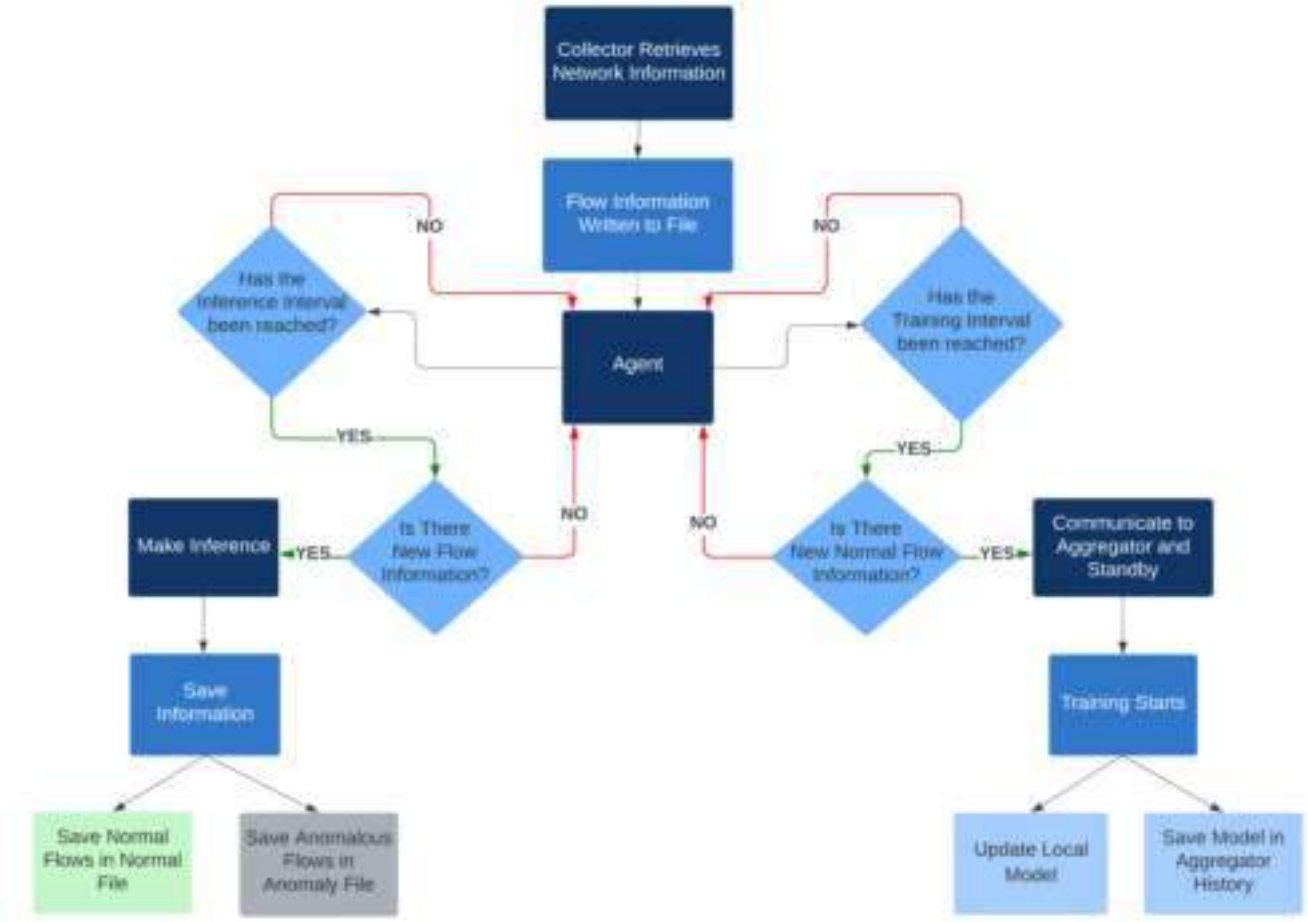


Figure 8: HSPF classification process

When the inference process is executed, its result is saved in the corresponding files, considering if the flow (or set of flows) were marked as *normal* or *malicious* traffic. The option of having two persistent files with the inference results, besides their persistence (similar to file mentioned before), is also related with the possibility of further analysis to the classification results, namely for continuous improvement, but also, for eventual audits that may be conducted to the HSPF.

The data hereby classified as *normal* is later used during the periodic training iterations of the algorithm. Before starting such process, the Agent reports to the Aggregator its readiness to start the process and waits for the confirmation to start. Even though this step is not needed in isolated scenarios (that is, where the HSPF is only providing security to a Network Application in one location, *e.g.*, Kubernetes cluster), it is needed for cases where the Network Application is deployed in multiple locations, namely in multiple project testbeds, or even in deployments following the EDGE-CORE approach. After the confirmation is received, the Agent triggers the training routine, which results in a newly trained algorithm instance. Afterwards, this new instance is evaluated with some validation data, and if its performance is better than the one currently in use, the new one replaces the previous one, and the inference will start to be conducted by this new instance. Both the results of the validation phase, along with some characteristics of the new instance, are shared with the Aggregator and stored.

3.3 Deployment process

Two methods have been envisioned to enable the quick deployment of the HSPF: (i) using a custom script, that promotes the injection of the HSPF components next to already executing vertical applications; (ii) allowing UC owners and third-party experiments to specify into their YAML component files that they want to install the HSPF components next to their micro-services.

At this point, the first method has been successfully implemented and validated in Kubernetes clusters, applying RBAC guidelines. Recently, efforts were conducted to achieve a preliminary deployment of the HSPF at UMA premises, thus achieving the first functional deployment of the HSPF, deployed in one of the project testbed infrastructures. This has been recently achieved using the first method described above, where only the *namespace* and the list of components to be monitored are used as input parameters for the installation script. Also, the UC owner or third-party experimenter has the possibility to only instantiate the HSPF security components next to a set of components at their choice, which provides more flexibility and control. *Annex V: Latest HSPF implementation architecture* contains debug information that demonstrates the correct behaviour of the HSPF internal components.

The second method is currently under development. Even though this is not mandatory for the use of the HSPF, it has been defined as a goal to be achieved, which is expected by M31. The rationale behind the development of the second method is the possibility for the Network Application developers to specify their intention of using the HSPF components directly on each *yaml* file of their Network Applications, which is expected to be a simpler and direct process, thus reducing potential barriers for the usage of HSPF by the majority of UC owners and third-party experimenters.

The requirements to deploy the HSPF in a Kubernetes-based cluster that an administrator must assure are the following:

- Provide a way to reach the namespace, where the framework will be deployed (*e.g.*, a *config* file, to be used with the *kubectl* client command).
- Authorize OPA to enforce traffic policies at the Ingress level.
- Enable Istio in the cluster.

When it comes to the UC owner, or third-party experimenter, the key step is the selection of the set of micro-services that the HSPF should monitor and provide security to. It is also possible to only specify the namespace, where it is intended to deploy the HSPF. However, if no list of micro-services is specified, the framework will assume that it must monitor all of them and deploy the necessary components accordingly.

3.4 Evolution towards a Network Application

The first version of the HSPF was the result of a vast analysis of the state of the art at the time (M12, D2.1), and resulted in a conceptual reference architecture with three key components: a security engine, a policy engine, and an AI engine. This was considered the first step towards the detection of malicious traffic, aiming to provide security for the Network Applications being experimented under the scope of 5G-EPICENTRE.

The former architecture led to the first HSPF implementation, whose architecture presented significant changes when compared to the conceptual one, mainly because of the experimentation process that took place to achieve such version. With two major components (the set of *collection agents* and the AICO component), the HSPF was starting to get close to a Network Intrusion and Detection System (NIDS) solution, allowing the identification and mitigation of threats (M24, D2.1 revision).

The latest version is the result of all the previous knowledge, combined into a new implementation architecture, which besides providing an improved version of the previous functionalities (*e.g.*, data security, traffic anomaly

detection), now also delivers a Dashboard (to provide crucial information in a user-friendly manner) and a Report Interface (through which real-time analysis related data may be easily provided).

The development of Network Applications, as defined in one of the latest 5G-PPP Software Network Working Group White Paper [4], aims at simplifying both the implementation and the deployment of Vertical Applications over the 5GC network. Bearing this in mind, the HSPF, despite of not being originally foreseen to be one, may now also be considered as a Network Application in a sense that:

- It does not represent a separated component in the 5G-EPICENTRE architecture.
- It is not mandatory, rather it is optional to whichever UC owner or third-party experimenter that may want to use it.
- It includes a way to provide data to the underlying Vertical Application, so it may take proper actions upon a specific event (*e.g.*, upon the detection of malicious traffic, choose to terminate a specific connection).

3.5 Validation Activities

For the purpose of achieving the first stable version of the detection module enabled by an unsupervised approach, it was necessary to perform a series of testing and assessment activities. To that end, the CIC-IDS2017 dataset has been used.

As mentioned before, the implemented approach was based on an Autoencoder. This type of approach presents two major phases to produce a satisfactory result: Reconstruction and Classification.

3.5.1 Reconstruction

As mentioned in Section 3.2.2, one of the first steps was to obtain a standardized version of the data. For contextualization, their representation is depicted in Figure 9, where the values of each feature for a normal instance are shown on the left and the values of each feature for an anomalous instance are shown on the right.

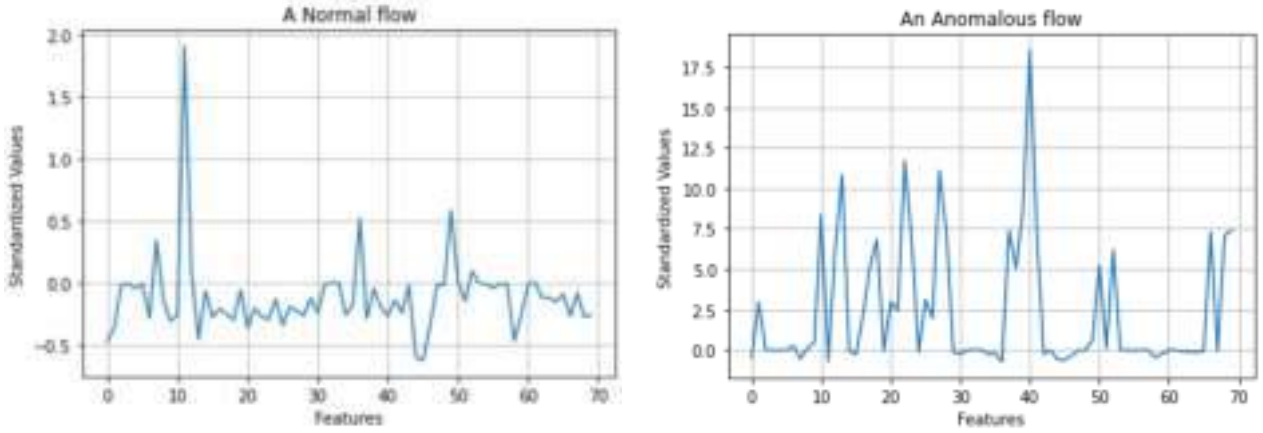


Figure 9: Standardized data

Considering that the selected approach was based on an Autoencoder, it was decided that performing any kind of feature reduction would not be appropriate, since Autoencoders already perform a dimensionality reduction. As such, the values of the original features were used without additional pre-processing, besides the standardisation process.

While seeking for the best parameters (of different layers) combination, a search grid has been applied. The considered layers and respective potential parameter values were the following:

- Number of layers: [7 (dimensions of each layer 70, 35, 18, bottleneck); 9 (dimensions of each layer 70, 50, 30, 10, bottleneck); 11 (dimensions of each layer 70, 55, 40, 25, 10, bottleneck)].

- Hidden layers' activation function: [ReLU, eLU, LeakyReLU].
- Optimizers: [Adam, Nadam].
- Bottleneck layer dimensions: [3, 5, 7].
- Learning Rate: [0.001, 0.0001, 0.00001].

The purpose behind testing different combinations of layers and respective parameters, was to explore how different combinations of layers, as well as how different decays in layer dimensions would impact the reconstruction learning step. The activation functions and the optimizers were selected considering a previous investigation on the matter by Carrera et al. [23], and some preliminary implementation tests.

The parameters for the bottleneck layer dimensions were selected aiming to be more difficult for the anomalous instances to be reconstructed, meaning if the bottleneck layer was higher, the anomalous instances would probably be easier to reconstruct. This contradicts the underlying goal of the algorithm, not being able to correctly reconstruct the anomalous instances, and thus presents a bigger reconstruction error in the end, which in comparison with a threshold can be used to classify the samples as either malicious or not malicious traffic.

The training was performed with 300 epochs, batch size of 64, early stop with patience of 20 and the loss was calculated using Mean Squared Error.

The best combination of hyperparameters was selected based on the validation loss. Figure 10 presents an example of training and validation losses, achieved during a training and validation routine.

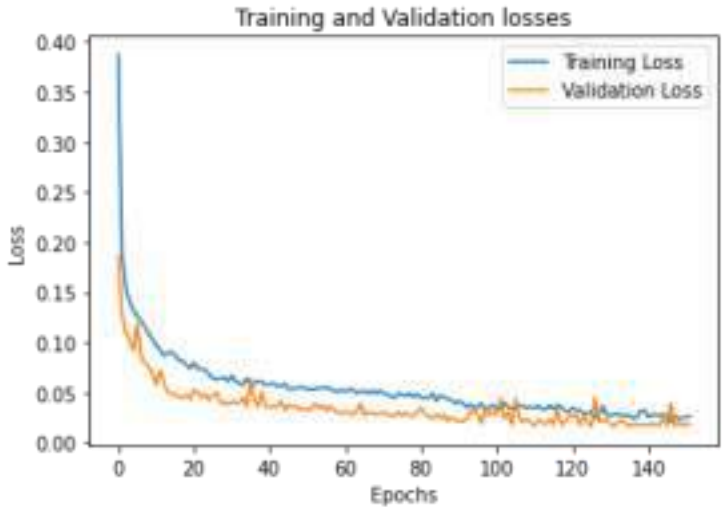


Figure 10: Training and Validation Loss

A representative sample of the best and worst results achieved during the entire experimentation phase are shown in Table 4. The entire list of results may be found in *Annex V: Latest HSPF implementation architecture*.

Table 4: Hyper-tunning parameters for Autoencoders reconstruction phase

Number of layers	Hidden layers activation functions	Optimizer	Bottleneck layer dimension	Learning rate	Validation Loss
11	elu	Nadam	7	0.0001	0.0182
11	leakyrelu	Adam	7	0.0001	0.0184
11	elu	Adam	7	0.0001	0.0187

9	leakyrelu	Nadam	7	0.0001	0.0196
9	elu	Nadam	7	0.0001	0.0201
9	elu	Adam	7	0.0001	0.0207
11	elu	Adam	5	0.0001	0.0210
11	leakyrelu	Nadam	7	0.0001	0.0217
7	elu	Adam	3	1e-05	0.1208
7	reLU	Adam	3	1e-05	0.124
9	reLU	Adam	3	1e-05	0.1293
7	eLU	Adam	3	0.001	0.1322
7	reLU	Adam	3	0.0001	0.1568
11	reLU	Adam	3	1e-05	0.1677
7	reLU	Nadam	3	0.0001	0.8150
11	reLU	Nadam	3	0.001	0.992

3.5.2 Classification

At this point, the reconstruction errors provided by the Autoencoder were used to classify the different samples. With that purpose in mind, a threshold needed to be defined, to allow a separation between normal and anomalous instances.

Two major steps were taken to find that threshold: (i) for each instance, the reconstruction error between the original instance and the reconstructed one was calculated using the Mean Squared Error, a process applied both to the training and validation sets; and (ii) the mean and standard deviation of the reconstruction errors were also calculated.

The threshold value was calculated using the expression below:

$$threshold = \mu + (x * \sigma) \qquad (1)$$

Considering the high variance of the reconstruction errors, the threshold that was being obtained with all the values did not allow us to perform a distinction division between malicious and not malicious traffic. As such, a method was applied to detect the presence of eventual outliers in the data, that could be potentially introducing some error in the calculus of this threshold. This detection was performed using an Interquartile Range based outlier detection approach [24]. First, the upper bound was calculated, which was the 75% percentile summed to 1.5 times the interquartile range (corresponding to the difference between the 75% percentile and the 25% percentile of the data). After that, all the errors that were above that upper bound were eliminated. Upon this pre-processing, the new threshold value was revealed to be much more appropriate.

With the objective of achieving the threshold that gave us the best performance for the classification, the F1-score metric [25] was used. This metric was selected to evaluate the different results, considering that the most important features for our problem are Recall and Precision [26], and that F1-score metric is a harmonic mean between the former two. As previously stated, the threshold was calculated by summing the mean and the standard deviation multiplied by a constant (σ). The values that σ took were the following: [1, 10, 20, 30, 40, 50], and the F1-score values achieved are shown in Table 5.

Table 5: Hyper tunning parameter for the threshold formula

Value Multiplied	F1-score
1	0.649
10	0.714
20	0.741
30	0.755
40	0.759
50	0.753

The value that maximized the F1-score was 40, so this value was selected. By combining the best model in terms of reconstruction, and the best threshold formula, the testing set was used to analyse the performance of the model. The achieved results for not malicious flows are depicted in Figure 11 on the left, with the results for the malicious flows depicted in the same Figure on the right.

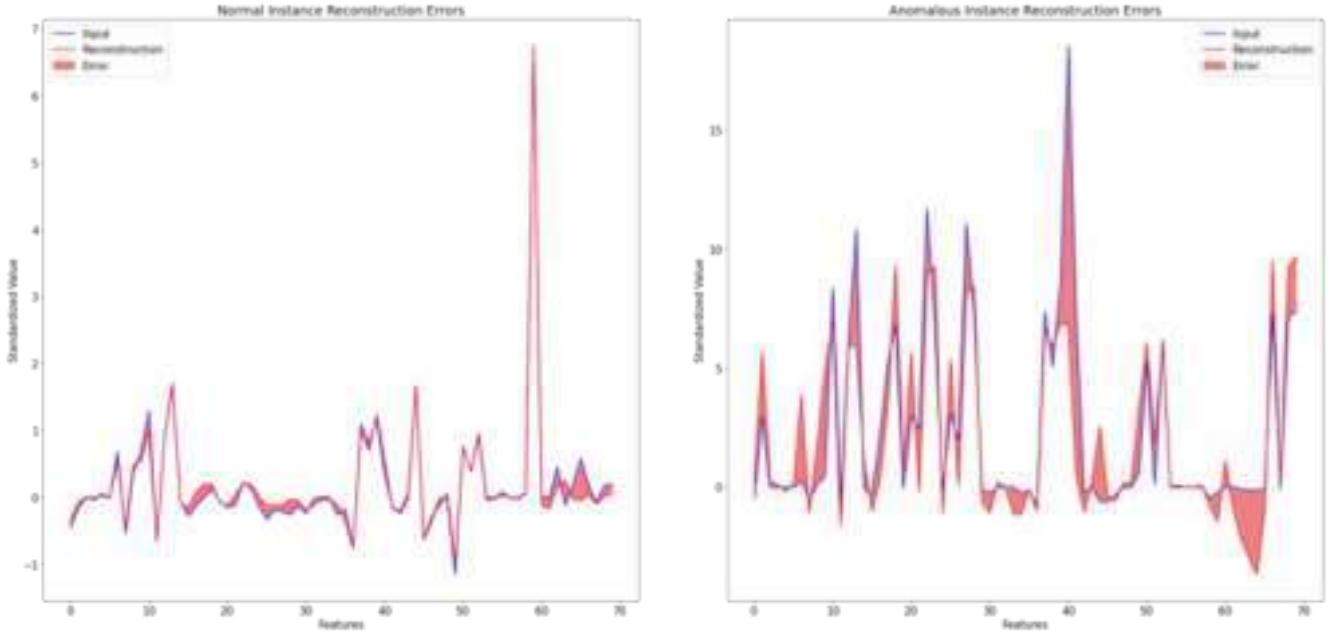


Figure 11: Reconstruction error for normal and malicious flows

In addition, the achieved classification metric values and corresponding confusion matrix [27] are presented in Table 6 and Table 7, respectively.

Table 6: Classification results

Metric	Score
Precision	0.9232
Recall	0.6448
F1-score	0.7593
False Negative Rate	0.3552
Accuracy	0.8673

Table 7: Confusion matrix

	Actually Positive	Actually Negative
Predicted Positive	172668	14357
Predicted Negative	95103	542859

The achieved *precision* is quite high, meaning the model is identifying most of the normal instances accurately, which is one of the key aspects, as the model must not produce a high number of False Positives (because it would potentially cause issues with normal traffic). This behaviour could result later in the disruption of normal operation of the underlying Vertical Application. In comparison to the literature [23], this approach presents similar results for most of the metrics for the different algorithms present in this article, and even outperforms some of them, as may be seen in Table 8 (translated from the referred publication for easier comparison).

Table 8: Results from testing unsupervised algorithms on the CIC-IDS2017 [23]

Algorithm	Precision	Recall	F1-score	Accuracy
EC	0.6264	0.5961	0.6053	0.7804
EIF	0.7063	0.7063	0.7063	0.8142
DAGMM	0.6934	0.6934	0.6934	0.8060
DAGMM-EIF	0.7083	0.7040	0.7056	0.8182
DA	0.6774	0.6774	0.6774	0.7990
DA-EIF	0.7051	0.6912	0.6975	0.8148
DSEBM	0.6800	0.6800	0.6800	0.7975
MemAE	0.7146	**0.7146**	**0.7146**	0.8194
MemAE-EIF	**0.7431**	0.6972	**0.7146**	**0.8350**

As may be seen from the comparison between the two Tables, the implemented approach presents higher metric values for the precision and for the F1-score. On the other hand, it presents a lower value for the recall. Further fine-tuning activities will be conducted, aiming to improve the overall performance of the implemented approach.

3.5.3 Results Analysis

The ideal validation loss during the Reconstruction phase would be zero. Bearing in mind that, in practice, this value will always be quite challenging (if not impossible) to achieve, it is possible to notice that for the best achieved results, the validation loss was quite close to zero (meaning that the implemented approach is able to reconstruct the different samples with a diminished percentage of error).

As for the Classification phase, it was possible to verify the good performance of the implemented approach when comparing its performance for the CIC-IDS2017 dataset with other approaches reported in the literature. In fact, this approach outperforms the best reported approaches for some of the evaluation metrics and presents similar values for the remaining ones.

3.6 Roadmap

This section aims to present the major activities foreseen for the remaining period of Task 2.6 (8 months, M28-M36), which are shown in Table 9.

Table 9: Roadmap

Description	M28	M29	M30	M31	M32	M33	M34	M35	M36
Major Milestones	D2.7								D2.8
Fine tuning of the latest HSPF architecture components									
5G-EPICENTRE UCs Integration									
Third-party experimentation supporting activities									

The first milestone presented above foresees some fine-tuning related activities to the latest HSPF implementation architecture, as a result of further validation activities, which are expected to happen until the end of M29.

The second milestone corresponds to the integration between the HSPF and different project UCs, which is expected to happen between M30 and M33 at the latest. The first integrations will be conducted with the UCs deployed at UMA testbed, leveraging the fact that an initial deployment has already been completed there.

The third milestone is related with the support to third-party experimentation, which is foreseen to happen between M30 and M34 where support will be given to third-party experimenters that may want to use HSPF to secure their own Network Applications.

Naturally, the writing of D2.8 will be carried out in simultaneously with all other activities, with a special emphasis between M33 and M35, leaving M36 for the internal review and quality check procedures over the document.

4 Conclusions

The increasing complexity from deploying 5G scenarios, such as the ones brought by the 5G-EPICENTRE project, designed on top of heterogeneous Cloud-native cross-testbeds, poses several challenges from a security standpoint. They demand specific security-tailored approaches in terms of orchestration, analytics, and automation.

This document started by briefly presenting how the 5G architecture addresses the security challenges and standards, namely, through the identification of the 5G security features; the 5G core components in charge of their enforcement; and the newest attack surface, resulting from the adoption of cloud-native technologies.

Afterwards, an overview of the evolution of the HSPF since the beginning of the project is provided, starting with the reference architecture, going through the first stable HSPF implementation architecture and finalizing with a set of lessons learnt in the process, and with the latest architecture that is described in detail.

A major goal was the ability of deploying the HSPF next to any existent running 'Network Application' being executed in a Kubernetes environment, an objective that has currently been achieved. Some details on how this deployment may happen are provided, as well the next steps to improve the current process. As a result of the project initiatives towards the definition of what a 'Network Application' is, the HSPF has evolved and may now also be classified as one (see also D1.4 and D4.2).

Moreover, a change of course on the detection model approach has been made based on the results and insights of the past validation activities, with the current approach being based on an unsupervised approach. Despite the fact that only preliminary validation tests have been conducted, we strongly believe that unsupervised approaches are the best ones for dealing with the security challenges that 5G-EPICENTRE has to deal with. The validation activities conducted so far with the latest architecture are described, and a roadmap is provided containing the major future activities to be conducted, namely the integration with different 5G-EPICENTRE parties.

The document also includes five Annexes, with the first presenting cloud-native background related concepts; the second focused on cloud-native security, namely on the existent best practices and standards; the third describes the technologies chosen for the development of the framework; and the fourth providing complementary information to the evolution of the HSPF. Finally, the fifth depicts a set of initial experiments conducted during the development and validation of past versions of the security framework and presents some debug information collected from the initial deployment conducted at UMA.

In summary, the document presents the work conducted so far under the scope of Task 2.6, with the highlight of the description of the existent HSPF, specifically designed and developed to bring security for any solution following the concept of a 'Network Application', like the one serving as the basis of 5G-EPICENTRE. The results and lessons learnt from the integration with 5G-EPICENTRE UCs and testbeds, as well as any other relevant developments related with the HSPF, will be reported in D2.8, due by M36.

References

[1] ETSI. (2019). Security architecture and procedures for 5G system. *ETSI Technical Specification 133 501 V15.4.0 (2019-05)*. https://www.etsi.org/deliver/etsi_ts/133500_133599/133501/15.04.00_60/ts_133501v150400p.pdf.

[2] Remmert, H. (2021, June 08). 2G, 3G, 4G LTE network shutdown updates [Blog]. *Digi Blog*. https://www.digi.com/blog/post/2g-3g-4g-lte-network-shutdown-updates.

[3] *Specifications & Technologies*. https://www.3gpp.org/

[4] Sayadi, B., Chang, C.-Y., Tranoris, C., Iordache, M., Katsaros, K., Vilalta, R., et al. (2022). *Network applications: Opening up 5G and beyond networks* [White paper]. Zenodo. https://doi.org/10.5281/zenodo.7123919.

[5] *Cloud Native Computing Foundation*. Retrieved from: https://www.cncf.io/

[6] Hardt, D. (2012). The OAuth 2.0 authorization framework. *RFC, 6749*, 1-76. Internet Engineering Task Force (IETF). https://datatracker.ietf.org/doc/html/rfc6749.

[7] Nokia. (2019). *5G security. A new approach to building digital trust* [White paper]. https://onestore.nokia.com/asset/206609.

[8] AdaptiveMobile. (2021). *5G Network Slicing Security in 5G Core Networks* [White paper]. https://info.adaptivemobile.com/5g-network-slicing-security.

[9] Ordonez-Lucena, J., Tranoris, C., Rodrigues, J., & Contreras, L. M. (2020). Cross-domain slice orchestration for advanced vertical trials in a multi-vendor 5G facility. In *2020 European Conference on Networks and Communications (EuCNC)* (pp. 40-45). https://doi.org/10.1109/EuCNC48522.2020.9200940.

[10] Bernardos, C. J., Contreras, L. M., Vaishnavi, I., Szabo, R., Mangues, J., Li, X., et al. (2018). Multi-domain network virtualization. *IETF Internet Draft*. https://datatracker.ietf.org/doc/html/draft-bernardos-nfvrg-multidomain-05.

[11] Seals, T. (2021, July 30). Doki backdoor infiltrates docker servers in the cloud [Blog]. *ThreatPost*. https://threatpost.com/doki-backdoor-docker-servers-cloud/157871/.

[12] O'Keefe, M. (2019, January 23). Welcome to the service mesh era: Introducing a new Istio blog post series [Blog]. *Google Cloud*. https://cloud.google.com/blog/products/networking/welcome-to-the-service-mesh-era-introducing-a-new-istio-blog-post-series.

[13] Bartly, R. (2021). *Solution path for security in the public cloud*. Gartner.

[14] Aqua (2021). *Cloud security report: Cloud configuration risks exposed* [Report]. https://www.aquasec.com/resources/.

[15] Sysdig. (2019). *Container usage report: Five minute container life highlights need for specific security controls* [Report]. https://sysdig.com/blog/sysdig-2019-container-usage-report/.

[16] *Sysdig Secure*. Retrieved from: https://sysdig.com/products/secure/.

[17] Kindervag, J. (2010). Build security into your network's dna: The zero trust network architecture. *Forrester Research Inc*, 1-26.

[18] Grafana Documentation. Retrieved from: https://grafana.com/docs.

[19] Open Policy Agent. Retrieved from: https://www.openpolicyagent.org/.

[20] Tcpdump. Retrieved from: https://www.tcpdump.org/manpages/tcpdump.1.html.

[21] NFStream: Flexible network data analysis framework. Retrieved from: https://www.NFStream.org/.

[22] Intrusion Detection Evaluation Dataset (CIC-IDS2017). Retrieved from: https://www.unb.ca/cic/datasets/ids-2017.html.

[23] Carrera, F., Dentamaro, V., Galantucci, S., Iannacone, A., Impedovo, D., & Pirlo, G. (2022). Combining unsupervised approaches for near real-time network traffic anomaly detection. *Applied Sciences, 12*(3), 1759. https://doi.org/10.3390/app12031759.

[24] Vinutha, H. P., Poornima, B., & Sagar, B. M. (2018). Detection of outliers using interquartile range technique from intrusion dataset. In *Information and Decision Sciences: Proceedings of the 6th International Conference on FICTA* (pp. 511-518). Springer Singapore. https://doi.org/10.1007/978-981-10-7563-6_53

[25] F-score. Retrieved from: https://deepai.org/machine-learning-glossary-and-terms/f-score.

[26]Precision vs Recall in Machine Learning. Retrieved from: https://levity.ai/blog/precision-vs-recall.
[27]What is a Confusion Matrix in Machine Learning. Retrieved from: https://machinelearningmastery.com/confusion-matrix-machine-learning/.

Annex I: Cloud-native security

Standards and best practices

Embracing containers, as one of the central transformations in the Cloud-native environment, requires new security standards and best practices to be attained in the design of solutions following the standardization work from the use of a high number of technologies of the 5G puzzle.

To that aim, this Annex presents the concepts, standards and best practices aiming to contribute to Cloud-native security from specialized groups. Its adoption in the HSPF of 5G-EPICENTRE will contribute to stopping attackers and defending against their threats. Thus, this section addresses the Zero Trust Architecture, Security By Design, UML, SecDevOps, Digital Signatures, Policies, Authentication and Authorization, Compliance Auditing and Supply Chain Security.

Zero Trust Architecture

Zero trust term was coined by John Kindervag [A1],represents the evolution of the concept of *de-perimeterization* and represents a paradigm in cybersecurity. Such a paradigm departs from the principle that trust is not implicitly granted, but instead, it should be specially focused on continuous assessment of resource protection per transaction basis. In such a paradigm, protection should consist in minimizing access to resources, including data, processing units and applications/services to only those subjects and assets identified as needing access, as well, as continually authenticating, and authorizing the identity and security posture of each access request.

A zero-trust architecture provides a set of cybersecurity principles for enterprises that departs from the zero trust principles primarily focused on data protection to prevent breaches and limit internal/lateral movement. It can also be expanded to consider protection for assets (devices, infrastructure components, applications, virtual and cloud components) and subjects (end users, applications and other non-human entities that request information from resources). Zero Trust Architecture (ZTA) is an end-to-end approach to enterprise resource and data security that encompasses identity (person and non-person entities), credentials, access management, operations, endpoints, hosting environments, and the interconnecting infrastructure.

The initial focus should be on restricting resources to those with a need to access and grant only the minimum privileges in terms of their needed operations. Traditionally, the focus was put on perimeter defence and authenticated subjects were given authorized access to a broad collection of resources once on the internal network. As a result, unauthorized lateral movements within the environment have been one of the biggest challenges.

Moreover, in [A1] NIST also proposes a zero-trust architecture as the response to the increasing complexities from the typical enterprise's infrastructures. A single enterprise may operate several internal networks, remote offices with their own local infrastructure, remote and/or mobile individuals, and cloud services. This complexity has outstripped legacy methods of perimeter-based network security as there is no single, easily identified perimeter for the enterprise. Perimeter-based network security has also been shown to be insufficient, since, once attackers breach the perimeter, further lateral movement is unhindered.

Security-by-design

In an effort to mitigate any security-related threats, more and more organisations are adopting security-by-design (SBD) for automating their data security controls. In that way, cybersecurity is considered from the early development stages rather than being an afterthought once software has been designed and developed. Thus, through its implementation from the very beginning of a project, SBD purpose is to prevent any cybersecurity breach from happening rather than repairing issues that have not been prevented. SBD is a framework that aids organisations to constantly manage, monitor and maintain their cybersecurity risk governance and management.

In general, incorporating security from the early development lifecycle of a software could significantly reduce the attack surface, which can be achieved by taking actions such as keeping the attack surface small, implementing authentication for different components of the software, encrypting confidential data, constantly monitoring inputs and keep components of the system separated. Lastly, treating the project as a living system and therefore, continually scanning and reviewing for security-related weaknesses can further reduce the likelihood of a breach [A2].

In 5G EPICENTRE, SBD is utilised for addressing security and privacy threats from the early stages of the development, as PPDR is a domain that requires resilient communication that can cope with emergencies and crises. Thus, the communication network shall be protected against not allowed monitoring and intrusion. A detailed analysis of the SBD, its lifecycle, the roles, and responsibilities of individuals involved in the different phases of the system development lifecycle (SDLC), including the project manager, developer, security officer and system administrator, is provided in D1.5: "Security-by-design toolkit".

Unified Modelling Language

Unified Modelling Language (UML) is a modelling language that is used for visualising the design of a system in a common, standardised way. As it is a simplified visual tool rather than a programming language, it can be utilised as a common medium for individuals from different backgrounds that should cooperate for the design and development of a system, such as software engineers, system architects and business analysts. However, UML does not inherently contain security features nor does consider security aspects in its execution. SecureUML and UMLsec are UML extensions that have been created to fill this gap and address security concerns from the beginning of the SDLC. The integration of security can prevent negligence with regards to security, as it is constantly reminded during the design and development phases. Thus, security violations are continuously considered and examined, which makes any system security-centric.

SecureUML is based on role-defined access control whilst imposing authorisation constraints. Through this extension, roles and access levels are defined via tags in the model design, such as User, Role, and Permission. An example showcasing the usefulness of SecureUML models is their ability to generate access control infrastructures that prevent errors during the implementation of access control policies and enhance the development of secure systems.

On the other hand, UMLsec is a UML profile extension that evaluates a project's specifications for any vulnerability by encapsulating security engineering patterns. It is a tool that can address security from an early stage and can be utilised by developers who are not proficient in security aspects. Any system-related weaknesses are expressed as checklists that are made available to the developers, therefore, simplifying the process of security consideration and implementation [A3].

Both SecureUML and UMLsec extensions are valuable tools for any security-oriented UML-based system design. Considering the analysis of Security Processes that are foreseen in 5G EPICENTRE, as presented in D1.5: "Security-by-design toolkit", and the requirements of the project, UMLsec is the most suitable extension to use, as it has a straightforward approach and a dedicated separation of roles per user with their respective activities and responsibilities.

SecDevOps

DevOps is the approach encompassing a set of cultural values together with the necessary tools and practices that move the step of continuous development of software to the production environment [A4] linking the development team to the operations team. SecDevOps implements the SBD principle by using automated security review of code and automated application security testing. It is the process of integrating secure development best practices and methodologies into development and deployment processes.

Digital signatures

Digital signatures are recognized as an important approach for maintaining the integrity of any data transferred all around the web. It's an overall appropriate solution that provides much in the way of trust and security, in a world where credentials are at constant risk of being stolen and any machine is at constant risk for abuse, such as bitcoin mining. Typically, digital signatures are used for signing emails, or to sign apps in popular stores such as in Google Play or Apple's App Stores supported using TLS certificates.

Policies

Policies incorporate in a set of rules, the important knowledge about how organizations should comply. They may target different purposes, such as the legal requirements, technical constraints and avoid repeating mistakes. The rules dictating the policies can be written and can also be established conventionally as part of the culture of an organization. Policies can be manually enforced, encoded in software, or even specified declaratively.

By decoupling the policies from the software, it will help the organizations to adapt to the external environment, avoiding coding them in advance at the time the application was developed. Policies specified declaratively decouple them from the code. They can be updated at any moment, without recompiling or redeploying, and can also be enforced automatically. Such an approach will contribute to deploying software services at scale. It also turns them suitable to changing business requirements, improves the ability to discover violations and conflicts, increases the consistency of policy compliance, and mitigates the risk of human error.

Policies can be useful to describe the way how a cluster should be deployed. They can dictate the allowed network routes, impose rate limits, or identify which servers are trusted. Authorizations are examples of policies governing who can take the actions over which resources. For example, a cloud computing service could answer questions such as: "can compute capacity be added?"; "to which regions can that capacity be added?"; and even identify which instances are not running in the correct region.

By putting policies early on, it will allow applying CI/CD security to automate checks of this important pipeline. Policies can then be stored in documents and provide the source for permissions.

Authentication and authorization

The adoption of an Identity and Access Management (IAM) framework of policies enables organizations with security capabilities to ensure that users have the appropriate access by managing users' permissions, be it individuals, or groups. The framework can allow or deny access to the available resources according to the underlining categories of the assets to be securely managed, including the role of users, organizations, geography, *etc*. Administrators define the credentials for their users and when they access them, those credentials are requested and validated before granting them access to the requested resources. Moreover, the requested context is evaluated against the defined policies and after that, with all the policies in place, a specific and single unique access to the resource been requested is provided.

Compliance auditing

Compliance is about regulations and checking a list of methodologies to be applied. Continuously checking compliance after releasing software, is a very important activity to assure that the released software is still up to date, regarding security aspects and others. It's also useful to verify such compliance using appropriate frameworks such as GDPR and NIST PCI DSS.

This endeavour should start in the early stages by performing experiments and engaging the compliance teams. It will be important to depart from an approach in terms of security by considering the basics. Therefore, it will be important to bring visibility over the assets and take inventories of the cloud infrastructure to that aim. In

case they are not made visible, it will not be possible to apply security to them. Therefore, it will be important to start with visibility in place, before complexly is increased, and then implement policy as guardrails. After that, the focus should be put on automation, to reduce human error by assuring continuous assessment for security, and finally, plan for scale.

Supply chain security

The usage of images registries can contribute to security and prevent cyber-attacks. Attackers realized that they are capable to pull off a supply chain attack such as in the SolarWinds attack [A5]. Supply chain attack hackers got into SolarWinds source code repository, and they inserted malicious code. After that, they pushed out their hacked code to up to 180,000 customers. Moreover, they also included a backdoor to install additional code on deployed systems.

The concept of a secure software supply chain can be applied to assessing the security of image registries. Those will contribute to streamlining the complexities to have secure environments for running containers. It will be important to discuss the best way to collect data concerning the security of registries. This security mechanism can be implemented by scripts, or manually, by collecting the registry assessment, which can be achieved through parallel work with compliance teams. Container lifecycle management will help to secure Cloud-native solutions. It will bring security controls, namely achieved by the promotion of images before their deployment. The second part of custom policies will allow the organizations to better understand their requirements.

Cloud-native security specifications

Supported using the best security practices and standards, this section specifies the aspects aiming to protect Cloud-native applications as part of Security and Privacy Framework, specifically tailored to protect the different layers of the 5G-EPICENTRE architecture. The layers comprise the infrastructure of the Cloud, Kubernetes Cluster Layer, and the Application Code running in containers and integrating the best security practices offered by Kubernetes [A6].

This document assumes the adoption of Docker as a container technology and Kubernetes as the orchestration technology to implement the Cloud-native approach by the 5G-EPICENTRE project.

Zero-Trust model

The Privacy and Security Framework follows the concept of Zero-Trust. This is defined by a premise where security is never granted implicitly; instead, access control should be continuously verified. This cybersecurity paradigm has an end-to-end architecture that encompasses identities, credentials, endpoints, operations, among others. Traditionally, the focus of restrictions is on the perimeter, and after successful authentication, minor security policies are applied to the access of the different services. With such an approach, by default, a profile with the lower level of permissions is assigned to all users and further permissions are granted as required.

Namespaces and network policies

Namespaces contribute to protecting edge privacy by isolating the administration of tenants in case of quota resource management. Despite this segmentation, this support is not sufficiently capable to avoid exploitations such as [A7]. Similar scenarios can be prevented by setting up policies to restrict traffic between namespaces, pods, and external networks.

A network policy specification consists of specifying the pods that will be subject to the policy, or to which types of policies, when applied to ingress and/or egress. In the case of ingress rules, they are applied to inbound traffic to the target pods, and egress rules are applied to outbound traffic from the target pods. Those restrictions comprise the IP address blocks, namespace, labels, protocols, and ports that are able to communicate. Because

policies are additive, restrictions applied to the traffic result from the combination of the different ingress and egress rules.

Each rule is comprised of a *NetworkPolicyPeer* for selecting pods on the other side of the connection to/from which traffic is allowed, through a Classless Inter-Domain Routing (CIDR) notation that specifies IP address blocks, namespaces, or pod labels; and a *NetworkPolicyPort* that allows to explicitly specify ports or protocols that may communicate with the pods.

Infrastructure and Southbound security

This section provides the infrastructure and southbound security specifications for infrastructure and federation layers of the 5G-EPICENTRE architecture.

Cloud

Service Based Providers (SBP) can select from VMs, containers, public, private or hybrid cloud. The choice will depend on the culture, their subscriber base, in-house skill sets, *etc*. It may include co-located computers, or a corporate data centre as the trusted computing basis for Kubernetes clusters.

The design of a system architecture is tied to the Cloud selection, needed to identify possible threats associated with cloud systems. If the Cloud layer is unsafe or misconfigured, the security risk of their components will increase.

The access to the Kubernetes control plane is provided by the API Server. It also includes the kubelet, and other core components that cooperate to schedule and run the workloads in the cluster. Similar to the physical access to a machine or Secure Shell (SSH) access, also the control plane should be carefully controlled. The flexibility of the Kubernetes access control system allows to customize users' controls to support the implementation of specific requirements, but that flexibility raises security risks. The access to the API Server should be restricted to a set of IP addresses and should not be available to the public on the internet.

Nodes should only accept connections from the control plane, on the defined ports, using network access control lists. Thus, if possible, these nodes should not be completely accessible to the public internet.

The Kubernetes control plane and nodes require permissions from each cloud provider. For the resources it needs to manage, it's better to give the cluster cloud provider access that follows the principle of least privilege.

The control plane should be the only one with access to etcd as the datastore for Kubernetes. It is possible to use etcd via TLS, according to the etcd documentation.

In the context of Kubernetes, etcd contains the state of the whole cluster, which includes Secrets. Therefore, it will be particularly important to encrypt the data storage. Encryption is the process of securely encoding data to safeguard its confidentiality. Encryption at rest refers to the use of symmetric encryption to encrypt and decrypt large volumes of data. Encrypting all storage at rest is a good practice in general.

Kubernetes

This section addresses the safeguards for the Kubernetes [A6] cluster's Infrastructure comprising the securing, the cluster's customizable components and keeping the apps in the cluster safe. To that aim, this section discusses the different approaches to have those components secure, supported using Role-Based Access Control (RBAC) [A7], authentication and authorization, and the use of secrets.

It is important to follow the recommendations to protect the cluster from accidental or malicious access.

An effort must be done to collect information on the surface of applications that present a higher exposure to attacks and from those, which, in case of attack, produce a bigger impact on the entire system. For instance, if an application includes an essential service A and other service B subject to a resource exhaustion attack, the

danger of service A becoming compromised increases considerably if resources of service B are not limited. Next, the areas of security concern and specifications for safeguarding Kubernetes workloads are addressed.

Node and Container runtime hardening. The security of the container-to-host barrier should be considered, and some examples of vulnerabilities are presented in Ruby YAML parsing (CVE-2013-0156) and Shellshock (CVE-2014-6271).

RBAC. Role-based access control (RBAC) represents a method to control access to a computer or network resources. It is based on roles associated with individual users in an organization.

A minimal security requirement is to enable RBAC in the cluster. Latest versions have RBAC enabled by default, which departs from the least privilege for humans and programs using the Kubernetes API. An appropriate configuration should enforce that each component runs with the most restrictive permissions and also that trusted components do not open doors for less privileged users.

The Kubernetes RBAC API comprises four kinds of Kubernetes object: Role, ClusterRole, RoleBinding and ClusterRoleBinding. An RBAC Role or ClusterRole contains rules that represent a set of additive permissions.

For humans, it might make sense to integrate authentication with corporate identity systems, and Kubernetes already provides plug-ins to implement custom integration authentication processes. Namely, integration with any compliant OpenID provider, such as Google and GitHub, is possible.

Authentication and authorization. The users access the Kubernetes API through the usage of kubectl, client libraries, or by REST requests. Both human users and Kubernetes service accounts can be authorized for API access. In a typical Kubernetes cluster, the API serves on port 443, protected by TLS. The API server presents a certificate signed using a private Certificate Authority (CA), or based on a public key infrastructure linked to a generally recognized CA. If a given cluster uses a private certificate authority, a copy of that CA certificate should be configured on the client, so that it is possible to trust the connection and be confident it was not intercepted.

A client can present a TLS client certificate at this stage. Once TLS is established, the HTTP request moves to the Authentication step. There, a cluster creation script, or a cluster admin takes the responsibility to configure the API server to run one or more Authenticator modules.

The input to the authentication step is the HTTP request. It examines the headers and/or client certificate. Authentication modules include client certificates, password, and plain tokens, bootstrap tokens, and JSON Web Tokens for service accounts. Multiple authentication modules can be specified, in which case each one is tried in sequence until one of them has succeeded. If the request cannot be authenticated, it is rejected with HTTP status code 401. Otherwise, the user is authenticated with a specific username and is available to perform subsequent steps of his decision.

Some authenticators also provide the group memberships of the user. While Kubernetes uses usernames for access control decisions and in request logging, it does not have a User object, nor does it store usernames or other information about users in its API.

After requests are authenticated as coming from a specific user, the request is authorized. A request must include the username of the requester, the requested action, and the object targeted in the action. The request will be authorized in case of policy dives to permissions to that the user in order to complete the requested action.

Secrets. A Secret is the Kubernetes object that can contain a small amount of sensitive data such as a password, a token, or a key. Secrets are also similar objects to ConfigMaps but are specifically intended to hold confidential data. Such information might otherwise be put in a pod specification or in a container image. Using a Secret means that developers do not need to include confidential data in their application code.

Because Secrets can be created independently of the Pods that use them, contributes to reduce the risk of the contents and data in that Secret being exposed along the workflow of creating, viewing, and editing pods. Kubernetes, and applications that run in the cluster, can also take additional precautions with Secrets, such as avoiding writing confidential data to non-volatile storage. When creating a Secret, it is possible to define its type to help the programmatic handling of different kinds of confidential data. Kubernetes provides several built-in types for some common usage scenarios. These types vary in terms of the validations performed and the constraints Kubernetes imposes on them.

Network access control. Network-level access control provides in-depth defence mechanisms, such as the protection against vulnerabilities, namely, the Heartbleed OpenSSL vulnerability (CVE-2014-0160) [A8].

The network policies can be used to limit inbound traffic to a pod, based on the namespace and labels of the originating pod, as well as the IP address for traffic coming from outside the cluster. With the same set of selectors, the network policy can likewise limit outbound traffic. Restricting ingress to the own application namespace is a good start point.

The Container Network Interface (CNI) provider takes the responsibility for enforcing network policies. A provider that implements the network policies should be used, such as Calico [A9].

In specific scenarios where an attacker has compromised one of the applications and exploited the container runtime or kernel, it is not possible to trust the node to enforce network access controls, therefore, it's necessary to apply the ingress/egress rules at the network level.

The limitation on the use of federating network policies across multiple Kubernetes clusters is related to the lack of granular access control at the network level.

Pod Security Policies

There exist three different isolation levels for pods in Kubernetes provided by pod security profiles. These profiles allow controlling how the pod activity is managed. This section will present the different regulations included in Kubernetes profiles, including Privileged, Baseline and Restricted that cover a wide range of security concerns. These regulations are cumulative and can range from extremely liberal to extremely narrow.

By decoupling the definition of policies from its instantiation it will contribute to the understanding and to the communication across clusters, regardless of the underlying enforcement method. Mechanisms will be defined on a per-policy basis as they mature. Individual policies' enforcement techniques are not defined here.

The security context defines privilege and access control settings for a Pod or Container. Security context settings include, but are not limited to:

- **Access Control:** Permission to access an object based on user ID (UID) and group ID (GID).
- **Security Enhanced Linux (SELinux):** Security labels are assigned to objects; Running as privileged or unprivileged.
- **Linux Capabilities:** Give a process some privileges, but not all the privileges of the root user.
- **AppArmor:** Use program profiles to restrict the capabilities of individual programs.
- **Seccomp:** Filter a process's system calls.
- **AllowPrivilegeEscalation:** Controls whether a process can gain more privileges than its parent process. This boolean directly controls whether the no_new_privs flag gets set on the container process. AllowPrivilegeEscalation is true always when the container run as Privileged or has *CAP_SYS_ADMIN*.
- **ReadOnlyRootFilesystem:** Mounts the container's root filesystem as read-only.

Baseline

The baseline is a minimally restrictive policy and prohibits privilege escalations. It allows using the default non-specific Pod setup. A baseline policy is designed to be simple to implement for common containerized workloads while avoiding known privilege escalations. This strategy is envisioned for non-critical application operators and developers. Next, the details of the available controls are presented.

HostProcess. Kubernetes introduced HostProcess functionality for clusters that include Windows nodes containers. Windows pods grant privileged access to the Windows node, by allowing HostProcess containers to execute. The basic policy forbids privileged access to the host.

Host namespaces. Namespaces segregate resources within a single cluster. Within a namespace, resource names are unique, but not between namespaces. Each one of Kubernetes resources can only be assigned to one namespace, and namespaces cannot be nested inside one another. The namespace-based scope is restricted to namespaced items, including Deployments and Services and not to cluster-wide objects such as StorageClass, Nodes, PersistentVolumes, *etc*. Namespaces are suitable in scenarios with large teams, with users spread across several teams or projects.

Privileged containers. A container cannot access any devices on the host by default, but a Privileged container can have access to all devices on the host. This gives the container almost complete access to the same resources as processes operating on the host. This is useful for containers aiming to use Linux features like network stack manipulation and device access. Most security methods are disabled by Privileged Pods, so, special attention is needed to the operations led by these Pods.

Capabilities. According to the Linux kernel capabilities FAQ, a capability represents a token used by a process to prove that it is allowed to do an operation on an object. The capabilities are individual units of privilege in a Linux kernel that can be independently enabled or disabled.

The most significant number of capabilities are required to influence the kernel/system, and they are used by the container runtime. Despite this, their use by processes operating inside the container is unusual. Some containers, however, require specific capabilities. For example, a container process requires *setuid* or *setgid* capabilities to drop privileges. Therefore, it will be important to have a balance between security and productivity.

HostPath Volumes. The files as part of containers are ephemeral because they live in running containers. In case of a container crash, these files are lost, because *kubelet* restarts the container in a clean state. A second problem arises when sharing files between containers running besides in the same pod. The Kubernetes volume is the abstraction mechanism that solves both problems.

A Docker volume corresponds to a directory on disk or in another container. At its core, a volume is a directory accessible to the containers in a pod. The volume type defines the source location of the directory, the medium that backs it, and its contents.

A hostPath volume mounts a file or directory from the host node's filesystem into the pod.

Therefore, the use of HostPath volumes must be forbidden.

Host Ports. Although the limitation of port ranges for communication may seem as self-evident, the ones being exposed by the service should be the ones strictly necessary for communication or metric collection.

The use of Host Ports should be forbidden, or at least significantly limited to a known list according to the specific purpose of the service. Therefore, it will be important to define how to expose applications running on the Kubernetes cluster to the outside world and define how they can have access to the Kubernetes Pods from outside of the cluster.

AppArmor. AppArmor is a Linux kernel security module that extends the usual Linux user and group permissions to limit the resources that applications can access. AppArmor can be configured for an application to limit its

attack surface and to provide further in-depth defence mechanisms. It is customized through profiles that are adjusted to give a certain program or container the access it requires, such as Linux capabilities, network access, file permissions, and so on. Each profile can be run in enforcing or complaining mode, where in the first blocking policies are enforced, originating blocks to prohibited resources, while in the second, it just reports the infractions.

AppArmor can help to operate a more secure deployment by limiting what containers can do and/or providing better auditing capabilities through the usage of system logs.

The runtime/default AppArmor profile is used by default on supported hosts. Overriding or removing the default AppArmor profile should be forbidden in a Kubernetes cluster, or, at least, be limited to a certain set of profiles.

SELinux. Security-Enhanced Linux (SELinux) is a security architecture for Linux systems that allows administrators to define who can access the system. SELinux system is supported by using labels, wherein every single process has a label. Also, every computing resource object, such as a file, directory, and system object have a label. Policy rules control access between 51 labelled processes and labelled resource objects. The kernel enforces the application of these rules.

Some compliance and security requirements, as well as policies, can be codified as SELinux rules, which can provide developers with the flexibility to build different types of apps that don't require manual auditing, while still serving as evidence of due diligence from a security perspective, which is something often required by large organizations. Setting a custom SELinux user or role option should be forbidden, as is changing the SELinux type.

The /proc Mount Type. The /proc directory contains virtual files that are windows into the current status of the running Linux kernel. The default /proc masks are set up to reduce the attack surface and this should be required.

Seccomp. Seccomp is a security mechanism for Linux processes to filter system calls (syscalls) based on a set of defined rules. Applying *seccomp* profiles to containerized workloads is one of the key tasks when enhancing the security of the application deployment. Developers, site reliability engineers and infrastructure administrators have to work hand in hand to create, distribute and maintain the profiles over the application life-cycle.

Kubernetes introduced an additional security layer on top of the existing *seccomp* support. Now, it is possible to have a securityContext defined in a field of Pods and their respective containers can be used to tune security related configurations of the generated workloads. This enhancement helps specify if the whole pod or a specific container should run as:

- **Unconfined**: *seccomp* will not be enabled.
- **RuntimeDefault**: the container runtimes default profile will be used.
- **Localhost**: a local node profile will be applied, which is being referenced by a relative path to the *seccomp* profile root of the *kubelet*.

Kubernetes now includes a *kubelet* feature gate *SeccompDefault*. It was initially included as an alpha feature state, meaning that it is disabled by default. It can be manually enabled for every single Kubernetes node. The feature changes the default *seccomp* profile from *Unconfined* to *RuntimeDefault*. In case it is not specified differently in the pod manifest, then the feature will add a higher set of security constraints by using the default profile of the container runtime. CRI-O or container runtimes may implement these profiles differently and they also may differ in terms of hardware architectures. Those default profiles block the more dangerous *syscall* and allow a common number of other ones, which are unlikely or unsafe to be used in a containerized application. Therefore, it is not necessary to set the Seccomp profile to Unconfined.

Sysctls. Sysctls can deactivate security measures affecting all containers on the host, therefore they should be avoided unless a "safe" subset is allowed. If a sysctl is namespaced in the container or Pod, and isolated from other Pods or processes on the same Node, it is deemed to be safe.

Privileged

The Privileged policy is characterized by the absence of constraints and has no restrictions, thus granting a larger number of permissions. As result, some known privilege escalations are allowed under this policy. This policy should be applied to workloads controlled at the system and infrastructure levels by privileged, trusted users. This policy is characterized by the lack of imposed limitations, instead, an instantiated profile for allow-by-default enforcement methods is used. Next, the controls that should be enforced or disallowed are presented.

Restricted. The restricted policy departs from the baseline profile and includes a significant number of constraints, following pod hardening best practices. It is aimed at security-critical application operators and developers, as well as lower-trust users. The controls stated below should be enforced or disallowed:

- **Volume Types:** The policy restricts the use of non-core volume types to those described by PersistentVolumes, in addition to HostPath volumes.
- **Privilege Escalation:** Privilege escalation through the use, for instance of set-user-ID or set-group-ID file mode, should be avoided.
- **Running as Non-root:** Containers must be required to run as non-root users.
- **Non-root groups (optional):** Containers should not be allowed to use a root primary or supplemental GID.
- **Seccomp:** it must be expressly set in the Seccomp profile. The Unconfined profile, as well as the lack of a profile, are both forbidden.
- **Capabilities:** Containers are only allowed to bring back the NET BIND SERVICE capability after dropping all other capabilities.

Quality of Service. When a pod is created in a Kubernetes cluster it is possible to assign one of the following QoS classes: Guaranteed, Burstable and BestEffort. Next, the conditions to be met in the application of QoS classes are discussed.

A pod can have assigned a QoS class of Guaranteed in the case:

- Every container in the pod must have a memory limit and a memory request.
- For every container in the pod, the memory limit must equal the memory request.
- Every container in the pod must have a CPU limit and a CPU request.
- For every container in the pod, the CPU limit must equal the CPU request.

A pod is given a QoS class of Burstable in the case:

- The pod does not meet the criteria for QoS class Guaranteed.
- At least one container in the pod has a memory or CPU request.

For a pod to be given a QoS class of BestEffort:

- Containers in the Pod must not have any memory or CPU limits or requests.

Network Policies. To manage the traffic flow at the IP address or port level of the Open Systems Interconnection Model (OSI) layer 3 or 4, the use of NetworkPolicies in Kubernetes for certain applications should be evaluated. A pod is able to describe how it can communicate with the different network entities, endpoints, and services using NetworkPolicies, an application-centric concept. A combination of the three IDs below is used to identify the entities with whom a Pod can communicate:

- Other allowed pods.
- Allowed namespaces.
- IP blocks, which exception is the traffic to and from the node where a pod is running. This traffic is always allowed, regardless of the IP address of the pod or the node.

When defining based NetworkPolicy based in a pod or namespace, a selector will specify the allowed traffic to and from the pods matching the selector. Meanwhile, when IP based NetworkPolicies are created, the policies based on IP blocks (CIDR ranges) are defined. Network policies are implemented by the network plugin. To use network policies, it will be required to use a networking solution supporting NetworkPolicy. Creating a Network-Policy resource without a controller that implements it, will not produce any effect.

By default, pods are non-isolated and can accept traffic from any source. Pods become isolated by having a NetworkPolicy that selects them. Once there is a NetworkPolicy in a namespace selecting a particular pod, that pod will reject all connections that are not allowed by that NetworkPolicy, however, other pods in the namespace that are not yet selected by any NetworkPolicy will continue to accept all traffic.

Network policies are additive, and, in that sense, they can conflict between them. In case a policy selects a pod, the pod is restricted to what is allowed by the union of those policies' ingress/egress rules. Thus, the order of evaluation does not affect the policy result.

Therefore, in order to allow the network traffic between two pods, both the egress policies on the source pod and the ingress policy on the destination pod, need to allow the wanted traffic. If either the egress policy on the source or the ingress policy on the destination denies the traffic, the traffic will be denied.

mTLS and TLS for Ingress. Strong application-layer authentication, such as mutual TLS (mTLS), can represent a solution to the problem of network-level access controls.

The encryption of all the communications by the use of TLS handshake with the client ahead of time in case the application code needs to interact through TCP. Encrypting network communication between services will be important and mTLS authentication is a mechanism that allows for that verification, performing a two-sided verification over communications.

The Kubernetes Ingress can help secure an Ingress resource by specifying a Secret that contains a TLS private key and a certificate. This Ingress resource only supports a single TLS port, 443, and assumes TLS termination at the ingress point (traffic to the Service and to the Pods are in plaintext). If the TLS configuration, in an Ingress, specifies different hosts, they are multiplexed on the same port according to the hostname specified through the SNI TLS extension (provided the Ingress controller supports SNI). The TLS secret must contain a certificate and a private key to use for TLS, named *tls.crt* and *tls.key*.

Devops Security at Back-End Layer

This section presents the DevOps security approaches at the backend layer, hardening images, supply chain security, container security, third party dependency, source code analysis tools and dynamic probing attacks and finally VM-level container isolation. Besides these, the already explained concept of SBD will also be followed.

Hardening images

This section presents the options for hardening the container images. They are: (i) restriction of the accessibility of the port; (ii) adoption of a strategy to reduce the data image size; (iii) implement a methodology to divide the segments of the network and reduce the overall exposure to the web.

It will be important to restrict network port accessibility. During the creation of a container, a developer might allow access to additional network ports for troubleshooting or debugging purposes, but, during the execution phase, the open ports should be reduced to the possible minimum.

It's usual to maintain log files containing API secrets, and other data along the container image build process that won't be in the final container image. Using *pf.dockerignore*, a specific set of files and folders can be explicitly defined to not be present in the *build* environment. This prevents the unintentional disclosure of any sensitive information or credentials.

A third approach for hardening is to shorten the base image length to make it as thin as feasible. By including the lower number of system files or apps in the container image, horizontal network movement possibilities are reduced, therefore, the chances of a container being hacked are also smaller. The images should depart from Alpine Linux container images in order to achieve this goal.

Containers running in production mode should enforce a strict read-only policy. Any web service that is exposed to the public internet should have no unnecessary writeable hard drive locations. To store or handle client information, the service should instead rely on secure network connections to databases.

The fourth step is to properly partition the network based on the architecture of the application. As a result, public and private services can be divided, so that all containers are not placed on the same flat network. Web servers are an example of a service that can be included in a public network segment, although backend services, such as database containers, do not need to be exposed to the public internet. As a result, web services should only have a very narrow network link to the databases, reducing the risk of the databases being exposed to the internet, during some security breach.

Supply chain security

Container build, test, and orchestration are the stages in a supply chain of container images.

Access control can be applied using well-defined access policies together with corporation ones in order to understand who is doing what. (Who is signing, who is promoting, who is sharing which build images, *etc*.). Image scanning is also a must at the binary level and it's crucial to validate the security of a specific image using common vulnerabilities, usually specified in security vulnerability databases (CVE DB). Image signing is another way of guaranteeing the authenticity of the image, as well, as of the publisher. CI tools can also apply signatures and verify them before running, which prevents the execution of not genuine images.

In terms of Image lifecycle management, it is important to always keep the installed image in its most updated version.

Container security

This section provides the specifications regarding the security of the container layer. The following topics are addressed: vulnerability scanning and OS dependency security, image signing and enforcement, the definition of the privileged users, the use of container runtime classes.

Vulnerability Scanning. Containers are checked for known vulnerabilities along the image build stage. Those vulnerabilities could be in the container image's configuration, with the instructions for how to launch it, or in any of the components included as dependencies that the application requires.

Image Signing and Enforcement. Despite Kubernetes not including native support for image signatures or their verification, to maintain a system of healthy containers, images for containers should be signed.

For signing images, it is possible to include a specific tag or a digest. The image content can change over time and be overwritten by subsequent versions of the image. Because the digest is a SHA256 hash of the image content, that cannot be modified, and it is unique for each image. As a result, an image can be tagged with either image:tag or image@sha256:d15754...

Docker Content Trust, [A10] Docker-developed method for signing images, is inextricably tied to Notary, another Docker-developed technology. *Notary* is a service that preserves manifest files signed by a trusted party for "trusted resources". This means that all signed, or soon-to-be-signed images are "trusted resources" for which manifest files are available in Docker Content Trust. The manifest files are signed with a private key and give a 1:1 mapping between tags and digests.

Connaisseur [A11] is an example of a Kubernetes admission controller to increase security. It allows only signed images in a cluster and ensures only trusted and unmodified content is deployed. To do so, it intercepts resource

creation or update requests, sent to the Kubernetes cluster, identifies all container images, and verifies their signatures against pre-configured public keys. Based on the result, it either accepts or denies those requests.

Privileged users. It will be important that the users created within each container will have the least amount of operating system permissions needed to accomplish the purpose of the container. This is aligned with the Zero-Trust model, already presented in Section Annex I: Cloud-native security.

Container runtime classes. Runtime classes for containers are able to bring additional isolation. RuntimeClass is a feature for configuring the container's runtime. Containers in a pod are operated using the container runtime configuration. To achieve a mix of performance and security, it is possible to assign various RuntimeClasses to separate Pods. For example, if a given workload requires a high degree of data protection, those Pods can be scheduled to operate in a container runtime supported by the use of hardware virtualization. Therefore, it will be possible to benefit from the alternative runtime's improved isolation at the cost of some additional overhead.

RuntimeClass may also be used to run several Pods with the same container runtime with different configurations.

Third-Party dependency, source code analysis tools and dynamic probing attacks

It's important to regularly perform a scan on applications' third-party libraries for known security vulnerabilities. Such a process may be performed automatically in any programming language.

Most programming languages allow the assessment of the source code for any potentially dangerous coding practices. Source code analysis tools, known as Static Application Security Testing (SAST) tools, can help to assess source code or compiled versions of code to help find security flaws. SAST tools can be added to the Interface Development Environments (IDE) and used to detect issues during software development. The achieved feedback can save time and effort, especially in comparison to finding later vulnerabilities across the development cycle.

The adoption of automated tools that can scan codebases for common security problems whenever possible should be considered. Automated tools exist to test some of the most well-known service threats against services, including SQL injection, CSRF, and XSS. The OWASP Zed Attack proxy tool is one of the most popular dynamic analysis tools. An extensive list of the tools is available at [A12].

VM-level container isolation

Traditional NFV decouples NFs from hardware, where VNFs running on VMs are interconnected to serve as building blocks for the setup of virtualized environments. eMBB, mMTC and URLLC often contradict in terms of requirements. Despite that, their needs can be satisfied through Cloud-native technologies, both at the network edge and at the network core. To this end, containerization has emerged as a key enabling software technology, particularly lending itself to deploying Cloud-native applications.

Containers can be defined as 'packages', containing all the software code and dependencies so that the application can be executed rapidly and reliably in different computing environments. By utilizing containers, developers break down their purpose-built hardware and software solutions into packages of microservices and implement them in containers. To this end, application component instances are built to be executed inside containerised environments, such as Docker and Kubernetes. These can tap into cloud-based support layer services (such as databases and middleware) that also run as microservices inside their own containers.

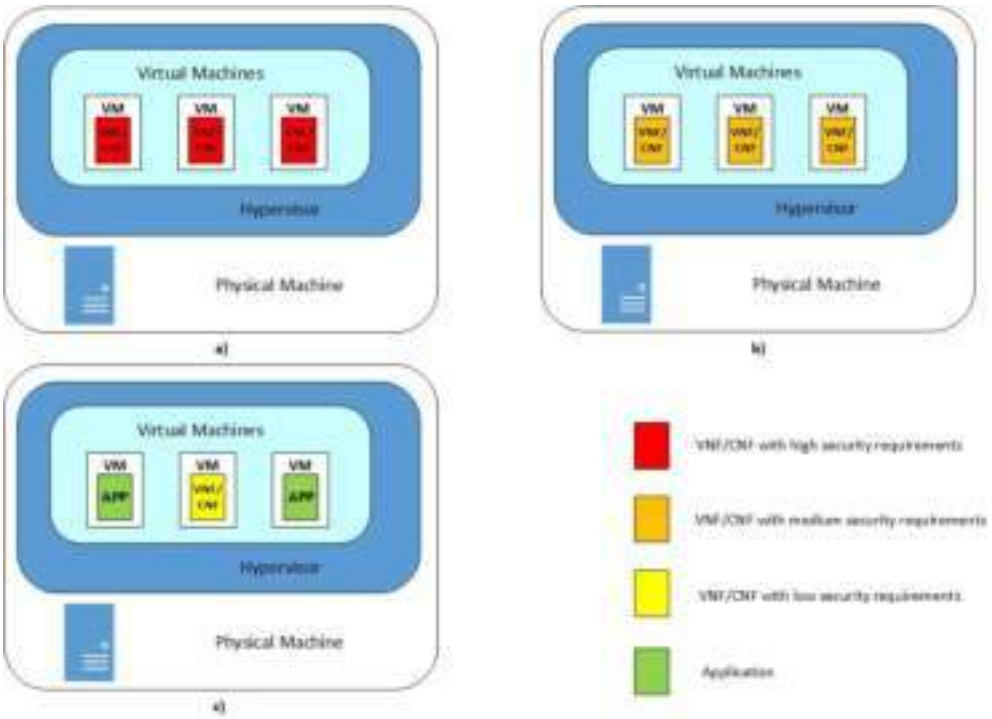
While not mutually exclusive, containers and VMs often represent different approaches, each with its own sets of pros and cons. Containers consume much less in both memory and disk space and are much faster to start. VMs provide more isolation and better security. The VM Hypervisor makes partitions of the physical resources and makes isolated entities and allows each VM to run independently. Isolation and Abstraction characteristics of the Hypervisor offer an additional security feature. In case if any VM is compromised, Hypervisor removes this VM and restores it to pre-attack state. However, the heavy footprint of current NFV platforms hinders the use

of VMs at the network edge. Security-by-design philosophy must be rigorously applied so as not to compromise security aspects during the transformation from monolithic VM-based approaches to container-based approaches.

VMs and containers have different architectures, though they share some similarities. Both containers and VMs provide isolation to varying degrees. VMs are self-sufficient, have their own operating system and do not share resources with other VMs. Containers share hosts with other containers, complicating the idea of a secure boundary.

A viable approach, concerning security, could be to deploy Kubernetes clusters within VMs. A Kubernetes Virtualized Infrastructure Manager (VIM) can coexist with a conventional *(e.g.*, OpenStack) VIM, sharing resources under a common orchestrator (*e.g.*, ETSI OSM). Containers with the same security requirements can be grouped in the same VM, as VMs have better isolation. Based on the criticality/security requirements we can define 3 security perimeters described below and as depicted in Figure 12:

- **Security perimeter 0**. Contains only VMs of high security requirements. The VMs are in dedicated (physical) servers. The VNFs/CNFs with high security requirements are put in such VMs.
- **Security perimeter 1**. Also contains VMs with low security requirements, shared in the same servers. VNFs/CNFs with different security requirements are put in separate VMs (but in the same physical machine). The isolation is ensured by the hypervisor (Nova in Openstack).
- **Security perimeter 2**. Contains the applications and everything else.



a) Security perimeter 0, b) security perimeter 1, c) security perimeter 2

Figure 12: Security perimeters and VNF/CNF placement

References

[A1] Kindervag, J. (2010). Build security into your network's dna: The zero trust network architecture. *Forrester Research Inc*, 1-26.

[A2] Security By Design: What Is It and How to Do It Right? https://www.spiceworks.com/it-security/cyber-risk-management/articles/what-is-security-by-design/

[A3] Jürjens, J. (2005). Sound methods and effective tools for model-based security engineering with UML. In *Proceedings of the 27th international Conference on Software Engineering* (pp. 322-331). https://doi.org/10.1145/1062455.1062519.

[A4] Davis, J., & Daniels, R. (2016). *Effective DevOps: building a culture of collaboration, affinity, and tooling at scale*. O'Reilly Media, Inc.

[A5] NPR. (2021). A 'worst nightmare' cyberattack: the untold story of the solarwinds hack [Blog]. *Untangling Disinformation*. https://www.npr.org/2021/04/16/985439655/a-worst-nightmare-cyberattack-the-untold-story-of-the-solarwinds-hack?t=1628521916151.

[A6] *Kubernetes Documentation: Security*. Retrieved from: https://kubernetes.io/docs/concepts/security/.

[A7] Felix, C., Garg, H., & Dikaleh, S. (2019). Kubernetes security and access management: a workshop exploring security & access features in Kubernetes. In *Proceedings of the 29th Annual International Conference on Computer Science and Software Engineering* (pp. 395-396). Retrieved from: https://dl.acm.org/doi/abs/10.5555/3370272.3370337

[A8] *CVE-2014-0160 Detail*. (2021). Retrieved from: https://nvd.nist.gov/vuln/detail/CVE-2014-0160.

[A9] *Calico*. Retrieved from: https://www.tigera.io/project-calico/.

[A10] *Docker docs: Content trust in Docker*. Retrieved from: https://docs.docker.com/engine/security/trust/.

[A11] *Connaisseur*. Retrieved from: https://github.com/sse-secure-systems/connaisseur.

[A12] OWASP. (2021). *Source Code Analysis Tools*. https://owasp.org/www-community/Source_Code_Analysis_Tools.

Annex II: Proposed Security and Privacy Framework – Background

The adoption of Service Mesh

The classical in-depth network security approaches, such as perimeter firewalls cannot be easily applied to Cloud-native scenarios, such as the ones envisioned by 5G-EPICENTRE container-based approaches. These require finer control regarding the network communications between all the different containers.

In the literature, the Service Mesh concept is discussed as a Cloud-native approach to bring additional security features. This concept can support different security capabilities including logging of API traffic, observability tagging, network traffic encryption, authentication, and authorization. Beyond the centralized management of the policies, it can also be used to support policy enforcement between different edge/cloud network traffic in Cloud-native 5G scenarios.

A Mesh entails a group of hosts that are coordinated to provide a consistent network topology. A Service Mesh is a method that allows having visibility and control of how the different components of an application share data. A Service Mesh is a dedicated infrastructure layer included directly in an application. Considering its visibility over the infrastructure layer it can report the performance of the interactions occurring between the different components. It also facilitates the optimization of communication, avoiding "downtime", as the application evolves. Each component of the application is a service depending on others to meet the needs of users.

A Service Mesh introduces a dedicated infrastructure supporting microservices-based application services such as authentication and authorization, and security monitoring. In the case of Cloud-native applications, created in a microservice architecture, the Service Mesh is a way of encompassing a large number of different services in a functional application.

In the case of a security incident or other error, Service Mesh logs can be used to identify the root cause of the incident, since it records the interactions between the different services.

The Service Mesh concept mainly relies on the usage of network proxies (the sidecars), deployed together with each container instance, for intercepting and controlling the network traffic among microservices. They allow monitoring and are suitable for security-related concerns. They also offer an abstraction layer for individual services or applications, by providing a sidecar data plane at every app (CNF container).

Service Mesh can intercept all ingress and egress container traffic. This capability enables CNF sidecar traffic capture, including intra-node CNF traffic and *preencryption* tapping, and reduces SSL load for brokers. The service proxy easily integrates with existing infrastructure, provides full packet visibility, is scalable and extensible, and uses existing packet broker APIs.

The Service Mesh does not introduce functionalities to the application's execution environment. It manages the communication between the services from the individual context to an infrastructure layer. The requests are routed between microservices using proxies in their own infrastructure layer. The proxies implementing the Service Mesh are called **sidecars** because they are decoupled and executed in parallel from each service.

Traffic management, security and visibility are the important features that will contribute to the adoption of distributed microservice architectures. Traffic management includes the configuration of the rules and traffic routing. It also makes possible the control of the traffic flow and communication between services. Security capabilities can be applied over the communication channels for managing the authentication, authorization, and encryption of the communication of services at scale. It also includes the capabilities to consistently apply the policies to different protocols and execution environments, while not demanding changes in applications.

Generally, modern applications comprise several business services in a network-dependent on each other. In case of some services become overloaded, the Service Mesh can forward requests from one service to the next and optimize how all those elements can work together.

Each new service added to the application, or a new instance of an existing service executed in a container, introduces new points for possible failures, meaning additional complexity for the communication environment. In such a complex microservice architecture, it is difficult to locate where problems occur without a Service Mesh. This happens because the Service Mesh also captures all aspects of service-to-service communication as performance metrics. Over time, it is possible to apply the data provided by the Service Mesh to the inter-service communication rules to increase the efficiency and reliability of service requests.

For example, in case a given service fails, the Service Mesh can register the downtime of the service. Since the service failure periods accumulate, it is possible to write rules to determine the ideal waiting time before making a new attempt. Such a process will contribute to having the system not overloaded with unnecessary attempts.

Different approaches can be adopted for deploying Service Mesh components. Therefore, they can be embedded into the application as a microservice, coupling them to the application code by implementing them as libraries, or implementing them as service proxies independently of application code. This last alternative has been highlighted as the most efficient one in terms of scalability and flexibility in many scenarios supporting infrastructure for microservices-based applications.

Without the Service Mesh, it would be necessary to code each microservice with the logic that manages service-to-service communication, resulting in less time for programmers to focus on business goals.

Service Mesh will facilitate the DevOps teams by reducing the complexity of deployments. With a Service Mesh, the DevOps teams can deal with the change from monolithic applications to Cloud-native applications, including the collection of small, independent, and lightly coupled microservice applications, at a reduced effort.

Mutual TLS (mTLS) is an important tool that enables to securely encrypt and trust in the communications occurring between the different components in the network, which is increasingly important to Service Mesh deployments.

Technologies and tools

This section presents the technologies and tools used in the instantiation of the HSPF.

Istio

Istio [B1] is a realization of an open-source Service Mesh platform enabling to control the way how microservices share data between them. It comprises a set of layered distributed applications providing traffic management, security, and observability at the Service Mesh level.

It contains concepts as roles and bindings and allows the granting of permissions to identities, throughout RBAC policies. Their APIs can enforce access control at mesh, namespace and service levels. As a result of applying policies over a Kubernetes' network, its benefits result in the ability to configure security between pod or service communication at the network and application layers.

The available APIs provide support for integration with other components, such as telemetry or policies systems. It was also developed to be executed over distinct environments, including on-premises, hosted in the cloud, Kubernetes containers or even in services running on VMs.

Istio is an open-source technology, which aims to bring security, management, and monitoring of services to a more transparent and centralized level. It is distinguished by its properties of load balancing, secure service-to-service communication (using TLS encryption) and access control, which are achieved through a pluggable policy layer. It runs over Kubernetes and allows a very large set of operations, such as: adding applications to a cluster; extending the mesh to other clusters; connect VMs or other machines located outside of Kubernetes. In terms of internal operations, its logical components are split into data and control plans.

The data plan comprises Envoy proxies as sidecars handling the compatibility issues. Those sidecars proxies intermediate the communication with other proxies and run parallel to the microservice. All those proxies together form a mesh network intercepting the communication between microservices, mediating, and controlling all network communications and collecting telemetry. Communications are mediated by proxies, specially designed to intercept, and route all the network traffic.

At the control plane, the proxies provide the capabilities to configure the components that apply policies and collect telemetry beyond the management and configuration for routing the traffic.

Kiali

Kiali [B2] is a management console for service fabrics based on Istio. It provides the representation of the structure of the service fabric by inferring the traffic topology and displays the health of the fabric. Kiali provides detailed metrics and allows the creation of validation policies. It also allows integration with Grafana and Jaeger.

It was one of the used tools to collect metrics, observe the health of microservices and even to apply security policies. It also helps to define, validate and observe the network of Istio services.

Kiali provides an interactive graphical view of each namespace in real-time that provides visibility into features such as circuit breakers, request rates, latency and even traffic flow graphs. The information made available is related to the components at various levels, applications, services, and workloads. It can display interactions with contextual information or in graphical form. Kiali also offers the ability to validate Istio configurations such as gateways, destination rules, virtual services, fabric policies and much more.

Grafana

Grafana [B3] is an open-source monitoring solution that can be used to configure dashboards for Istio. Grafana can help monitoring the health of Istio and applications in the service fabric. This tool allows to study, analyze and monitor data over a period of time. It may also be used to represent the user behaviour, application behaviour, how often errors arise in production or in a pre-production environment, types of errors that appear, and contextual scenarios, among other scenarios.

Dashboards extract information from connected data sources such as Graphite, Prometheus, Influx DB, ElasticSearch, MySQL, PostgreSQL *etc*. These are some of the many data sources that Grafana supports by default. Dashboards contain a range of visualization options, such as geographic maps, heat maps, histograms, among others. Basically, it provides all the variety of tables and graphs that a business typically requires to analyse data.

Curiefense

Curiefense [B4] is an API-first, GitOps-based web-defense HTTP-Filter adapter for Envoy Proxy. It provides various security technologies (WAF, application layer DDoS protection, *bot* management and more) along with real-time traffic and transparency monitoring. Curiefense is fully programmatically controllable. All configuration data (sets of security rules, policies, *etc*.) can be kept individually or as different branches for different environments according to the user needs. It adds traffic filtering capabilities to containers, service fabrics, inbound gateways and many other components of modern topologies.

Curiefense eliminates the need for external solutions; all traffic filtering is done within its perimeter. Traditional traffic filtering is performed outside the protected entity (application, service, API, *etc*.). Traffic filtering can be configured differently for different environments (*e.g.*, dev / qa / prod), and all can be administered from a central cluster.

Curiefense may be used to protect different types of resources, such as a website, an application, a service or an API. Users represent a source of traffic trying to access that resource. Incoming traffic passes through Envoy,

which uses Curiefense as an HTTP filter, so the most hostile requests will be blocked. The other components represent the Curiefense platform itself, as follows:

- Proxy Curiefense connects to Envoy proxy and performs traffic filtering.
- DB logs: Curiefense stores traffic data (headers, payloads, *etc*.) of all requests here.
- Metrics: A Prometheus traffic metrics store.
- Dashboard: Dashboard(s) using Grafana with visual displays of traffic metrics.
- Web UI: Curiefense web console for platform configuration.
- Configuration server, a service that:
    a) Receives web UI configuration edits.
    b) Receives configuration edits from API calls.
    c) Creates new configuration versions in response to edits.
    d) Stores the new version in one or more Cloud Storage ranges.
- Cloud Storage: Stores versioned configurations. Each Curiefense proxy periodically checks the Cloud Storage, and when a new version is found there, the proxy downloads and updates its security posture.

Open Policy Agent

The Open Policy Agent (OPA) [B5] is an open-source, general-purpose policy engine that unifies the implementation of policy enforcement procedures across the IT environments, such as the ones involving Cloud-native applications. OPA was originally created by Styra and has since been accepted by the CNCF. OPA provides a high-level declarative language to specify policy as code, and simple APIs to offload policy decision-making from software. OPA can enforce policies in microservices, Kubernetes, CI/CD pipelines, API gateways, and more. OPA can be used to control authorization, admission, and other policies in Cloud-native environments, with a focus on Kubernetes.

OPA is a lightweight general-purpose policy engine that can be co-located to the existing services. OPA can be integrated as a sidecar, host-level daemon, or library. OPA allows to decouple policy decisions from software for policy enforcement. It also avoids the development of custom languages for policy management, including the definition of a syntax, semantics, and the development of an evaluation engine (that would need to be carefully designed, implemented, tested, documented, and then maintained to ensure correct behaviour and a positive user experience).

OPA is a general-purpose policy engine that decouples policy decision-making from policy enforcement. Its high-level declarative language provides intuitive ways of specifying policies. It can be used to enforce policies on several environments, namely on microservices and Kubernetes. Whereas Istio policies are limited to networks, OPA allows a more comprehensive strategy to implement distinct policies and take more control over deployments and containers.

Services offload policy decisions to OPA by executing queries. OPA evaluates policies and data to produce query results for the client(s). Policies are written in a high-level declarative language and can be loaded dynamically into OPA remotely via APIs or through the local filesystem.

Declarative policies in OPA's policy are defined in Rego language. OPA generates policy decisions by evaluating the query input against policies and data. OPA and Rego can be used to describe many different policies and they are agnostic to the domain, such as in the following cases:

- Which resources are allowed to each user.
- Which traffic should be allowed to each subnet by the egress controller.
- To which clusters, a certain workload must be deployed to.
- From which locations can the registries binaries be downloaded from.

- Which OS capabilities may a container use.
- To which periods of the day may the system can be accessed at.

Policy decisions are not limited to simple yes/no or allow/deny answers. Like query inputs, policies can generate arbitrary structured data as output. An example of a security policy to be implemented can dictate that the servers should be reachable from the Internet and must not expose the insecure 'http' protocol. A second security policy may involve the servers that are not allowed to expose the 'telnet' protocol. The policy needs to be enforced when servers, networks, and ports are provisioned, and the compliance team needs to periodically audit the system to find out if any servers are violating the policy.

OPA is powering AQUA security solutions [B7] to control workload admission using Kubernetes attributes to shift-left workload security and gain a clear perspective of the workload's security posture. New Kubernetes Assurance Policies allow the application of dozens of out-of-the-box rules, or simply adding custom ones using regular expressions.

References

[B1] Istio. Retrieved from: https://istio.io/.
[B2] *Kiali Documentation*. Retrieved from: https://kiali.io/docs/.
[B3] Grafana Documentation. Retrieved from: https://grafana.com/docs.
[B4] *Curiefense Documentation*. Retrieved from: https://docs.curiefense.io/.
[B5] Open Policy Agent. Retrieved from: https://www.openpolicyagent.org/.
[B6] OWASP. (2021). *Source Code Analysis Tools*. https://owasp.org/www-community/Source_Code_Analysis_Tools.
[B7] AQUA. (2021). *Holistic Kubernetes Security for the Enterprise*. https://www.aquasec.com/products/kubernetes-security/.

Annex III: Security Framework – Initial Experiments

This section describes the preliminary experimental work regarding the proposed security framework pursuing the demonstration of concepts and respective validation of the solutions carried out in a Kubernetes cluster.

The experimental work and its objectives are presented in Table 10. The first Networking experiment seeks to demonstrate aspects of networking and the communication aspects in Kubernetes. The Observability experiment demonstrates the metrics helping to provide observance of network traffic through the use of a Service Mesh. The Security experiment aims to demonstrate security aspects in a cluster of Kubernetes.

Table 10: List of experiments

Experimental work	Objective
Networking	Demonstrate communication logic on a Kubernetes cluster
Observability	Demonstrate observability on a Kubernetes cluster
Security	Demonstrate security aspects on a Kubernetes cluster
Machine Learning security: a DoS use case	Demonstrate application of ML mechanisms applied to security

In the first experiment, Kubernetes and Istio capabilities were explored as technologies, helping in the implementation of secure distributed systems in microservices. It was possible to understand the communication mechanisms between applications, pods and containers and understand the role of exposed services of an application. Moreover, such technologies also helped to understand the configurations need for external and internal accesses and how to expose internal applications to the Internet.

From a second experiment, it was possible to explore the Kiali capabilities for monitoring a Service Mesh on the visualization of services, applications, pods, traffic, and the aggregation of their metrics. It was possible to understand how Kiali facilitates the orchestration and administration process of a Service Mesh.

Regarding the third experiment, it was possible to infer that the Service Mesh concept is key for protecting distributed systems, namely by adding authorization and authentication, separately from the applications.

In a fourth experiment, the use of ML to classify the ongoing traffic was explored. Since it is unlikely that a model achieves 100% accuracy, even considering the small size of the dataset (6623 instances), proper analysis has been conducted. It revealed that there were some issues regarding how packets have been aggregated.

Table 11 presents the applications and scripts developed along with the implementation of the architecture, supporting the running experiments:

Table 11: Applications and scripts

Application/Script	Description
Netcap	Tool to capture the traffic over a network interface
Extractor	Application to process the network packets
MQTT Client	MQTT Client to communicate to the message broker
MQTT Attacker	MQTT Client to run and perform DoS attacks to the message broker
Patch	Script to inject a container in the pod

The *Netcap* application allows capturing network packets over a network interface. The Extractor is the application that departs from the process's files in *libpcap* format to extract the network packets and produce a dataset. To that dataset, it will be applied an ML model to find out the anomalous network flows. The MQTT Client application mimics the behaviour of the UEs as the application implements the MQTT protocol client for communicating with a message broker. The MQTT client is the application running the DoS attack targeting the message broker. The Patch script is responsible for injecting in real-time a container into a pod.

Networking Experience

In this experiment the communication mechanisms between pods and containers in a Kubernetes cluster are explored. Moreover, the configuration of external and internal access to the cluster supported by the use of Service Mesh concepts supported in Istio technology was explored. Istio was installed in a Kubernetes cluster with *ingressgateway* and *egressgateway* components. Automatic sidecar injection has been set up. It was implemented a communication scenario between services in a cluster. Some examples of types of communications include the ones occurring internally and externally to the cluster, including the inbound and outbound communications.

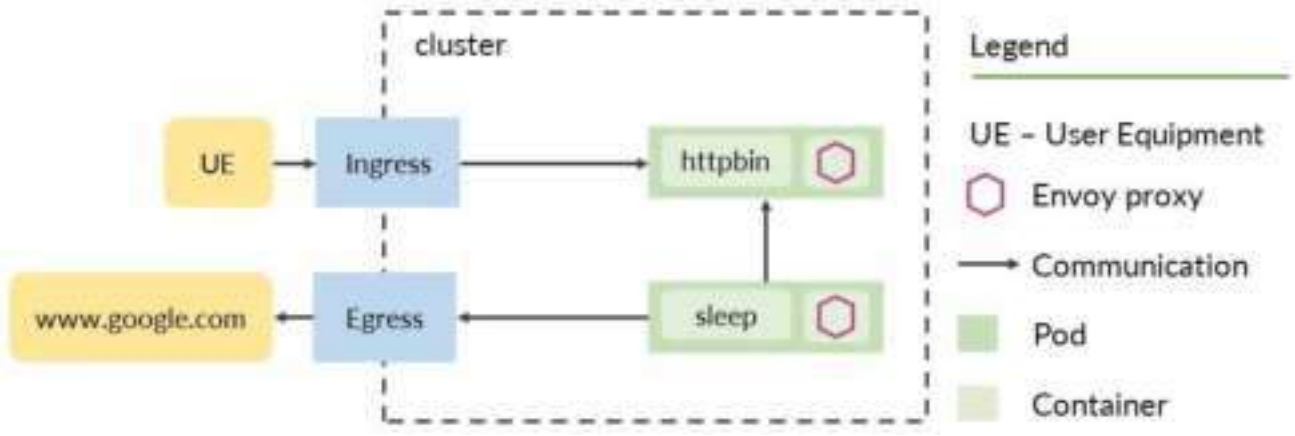


Figure 13: Communication scenario between services

Figure 13 illustrates the communication scenario between the *httpbin* and sleep applications. The *ingressgateway* component forwards external traffic to the *httpbin* service and *egressgateway* forwards internal traffic to the www.google.com website. The UE represents the equipment used to communicate with the *httpbin* service, the node terminal.

Observability experiment

The detection of attacks in real-time may avoid severe consequences over the business operations. That consists in trying to fix the issues promptly and reduce the costs (*e.g.*, the ones related to downtime). To do that, it will be necessary to identify the tracing events until the root cause is found. After that, it will be also important to review and evaluate and even plan future improvements. In the case of distributed environments, such as Kubernetes, that task will be particularly painful due to the number of components and complexity of their flows.

Observability over the monitoring data has more value if data can be centralized instead of living in silos to be queried by everybody. Monitoring such an amount of data requires a scaling and flexible approach.

Observability relies upon three pillars, such as tracing, logging, and metrics. Achieving observability in microservices in a Kubernetes cluster involves combining these three, as they represent the only way to understand complex environments. Metrics can provide the response to "what". Logs help to answer the "why" and the traces provide the information of "where". Trying to find specific logs in a pool of billions of logs of microservice

executions can take hours or even days. Moreover, the correlation of those logs across with the other metrics values turns out to be a very difficult task and, in the future, a lot of instrumentation will be required to extract the insights from the cluster.

After the extraction process, there are still a few questions to be answered:

- How do metrics and logs correlate?
- How does data between different microservices correlate?
- How to know where and when something will fail?

Tracing is the technique that helps answer those questions. A trace provides information about the transaction or workflow in a distributed system. One way to achieve this is to have a distributed solution able to collect logs and process them to trigger the alerts in a well-established time-bound threshold. Such alerts should also trigger notifications to involved parties. These alerts will later help to troubleshoot the root cause of events.

Therefore, it is of high importance to know how microservices are interconnected, from the design until the implementation phase while already in the production environment. This way, it will be possible to identify the anomalies and fix performance issues and reduce the Mean Time To Repair (MTTR). Moreover, it will be important to understand the computing latency in each one of the microservices. The adoption of a distributed solution for tracing purposes can help to have observability over the ongoing processes, improve performance and leverage the adoption of proactive approaches rather than reactive ones.

The monitoring solutions should be able to manage at scale and support for long term storage. Some technologies that meet those requirements are Prometheus and Thanos, Telegraph and InfluxDB, Victoria Metrics or even Cortex.

The following experiment addresses the underline complexities to have observability over the ongoing traffic inside the Kubernetes cluster wherein two containers participate. This experiment, depicted in Figure 14, comprises inbound and outbound communication in the cluster.

In the case of inbound communication, the *istio-ingressgateway* component acts as a gateway, allowing traffic to enter the cluster by exposing a port in a Kubernetes node. This traffic is forwarded to the *httpbin* service, which, in turn, will forward it into the *httpbin* container. This approach contributes to securing external applications to have and to have safe access to the ones within the cluster.

In the case of outbound communication, the *istio-egressgateway* component acts as a gateway by letting the traffic in the cluster flow from inside to the outside. The *egressgateway* represents a convenient strategy to have applications running within the cluster that access the external API's without jeopardizing the cluster's security. Despite not being visible in Figure 14, a sleep container communicates with the external service at Google. Such a service is defined within Istio's internal service register. For safer communication, traffic is forwarded to the *egressgateway*, which, in turn, redirects it to an external service.

The next experiment explores the observability capabilities in a Service Mesh architecture. Kiali was the used technology, providing visibility over the communications between pods and containers. It was possible to understand the offered opportunities from the adoption of a Service Mesh architecture and Kiali.

This experiment started by deploying Istio into the cluster with an *ingressgateway* and *egressgateway* components. This deployment also includes Prometheus and Kiali containers. In the next stage, the sidecars are injected into the running pods. The depicted communication scenario is like the Networking Experiment, as illustrated in Figure 13 and detailed in the corresponding section.

Once the communication has started, using the Kiali dashboard, it was possible to visualize the network traffic as a result of the Application activity over the Service Mesh inside the cluster.

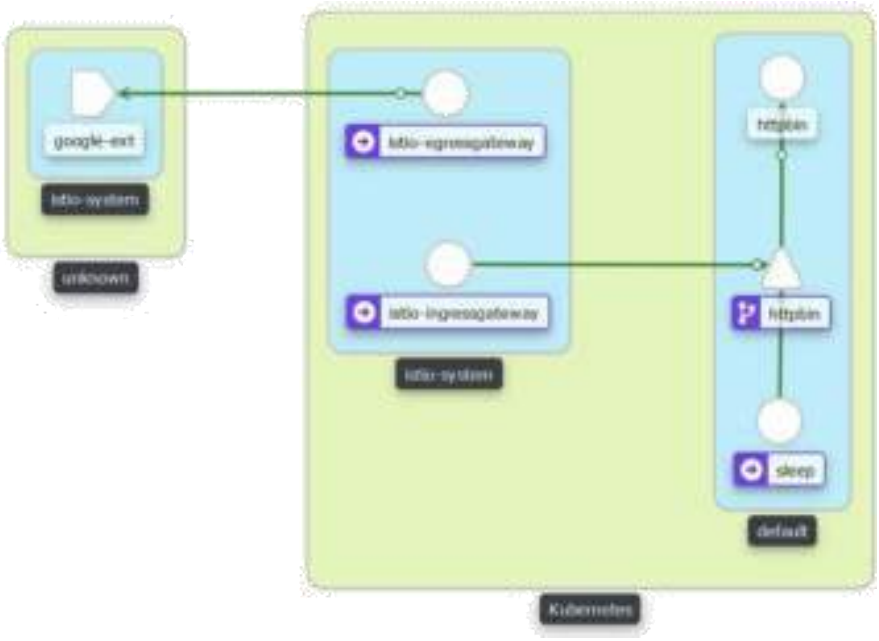


Figure 14: Kiali Network graph Mode in Istio

Figure 15 depicts the volume (left chart) and duration (right chart) of the network traffic over a period of five minutes. Both graphics depict the traffic coming from the Istio *ingressgateway* regarding the outside communications to the *httpbin* and *sleep* containers, as well as the internal communications between these two containers.

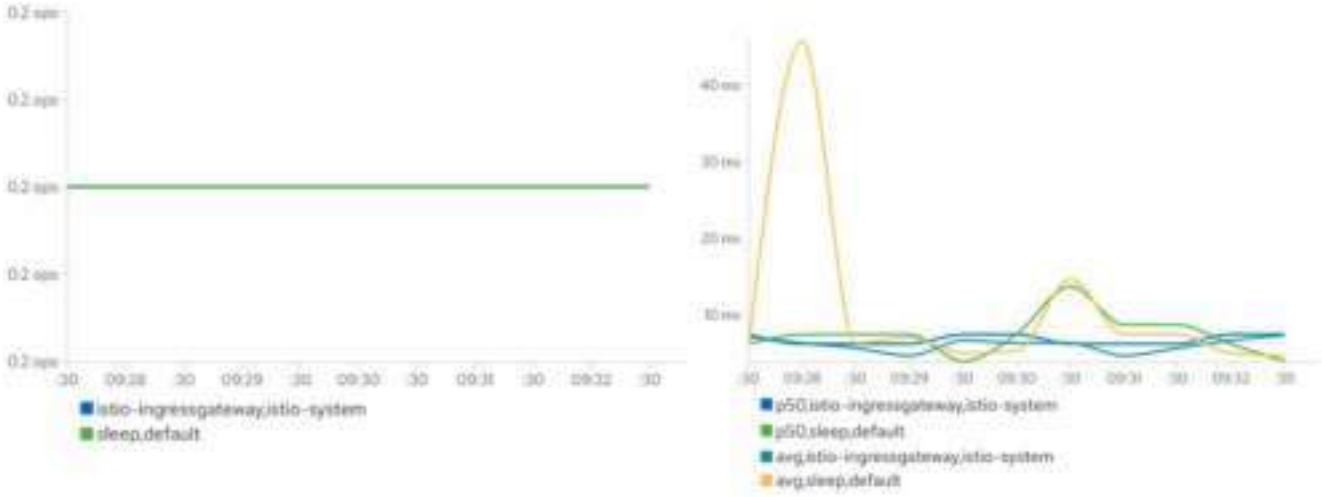


Figure 15: httpbin application metrics: order volume (left) and order duration (right)

Internal communication experiment

The experiment illustrated in Figure 16 did not require further communication configurations since Kubernetes enables the communication between *httpbin* and *sleep* without involving Istio.

In the next step, the internal communication between *httpbin* and sleep was performed. The traffic was generated by having access to the *sleep* container running GET requests to the *httpbin* service at port 8000.

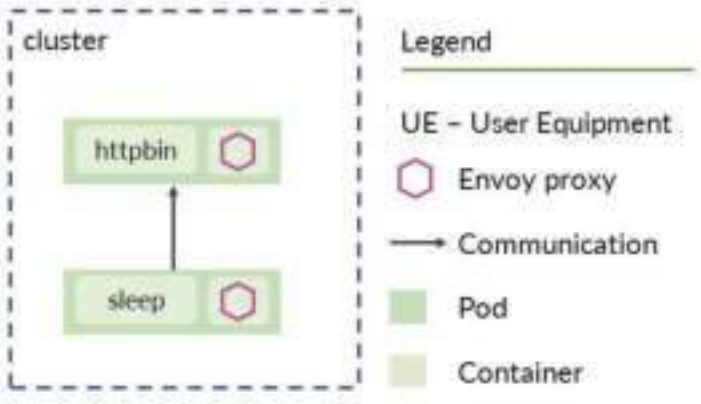


Figure 16: Internal communication diagram

Inbound communication experiment

The inbound traffic (from the outside of the cluster to the inside) requires some rules to be defined for redirecting Istio traffic. The gateway resource was used to configure the *ingressgateway*, and the Virtual Service was used to forward the communication from the *ingressgateway* to *httpbin*.

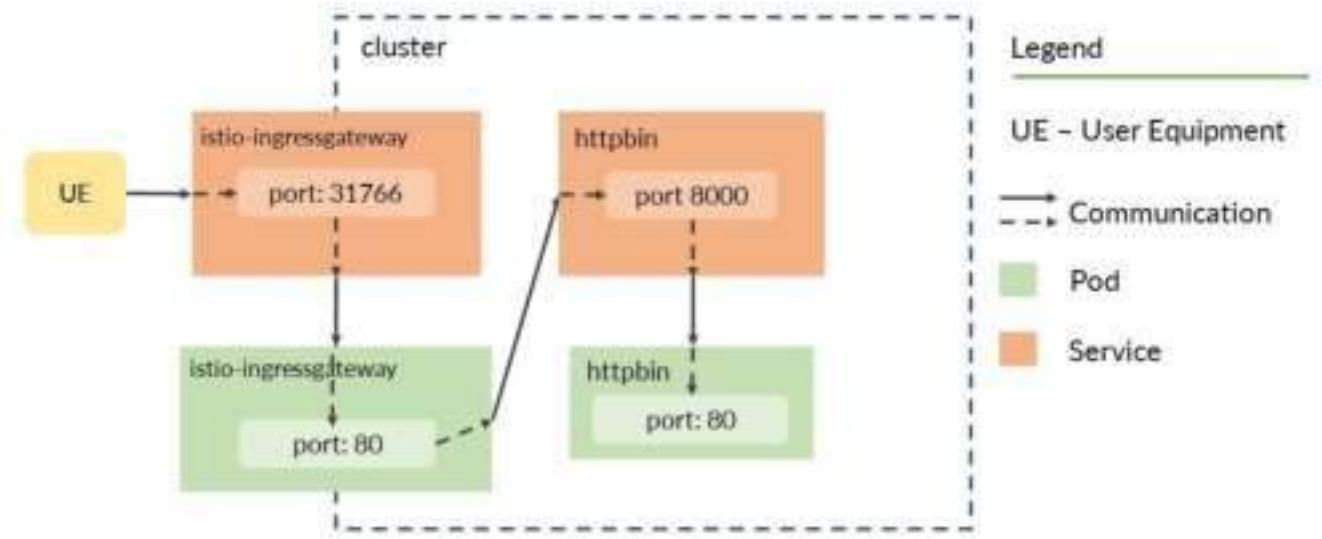


Figure 17: Cluster Inbound Communication

Figure 17 illustrates the inbound communication into the cluster. The Gateway component exposes the port 80 on *istio-ingressgateway*, and Virtual Service forwards traffic received in the port 80 on *istio-ingressgateway* to *httpbin* service on port 8000.

Figure 18 depicts the sequence diagram of the communication flows between the UE and *httpbin*. An HTTP GET request was made from the UE into the *istio-ingressgateway* service. After that, the request was forwarded to the *httpbin*. Considering that *istio-ingressgateway* is deployed as a service of type NodePort, that means that a port (31766 was the one used) has been opened in the node and the service is available, from the node terminal, on the exposed port.

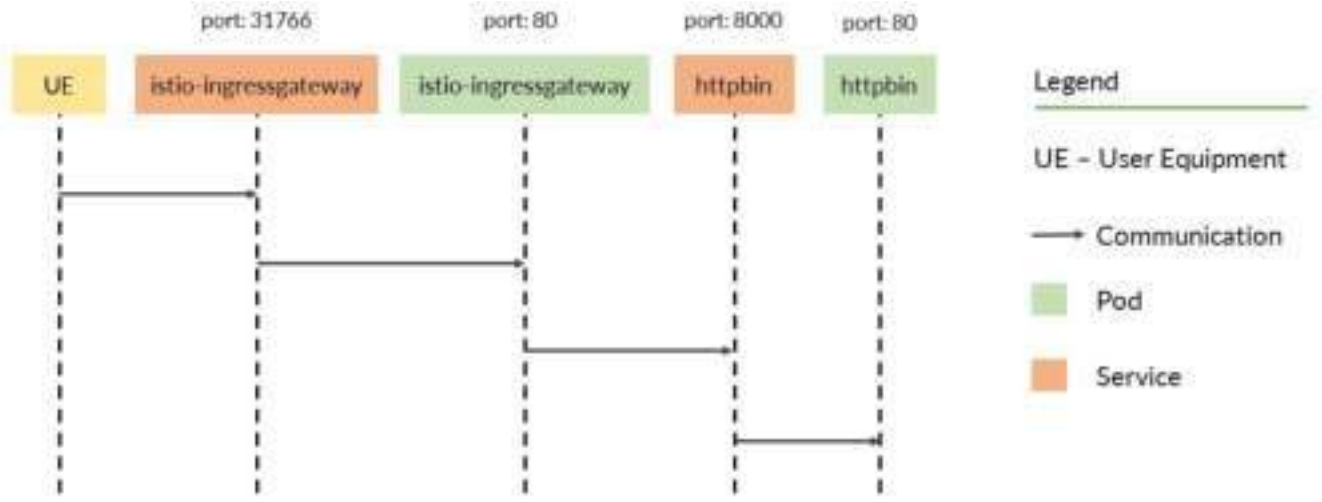


Figure 18: Inbound Communication Sequence Diagram

The next step was to proceed with the communication. It started by getting the node address and the exposed port and generating an HTTP GET request, which was successful. This request was made through the use of a *curl* command.

Outbound communication experiment

To demonstrate outbound communication of the resources such as the gateway, it was necessary to configure the *egressgateway*, a Virtual Service routing traffic and also a Service Entry to add the external service to Istio's services registry.

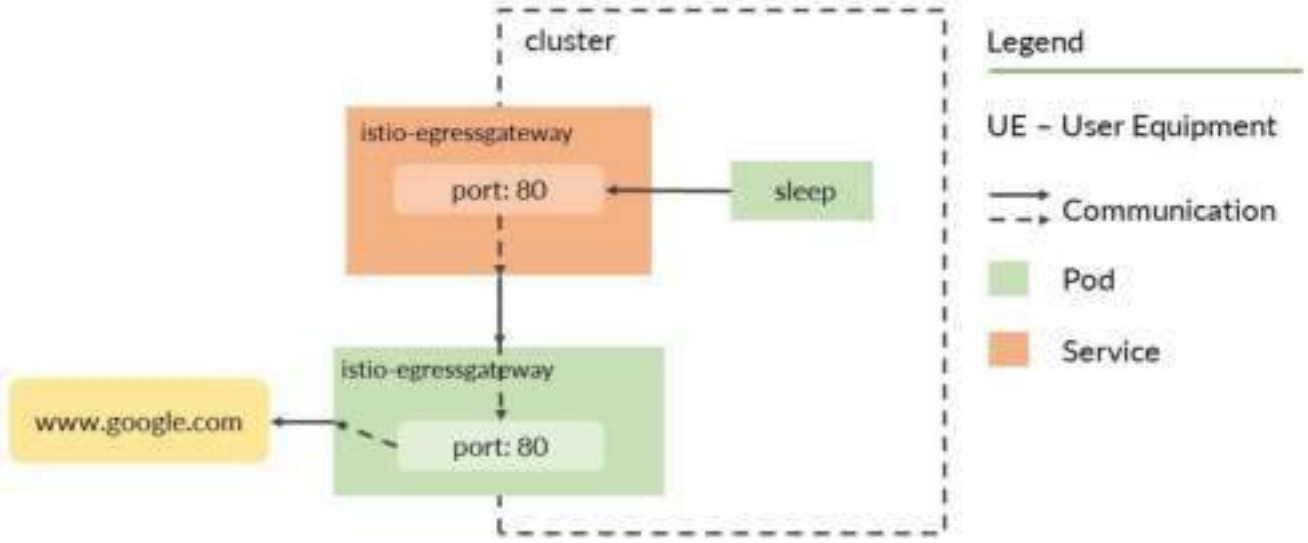


Figure 19: Indoor-outdoor communication diagram

Figure 19 illustrates the cluster outbound communication. The gateway component exposes the port 80 on the *istio-egressgateway*. The Virtual Service forwards traffic from the sleep pod to port 80 on the *istio-egressgateway*, and then from the *istio-egressgateway* pod to the external service at Google.

The sequence diagram in Figure 20 describes the communication flows. Traffic was generated from HTTP GET requests, departing from the *sleep* container towards a Google service.

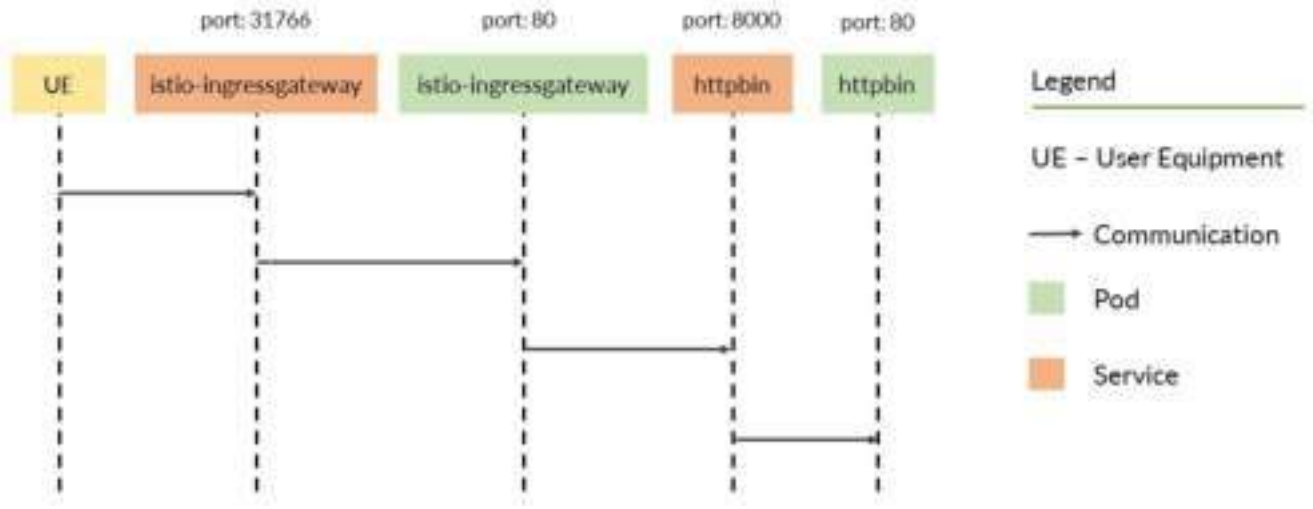


Figure 20: Outbound communication sequence diagram

Kubernetes authorization and authentication policies experiment

This experiment explored the application of security concepts in a cluster of Kubernetes using Istio. It was possible to understand how authorization and authentication mechanisms on applications contribute to a more secure system.

This experiment started by preparing a cluster environment with Istio installed besides *ingressgateway* and *egressgateway* components. Then, the sidecars were automatically injected.

In this experiment, the Istio security policies were applied to demonstrate the capabilities related to security in a cluster. The communication scenario is like the Networking experience at Kubernetes as illustrated in Figure 13 and detailed in the respective section.

The following sections report on how the security concepts regarding authorization and authentication were applied.

Authorization

The authorization policies were enforced using the AuthorizationPolicy feature. A policy was applied to block all communications within the cluster, and it was possible to validate the expected behaviour since the request made to the *httpbin* was refused with a response containing the 403 Forbidden status code. The policy was removed, and another policy was applied to allow access control to the *httpbin* application, but simultaneously denying HTTP GET requests from a specific IP.

The HTTP GET requests from the IP were classified as a threat, while the communications coming from other IP addresses were kept being allowed. In this sense, the policy was applied to block traffic originating from the *sleep* application.

Finally, it was possible to depict the requests being made from the sleep application, being blocked. The policy was later removed, and all the communications were again allowed.

Authentication

Authentication policies may be enforced using the PeerAuthentication feature. Another *sleep* and *httpbin* application were installed in a different namespace, this time without the Istio sidecar. This configuration had the purposed of demonstrating the authorization process in both secured and unsecured applications.

Then, the applications were deployed without the sidecars. The pods within the default namespace are made up of two containers, while pods within the no-sidecar namespace have only one container.

A mesh-level mTLS policy was subsequently applied. This policy forces applications to accept only encrypted communications with mTLS. To easily test the effects of the policy, a script evaluated the communication between all applications.

Then, the request being sent by the *sleep* application without the sidecar is refused, as expected, because the request did not comply with the mTLS policies, resulting in an HTTP code 403.

Finally, the policy was removed, therefore, compliance with mTLS was no longer mandatory, and communication returned to the starting point.

Machine Learning for automation of security: DoS experiment

The previous experiments had the purpose to verify the use of third-party tools to justify their use in the HSPF. The experiment described in this Section explores the use of one ML technique to detect anomalies in the network traffic in the Kubernetes cluster. It was possible to dive into the NIDS integration mechanisms and in the automation of the security of a system and its applications.

In order to run this experiment, Istio was deployed in a Kubernetes cluster with the *ingressgateway* and *egressgateway* components. Next, the sidecars were injected into the *netcap, extractor, mqtt-client and mqtt-attacker* applications.

In this experiment, the involved processes in data collection, pre-processing and classification of the NIDS pipeline for traffic classification were implemented. To this end, an attack scenario in Kubernetes was defined.

Figure 21 illustrates the experimental communication scenario, including applications such as *message-broker*, *mqtt-client* and *mqtt-attacker*. The mqtt-client application mimics the behaviour of an IoT device communicating with the message broker each twenty-second interval. The *mqtt-attacker* application launches a DoS attack on the monitoring agent every 120 seconds.

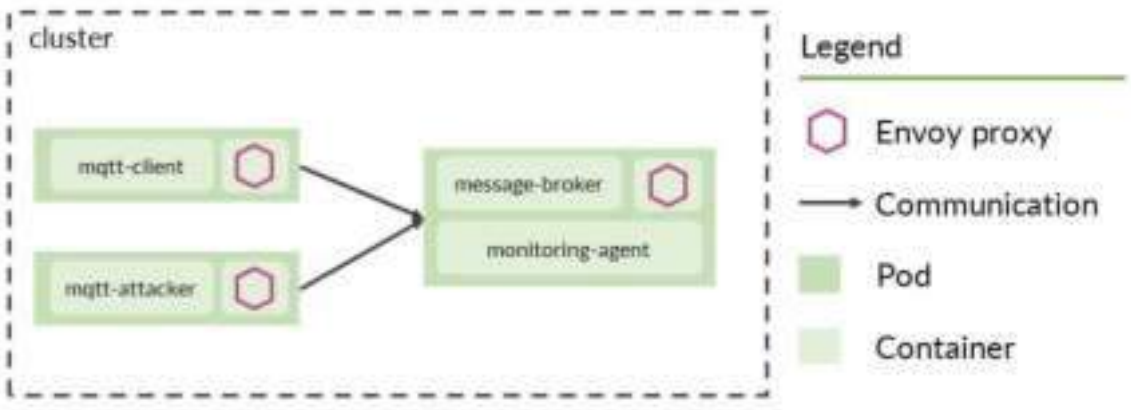


Figure 21: Communication scenario

Data Collection and Processing

The collection of data was performed in the monitoring-agent container in the message-broker pod. Next, its *netcap* program starts capturing the network packets to be saved in a file in *libpcap* format.

Once the data has been collected, the next stage for processing the data initiates. Herein, it was important to define the structure of the dataset and the selection of the relevant features to be extracted from the network packets. Considering that the chosen attack is not specific to the MQTT protocol, features related to it were

ignored. Given the nature of the attack, an analysis based on bidirectional communication flows was chosen, instead of a per-packet analysis. This decision was due to the need to have features, such as the number of packets and bytes, to allow the detection of anomalies. The rationale being aggregating packets into streams is implemented by the NFStream library.

Table 12 presents the features attained from the dataset, extracted from the captured network packets.

Table 12: List of features in a dataset

Feature	Description
duration	Flow duration
src_ip	Source IP address
src_port	Source port
dst_ip	Destination IP
dst_port	Destination port
pkts	Total packets
bytes	Total bytes
pkts_fwd_sd	Total packets in direction source→ destination
pkts_bwd_ds	Total packets in direction destination → source
bytes_fwd_sd	Total bytes in direction source→ destination
bytes_bwd_ds	Total bytes in direction destination→ source
bytes_bwd_ds	Flow duration

NFStream was the Python library package providing aggregation capabilities to process the packets. By default, the *NFStream* library allows the extraction of the required features, presented in Table 12.

The mqtt-attacker IP address (172.17.0.7) and mqtt-client (172.17.0.8) were used to classify traffic as normal (0) or anomalous (1).

Table 13 shows a part of the final dataset including the labels from the classification stage including 6623 communication flows.

Table 13: Classified Dataset

# flow	src_ip	dst_ip	…	duration	pkts	bytes	Anomaly
0	172.17.0.7	172.17.0.9	…	32389	8	4489	1
1	172.17.0.7	172.17.0.9	…	32331	8	4489	1
2	172.17.0.7	172.17.0.9	…	32373	8	4489	1

3	172.17.0.7	172.17.0.9	…	32349	8	4489	1
4	172.17.0.7	172.17.0.9	…	32381	8	4489	1
…							
6618	172.17.0.7	172.17.0.9	…	5	6	4297	1
6619	172.17.0.7	172.17.0.9	…	3	6	4297	1
6620	172.17.0.7	172.17.0.9	…	3	6	4297	1
6621	172.17.0.7	172.17.0.9	…	3	6	4297	1
6622	172.17.0.7	172.17.0.9	…	2	5	4198	1

Training

For the training phase, the following features were selected: *duration*, *pkts*, *bytes*, *pkts_fwd*, *pkts_bwd*, *bytes_fwd* and *bytes_bwd*. The features *src_ip*, *src_port*, *dst_ip* and *dst_port*, were excluded to avoid a wrong generalization of the algorithm, since all the flows originated from the IP address 172.17.0.7 were classified as anomalous. Random Forest [C1] was selected as the algorithm since it has been giving proof of good behaviour while dealing with problems of network anomaly detection.

The model was trained and evaluated with a subset from the initial dataset. Figure 22 presents the confusion matrix depicting the performance of the model classifying the traffic. It is possible to depict, the model has correctly classified all samples in the test: including 14 samples of non-anomalous communication, 14 were classified as such, and of 1642 records of malignant communication, 1642 were classified accordingly. In this way, the model achieved 100% accuracy.

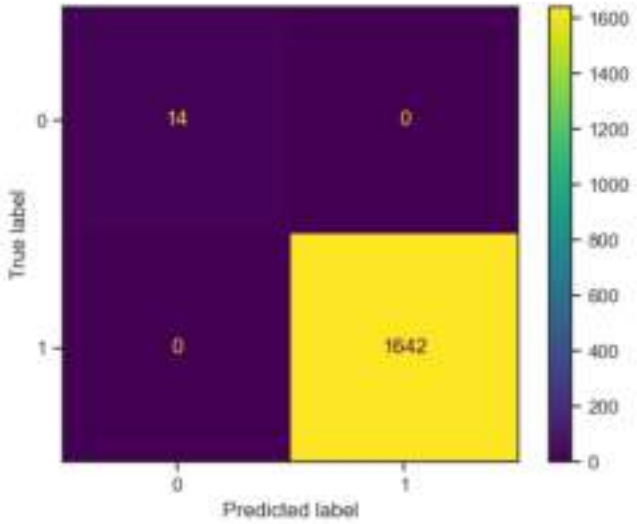


Figure 22: Confusion Matrix

Evaluation and validation

The initial classification revealed an unbalanced set of 6560 anomalous flows, which is near 100 times greater than the number of non-anomalous flows, which were only 63.

The identified issues were related to the aggregation logic. The aggregation algorithm split the received flows, considering the origin port of each packet. Knowing that each launched attack represented 1024 connections to the message broker and that these happened simultaneously, then each communication flow had a different source port. Thus, the packet aggregator produced 1024 identical communication streams for each launched attack.

Regarding the *mqtt-client*, the packet aggregator produced only a single communication flow for each execution, because only one connection was made per cycle. This can justify such a high discrepancy in the number of normal and abnormal traffic, leading to an unbalanced dataset and, consequently, justifying some over-fitting of the algorithm towards the abnormal traffic. After this analysis, it can be stated that the dataset and ML model generated is not suitable to be used due to the unbalanced data used in its training process.

Security policies enforcement

The objective of this experiment was to apply security mechanisms to the UC4 platform which includes several microservices in a Kubernetes cluster. These microservices communicate with each other and receive requests from outside of the cluster. The objective was to enforce the security policies for requests classified as anomalies that might be received by the aforementioned microservices. To meet that goal, two different options were explored.

In the first, the policies were applied with Istio, that is, it was necessary to create a program that, upon receiving the traffic, evaluates it and applies policies as needed. This program was defined in a YAML file, which was later applied using a Kubectl command.

The second solution extends the first one with the application of OPA, which was responsible for applying the defined policies.

The objective was to create two distinct scenarios, demonstrating the application of run-time policies and thus block one/several attack/s. The distinct scenarios correspond to the two solutions presented in the following sub-sections.

UC4 Scenario

The UC4 platform represents the scenario in this experiment. It includes the following set of services:

- End-user Device Simulator.
- Message Broker.
- PostgreSQL.
- InfluxDB.
- Telegraf.
- Kapacitor.
- Orchestrator.
- Monitor.
- Gateway.
- WebRTC Server.
- Portal.

This experiment included the pods for the Server and another one for EMQX, containing the Envoy proxy and a worker container. The role of this *worker* was to capture the traffic to be processed and classified by an ML model. Finally, this worker performed requests to the Kubernetes control API to stop the communication from the attacker.

In the last control stage, two different approaches were tested. In the first, requests were made to the Istio API which in turn makes requests to the Kubernetes API. In the second, requests were made to the OPA's API, and for that, TCP requests were used, containing the policies meant to be applied.

The flow of events of the processing pipeline since the traffic is captured was as follows:

- Capture of the network traffic.
- Process of the dataset with the extracted features.
- Classification of the captured traffic using the ML model.
- Enforcement of the security policies.

Security policy enforcement with Istio experiment

In a first experiment, Istio supports the application of security policies. The pods running UC4 microservices also run a sidecar proxy. The policies were applied as follows:

1. Departing from the classified flows in the collected traffic, a policy is created to block the incoming communications from the IP address of the source classified as a threat.
2. To that aim, a YAML file is created to enforce the policy to deny the communications coming from the identified malignant IP address.
3. Use of the kubernetes command “kubectl” to “apply” this YAML file with the policy.
4. Once the policy has been applied, the Envoy proxy's is now aware of them, and they are the ones who will allow/deny communication between the services.

Security policy enforcement with OPA experiment

The second experiment is also supported by Istio and the respective envoy proxies, but it included OPA as a component. OPA was used as a decision mechanism for the acceptance or denial of the incoming and outgoing communications. Thus, in this case, the decision component was independent of the component responsible for the application of the policies.

This experiment explored the application of policies departing from the previous solution, to apply a policy over the OPA API. OPA is an external authorization service, so the following steps are needed:

1. A security policy was created to deny connections from the malignant IP address. This consisted in creating a “rego” file containing the policy to deny communications from such IP address.
2. A request is made to the OPA API with the “rego” file in the message body.
3. From this moment on, each time a service receives a request, its envoy proxy “asks” the OPA for authorization.
4. The OPA answers inform the proxy about the decision to accept or block the request.

Attack to EMQX experiment

This scenario addresses an attack on the EMQX[1] service. The EMQX service is an MQTT broker, that is responsible for enabling the communications between the different UC4 components and between the User Equipment (UEs).

Both EMQX and attacker containers were deployed. After the attack has started it was possible to observe the traffic under the Grafana component. Finally, a specifically tailored policy to deny the communication from the source IP was applied. Finally, and at the end, it was possible to observe the traffic being stopped.

The objectives of this attack scenarios were the following:

[1] https://www.emqx.io/

- Simulation of an attack on the MQTT broker (EMQX).
- Application of the policies to stop the attack by exploring Istio and OPA.
- Collecting traffic metrics through Prometheus.
- Provide observability over the traffic with Grafana (metrics collected by Prometheus).

This experience was focused on the attack of the EMQX and it was conducted as follows:

Step 1. The EMQX was deployed as HEML (package manager).

Step 2. Build the needed different Docker images.

Step 3. Deploy the Attacker.

Step 4. Execute the "TCP Dump" container capturing the requests to the EMQX component into a file. Later, this file was extracted and analysed with Wireshark and afterwards exported to a ".csv" file to be processed as a training dataset by a ML model, aiming to produce an algorithm capable to identify malicious connections.

Step 5. Run the attacker with 500 connections, implemented with a Python application to produce the DoS to the EMQX pod.

Step 6. Apply security policies to block the attack.

Step 7. Observe the traffic being generated before, during and after the attack be blocked as a result of the policies enforcement (Figure 23).

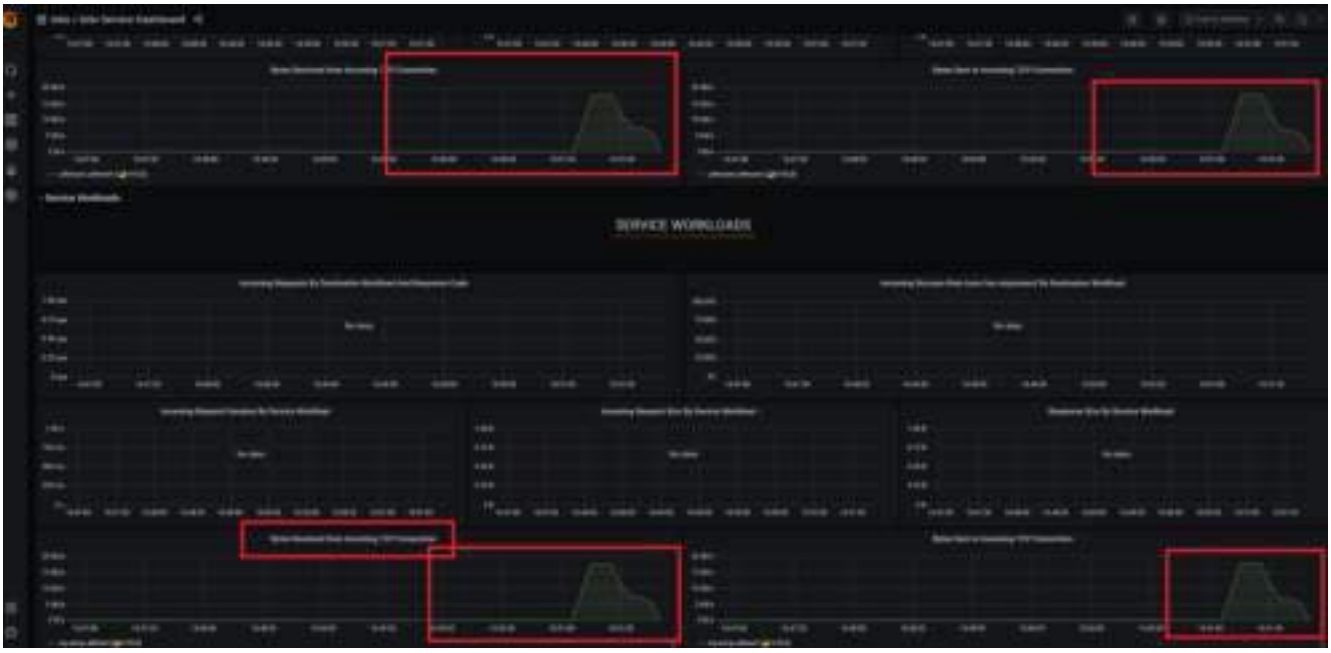


Figure 23: Traffic in Grafana

References

[C1] Donges, N. (2021). Random Forest Classifier: A Complete Guide to How It Works in Machine Learning [Blog]. *Built In*. https://builtin.com/data-science/random-forest-algorithm.

Annex IV: HSPF Evolution

This Annex aims to provide additional information what is already presented in Section 3.1.

Reference Architecture

Figure 1 depicts the initial reference architecture of the proposed Security and Privacy Framework, where three key elements reside: a security engine, a policy engine, and an AI engine.

The security engine encapsulates all the security protocols, software, and storage implementations for the different layers of architecture, including mechanisms for protecting the physical hardware and communications, and vulnerability scanning for breach mitigation and response. The security engine also considers the protection to the underlying host OS running the containers and the shared infrastructure. The following concepts will be further researched: access control via single sign-on, authentication and network traffic encryption, container isolation techniques and malicious intent detection.

The policy engine monitors the connections between sources and resources and makes the ultimate decision to grant, deny, or revoke access to the resource. This component can include a process to apply policy administration decisions. It also centralizes the configuration of the policies enforced at the network and container levels. For instance, applying policy profiles at the container level, can target multi-tenant environments, manage the resource access, or enforce process kernel restrictions, enhancing its isolation. This component is also responsible for session-specific authentication processes, including the generation of tokens or credentials to be used by a client to have access to a resource. In case the session is authorized, and the request authenticated, it allows the session to start. Otherwise, the session is denied, and the connection is blocked. In some implementations, the decision and the policy manager can be divided into two logical components. These components communicate via the control plane.

An AI engine assists in taking decisions, in order to enforce adequate security policies. For instance, it might help to identify anomalous flows based on the observability of the network traffic and support the enforcement of automated response policies and actions. Operators face a rise of unknown threats that compromise their systems. Threats like DoS attacks and Advanced Persistent Threats (APTs) exploiting different protocols have shown that perimeter walls and relying solely upon rule-based firewalls supported by a single protocol analysis are not enough anymore. Hence, rule-based firewalls should also consider artificial intelligence for enabling an immediate response to yet unknown security threats, which are undetectable by firewall vendors. By applying layered protection that includes static and heuristic analysis, anomaly detection powered by ML, and sandboxing, it is possible to provide a comprehensive defence with real-time, multi-layered threat detection.

Finally, in addition to the proposed framework, the 5G-EPICENTRE project will foster the use of SBD philosophy, which aims to incorporate security and privacy concepts from the early stages of the design and development lifecycle. Concepts and approaches such as UMLsec, SecureUML and SecDevOps will be considered during the design of the overall 5G-EPICENTRE architecture components. Further information over these and other concepts, as well as used technologies are presented into Annex I: Cloud-native security.

HSPF Implementation Architecture

The first stable version of the HSPF implementation (depicted in Figure 2) has two major components: the set of *collection agents* and the AICO, which is composed by four components: *Data Collection*, *Analytics*, *Intelligence* and the *Control & Orchestration*.

The *Data Collection* is the mechanism used to collect communications traffic from the several *collection agents*. The *Analytics* block is responsible for extracting the set of relative features and applying pre-processing techniques to the collected data. The processed data is then consumed by the *Intelligence* block, which is empowered

by AI algorithms to detect anomalous flows and identify them as potential threats to the system. Such classification is handed over to the *Control & Orchestration* mechanism, which then applies the appropriate security policies, namely, to mitigate threats upon their detection.

The deployment of this framework assumes a Kubernetes environment with Istio as a Service Mesh. The usage of Kubernetes contributes to an easier integration of this framework, as it allows for the injection of collection agents at run-time. Additionally, the use of Kubernetes has been defined by 5G-EPICENTRE as the main technology to be used in the testbeds, which greatly aligns with this assumption.

This architecture also foresees the existence of two different types of attackers: outsider and insider. The first aims to represent attacks that are originated outside of the platform (*e.g.*, dictionary attacks over some authentication service). The second aims to represent situations where an attacker would somehow have gained access to a component within a cluster (*e.g.*, a service running in a container) and started to cause the disruption of the normal behaviour of the application (*e.g.*, causing the overload of a service with malicious traffic).

Annex V: Latest HSPF implementation architecture

In this Annex, complementary information related with the latest HSPF implementation architecture will be presented.

Report Interface

This Section aims to provide additional information on the HSPF Report Interface. Thought to provide real-time information over the underlying system, this interface currently supports integration with a MQTT broker, but additional methods are envisioned to be added. The initial support to communications via MQTT was designed with the aim of allowing the integration with the project's Publisher component.

Figure 24 depicts the existent message schema (on the left) and an example (on the right).

Figure 24: HSPF Report Interface – Message Schema and Example

The schema of this message is aligned with the schemas defined by the Consortium for the different interactions with the Publisher component, which have been initially defined in D4.1 and later on refined in D4.4. The *data* field attributes are described hereafter:

- source_ip: Corresponds to the source IP of the detected attack.
- destination_ip: Corresponds to the IP of the service targeted.
- destination_port: Corresponds to the list of ports targeted by the attack.
- classifier_ip: Corresponds to the classifier IP that classified the flow(s) as malicious.

- model_ip: Corresponds to the unique model identifier that performed the traffic classification.
- flow_filename: Corresponds to the filename containing the flow characterized as malicious.
- flow_ids: Corresponds to the list of flows IDs classified as malicious.
- anomalies_detected: Corresponds to the internal threshold that sets the number of anomalies apart from which the origin component is classified as an attacker and a message of detected attack is generated.
- description: Provides a description over the detected attack.
- additional_info: Provides additional information to the description.
- first_occurrence: Corresponds to the timestamp of the first anomaly detected that led to the issue of this message.
- last_occurrence: Corresponds to the timestamp of the last anomaly detected before the source IP be characterized as an attacker and this message be triggered.
- anomaly_threshold: Corresponds to the threshold used to characterize the flow(s) as malicious or not malicious, upon the flow reconstruction by the ML model.
- reconstruction_error: Corresponds to the reconstruction error obtained by the last flow before the source IP being characterized as an attacker and this message be triggered.
- comparison_metric: Corresponds to the comparison metric used to calculate the reconstruction error.
- tf_version: Corresponds to the TensorFlow [D1] version being used in the current deployment.
- nfstream_version: Corresponds to the NFStream [D2] version being used in the current deployment.

Validation Activities

Table 14 presents the full list of results attained during the validation activities with the AI model currently in use.

Table 14: Validation Activities - Results

Number of layers	Hidden layers activation function	Optimizer	Bottleneck Layer dimension	Learning rate	Validation Loss
11	elu	Nadam	7	0.0001	0.0182
11	leakyrelu	Adam	7	0.0001	0.0184
11	elu	Adam	7	0.0001	0.0187
9	leakyrelu	Nadam	7	0.0001	0.0196
9	elu	Nadam	7	0.0001	0.0201
9	elu	Adam	7	0.0001	0.0207
11	elu	Adam	5	0.0001	0.021
11	leakyrelu	Nadam	7	0.0001	0.0217
11	leakyrelu	Adam	5	0.0001	0.0249
11	leakyrelu	Nadam	5	0.0001	0.0257
9	relu	Adam	7	0.0001	0.0274
9	elu	Nadam	5	0.0001	0.0276
11	elu	Nadam	5	0.0001	0.0277

9	leakyrelu	Adam	7	0.0001	0.02814
9	leakyrelu	Nadam	5	0.0001	0.0282
9	relu	Nadam	7	0.0001	0.0296
11	relu	Nadam	7	0.0001	0.0311
9	leakyrelu	Adam	5	0.0001	0.0319
11	elu	Adam	7	1e-05	0.0331
9	elu	Adam	5	0.0001	0.0341
9	leakyrelu	Nadam	7	0.001	0.0342
11	elu	Nadam	5	1e-05	0.0349
9	relu	Adam	7	0.001	0.0361
11	elu	Adam	7	0.001	0.0364
11	elu	Nadam	7	1e-05	0.0366
9	relu	Adam	5	0.0001	0.0375
9	relu	Nadam	5	0.0001	0.0382
7	elu	Nadam	7	0.0001	0.0389
11	relu	Adam	5	0.0001	0.0392
9	elu	Adam	5	0.001	0.0394
9	elu	Nadam	7	0.001	0.0397
7	leakyrelu	Nadam	7	0.0001	0.0403
9	elu	Adam	7	0.001	0.0404
9	elu	Nadam	5	0.001	0.0406
11	elu	Nadam	7	0.001	0.0411
11	elu	Adam	5	1e-05	0.0416
11	elu	Nadam	5	0.001	0.0419
7	elu	Adam	7	0.001	0.0420
9	leakyrelu	Adam	7	0.001	0.04219
11	elu	Adam	3	0.0001	0.0422
7	relu	Adam	7	0.0001	0.0423
7	leakyrelu	Adam	7	0.0001	0.0423
7	elu	Adam	7	0.0001	0.0426
9	elu	Nadam	7	1e-05	0.0430
9	elu	Adam	7	1e-05	0.0436
7	elu	Nadam	5	0.0001	0.0438

7	relu	Nadam	7	0.001	0.0442
7	elu	Adam	5	0.0001	0.0450
11	relu	Adam	3	0.0001	0.0453
11	relu	Nadam	5	0.0001	0.0456
11	elu	Nadam	3	0.0001	0.0457
9	elu	Nadam	5	1e-05	0.0462
11	leakyrelu	Nadam	3	0.0001	0.0465
11	leakyrelu	Adam	3	0.0001	0.0486
7	relu	Adam	5	0.0001	0.0488
11	relu	Adam	7	0.0001	0.049
11	leakyrelu	Adam	7	0.001	0.0491
11	relu	Adam	7	1e-05	0.0494
7	leakyrelu	Nadam	5	0.0001	0.0501
11	elu	Adam	5	0.001	0.0502
9	elu	Adam	3	0.0001	0.0505
9	elu	Nadam	3	0.0001	0.0511
7	relu	Nadam	7	0.0001	0.0517
9	relu	Adam	3	0.0001	0.0518
11	leakyrelu	Adam	7	1e-05	0.0523
7	leakyrelu	Adam	5	0.0001	0.0536
11	leakyrelu	Nadam	7	1e-05	0.0551
11	relu	Nadam	3	0.0001	0.0556
7	elu	Adam	5	1e-05	0.0572
7	elu	Adam	7	1e-05	0.0578
7	elu	Nadam	7	1e-05	0.0593
7	relu	Nadam	5	0.0001	0.0598
9	leakyrelu	Nadam	7	1e-05	0.0603
11	relu	Adam	5	1e-05	0.0606
11	relu	Nadam	5	1e-05	0.0615
9	relu	Adam	5	0.001	0.0619
9	leakyrelu	Nadam	3	0.0001	0.0647
11	leakyrelu	Nadam	5	1e-05	0.0655
9	leakyrelu	Nadam	5	1e-05	0.0657

11	elu	Adam	3	1e-05	0.0664
11	relu	Nadam	7	1e-05	0.0665
7	relu	Nadam	7	1e-05	0.0669
7	relu	Adam	7	0.001	0.0671
7	elu	Nadam	7	0.001	0.0673
11	leakyrelu	Adam	5	1e-05	0.0683
7	leakyrelu	Nadam	7	0.001	0.0695
7	leakyrelu	Nadam	5	1e-05	0.0697
9	elu	Adam	3	0.001	0.0698
7	leakyrelu	Adam	7	1e-05	0.0702
7	elu	Nadam	3	0.0001	0.0706
7	relu	Adam	7	1e-05	0.0721
9	relu	Nadam	7	1e-05	0.0724
7	elu	Nadam	5	0.001	0.0725
7	leakyrelu	Adam	7	0.001	0.0726
9	relu	Adam	7	1e-05	0.0728
9	leakyrelu	Adam	3	0.0001	0.0730
7	leakyrelu	Adam	3	0.0001	0.0737
11	relu	Nadam	7	0.001	0.0739
11	relu	Adam	7	0.001	0.0741
7	leakyrelu	Nadam	3	0.0001	0.0743
7	elu	Adam	5	0.001	0.0744
9	leakyrelu	Adam	7	1e-05	0.0747
7	elu	Nadam	5	1e-05	0.0755
9	leakyrelu	Adam	5	0.001	0.0759
9	leakyrelu	Nadam	5	0.001	0.0760
11	elu	Nadam	3	1e-05	0.0762
7	leakyrelu	Nadam	5	0.001	0.0778
9	relu	Nadam	7	0.001	0.0782
11	elu	Adam	3	0.001	0.0792
9	relu	Nadam	5	1e-05	0.0805
11	leakyrelu	Adam	5	0.001	0.0805
11	leakyrelu	Nadam	3	1e-05	0.0808

11	relu	Nadam	5	0.001	0.0822
7	leakyrelu	Adam	5	0.001	0.0823
11	leakyrelu	Nadam	5	0.001	0.0825
7	relu	Adam	5	0.001	0.0826
11	elu	Nadam	3	0.001	0.0830
11	leakyrelu	Nadam	7	0.001	0.0830
7	relu	Nadam	5	0.001	0.0832
11	relu	Nadam	3	1e-05	0.0845
9	elu	Adam	5	1e-05	0.0861
7	leakyrelu	Adam	3	1e-05	0.0874
9	leakyrelu	Adam	5	1e-05	0.0876
9	relu	Nadam	5	0.001	0.0877
11	leakyrelu	Adam	3	1e-05	0.0885
11	relu	Adam	5	0.001	0.0888
7	leakyrelu	Adam	5	1e-05	0.0889
9	leakyrelu	Nadam	3	1e-05	0.0889
11	relu	Adam	3	0.001	0.0895
7	relu	Nadam	5	1e-05	0.0902
9	relu	Nadam	3	0.0001	0.0910
7	relu	Adam	5	1e-05	0.0912
9	leakyrelu	Adam	3	0.001	0.0913
9	leakyrelu	Nadam	3	0.001	0.0913
9	relu	Adam	3	0.001	0.0919
11	leakyrelu	Nadam	3	0.001	0.0938
11	leakyrelu	Adam	3	0.001	0.0947
9	elu	Nadam	3	1e-05	0.0954
7	elu	Adam	3	0.0001	0.0963
9	relu	Nadam	3	0.001	0.0977
7	leakyrelu	Adam	3	0.001	0.0983
7	elu	Nadam	3	0.001	0.0991
7	leakyrelu	Nadam	3	0.001	0.0994
7	leakyrelu	Nadam	7	1e-05	0.0995
7	relu	Adam	3	0.001	0.1018

7	elu	Nadam	3	1e-05	0.1021
7	relu	Nadam	3	0.001	0.1064
9	relu	Adam	5	1e-05	0.1086
7	leakyrelu	Nadam	3	1e-05	0.1120
9	leakyrelu	Adam	3	1e-05	0.1125
7	relu	Nadam	3	1e-05	0.1129
9	relu	Nadam	3	1e-05	0.1156
9	elu	Adam	3	1e-05	0.1167
9	elu	Nadam	3	0.001	0.1180
7	elu	Adam	3	1e-05	0.1208
7	relu	Adam	3	1e-05	0.124
9	relu	Adam	3	1e-05	0.1293
7	elu	Adam	3	0.001	0.1322
7	relu	Adam	3	0.0001	0.1568
11	relu	Adam	3	1e-05	0.1677
7	relu	Nadam	3	0.0001	0.8150
11	relu	Nadam	3	0.001	0.992

First HSPF deployment into a 5G-EPICENTRE testbed

Recently, the first HSPF deployment has been successfully achieved at UMA testbed.

Figure 25 presents an example of the logging information registered by a Classifier, injected next to a micro-service. Data in the image corresponds to a series of flows, which are recorded in the same format that is later used by the Agent for inference purposes.

Figure 26 depicts an example of the logging information registered by an Agent injected next to a micro-service. Data in the image presents the results of the MSE inferred from the newest flows made available by the Collector, as well as their respective classification into *malicious* and *not malicious* data. It is also possible to notice that the Agent correctly establishes the connection to the Aggregator.

Figure 27 includes an example of the logging information registered by an Aggregator. From the data in the Figure, it is possible to notice that the Aggregator is aware of the two components that are currently being monitored, as well to see that it can receive the layers' weights from one of the clients (*e.g.*, one of the Agents).

Figure 28 shows an example of the logging information registered by an Aggregator in a later stage, where the periodic training routine has started, with the Aggregator performing a series of training batches that take into consideration the layers' weights shared by each Agent.

Figure 25: HSPF Collector - Logging Example

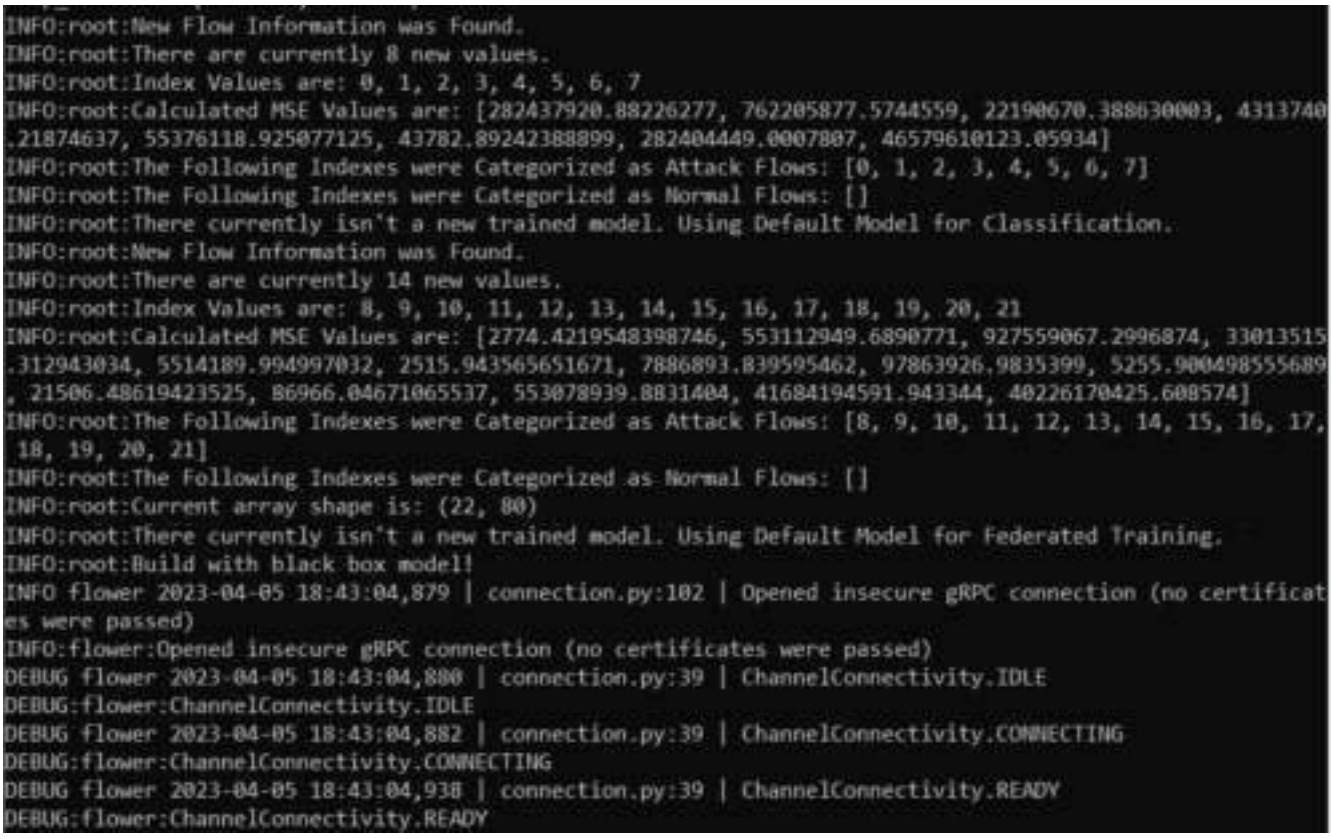


Figure 26: HSPF Agent - Logging Example

```
root@uma-gw:~/.kube# kubectl logs aggregator-768bc86558-rnqs9
INFO:root:Federated Training Window set to every 5 minute(s).
INFO:root:Training List: ['mt-orchestrator', 'message-broker']
Pod external ip is: 10.11.23.99
mt-orchestrator message-broker
['node3', 'node2']
INFO:root:Connected to MQTT Broker!
INFO:root:Received `Started FL.` from `testtopic/aux` topic
INFO flower 2023-04-05 18:41:29,478 | app.py:109 | Flower server running (20 rounds)
SSL is disabled
INFO:flower:Flower server running (20 rounds)
SSL is disabled
INFO flower 2023-04-05 18:41:29,479 | server.py:128 | Initializing global parameters
INFO:flower:Initializing global parameters
INFO flower 2023-04-05 18:41:29,479 | server.py:327 | Requesting initial parameters from one random clie
nt
INFO:flower:Requesting initial parameters from one random client
INFO flower 2023-04-05 18:43:05,076 | server.py:330 | Received initial parameters from one random client
INFO:flower:Received initial parameters from one random client
INFO flower 2023-04-05 18:43:05,077 | server.py:130 | Evaluating initial parameters
INFO:flower:Evaluating initial parameters
INFO flower 2023-04-05 18:43:05,077 | server.py:143 | FL starting
INFO:flower:FL starting
root@uma-gw:~/.kube#
```

Figure 27: HSPF Aggregator - Logging Example

```
INFO:root:Received `Ended FL.` from `testtopic/aux` topic
Pod external ip is: 10.11.23.99
mt-orchestrator message-broker
['node3', 'node2']
INFO:root:Received `Started FL.` from `testtopic/aux` topic
INFO flower 2023-04-05 19:38:44,009 | app.py:109 | Flower server running (20 rounds)
SSL is disabled
INFO:flower:Flower server running (20 rounds)
SSL is disabled
INFO flower 2023-04-05 19:38:44,010 | server.py:128 | Initializing global parameters
INFO:flower:Initializing global parameters
INFO flower 2023-04-05 19:38:44,010 | server.py:327 | Requesting initial parameters from one random clie
nt
INFO:flower:Requesting initial parameters from one random client
INFO flower 2023-04-05 19:39:54,790 | server.py:330 | Received initial parameters from one random client
INFO:flower:Received initial parameters from one random client
INFO flower 2023-04-05 19:39:54,790 | server.py:130 | Evaluating initial parameters
INFO:flower:Evaluating initial parameters
INFO flower 2023-04-05 19:39:54,791 | server.py:143 | FL starting
INFO:flower:FL starting
DEBUG flower 2023-04-05 19:39:55,659 | server.py:269 | fit_round: strategy sampled 2 clients (out of 2)
DEBUG:flower:fit_round: strategy sampled 2 clients (out of 2)
DEBUG flower 2023-04-05 19:39:56,770 | server.py:281 | fit_round received 2 results and 0 failures
DEBUG:flower:fit_round received 2 results and 0 failures
DEBUG flower 2023-04-05 19:39:56,789 | server.py:215 | evaluate_round: strategy sampled 2 clients (out o
f 2)
DEBUG:flower:evaluate_round: strategy sampled 2 clients (out of 2)
DEBUG flower 2023-04-05 19:39:56,856 | server.py:227 | evaluate_round received 2 results and 0 failures
DEBUG:flower:evaluate_round received 2 results and 0 failures
DEBUG flower 2023-04-05 19:39:56,857 | server.py:269 | fit_round: strategy sampled 2 clients (out of 2)
DEBUG:flower:fit_round: strategy sampled 2 clients (out of 2)
DEBUG flower 2023-04-05 19:39:57,066 | server.py:281 | fit_round received 2 results and 0 failures
DEBUG:flower:fit_round received 2 results and 0 failures
DEBUG flower 2023-04-05 19:39:57,085 | server.py:215 | evaluate_round: strategy sampled 2 clients (out o
f 2)
DEBUG:flower:evaluate_round: strategy sampled 2 clients (out of 2)
DEBUG flower 2023-04-05 19:39:57,158 | server.py:227 | evaluate_round received 2 results and 0 failures
DEBUG:flower:evaluate_round received 2 results and 0 failures
DEBUG flower 2023-04-05 19:39:57,158 | server.py:269 | fit_round: strategy sampled 2 clients (out of 2)
DEBUG:flower:fit_round: strategy sampled 2 clients (out of 2)
DEBUG flower 2023-04-05 19:39:57,459 | server.py:281 | fit_round received 2 results and 0 failures
DEBUG:flower:fit_round received 2 results and 0 failures
DEBUG flower 2023-04-05 19:39:57,478 | server.py:215 | evaluate_round: strategy sampled 2 clients (out o
f 2)
DEBUG:flower:evaluate_round: strategy sampled 2 clients (out of 2)
```

Figure 28: Aggregator Collector - Logging Example 2

References

[D1] TensorFlow. Retrieved from: https://www.tensorflow.org/.

[D2] NFStream: Flexible network data analysis framework. Retrieved from: https://www.NFStream.org/.