Grant Agreement No.: 101070473
Call: HORIZON-CL4-2021-DATA-01
Topic: HORIZON-CL4-2021-DATA-01-05
Type of action: HORIZON-RIA

D5.1 SEAMLESS, ZERO-TRUST SECURITY AND PRIVACY

Revision: v.1.0

Work package	WP 5
Task	All
Due date	30/11/2023
Submission date	30/11/2023
Deliverable lead	CYSEC
Version	1.0
Authors	Emna Amri (CYSEC), Eduard Marin (TID), Domenico Siracusa (FBK), Matteo Franzil (FBK), Marco Zambianco (FBK), Daniele Santoro (FBK), Eduardo Cánovas (UMU), Jose Manuel Bernabe Murcia (UMU), Elisa Albanese (RSE), Francesco Pizzato (Polito), Guillem Garí (ROB), Andy Edmonds (TerraviewOS), George Kornaros (HMU)
Reviewers	Guillem Garí (ROB), Nasir Asadov (TUB)

Abstract	This document provides an overview of the security services and features that are being implemented within the FLUIDOS stack in order to enforce a seamless, zero-trust security and privacy during all the phases of creation and utilisation of the fluid continuum.
Keywords	security; confidentiality; integrity; isolation; authorization; authentication; TEE; intrusion-detection; cyber-deception; attestation; isolation

Document Revision History

Version	Date	Description of change	List of contributor(s)
V0.1	25/07/2023	First version of the document content	Emna Amri (CYSEC)
V0.2	27/10/2023	First version of the draft, ready for review	Emna Amri (CYSEC), Eduard Marin (TID), Domenico Siracusa (FBK), Matteo Franzil (FBK), Marco Zambianco (FBK), Daniele Santoro (FBK), Eduardo Cánovas (UMU), Jose Manuel Bernabe Murcia (UMU), Elisa Albanese (RSE), Francesco Pizzato (Polito), Guillem Garí (ROB), Andy Edmonds (TerraviewOS), George Kornaros (HMU)
V0.3	20/11/2023	Draft with integrated review	Emna Amri (CYSEC)
V1.0	29/11/2023	Final version of the deliverable, ready for submission	Emna Amri (CYSEC), Eduard Marin (TID), Domenico Siracusa (FBK), Matteo Franzil (FBK), Elisa Albanese (RSE).

Disclaimer

Copyright Notice

Project co-funded by the European Commission in the Horizon Europe Programme

Nature of the deliverable:	R	
Dissemination Level		
PU	Public, fully open, e.g. web	
CL	Classified, information as referred to in Commission Decision 2015/444/EC	
CO	Confidential to FLUIDOS project and Commission Services	x

** R: Document, report (excluding the periodic and final reports)*

DEM: Demonstrator, pilot, prototype, plan designs

DEC: Websites, patents filing, press & media actions, videos, etc.

OTHER: Software, technical diagram, etc.

Funded by Horizon Europe Framework Programme of the European Union

EXECUTIVE SUMMARY

The deliverable D5.1 provides an overview of the security services and features that are proposed for the FLUIDOS stack in order to enforce a seamless, zero-trust security and privacy during all the phases of creation and utilization of the fluid continuum.

Part of the security considerations about the FLUIDOS stack is based on the requirements extracted from the considered use cases (Intelligent Power Grids, Smart Viticulture, and Robotic logistics). Each of these scenarios emphasizes the need for robust security measures tailored to their unique operational contexts.

Alongside use case requirements, a first-level threat analysis for the FLUIDOS architecture is also included, setting the foundation for understanding the potential vulnerabilities within the stack. This groundwork also maps out the security challenges of FLUIDOS across its various interactions.

These challenges are tackled, first, with the implementation of secure discovery and resources acquisition, facilitated by decentralized authentication. Subsequently, tools dedicated to the secure utilization of the FLUIDOS continuum are detailed. Topics covered encompass system integrity, confidential computing, and container isolation. Additionally, measures and strategies for threat and intrusion detection, in conjunction with cyber deception techniques, are discussed.

The final section documents scholarly dissemination activities, encompassing conference participations and a collection of papers and proceedings highlighting the security facets of FLUIDOS.

Funded by Horizon Europe Framework Programme of the European Union

TABLE OF CONTENTS

Funded by Horizon Europe Framework Programme of the European Union

Funded by Horizon Europe
Framework Programme of the European Union

LIST OF FIGURES

Funded by Horizon Europe Framework Programme of the European Union

Funded by Horizon Europe Framework Programme of the European Union

LIST OF TABLES

Funded by Horizon Europe Framework Programme of the European Union

ABBREVIATIONS

AAA	Authentication, Authorization, and Accounting
AG	Attack Graph
AP	Attack Path
API	Application Programming Interface
BC	Betweenness Centrality
CDF	Cumulative Distributed Function
CDM	Clinical Data Management
CIDR	Classless Inter-Domain Routing
CNCF	Cloud Native Computing Foundation
CNI	Container Network Interface
CNN	Convolutional Neural Network
CPU	Central Processing Unit
CRD	Custom Resource Definition
CVE	Common Vulnerabilities and Exposures
CVSS	Common Vulnerability Scoring System
DAP	Deceptive Attack Path
DB	Data Base
DC	Data Center
DDoS	Distributed Denial of Service
DID	Decentralized IDentifier
DLT	Distributed Ledger Technology
DoS	Denial of Service
DP	Differential Privacy
DST	DeSTination
ECM	Exploit Code Maturity

Funded by Horizon Europe Framework Programme of the European Union

EM	Exploitability Metrics
FL	Federated Learning
FLAD	Federated Learning Approach to DDoS attack detection
FLDDoS	Federated Learning DDoS
FLUIDOS	FLexible scalable secUre and DecentralIzed Operating System
FNR	False Negative Rate
GDPR	General Data Protection Regulation
GPU	Graphics Processing Unit
HSM	Hardware Security Module
HTTP	HyperText Transfer Protocol
ICT	Information and Communication Technology
IDS	Intrusion Detection System
IoT	Internet of Things
KVM	Kernel-based Virtual Machine
LED	Leaf Edge Device
LUKS	Linux Unified Key Setup
MBGD	Model-Based Geometric Design
MFA	Multi Factor Authentication
MiTM	Man in The Middle
ML	Machine Learning
MLP	Multi-Layer Perceptron
MQTT	Message Queuing Telemetry Transport
NIDS	Network Intrusion Detection System
NIST	National Institute of Standards and Technology
ORM	Object-Relational Mapping
OS	Operating System
OWASP	Open Web Application Security Project

Funded by Horizon Europe Framework Programme of the European Union

OVMF	Open Virtual Machine Firmware
PA	Policy Administrator
PARSEC	Platform AbstRaction for SECurity
PDC	Phasor Data Concentrator
PE	Policy Engine
PEP	Policy enforcement Point
PMU	Phasor Measurement Unit
PoC	Proof of Concept
PSM	Privacy and Security Measures
QoS	Quality of Service
RA	Remote Attestation
RAM	Random Access Memory
RBAC	Role-Based Access Control
RCE	Remote Code Execution
REAR	REsource Advertisement and Reservation
RFE	Recursive Feature Elimination
RNN	Recurrent Neural Network
ROS	Robot Operating System
SCTP	Stream Control Transmission Protocol
SEV	Secure Encrypted Virtualization
SGX	Software Guard Extensions
SNP	Secure Nested Paging
SoC	System on Chip
SRC	SouRce Code
SSH	Secure Shell
SSI	Self Sovereignty Identity
SVID	SPIFFE Verifiable Identity Documentation

Funded by Horizon Europe Framework Programme of the European Union

TCB	Trusted Computing Base
TCP	Transmission Control Protocol
TDX	Trust Domain Extensions
TEE	Trusted Execution Environment
TLS	Transport Layer Security
TPM	Trusted Platform Module
TTP	Tactics Techniques and Procedures
UC	Use Case
UDP	User Datagram Protocol
URI	Uniform Resource Identifier
VC	Verifiable Credentials
VM	Virtual Machine
W3C	World Wide Web Consortium
ZTA	Zero Trust Architecture

Funded by Horizon Europe Framework Programme of the European Union

1 INTRODUCTION

The evolution of computing has introduced transformative paradigms, with cloud computing at the forefront. This shift revolutionized the conventional model of single-application servers, introducing dynamic resource allocation, cost savings, and enhanced service agility. However, the rapid proliferation of sensors and IoT devices, coupled with the persistent reliance on centralized cloud computing, has created a pressing need for a more resilient and secure infrastructure.

The FLUIDOS project, short for "Flexible, scalable, secure, and decentralized Operating System," has been introduced as a response to this challenge. Besides offering a borderless, decentralized continuum that integrates the edge with the cloud, FLUIDOS acknowledges the crucial importance of security in this evolving landscape. As computing extends to the edges of networks, where data is generated and actions take place, the potential for vulnerabilities and threats escalates significantly. In this context, FLUIDOS features a comprehensive security initiative that aims at safeguarding the integrity and confidentiality of data and services across the continuum – from the cloud to the edge.

FLUIDOS's mission extends beyond technological innovation; it seeks to redefine the paradigm of edge computing by placing security at the core of its design.

In this document, we dive into the landscape of security challenges that FLUIDOS addresses, and the comprehensive array of measures, services, and tools implemented to ensure the highest levels of security across this distributed ecosystem.

1.1 THE FLUIDOS ARCHITECTURE

The FLUIDOS detailed architecture was described in D2.1 *"D2.1 Scenarios, Requirements and Reference Architecture – v.1"*. Here we provide a reminder of the main concepts required for a better understanding of this document.

1.1.1 The FLUIDOS Software Stack

The FLUIDOS software stack is composed of four main layers, as depicted in Figure 1.1.

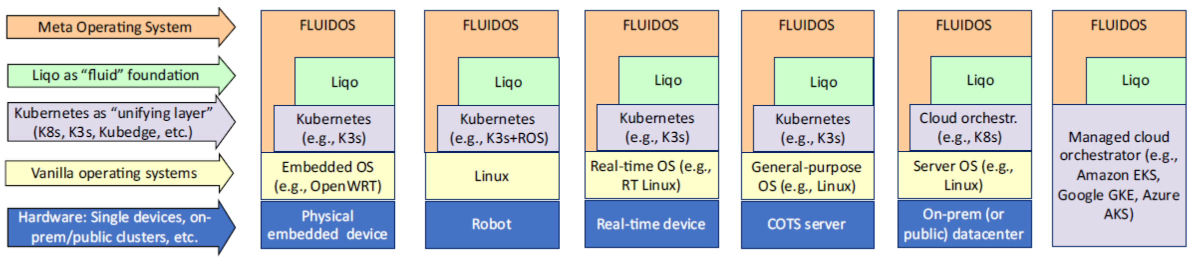

Meta Operating System
Liqo as "fluid" foundation
Kubernetes as "unifying layer" (K8s, K3s, Kubedge, etc.)
Vanilla operating systems
Hardware: Single devices, on-prem/public clusters, etc.
FLUIDOS
Liqo
Kubernetes (e.g., K3s)
Embedded OS (e.g., OpenWRT)
Physical embedded device
FLUIDOS
Liqo
Kubernetes (e.g., K3s+ROS)
Linux
Robot
FLUIDOS
Liqo
Kubernetes (e.g., K3s)
Real-time OS (e.g., RT Linux)
Real-time device
FLUIDOS
Liqo
Kubernetes (e.g., K3s)
General-purpose OS (e.g., Linux)
COTS server
FLUIDOS
Liqo
Cloud orchestr. (e.g., K8s)
Server OS (e.g., Linux)
On-prem (or public) datacenter
FLUIDOS
Liqo
Managed cloud orchestrator (e.g., Amazon EKS, Google GKE, Azure AKS)

FIGURE 1.1: THE FLUIDOS STACK

Funded by Horizon Europe Framework Programme of the European Union

- Vanilla operating systems: which abstract the underlying hardware capabilities. There are strong technical and cultural reasons for having multiple operating systems on real devices, although many of them are Linux-based: necessity to support a given set of features (e.g., real-time processing), widespread adoption within a given community (e.g., Container OS), hardware requirements (e.g., low-cost devices), user constraints (e.g., usability in consumer-oriented electronics or smartphones).
- Kubernetes [1]: which introduces a uniform layer on top of different infrastructures, regardless of whether they are end-user devices or larger cloud/edge data centers. Accounting for heterogeneous characteristics, Kubernetes distributions are available in different flavours: full-fledged Kubernetes (i.e., K8s) for small/large DCs, other distributions (e.g., Microk8s, K3s, KubeEdge) that target individual devices or small swarms of devices at the edge of the network. Kubernetes provides the minimum common denominator for FLUIDOS to build its services upon, leveraging its user-oriented primitives (e.g., replicas, deployments, stateful sets, services, etc.) that enable the deployment of user applications independently of the current distribution (e.g., K8s vs K3s), the size of the DC/node, and other characteristics (e.g., CPU architecture, i.e., Intel vs. ARM).
- Liqo [2]: which brings in a multi-cluster abstraction on top of Kubernetes, enabling seamless offloading of workloads from one cluster to another. At the same time, it handles all the additional aspects required to make this process transparent from both the users and the applications point of view, including resource negotiation, cross-cluster network fabric setup, and synchronization of the appropriate artifacts. Overall, Liqo provides the foundation to enable the "resource virtualization" layer, exposing at the same time appropriate extension hooks to further enrich it with domain specific capabilities.
- FLUIDOS: which implements the full Meta Operating System capabilities. It builds on top, and it leverages the extension hooks made available by the underlying layers to enable the most advanced and domain specific capabilities.

The overall software stack shall acknowledge and support the possible usage of additional middleware frameworks widely adopted in certain communities, such as ROS and MQTT.

1.1.2 The FLUIDOS Node Architecture

A FLUIDOS node builds on top of Kubernetes, which takes care of abstracting the underlying (physical) resources and capabilities in a uniform way, no matter whether dealing with single devices or full-fledged clusters (and the actual operating system) while providing at the same time standard interfaces for their consumption. Specifically, it properly extends Kubernetes with new control logic responsible for handling the different node to node interactions, as well as to enable the specification of advanced policies and intents (e.g., to constrain application execution), which are currently not understood by the orchestrator.

Funded by Horizon Europe Framework Programme of the European Union

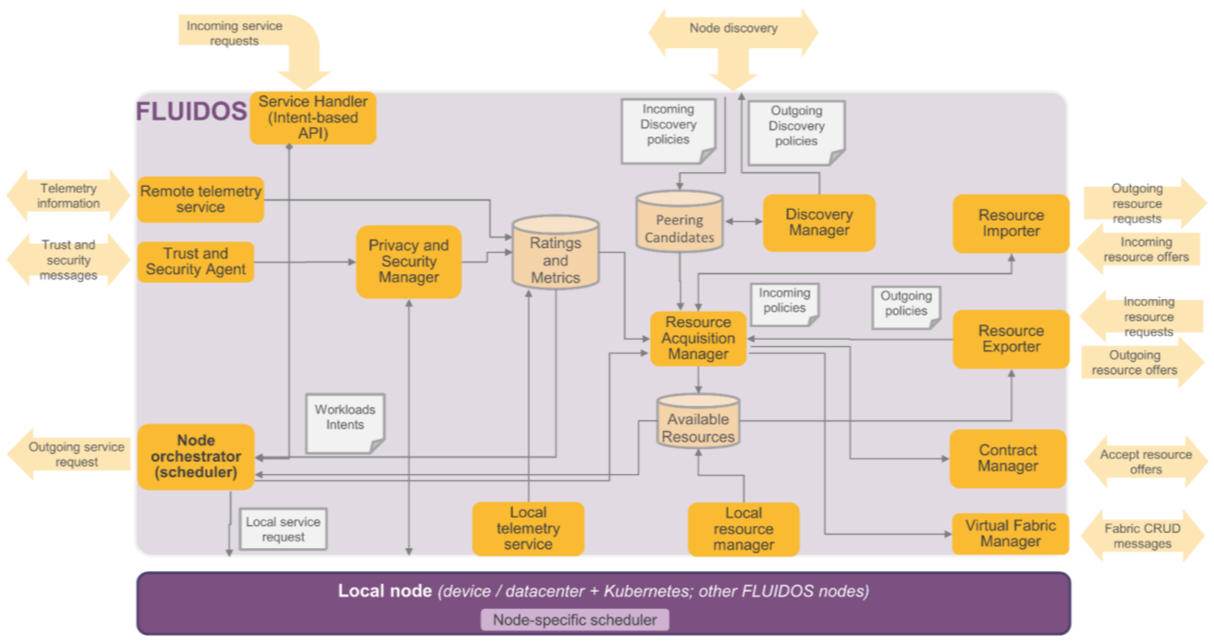


FIGURE 1.2: THE FLUIDOS MAIN ARCHITECTURE

Given this precondition, the main architectural components of a FLUIDOS node are depicted in Figure 1.2, and mainly consist of:

- Discovery manager, responsible for the discovery of other FLUIDOS nodes, producing as output a local database of feasible peering candidates (see Figure A Appendix A for a description of the Discovery Workflow).
- Node orchestrator, responsible for the orchestration of the service requests, either on the local node, or offloading them to a remote FLUIDOS node. Figure B of Annex A describes the FLUIDOS service request process.
- Resource acquisition manager, responsible for the negotiation process performed to acquire resources and services from remote FLUIDOS nodes. It can be triggered either proactively, based on policies, to ensure that a given amount of resources is always available to fulfil foreseen future requests, or by the node orchestrator, reacting to the lack of matching resources to satisfy a service request. Figure C of Annex A describes the FLUIDOS resource acquisition process.
- Virtual network fabric, responsible for establishing the computing continuum abstractions to enable the seamless execution of workloads spread across multiple nodes.
- Privacy and security manager, in charge of guaranteeing the security of the different parties involved in the resource continuum
- Telemetry service, responsible for the monitoring of the infrastructure, including the collection of all the observability parameters key to enforce and verify the satisfaction of the workload requirements expressed through the intent-based API.
- Cost manager, responsible for evaluating the burdens of carrying out a computational load on a node.

Funded by Horizon Europe Framework Programme of the European Union

In addition, for what concerns the interaction between different nodes, we designed the REAR (REsource Advertisement and Reservation) protocol, which enables different actors such as cloud providers and customers to advertise, reserve (and then consume) resources (e.g., virtual machines and their characteristics in terms of CPU, RAM; a Kubernetes cluster, etc.), and services (e.g., a database as a service).

The design of the FLUIDOS node was mainly based on the functional requirements provided by the different use-cases.

In the following we give an overview of the use cases scenarios with a focus on their security requirements that will drive the security architecture described in this document.

1.2 USE CASES SCENARIOS AND SECURITY REQUIREMENTS

In this section, we provide a high-level overview of the three use-cases associated with FLUIDOS, for the sake of context for the security requirements. For a more in-depth exploration, please refer to Deliverable D2.1.

1.2.1 UC1 Intelligent Power Grid – Energy - RSE

Power distribution grids are evolving into more complex structures, integrating distributed generation, dual load-generator entities (so-called prosumers), energy storage, and new equipment and services. Digital technologies, sensors and software are used to better match the supply and demand of electricity.

Due to the increasing dynamics and uncertainty changing behaviour of the actors in distribution grids, real-time collection of data has a critical role in guaranteeing energy delivery to end-users. To monitor, control and protect distribution grids, the use of synchro-phasors from Phasor Measurement Units (PMUs), traditionally employed in the transmission grid, is promising. However, the use of PMUs presents challenges related to resilience, scalability, availability, security and cost of the measurement infrastructure, which includes the devices and their communication systems. PMU are devices strategically placed in the power grid to measure electrical quantities and transmit this data to Phasor Data Concentrators (PDCs), with PMUs sending multiple measurements at high sampling frequencies. PDCs collect, aggregate, and synchronize this data and can be located in substations and control centers, forming a hierarchical architecture. The actors involved are the following: ICT operator, responsible for the maintenance and management of the ICT infrastructure, grid maintenance operator, responsible for the maintenance of the power grid in case of failures, grid administrator, responsible for technical and economic optimizations.

The expected scenario will be to expand the PMUs and PDCs infrastructure to the distribution grid. To support this transition, the infrastructure needs to scale and handle a large number

Funded by Horizon Europe Framework Programme of the European Union

of PMUs and PDCs, processing the data complying with QoS and latency requirements. Furthermore, in case of ICT maintenance, outage and grid reconfiguration, the infrastructure should be highly resilient, orchestrating PDCs, even on different hardware and allowing for a seamless phasor data concentration and grid state computation. Orchestration should reduce the required redundant hardware and therefore associated costs, and allow local edge processing in case of communication breakdowns.

The security requirements for this UC are based on security guidelines for Intelligent Power Grids [3] [4].

1.2.2 UC2 Smart Viticulture - TER

TerraviewOS, is a unified platform for viticulture, enables the grower to manage information from many sources, returning high-value info such as yield estimation, smart irrigation, and disease prediction and diagnosis. Since TerraviewOS is cloud-based, a key problem is the interaction with on-field devices in the presence of poor network connectivity. Operations such as drone aerial surveys (approximately 30-40 GB of data for a modest area of 10 hectares) may return their valuable results with large delays, resulting in potentially poor user experience or indeed non-operation for customers.

Through this use case, a solution for Terraview will be to create edge/device-tailored distributions of its OS to enable edge computation delivered by FLUIDOS. Due to the complexities, Terraview needs an underlying system that relieves their software from managing heterogeneous devices and cloud-native workloads. It is this system that will be enabled by FLUIDOS. The prototype hardware that the solution will be deployed to is specifically targeted for agricultural use. The specifications of the hardware can be viewed online and have only support for generalized processing workloads. We will not need the use of GPUs, however there are TEE capabilities onboard, those will be leveraged to secure workloads executing upon the edge hardware.

1.2.3 UC3 Robotic logistics - Robotnik

The robotics logistics use case involves mobile robots in industry 4.0, smart logistics, and retail scenarios, where these mobile robotic platforms can be seen as self-organized mobile battery-powered systems with resource constraints in terms of computation. The productivity of this use case depends on the ability of the mobile robots to perform complex tasks in a short time, the optimization of the duration of their battery life, their effective coordination with the fleet, and the time-consuming deployment operations.

The main reason to introduce FLUIDOS in this scenario is to manage fleet computational capacity, aiming to increase efficiency, execute heavy workloads without draining robot batteries and reduce deployment and hardware costs. Externalising and moving dynamically workloads between the different computational resources (to other robots, to the edge, or

Funded by Horizon Europe Framework Programme of the European Union

to the cloud) based on some predefined rules, will increase the productivity of the whole system and will offer a completely new approach to the market of the autonomous mobile robots in logistics.

The security aspects in this use-case are mainly related to the generated data:

- The robots produce data such as video streams that could be regulated by GDPR and outsource it to external devices. This data must be only accessible by the customer.
- With the increase of confidential robotic information traveling around the network with FLUIDOS, we need to ensure that data flow is encrypted and only readable by the workload that will consume the data.

1.2.4 Security Requirements

The table below summarizes the security requirements provided by the three use-cases described above.

TABLE 1.1: SECURITY REQUIREMENTS OF FLUIDOS USE-CASES

Req ID	Req Description	Criticality
REQ-RSE-01	**Authentication and RBAC authorization:** in the FLUIDOS environment role-based authentication and authorization should be guaranteed among FLUIDOS nodes.	high
REQ-RSE-02	**Pod Security Standards and Pod Security Admission Labels:** Pod Security Standard and Pod Security Admission Labels should be respected when shifting workloads to another FLUIDOS node in order to avoid privilege escalation.	high
REQ-RSE-03 REQ-TER-02	**Workload Isolation:** pods should be completely isolated from other tenants of the system, which share common resources and the hosting FLUIDOS node so that workloads are secure and cannot be tampered with; isolation is required also in the case of non-different administrative domains because FLUIDOS nodes can host multiple applications with different levels of criticality for the power grid management.	high
REQ-RSE-04 REQ-TER-02 REQ-ROB-02	**Encrypted communication:** communication between FLUIDOS nodes should be encrypted so that confidentiality is preserved.	high
REQ-RSE-05	**Network policies:** pods should maintain network policies when shifted to another FLUIDOS node so that the integrity and confidentiality of communications are maintained.	high

Funded by Horizon Europe Framework Programme of the European Union

REQ-RSE-06	**Security Alert:** FLUIDOS should send alerts in case of critical events in order to have a situational-aware system.	high
REQ-RSE-07 REQ-TER-05	**Monitoring:** monitoring dashboards and logs should be accessible so that real-time monitoring and post-incident analysis can be done in the FLUIDOS environment.	high
REQ-RSE-08	**Defensive measures** against cyber-attacks: defensive measures are needed against cyber-attacks to improve detection, mitigation and resilience in the FLUIDOS environment.	high
REQ-TER-01	**TEE-Enabled Processing:** Crates should be capable of executing all processes supported by the use of a TEE through FLUIDOS stack.	high
REQ-TER-04	**Integration:** An AAA provider that supports TerraviewOS must be used in order to aid integration with the existing system.	high
REQ-ROB-01	**Data Confidentiality and Privacy:** Confidential and private data should be guaranteed to be only accessible by the designated consumer.	high

1.3THREATS ANALYSIS AND SECURITY RISK ASSESSMENT

To identify the key vulnerabilities and security challenges inherent in the FLUIDOS architecture, we initiated a high-level threat analysis. This modeling process will be iterative and will be continually updated to accommodate emerging insights and evolving threat landscapes.

To maintain consistency with the key FLUIDOS resources, schemas and workflows described in the D2.1 deliverable, we have identified three key areas in which we performed the threat analysis study: **the discovery phase, the resource acquisition phase and the service request phase.**

1.3.1 Methodology

First, for each of these areas we draw a data flow diagram that summarizes the main components that are involved in that phase. Each diagram incorporates information from current status and workflow diagrams included in D2.1. We do not focus on the sequence of actions performed by each component, as that information is already conveyed by each workflow diagram in the deliverable. Instead, we identify trust boundaries between

Funded by Horizon Europe Framework Programme of the European Union

components, as a means of displaying the extent of each entity's trust zone. Thus, with a dashed line we group together components that should be able to interact with each other, safe from possible outside attackers.

Given FLUIDOS's inherently distributed architecture, the ideal approach might have been to establish strict boundaries between its components. However, such a strategy would have introduced complexity in visual representations and hindered analytical efforts. As a result, we have chosen a more conservative approach, which involves grouping the majority of FLUIDOS node components. Each trust zone is depicted by rounded rectangles, external entities by standard rectangles, data stores by cylinders, and data flow is represented through arrows. This approach strikes a balance between clarity in diagrammatic representation and practicality for analytical purposes.

In each diagram, we apply the STRIDE framework to assess the various threats that could be concerned with that particular phase. STRIDE is an acronym for Spoofing, Tampering, Repudiation, Information Disclosure, Denial of Service, and Elevation of Privilege. Each letter refers to a particular category of threats that violates a certain desired property:

TABLE 1.2: STRIDE FRAMEWORK DESCRIPTION

	Category	Desired Property
S	Spoofing	Authentication
T	Tampering	Data Integrity
R	Repudiation	Non-Repudiation
I	Information Disclosure	Confidentiality
D	Denial of Service	Availability
E	Elevation of Privilege	Authorization

For each of these categories and corresponding violated properties, we enumerate all the threats that fall down within the category and are involved in that particular phase. To maintain conciseness, we avoid outlining threats that are not strictly part of the FLUIDOS environment, in particular regarding Kubernetes threats. Some online resources such as the OWASP Kubernetes Cheat Sheet already cover them in great detail and we expect FLUIDOS administrators to apply such best practices to their clusters as much as possible.

In each table, every threat is labeled with a distinct id, of the form:

Funded by Horizon Europe Framework Programme of the European Union

[THREAT TYPE]-[PHASE NUMBER]-[THREAT NUMBER]

So, for example, the first threat of the Discovery phase that falls into the Spoofing category is labeled S11. Discovery phase threats are the first (X1Y), followed by the resource acquisition phase ones (X2Y) and finally the service request phase ones (X3Y).

Finally, a mitigation table is provided following the compilation of all threat tables. Each threat is identified by its label and is associated with an assigned mitigation strategy. For the sake of brevity, threats that can be addressed with the same mitigation approach are grouped together.

1.3.2 Threat Analysis Results

1.3.2.1 Discovery Phase

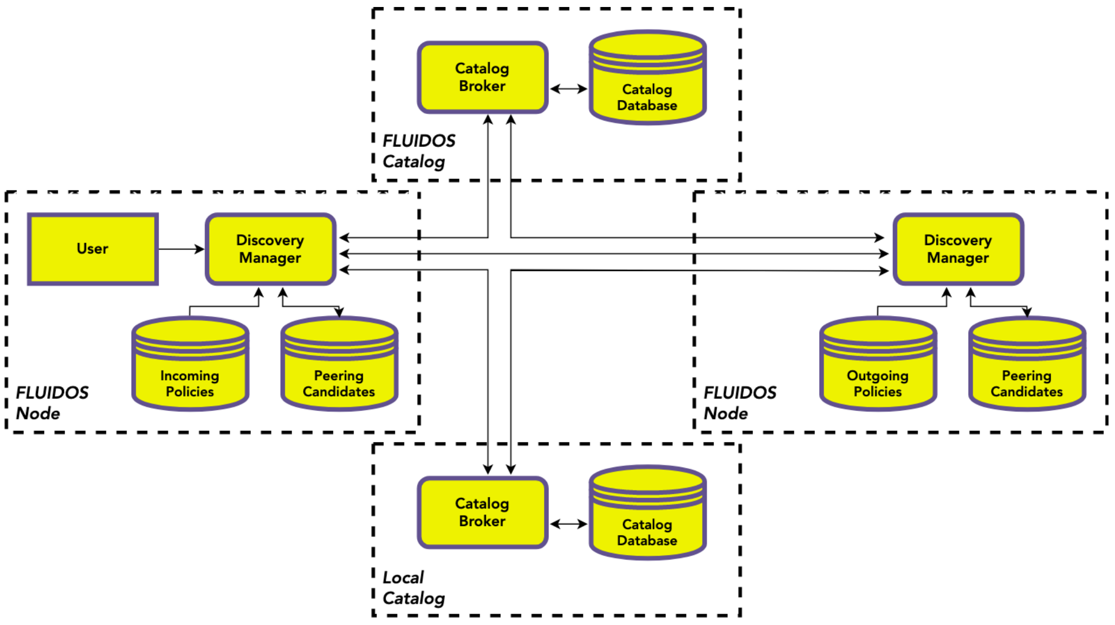


FIGURE 1.3: THE DATA FLOW OF THE DISCOVERY PHASE

The following diagram describes the data flow of the discovery phase (1).

Applying the STRIDE framework to the delineated boundaries, the subsequent threats have been identified.

TABLE 1.3: LIST OF THREATS FROM THE DISCOVERY PHASE

	Threat ID	Threat Description
S	S11	Malicious entity could pose as a FLUIDOS node

Funded by Horizon Europe Framework Programme of the European Union

	Threat ID	Threat Description
	S12	Malicious entity could pose as a local catalog
	S13	Man-in-the-middle attack between the direct discovery of two nodes
	S14	Man-in-the-middle attack between a single node and a catalog (both registration and advertisement)
	S15	Node with low trust score could change ID and pretend to be a new one, re-advertising
T	T11	Malicious peer modifying the peering candidates database due to missing and/or improper validation of data sent through API
	T12	Malicious peer modifying the list of available peers due to missing and/or improper validation of data sent through API
	T13	Malicious manipulation of incoming policies and outgoing policies database, leading to acquisition (or vice versa) of resources that do not meet the requested security policies
R	R11	A FLUIDOS node claiming he never advertised to a local or the global FLUIDOS catalog.
I	I11	Enumeration of other nodes and their characteristics in (private) catalog by malicious nodes that gained access
	I12	Excessive information provided by nodes during advertisement (e.g. accidental information about OS)
D	D11	Continuously generating new IDs and advertising to the catalog
	D12	Continuously generating new IDs and advertising to other peers
	D13	Distributed DoS attack to the catalog(s) (e.g. SYN attacks)
E	E11	Insufficient authorization in catalog API, allowing privilege escalation by nodes (e.g. acting as supernodes)
	E12	Privilege escalation in discovery with a malicious node falsely acting as a supernode

Funded by Horizon Europe Framework Programme of the European Union

1.3.2.2 Resource acquisition phase

The following diagram describes the data flow of the Resource acquisition phase (2).

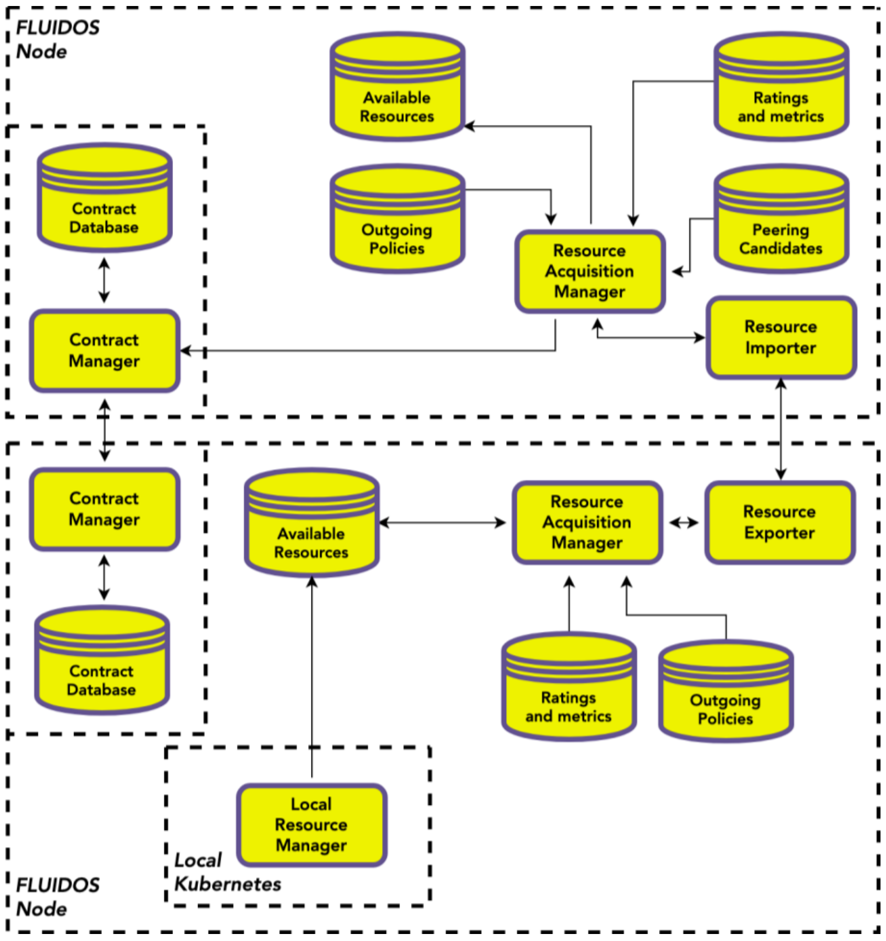


FIGURE 1.4: THE DATA FLOW OF THE RESOURCE ACQUISITION PHASE

Applying the STRIDE framework to the delineated boundaries, the subsequent threats have been identified.

TABLE 1.4: LIST OF THREATS FROM THE RESOURCE ACQUISITION PHASE

	Threat ID	Threat Description
S	S21	Man in The Middle (MiTM) between the contract manager and Resource Acquisition Manager
	S22	Unauthorized user using the contract manager
	S23	MiTM between the resource importer and exporter, pretending to be a third node

Funded by Horizon Europe Framework Programme of the European Union

	Threat ID	Threat Description
	S24	MiTM in front of the resource importer, attempting to make the node negotiate with an attacker pretending to be the same node (in other words, what happens if a node tries to negotiate with itself?)
	S25	Broken/no API authentication between resource importer/resource exporter
	S26	Broken no/authentication in the available resources database, leading to fake K8s clusters accessing it
T	T21	Improper data validation in the resource importer (e.g. a node claiming he can export different resources than he has)
	T22	Improper data validation in the resource exporter (e.g. deceiving other nodes)
	T23	Fake clusters submitting/editing data in the available resources database
	T24	Malicious manipulation of incoming policies and outgoing policies database, leading to acquisition (or vice versa) of resources that do not meet the requested security policies
	T25	Attacker tampering with the signed contracts, deleting them, or signing non-existent ones
R	R21	Insufficient validation of contract data (e.g. wrong or non-existent information, expired digital signature, etc…)
	R22	Contract data insecurely stored allowing repudiation by other node
	R23	Insufficient logging or no updating in the available resources database (e.g. nodes claiming certain resources have never been shared)
I	I21	Contract data insecurely stored and viewable by other entities
	I22	Node gathering resource import requests and relaying them elsewhere (e.g. an attacker wishing to understand the patterns of deployment by a FLUIDOS node)
D	D21	Waiting a long period of time to respond to resource import requests, hanging nodes
	D22	Waiting a long period to sign contracts, rendering the shared resources unusable

Funded by Horizon Europe Framework Programme of the European Union

	Threat ID	Threat Description
E	E21	Privilege escalation in Service Handler API
	E22	Privilege escalation in the contract manager (e.g. user with read-only access allowed to sign contracts)

1.3.2.3 Service request phase

The following diagram describes the data flow of the Service request phase (3).

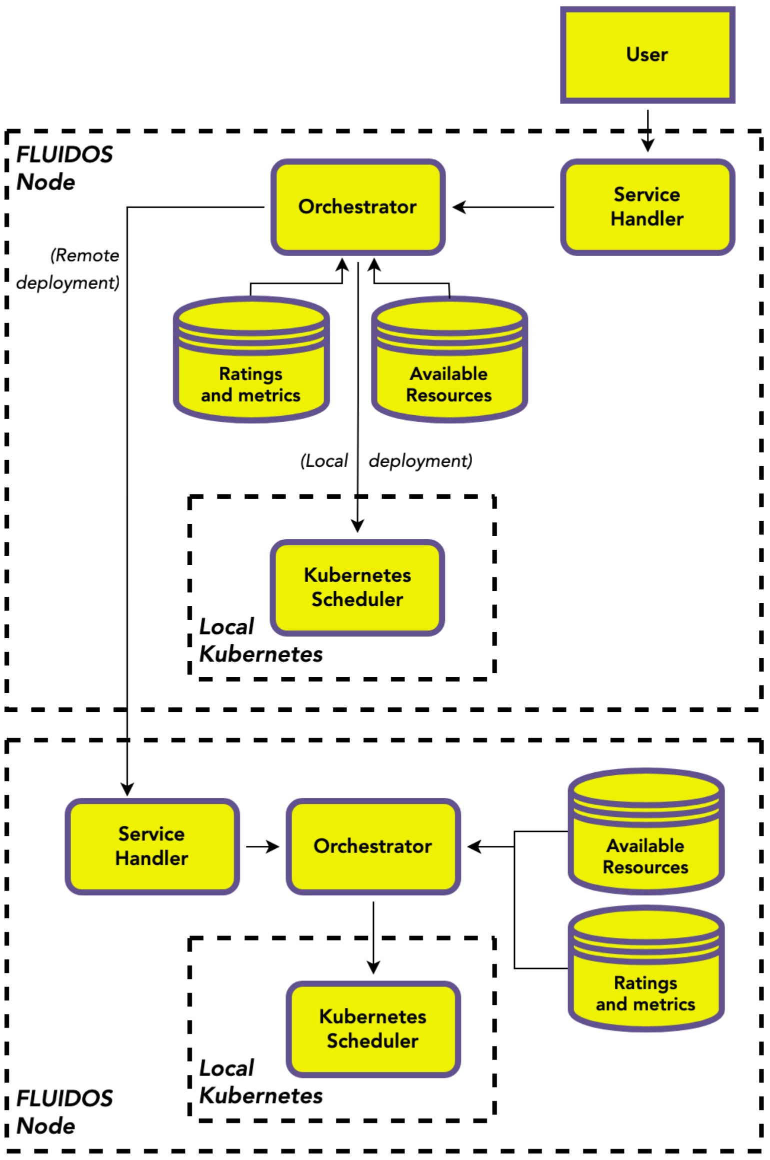


FIGURE 1.5: THE DATA FLOW OF THE SERVICE REQUEST PHASE

Funded by Horizon Europe Framework Programme of the European Union

Applying the STRIDE framework to the delineated boundaries, the subsequent threats have been identified.

TABLE 1.5: LIST OF THREATS FROM THE SERVICE REQUEST PHASE

	Threat ID	Threat Description
S	S31	Threat actor can pretend to be another FLUIDOS node and make unprompted Service Handler requests via the Intent API
	S32	Threat actor can pretend to be an already peered FLUIDOS node and pretend to handle service requests via the intent API
	S33	MiTM between the orchestrator and the local K8s cluster, interrupting job scheduling
	S34	Malicious user can deploy copycats servers in a remote node and spoof the original service
T	T31	Insufficient data validation in API communications with the Service Handler
	T32	Manipulation of ratings and metrics database (e.g. as a result of other attacks), an attacker could poison the DB forcing the node to select some higher-ranked node instead and attempt co-location
	T33	Destination node could run your workload on tampered nodes (e.g. a OS different than the required ones, without a Trusted Environment (TEE) even if requested, or not on a FLUIDOS architecture at all)
	T34	Node could ask to remotely schedule a workload that uses tampered or compromised images, or that downloads container images from an untrusted registry
	T35	Malicious user, with workloads scheduled both remotely and locally, can cross boundaries and contact services they should not
	T36	Malicious user could exploit unfiltered system calls to tamper with the host's file system, e.g., calling write() to files he should not access
R	R31	Insufficient logging in recursive service requests (e.g. node A relays to node B which relays to node C, but node B keeps no trace of this and node C cannot keep track of the original owner)
I	I31	Orchestrator leaking information when deploying workloads locally (e.g. service accounts being available from containers, that could be exploited by external users of the service)

Funded by Horizon Europe Framework Programme of the European Union

	Threat ID	Threat Description
	I32	Orchestrator leaking information when deploying workloads remotely (e.g. service accounts viewable from containers, location, OS details, exploitable by other FLUIDOS users)
	I33	Node could snoop on other tenants' workloads, for example by inspecting their system calls, or attempting to understand what is being run in a TEE environment
	I34	Malicious user could exploit unfiltered system calls to exfiltrate data from the host
D	D31	User could abuse the resources of other nodes to launch DoSs (e.g. running CPU or memory intensive workloads)
	D32	User could repeatedly ask for deployments in remote resources regardless of the success of the requests
	D33	Malicious node could collect service requests from various other nodes and recursively relay all of them to a single victim node
	D34	Ignoring recursive service requests (a malicious node could receive a request, claim it was recursively scheduled elsewhere but in reality it was never scheduled at all)
	D35	Malicious user could exploit unfiltered system calls to mount a DoS attack to the host, e.g., opening sockets or creating files
E	E31	Insufficient authorization to access Service Handler API (e.g. unauthenticated nodes accessing it)
	E32	Malicious user could exploit underlying kernel vulnerabilities triggered through system calls and achieve privilege escalation

1.3.3 Mitigation Mechanisms

Subsequent to the identification of security threats, our focus shifted towards the exploration of potential mitigation mechanisms for each of them.

As mentioned previously, the following table presents a brief recap of some possible mitigation strategies that could be pursued to address the threats that have been found in this analysis.

Funded by Horizon Europe Framework Programme of the European Union

TABLE 1.6: LIST OF MITIGATION MECHANISMS

Mit. ID	Threat ID	Mitigation
M01	S31 T33, T34 R31 D33, D34	Provide a form of remote attestation to ensure the party the node of the OS or the workload is what they claim to be and is doing what they should.
M02	S13, S14 S21, S23, S24 S33	Provide a secure cryptography channel between nodes.
M03	S15 T32	Create a globally-accessible, FLUIDOS-maintained trust database. This database could be addressable by ID, allowing any node to (anonymously) submit their trust scores and enabling other nodes to assess them. To account for further threats, authentication and possibly geographic distribution is recommended.
M04	T11, T12 T21-T23 R11 R23 T31, T32	Ensure the resource is resistant to attacks such as Injection, Cryptographic Failures and Security Misconfigurations [5]. In particular, assume any input that is being sent by nodes could be potentially malevolent and sanitize all incoming fields.
M05	T13 T24	*(only if the resource is to be handled locally by administrators)* Ensure modifications to the resource are properly timestamped and revertible, and/or assess the trust of the administrator making such modifications.
M06	T25 R11 R22, 23 I21	Ensure the security of the storage medium holding the data, e.g., using data encryption, redundancy, and access control. Provide Trusted Execution Environments (TEEs).
M07	T25 R11 R23 R31	Ensure logs are properly configured, sorted, and accessible to the parties managing it.
M08	R21	Ensure the cryptographic validity of the exchanged data, and/or do a background check on entities whose contract is being signed
M09	I11	Implement rate limiting (or equivalent measure) and monitoring when querying catalogs, blocking too-curious nodes

Funded by Horizon Europe Framework Programme of the European Union

Mit. ID	Threat ID	Mitigation
M10	I12 I31, I32	Ensure the information inserted in API requests does not link private, sensitive, or unimportant data
M11	I22	Accept the risk of threat, or, consider using privacy-preserving solutions, e.g., differential privacy, at the cost of misusing resources.
M12	I33	Provide some form of differential privacy by, e.g., injecting noise into system calls
M13	D11-D13 D32 D33	Implement DoS/DDoS detection to detect repeated offenses even when the nodes' advertised ID does not match
M14	D21, D22 D34	Implement aggressive timeouts and adopt a *pessimistic* approach, e.g., assume the request will likely fail and in parallel ask for resources to more nodes
M15	D31	Intercept requests from users and trim those that generate an excessive amount of workload or traffic.
M16	S26 E21 E31	Implement authentication strategies in the APIs (e.g. strong password requirements, MFA) for potentially destructive/administrative actions
M17	S22, T25 E22	Implement MFA authentication for the contract manager.
M18	S11, S12 E11, E12 S25	Ensure proper authentication mechanisms are put in place (e.g. Decentralized IDentifiers (DIDs) and Verifiable Credentials (VCs)) between each FLUIDOS Node and the contacting FLUIDOS Supernode, preventing the former from acting as the latter and vice versa.
M19	S34 T35	Ensure correct network isolation is employed to prevent unauthorized contacts between different tenants' services.
M20	T36 I34 D35 E32	Limit the set of system calls available to workloads, processes and users

Note that the table above presents the mitigation that can be implemented independently of what we currently have and intend to do to guarantee the security of FLUIDOS. In light of

Funded by Horizon Europe Framework Programme of the European Union

the outcomes derived from the threat analysis and the consideration of security requirements associated with our use cases, our efforts have been directed towards the design and development of the security services and mechanisms described in the forthcoming section.

1.4APPROACH OVERVIEW: SECURING THE FLUIDOS ECOSYSTEM

Based on priorities, available resources, and capabilities, we categorized our contributions for the security of FLUIDOS along **three distinct dimensions**: the phases of creation and utilization of the fluid continuum, the origin of threats and potential targets (results of the threat model), as well as the expected outcomes (i.e., a Software tool, a Method, such as an algorithm or a procedure, or a combination of both). The following figure illustrates the **eight core contributions to WP5**, categorized according to these three dimensions.

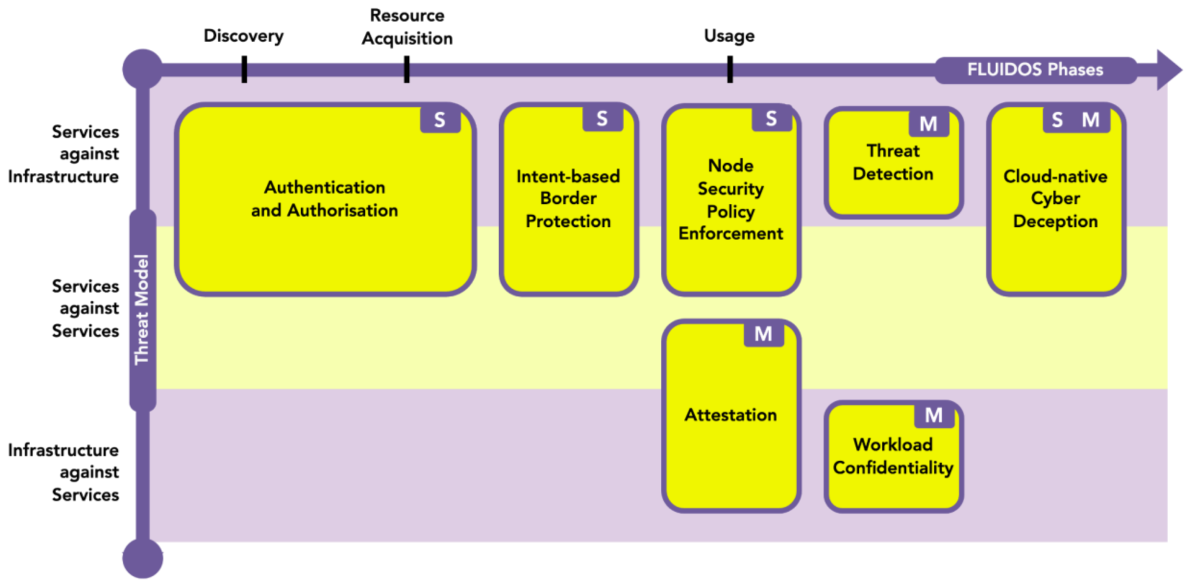

Discovery
Resource Acquisition
Usage
FLUIDOS Phases
Threat Model
Services against Infrastructure
Services against Services
Infrastructure against Services
S
Authentication and Authorisation
S
Intent-based Border Protection
S
Node Security Policy Enforcement
M
Threat Detection
S M
Cloud-native Cyber Deception
M
Attestation
M
Workload Confidentiality

FIGURE 1.6: WP5 CORE CONTRIBUTIONS, IN LINE WITH THE FLUIDOS PHASES AND THE ATTACK SCENARIOS. EACH CONTRIBUTION COULD BE A (S)OFTWARE TOOL OR A (M)ETHOD, OR BOTH.

In terms of the chronological horizontal dimension, we provide a summary of the FLUIDOS phases, encompassing the *discovery* of peering candidates, the *resource acquisition* (including negotiation, reservation, contract signing, and peering), and the *usage*. De-peering is omitted as no specific security measures were taken for that phase, except for the updating of trust scores, which is related to activities carried out during the discovery and resource acquisition phases. Details about the challenges addressed within each phase are given in the following chapters.

Concerning the origin of threats and potential targets (in the vertical dimension), we distinguish between three distinct scenarios in which a malicious actor could deliver an attack:

Funded by Horizon Europe Framework Programme of the European Union

- Directly to the infrastructure via services (e.g. a container deployed in a third-party FLUIDOS node initiating a DoS attack)
- To other services via its services (e.g., a container achieving co-location with other services and attempting to exfiltrate sensitive information)
- To services via the infrastructure (e.g., a curious infrastructure provider, owner of a FLUIDOS node, seeking to understand which business services are being operated by its customers).

The core WP5 contributions, flagged in Figure 1.6 as a novel Software tool ("S") or Method ("M", referring to algorithms, protocols, policies, etc.), are briefly described here and presented in more detail (together with other contributions) in the following chapters.

Starting from the top left of Figure 1.6, the establishment of trust is critical from the resource discovery and acquisition phases onwards, in which individual FLUIDOS nodes are aggregated into supernodes, and peering relationships are established. To address this, within T5.1, the project devised a distributed *authentication and authorisation* solution utilizing Distributed Ledger Technology (DLT) to issue credentials based on Decentralized Identifiers (DIDs). These proposed solutions, currently in the process of implementation as a set of software tools for FLUIDOS users, form the foundation of trust upon which nodes from various administrative domains can be evaluated. Please refer to chapter Secure Discovery and Resources Acquisition for more details.

While resources are being acquired and before their utilization starts, it is imperative to *segregate resources and restrict communications and interactions* to precisely control undesired exchanges between assets and resources from different administrative domains. This ongoing activity, aims to protect a provider's services and infrastructure from potentially malicious users. It resulted in the development of an initial set of security primitives that have already undergone testing and integration into Liqo. Please refer to the section Isolation of Containers for more information.

As resources across FLUIDOS nodes are peered and authorisation for use is granted, users expect verification that the system remains uncompromised and aligns with the agreed-upon specifications at the time of reservation. Furthermore, they require assurance that submitted workloads actually operate within this environment. During this first reporting period, our focus centred on *system attestation*, providing a methodology to be seamlessly integrated with the resource acquisition process. Please refer to Section System Integrity - Remote Attestation for more information.

Even when users are correctly authenticated, possess access to a limited set of resources and capabilities, and are guaranteed that workloads run in the intended environment, there remains a potential threat from actors seeking to exploit remaining interfaces, assets, and information for malicious purposes. Hence, one of our initial tasks was to develop a tool for the automatic discovery of syscalls and capabilities invoked by applications running within

Funded by Horizon Europe Framework Programme of the European Union

containers and restrict them to the bare minimum (Section Isolation of Containers). Yet, an honest-but-curious provider may still use the remaining interfaces between the containers and the kernel to gather business intelligence from workloads executed in the resources they share through FLUIDOS. Consequently, we have identified Workload confidentiality methods that could be employed for such attacks and proposed countermeasures.

At the same time, a malicious user could attempt to disrupt the proper functioning of a FLUIDOS node, or target other services within the infrastructure. To address such threats we initially considered the trade-off between detection accuracy and monitoring depth, aiming for fast yet precise Threats and Intrusion Detection. Subsequently, we considered how problematic it could be for small FLUIDOS nodes to train an accurate model with a small amount of data. To address this issue, we introduced an effective methodology that improves federated learning to collaboratively train a model for threat detection across different FLUIDOS administrative domains. The proposed method considers non-independent and identically distributed data and avoids sharing any attack data from each FLUIDOS node.

Recognising that these methods for threat and intrusion detection can be evaded (through adversarial machine learning, for instance), generate a substantial number of alarms, and are vulnerable to zero-day attacks, we put forth a proposal to augment them with a cloud-native approach to Cyber Deception This approach relies on orchestration capabilities to offer a resource-aware strategy for creating and deploying decoys. During the first reporting period, we developed an algorithm for the efficient selection of decoys and created a proof-of-concept solution to be delivered as a FLUIDOS service.

Finally, the following table summarizes, for each possible mitigation identified in Section 1.3.3 and UC requirements defined in Section 1.2.4, the solutions proposed by FLUIDOS, outlining the status of the work.

TABLE 1.7: OVERVIEW OF THE CURRENTLY-CONSIDERED MITIGATION MEASURES

Mitigation ID	UC Requirement	Status	FLUIDOS Solution
M01	REQ-RSE-03 REQ-TER-02	Addressed partially in Year 1	Attestation
M02	REQ-RSE-04 REQ-TER-03 REQ-ROB-02	*Addressable with state of the art solutions*	
M03		*To be addressed in Year 2*	
M04		*Addressable with state of the art solutions*	

Funded by Horizon Europe Framework Programme of the European Union

M05		*Not addressed (architecture-dependent)*	
M06	REQ-TER-01 REQ-ROB-01	*Addressable with state of the art solutions*	
M07	REQ-RSE-07 REQ-TER-05 REQ-ROB-01	*Addressable with state of the art solutions*	
M08-M10		*Addressable with state of the art solutions*	
M11-M12	REQ-RSE-03 REQ-TER-02	Addressed in Year 1	Workload confidentiality
M13	REQ-RSE-08	Addressed partially in Year 1	Threat detection Cloud-Native Cyber Deception
M14		*Addressable with state of the art solutions*	
M15		*Addressable with state of the art solutions, some effort will be dedicated in Year 2*	
M16		*Addressable with state of the art solutions*	
M17		*Addressable with state of the art solutions*	
M18	REQ-RSE-01 REQ-RSE-05 REQ-TER-02	Addressed partially in Year 1	Authentication and Authorisation
M19	REQ-RSE-03 REQ-RSE-05 REQ-TER-02	Addressed partially in Year 1	Intent-based Border Protection
M20	REQ-RSE-02	Addressed in Year 1	Node Security Policy Enforcement

The following chapters provide detailed descriptions of all the core WP5 contributions from requirement definition, to design, to implementation and testing.

Funded by Horizon Europe Framework Programme of the European Union

2 SECURE DISCOVERY AND RESOURCES ACQUISITION

2.1 AUTHENTICATION IN FLUIDOS

FLUIDOS is oriented towards the sharing and use of resources within what is known as the Computing Continuum, taking a step towards the transparency of this sharing, FLUIDOS reaches and evolves in the field of "liquid computing". FLUIDOS architecture allows independent actors to reach agreements in order to share their resources in a fluidified way.

Within this ecosystem, the various entities, represented as FLUIDOS nodes, must act as stewards of their own identities to maintain their autonomy. They should possess the capability to create and control their identities without relying on any centralized authority, a concept commonly referred to as Self-Sovereign Identity (SSI).

The architecture leverages Decentralized Identifiers (DIDs), Verifiable Credentials (VC), and Distributed Ledgers to create a decentralized authentication framework.

2.1.1 Decentralized Identifiers (DID)

The Decentralized Identifiers (DIDs), defined in “Decentralized Identifiers (DIDs) v1.0” [6] as W3C recommendations, represent a new type of globally unique identifier designed to enable individuals and organizations to generate their own identifiers using systems they trust. This is the main reason why they are well suited to the needs of an authentication architecture in which identity is self-managed.

By design, DIDs allow the holder to control them without the need for a third party. DIDs are URIs that associate a DID subject with a DID document allowing trustable interactions associated with that subject. Each DID document can express cryptographic material, verification methods, or services, which provides a set of mechanisms enabling a DID controller to prove control of the DID. Figure 2.1 shows the current FLUIDOS DID structure:

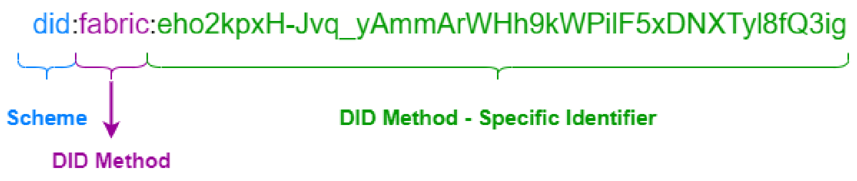

did:fabric:eho2kpxH-Jvq_yAmmArWHh9kWPiIF5xDNXTyl8fQ3ig
Scheme
DID Method
DID Method - Specific Identifier

FIGURE 2.1: FLUIDOS DID STRUCTURE

Maintaining the DID format from the specification, the generic DID scheme is a URI scheme conformant with [RFC3986].

Funded by Horizon Europe Framework Programme of the European Union

2.1.2 Verifiable Credentials (VC)

Another element necessary to authenticate the actors of an interaction are credentials. The credentials will provide information that will enrich the identity information of the person presenting it. W3C recommendation "Verifiable Credentials Data Model v1.1" [7] provides us with a model that aims to standardize this exchange of credentials.

Using verifiable credentials their holders can generate verifiable presentations to be shared with verifiers to prove they possess those verifiable credentials.

The possibility to transmit verifiable credentials and verifiable presentations rapidly, makes them convenient when trying to establish trust at a distance.

2.1.3 Distributed Ledgers (DL)

The objective of distributed ledgers, as components in FLUIDOS authentication architecture, is to provide immutable and auditable support for identity management and trusted data sharing.

DL component enriches the authentication architecture acting as secure storage of data related to identity and trust. This data is related to the identity management such as public cryptographic material and decentralized identifiers (DID) from the Self Sovereign Identity scheme of FLUIDOS.

2.1.4 FLUIDOS decentralized authentication architecture

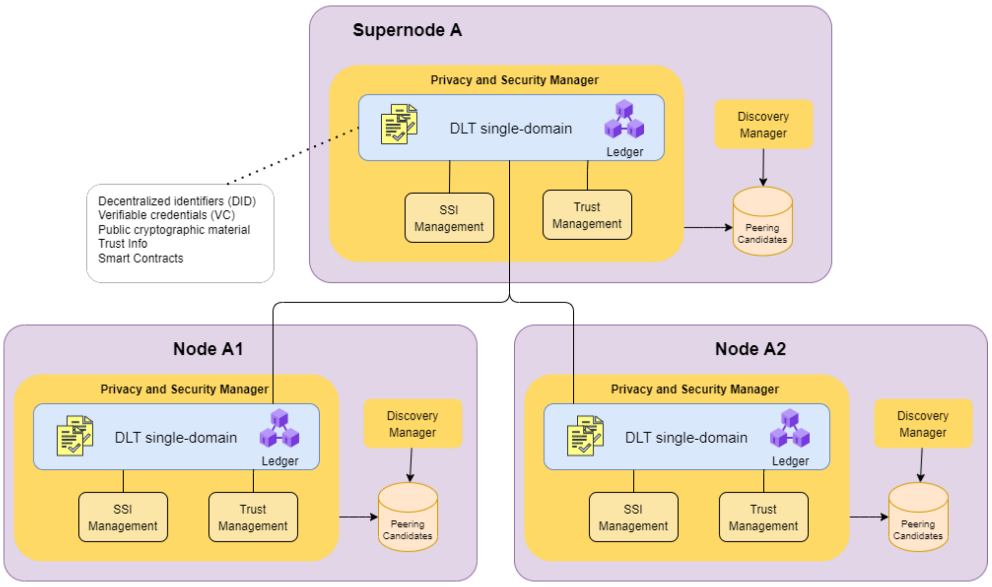


FIGURE 2.2: PRIVACY AND SECURITY MANAGER MODULE OF FLUIDOS

Funded by Horizon Europe Framework Programme of the European Union

In the proposed decentralized authentication architecture for FLUIDOS, the Privacy and Security Manager module will integrate the Self-sovereign identity (SSI) Management component.

This component is responsible for:

- Control the identity of the FLUIDOS Node generating its own identifier, acquiring its VCs and storing them in their own wallet and presenting these credentials or verifiable presentations to allow the identification of the node.
- Contribute to the identity of the other FLUIDOS Nodes inside the borders of its domain. Acting as issuer, it will provide VCs to other nodes inside its domain.
- Verify the VCs or verifiable presentations presented from other nodes. Each node inside the domain will be able to verify credentials received from other nodes, from the same domain or not.
- Publish in the DL public cryptographic material in a way that is accessible and auditable at any time. It enables the use of privacy-preserving credentials, decentralized IDs.

2.1.5 Usage scenario: Identity Bootstrapping

The first envisioned scenario is the generation of the identity of the FLUIDOS node. In this scenario there are two possibilities:

- **When the bootstrapping node is alone in the domain:** When the node starts in an empty domain it is expected that this node will be the supernode of that domain.
    a. As the starting node does not have a DID to use for its identification it should generate it. SSI Management component is asked to generate a new one.
    b. Using the newly generated DID the next step is to ask SSI Management to generate a verifiable credential. As this node is the supernode because it is the first node in the domain, it should generate a verifiable credential for itself.
    c. The newly generated verifiable credential should be stored in the supernode wallet and its signature can be verified by anyone with access to the DLT by obtaining the public key found in the DID (publicly available) of the issuer (supernode) that has issued that verifiable credential, in this case the issuer is himself.

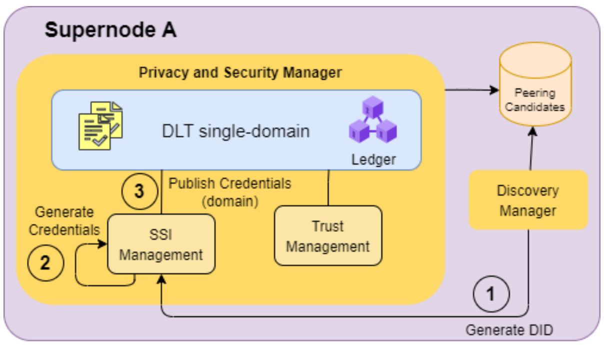


FIGURE 2.3: SOLE NODE BOOTSTRAPPING PROCEDURE

- **When there exists a supernode in the domain.** When a node starts in a domain with a supernode in charge, the flow is a bit different.
    a. As the starting node does not have a DID to use for its identification it should generate it. SSI Management component is asked to generate a new one.
    b. Using the newly generated DID the next step is to ask SSI Management to generate credentials. As a supernode exists in the domain, the starting node asks the supernode to generate a verifiable credential for him.   .
    c. The newly generated verifiable credential should be stored in the node wallet and its signature can be verified by anyone with access to the DLT by obtaining the public key found in the DID (publicly available) of the issuer (supernode) that has issued that verifiable credential.

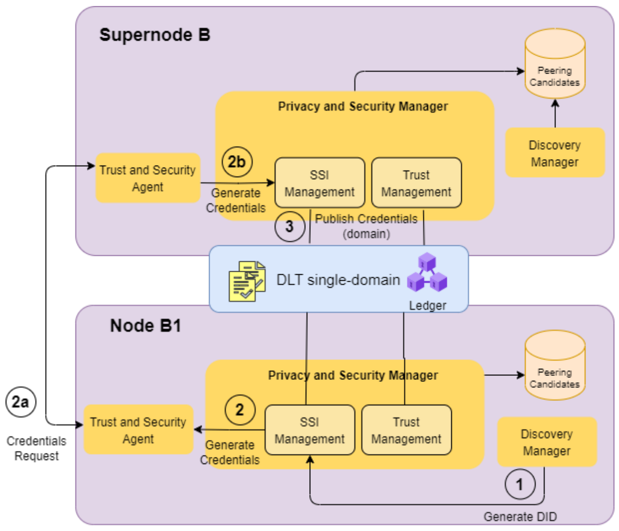


FIGURE 2.4: BOOTSTRAPPING PROCEDURE IN A MULTI-NODE DOMAIN

Funded by Horizon Europe Framework Programme of the European Union

2.2 AUTHORIZATION AND TRUST SCORES

Authorization in FLUIDOS, as happens with authentication, must maintain the characteristic of decentralization; to do so, in the Computing Continuum the trend is to design Zero Trust architectures (ZTA). Zero Trust paradigm starts from the premise that trust is never granted implicitly but must be continually evaluated. Resources must be restricted to those with a need to access and grant only the minimum privileges needed to perform the mission.

The basic Zero Trust tenets adapted to FLUIDOS are:

- All the components put in place to provide shared services in FLUIDOS are resources.
- All communication is secured regardless of network location, especially because that communication could be from a third party.
- Access to individual resources is granted on a per-session basis. Trust in the requester is evaluated before the access is given and it should be granted with least privilege basis. Authentication and authorization should be evaluated per resource, access to one resource will not automatically grant access to a different resource.
- The policy to access resources is dynamic and may include other behavioural and environmental attributes.
- FLUIDOS nodes will monitor and measure the integrity and security posture of all owned and associated resources. Trust is not inherent to the resources. FLUIDOS to implement ZTA should continuously monitor the state of resources and apply corrective measures as needed.
- All resource authentication and authorization are dynamic and strictly enforced before access is allowed. Accessing a resource is a constant cycle of obtaining access, assessing threats, reevaluating trust in ongoing communication.
- FLUIDOS nodes will collect as much information as possible about the current state of resources and use it to improve its security.

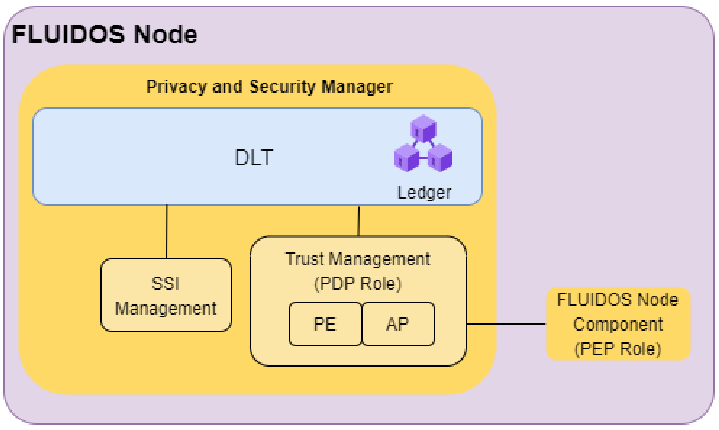

FLUIDOS Node
Privacy and Security Manager
DLT
Ledger
SSI Management
Trust Management (PDP Role)
PE
AP
FLUIDOS Node Component (PEP Role)

FIGURE 2.5: PRIVACY AND SECURITY MANAGER OF FLUIDOS NODE

Funded by Horizon Europe Framework Programme of the European Union

The ZTA reference design independent of deployment models consists of three types of core components: Policy Engines (PEs), Policy Administrators (PAs), and Policy enforcement points (PEP) PEPs.

- Policy engine (PE): PE is responsible for the decision to grant access to a resource for a given subject. The PE uses FLUIDOS policy as well as input from external sources (e.g., CDM systems, threat intelligence services) as input to a trust algorithm in charge of granting, denying, or revoking access to the resource. The PE is paired with the PA. PE makes and logs the decision (as approved, or denied), and the policy administrator executes the decision.
- Policy administrator (PA): PA is responsible for establishing and/or shutting down the communication path between a subject and a resource (via commands to relevant PEPs). It would generate any session-specific authentication and authentication token or credential used by a client to access a FLUIDOS resource. If the session is authorized and the request authenticated, the PA configures the PEP to allow the session to start, else it will send signals to the PEP to shut down the connection.
- Policy enforcement point (PEP): PEP is responsible for enabling, monitoring, and eventually terminating connections between a subject and a FLUIDOS resource. The PEP communicates with the PA to forward requests and/or receive policy updates from the PA. FLUIDOS components with security needs should include PEP functionality.

Besides these core elements, the FLUIDOS trust management depends on the SSI Management to make the functions of the ID management system that is mentioned on the ZTA. This is responsible for creating, storing, and managing FLUIDOS Nodes identity and credentials records in the Distributed Ledger. This system contains the necessary subject information (e.g., name, email address, certificates) and other node characteristics.

2.3 FLUIDOS TRUSTED EDGE

Besides trust between FLUIDOS nodes and super-nodes, it is important to establish security between the FLUIDOS Edge infrastructure and the edge/IoT devices. To do that, it is fundamental to support trusted computing devices, operating systems, edge microservice communications and networking. Trust between FLUIDOS Edge components and edge devices can be achieved by adopting protocols such as TLS which can offer mutual authentication and confidentiality (data encryption) between two communicating parties. Another important aspect related to security on the edge is to provide secure storage for sensitive data such as keys, certificates, and credentials.

In this direction, we integrate the SPIFFE/SPIRE framework to provide authentication and encryption between the edge infrastructure and IoT devices. We also explore the STM32Trust TEE Secure Manager to enable secure storage on high-performance STM32 microcontroller devices (secure elements). Finally, we introduce the PARSEC micro-service

Funded by Horizon Europe Framework Programme of the European Union

which is an abstraction layer that eliminates the complexity required by the edge infrastructure to communicate with the various secure elements (IoT devices) available today.

2.3.1 SPIFFE/SPIRE

SPIFFE [8] is a Cloud Native Computing Foundation (CNCF) framework composed of a set of open-source standards for securely identifying software systems in dynamic and heterogeneous environments. Systems that adopt SPIFFE can easily and reliably mutually authenticate wherever they are running. SPIFFE is the selected framework solution for providing secure identity in the form of a specially crafted X509 certificate. By using and further evolving SPIRE, a tool that implements the standards set by SPIFFE, we can issue certificates both for the FLUIDOS Edge entities and the leaf edge devices so that each party can authenticate itself against the other party. Another important role of SPIRE is that it provides node and workload (process, pod etc.) attestation.

Typically, a SPIRE deployment consists of a SPIRE server installed in one node and one or more SPIRE agents installed in each node of a Kubernetes environment (Figure 2.6).

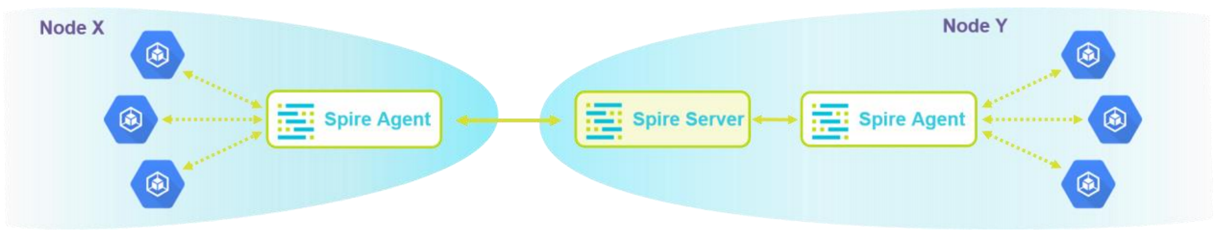


FIGURE 2.6: SPIRE ARCHITECTURE

A SPIRE server manages all identities in its configured SPIFFE trust domain. Apart from that, it performs node attestation to authenticate all the agent identities in its domain. Finally, it creates SPIFFE Verifiable Identity Documents (SVID) for workloads when requested by an authenticated agent.

A SPIRE agent requests SVIDs from the SPIRE server and caches them until a workload requests its SVID. Moreover, it performs workload attestation to verify the authenticity of the workload identities.

Funded by Horizon Europe Framework Programme of the European Union

2.3.2 STM32Trust TEE Secure Manager

The STM32Trust TEE Secure Manager is a suite of system-on-chip (SoC) security solutions that simplify the development of embedded applications with security features and services.

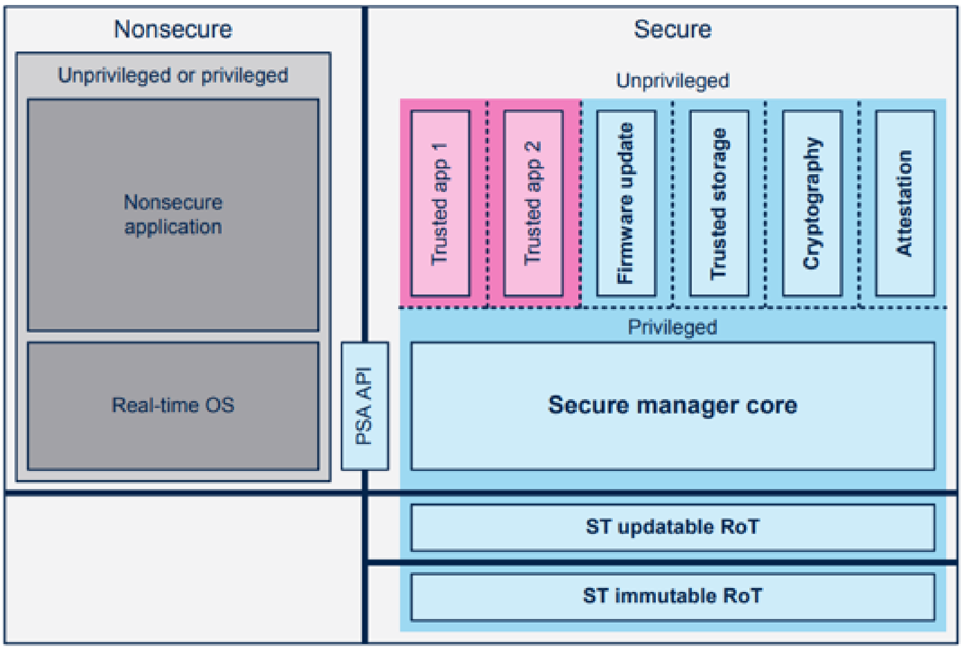


FIGURE 2.7: SECURE MANAGER EMBEDDED SOFTWARE EXPLOITING ARM® TRUSTZONE® FEATURES

It offers support for secure boot, root of trust, cryptographic functions, secure storage, attestation, and secure firmware updates. The ARM TrustZone [9], available in STM32 microcontrollers, can be configured to create secure memory regions and privileged access to peripherals (Figure 2.7). Among the numerous security solutions, we emphasize secure storage on the STM32 microcontrollers to provide secure elements for sensitive data of the FLUIDOS Edge environment.

2.3.3 The PARSEC Abstraction Layer

PARSEC (Platform AbstRaction for SECurity) [10], is an open-source initiative to provide a common API to hardware security and cryptographic services in a platform-agnostic way. This abstraction layer keeps workloads decoupled from physical platform details, enabling cloud-native delivery flows within the data center and at the edge. Computing platforms have evolved to offer a range of facilities for secure storage and secure operations such as Hardware Security Modules (HSMs), Trusted Platform Modules (TPMs) or firmware services running in Trusted Execution Environments (TEEs).

Funded by Horizon Europe Framework Programme of the European Union

Parsec is an abstraction layer that allows cloud-native workloads, programming languages and containers to access hardware secure storage/service facilities in an effortless and consistent way through its API without worrying about the underlying complexity and diversity of the hardware secure elements. In addition to the common API, the PARSEC service offers multitenancy, that is, the underlying security facilities can be shared amongst multiple client applications.

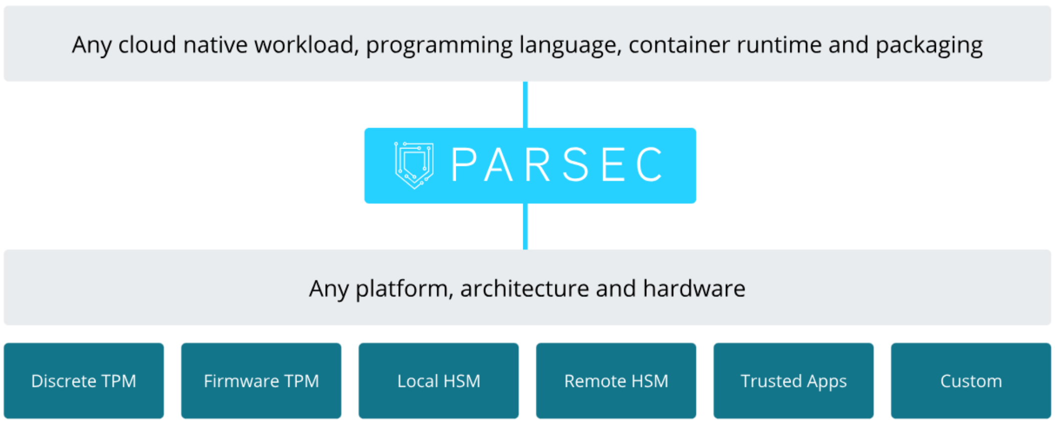

Any cloud native workload, programming language, container runtime and packaging
PARSEC
Any platform, architecture and hardware
Discrete TPM
Firmware TPM
Local HSM
Remote HSM
Trusted Apps
Custom

FIGURE 2.8: THE PARSEC SERVICE FACILITATES ANY WORKLOAD TO ACCESS ANY UNDERLYING SECURE ELEMENT [10]

2.3.4 Mutual Authentication

The role of the FLUIDOS Edge infrastructure is to make IoT resources (microcontrollers, sensors, etc.) and corresponding data available to the FLUIDOS node. The origin of these resources has to be trusted inside the FLUIDOS domain, so, an important step in this direction is to secure both the devices and the data path starting from the IoT leaf edge devices (LEDs) until they reach a cloud application.

In our implementation depicted in Figure 2.9, an InfluxDB application is deployed in the Cloud Node to provide a means of storage for data received by LEDs. A Mapper is deployed in the Edge Node to make an LED available to the FLUIDOS Edge environment. It uses the Bluetooth protocol to pair with the LED and start receiving data from the latter which are then published to the MQTT broker. The Router is a service deployed in the Cloud Node that receives LED data which in turn are forwarded to the InfluxDB application according to rules that were set earlier.

Based on the data flow described above there are three paths that have to be secured since they constitute communication with external systems/components:

- Mapper ↔ LED
- EdgeCore ↔ CloudCore

Funded by Horizon Europe Framework Programme of the European Union

- Router → Cloud application (e.g., InfluxDB)

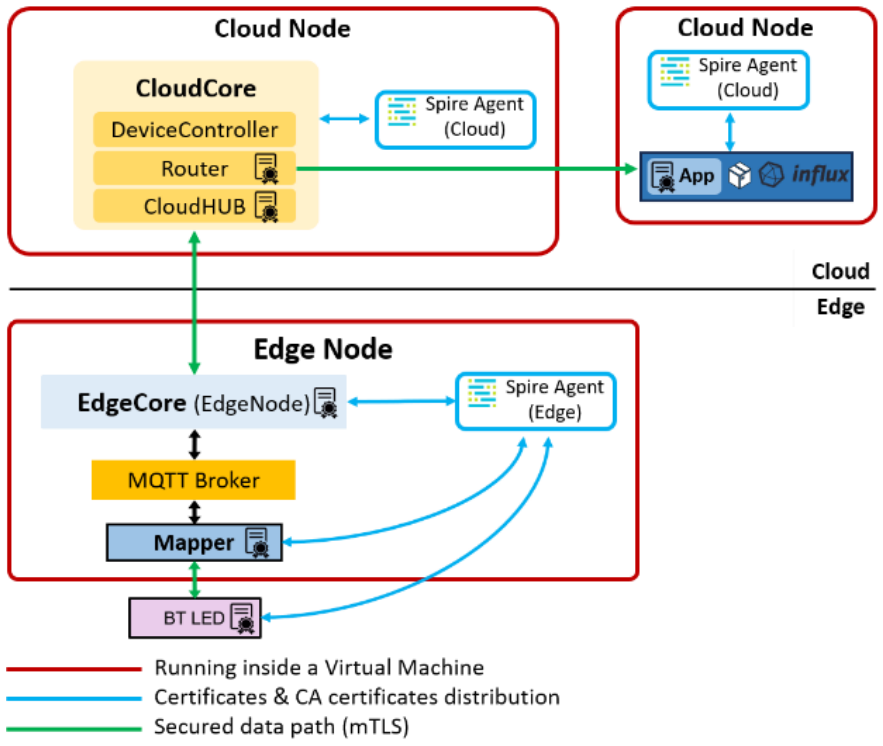


FIGURE 2.9: ARCHITECTURE AND DATA FLOW IN THE FLUIDOS EDGE ENVIRONMENT

At this point, we introduce SPIRE which is responsible for providing certificates to the FLUIDOS Edge entities and securing the three paths above. For this case, we utilize three SPIRE agents to perform the workload attestation, issue and deliver the certificates (SPIFFE identities) for the entities involved, that is, CloudCore modules, EdgeCore modules and Mapper, LED, and Cloud application. Once all certificates are delivered, all entities involved can mutually authenticate each other and securely communicate over TLS.

2.3.5 Edge Devices Secure Storage

Certificates issued by the SPIRE deployment or other sensitive data like login credentials and keys need to be securely stored. Our solution combines the STM32Trust Tee Secure Manager suite along with high-performance STM32 microcontrollers equipped with ARM® TrustZone® to offer secure storage solutions to the FLUIDOS Edge environment.

A SPIRE agent is responsible for delivering a certificate (SPIFFE identity) to an application that can take advantage of the underlying secure storage to write its certificate by integrating the PARSEC client library which communicates with the PARSEC micro-service through an API. The application passes its identity to PARSEC in the form of a token. In turn, PARSEC

Funded by Horizon Europe Framework Programme of the European Union

validates the authenticity of the document by querying the SPIRE agent. The verified identity is then stored in the STM32 secure element (Figure 2.10).

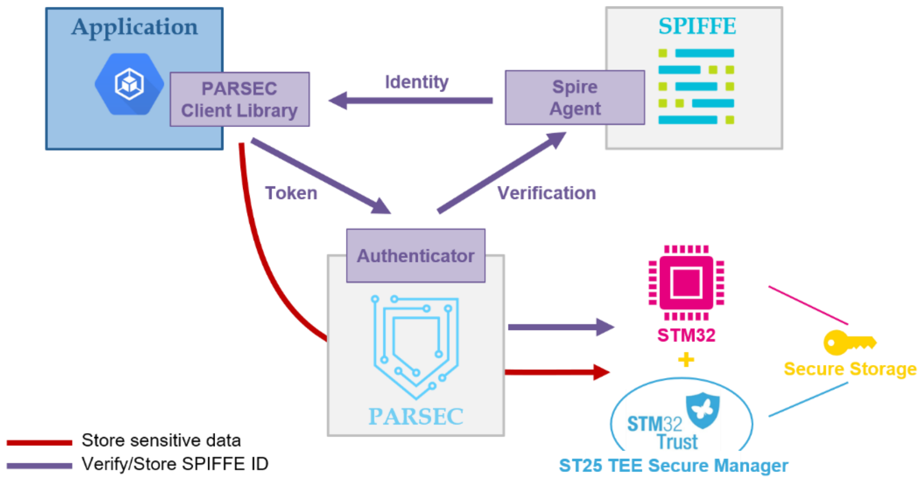


FIGURE 2.10: STM32 TRUST TEE SECURE MANAGER AND PARSEC FOR SECURE STORAGE

Funded by Horizon Europe Framework Programme of the European Union

3 SECURE USAGE OF FLUIDOS CONTINUUM

3.1 SYSTEM INTEGRITY - REMOTE ATTESTATION

One of the main security goals of FLUIDOS project is to ensure the secure execution of workloads, requests and response messages across the FLUIDOS ecosystem, leveraging the capabilities of Trusted Execution Environments (TEEs) and considering different possible implementations. The work in this area mainly focused on how to investigate the authenticity and integrity of an environment by means of Remote Attestation (RA) mechanisms.

There are various existing implementations of Remote Attestation with widely different use cases, supported platforms, security models, and integrations; as such, it can be interesting to consider quite broadly the foundational ideas behind RA. The objectives of using RA in the context of Confidential Computing can be summarized as follows: "*Ensuring that a remote system is trustworthy through its endorsement by a mutually trusted intermediary*"

This sentence can be dissected as such:

- A Remote System represents any system we must interact with, upon which we do not have absolute control, such as a cloud virtual machine or a smartphone.
- Trustworthy means that only trusted code and data are running within the system. Also, any relevant associated system metadata, such as debugging or security options, are as expected.
- Endorsement means that some entity can produce proofs of some kind as to the system's state. It is also implied that the entity has the technical ability necessary to ensure the proofs' validity.
- Mutually Trusted Intermediary A third party explicitly trusted by both the remote system and the entity requesting the remote attestation. Often the manufacturer of the Remote System or a critical component within (CPU, SoC, or TEE).

3.1.1 State of the Art

3.1.1.1 High-Level Definition of Remote Attestation

Conceptually, remote attestation consists of an untrusted entity providing claims to a third party that can be substantiated through the intervention of one or more commonly trusted entities.

While multiple ways exist to define the specifics of a remote attestation mechanism, this work uses the trust model from the "Remote Attestation Procedures Architecture" IETF draft [11] when describing the various existing remote attestation mechanisms. Namely, the following (non-exhaustive list of) relevant roles, which the same entities may hold, are present in their trust model:

- Attester: An entity that produces relevant evidence to be used by a Verifier. This evidence is most often related to the system managed by the Attester. This evidence may be cryptographically protected. In the case of AMD SEV-SNP, the Attester is the AMD Platform Security Processor, producing evidence (attestation reports) regarding a VM it manages and protects.
- Relying Party: An entity that depends upon the validity of information about the system managed by the attester. It relies on a Verifier to ensure that the information provided by the Attester about the system is valid and can be actioned upon. In the case of AMD SEV-SNP, the Relying party would be the entity that relies upon the VM being appropriately started and being in a trustworthy state (e.g., A bank that wishes to ensure their data is only provisioned to a trusted VM).
- Verifier: An entity that trusts the manufacturer of the Attester, it can rely on its endorsement of the Attester's evidence to be assured of its validity. It then produces actionable attestation results for the Relying Party. In the case of AMD SEV-SNP, the Verifier would be the application that requests the relevant certificate chains from AMD's key distribution servers and uses it to check the signature of the attestation report. Finally, it provides relevant information to the Relying Party from the contents of the attestation report.
- Endorser: An entity that endorses the validity of the evidence produced by the Attester and is trusted by the Verifier. In the case of AMD SEV-SNP, this endorser would be AMD which provides the signing keys to the Attester and ensures they are unusable by third parties. It then provides the associated public keys to the Verifier

While their model has other less significant roles, those four are the most important ones when considering and analysing various Remote Attestation implementations.

3.1.1.2 Proposed Simplified Taxonomy

Various technological trade-offs exist with the use and implementation of remote attestation mechanisms; as such, it is interesting to produce a relevant taxonomy. While detailed and motivated taxonomy exists for remote attestation within embedded systems [12], its usefulness is significantly restricted when also considering both smartphone and server-based remote attestation mechanisms.

As such, the following simplified taxonomy targeted at FLUIDOS relevant needs is proposed:

- Hardware-Based/Software-Based mechanism: This distinction is made on whether the remote attestation mechanism relies upon hardware-specific instructions dedicated to remote attestation or if the mechanism relies upon software running on a platform-specific or architecture-specific trusted execution environment (potentially

Funded by Horizon Europe Framework Programme of the European Union

with hardware-backed key management). This distinction informs the future flexibility of mechanisms in already shipped hardware. Indeed, software-based mechanisms could be much more evolutive through potential firmware updates, whereas hardware instruction sets could not see much evolution for shipped hardware.

- Software/System Attestation: This distinction is made on whether the remote attestation mechanisms attests to the state of a given software within a system or if it attests to the state of an entire system.
  It informs the use cases of such attestation mechanisms; software attestation is more axed towards ensuring that a given software was not tampered with, while allowing it to run within various untrusted systems. System attestation would be preferred to ensure the state of an entire OS.

- Online/Offline Verification: This distinction is made on whether or not the verification of attestation results requires communications between the Verifier and a third party, such as the Endorser.
  It informs on how reliant the Verifier will be upon the Endorser at runtime. Long term support for the mechanism by the Endorser may be impacted, along with increasing the complexity of using the mechanism within a fully isolated network.

- Flexible/Rigid Chain of Trust: This distinction is made on whether the chain of trust used by the Verifier to tie the attestation results to the Endorser is rigid or may be updated later for any purpose without introducing new hardware.
  It informs about how likely the mechanism may recover from a partial chain of trust breach, with flexible chains somewhat more likely to recover than rigid ones. Typically, rigid chains of trust are present in hardware-based mechanisms and flexible ones within software-based mechanisms, with some exceptions.

3.1.1.3 Remote Attestation Mechanisms

Current RA mechanisms are used in many contexts, hardware, and security requirements. The following is a non-exhaustive list of known RA mechanisms and a high-level description of their inner workings, uses, and detailed taxonomy as previously defined. This information is also available in Table 3.1. Table 3.2 describes the entities holding the various available roles.

TABLE 3.1: TAXONOMY OF VARIOUS REMOTE ATTESTATION MECHANISMS

Mechanism	Implementation	Scope	Verification	Chain of Trust
ARM TrustZone	Software	N/A	N/A	N/A
TPM 1.2/2.0	Hardware	System	Offline	Rigid

Funded by Horizon Europe Framework Programme of the European Union

Android Safety Net / Play Integrator	Software	Software/ System[1]	Online	Flexible
iOS Device Check	Software	Software[1]	Offline	Flexible
AMD SEV (-ES)	Software	System	Offline[2]	Rigid
AMD SEV-SNP	Software	System	Online	Flexible
Intel SGX	Hardware	Software	Online	Flexible
Intel TDX	Software	System	Online	Flexible

TABLE 3.2: REMOTE ATTESTATION ROLES WITHIN VARIOUS IMPLEMENTATIONS

Mechanism	Attester	Relying Party	Verifier	Endorser
ARM TrustZone	TEE Software	N/A	TEE or Endorser server	TEE software dev or SoC vendor
TPM 1.2/2.0	TPM	End user	End user	TPM Vendor/ Manufacturer
Android Safety Net / Play Integrator	Android TEE[3]	App developer	Google	Google
iOS Device Check	iOS TEE[4]	App developer	Apple or App developer	Apple
AMD SEV (-ES)	AMD Firmware	VM owner[5]	VM owner	AMD
AMD SEV-SNP	AMD Firmware	VM user[5]	VM user	AMD
Intel SGX	Intel ISA & Intel Enclave	Software developer	Intel	Intel
Intel TDX	Intel Firmware & Enclave	VM user	Intel	Intel

[1] While Android's SafetyNet/Play Integrity mechanism ensures the application and the system it runs on are trustworthy, iOS DeviceCheck only protects the application's trustworthiness without giving any assurances if the underlying device is modified.
[2] While this mechanism's initial provisioning phase requires online verification with Apple's servers, future verifications can be done without any communications with Apple's servers.
[3] Android TEE depends on platform architecture, ARM TrustZone for ARM Android versions. Unable to determine the exact TEE platform used for x86_64 versions.
[4] iOS TEE is the security coprocessor named "SecureEnclave".
[5] In this context, the VM owner and user represent two distinct entities. The VM owner is considered a privileged party to the proper execution of the VM, distinct from the VM Host. In the context of cloud providers, due to security concerns, this is most often the cloud service provider itself and not its end user. The VM user, however, is simply the entity that will use the VM once it has been launched.

Funded by Horizon Europe Framework Programme of the European Union

A detailed study of the remote attestation mechanisms provided by AMD SEV and Intel SGX/TDX were performed but will not be detailed here for the sake of simplicity. However, the outcome of these studies highly participates in the following design and development.

3.1.2 Attested Launch Protocol Design

"Attested Launch" is a concept that relies upon Remote Attestation to ensure that a given system may only be started if a given RA mechanism is carried out successfully and makes it inoperable any other way, through the late provisioning of critical cryptographic material.

When integrated with FLUIDOS, this protocol can substantially enhance the trustworthiness of dynamically shared resources across FLUIDOS clusters, ensuring that offloaded workloads are executed only on nodes with a verified and trustworthy OS. This is especially crucial in dynamic, decentralized environments where the risk of encountering a corrupted or malicious node is high. Notably, we believe that scenarios with a corrupted or malicious FLUIDOS node may include:

- Workload offloading: The attested launch protocol would prevent the workload from being offloaded if the node's OS does not match the verified, trustworthy state.
- Dynamic Resource Sharing: With the attested launch protocol in place, before any resources are shared or borrowed, the integrity of the target system is confirmed. A malicious node that has been tampered with would be identified and isolated, ensuring it doesn't participate in the resource-sharing pool.
- Data security: the attested launch protocol would ensure that any node handling sensitive data has an OS that hasn't been tampered with, protecting against data leaks or unauthorized data manipulation.

To demonstrate this, we developed a Proof-of-concept where attested launch is achieved by storing the system (Ubuntu 20.04)'s root partition behind a LUKS encrypted disk for which the key will only be transmitted upon a successful RA mechanism run. In the event of a RA failure, no key shall be delivered, and the system will not be able to start.

In this PoC we attested the launch of a VM representing the system to be attested. This attestation will also guarantee a certain amount of isolation for workloads and containers deployed within the VM.

3.1.2.1 Requirements and technical baseline

The remote attestation mechanism must be able to attest to the initial state of a system and not only an application. It must also be able to carry arbitrary data such that replay protection may be implemented in the form of a nonce within the attestation report. Furthermore, it must be possible to ensure that only code belonging to the VM owner may run within the VM hosting the system, or at the very least, that a strong chain of trust may be established from the initial VM image provided by the host (and properly audited), up to the OS level.

Funded by Horizon Europe Framework Programme of the European Union

After careful consideration of the remote attestation mechanisms provided by AMD and Intel, we decided to base the first proof of concept on the SEV-SNP remote attestation platform, with some considerations taken to ensure future compatibility with Intel TDX.

This protocol relies on the following entities :

- Guest VM: AMD SEV-SNP VM instance that is trying to prove it has not been tampered with to request cryptographic secrets.
- Verifier: The entity holding the required cryptographic secrets, provisioning them to the Guest VM if it can assert it has not been tampered with.

For the protocol to be usable with multiple VM instances simultaneously, it must be possible to segregate the LUKS secret between instances. In order to do so, a 32-byte identifier will be used during the protocol.

3.1.2.2 Threat Model

In the context of this attested launch protocol, the most important asset to secure is the cryptographic payload associated with a given identifier (the LUKS key that unlocks the instance's disks) . Note that we will exclusively focus on the Attested Launch protocol and not the underlying Remote Attestation mechanism. It is assumed that AMD takes the necessary steps to ensure the security of SEV-SNP through security advisories and updates to relevant firmware blobs. A high-level overview of the threat model can be found within Figure 3.1.

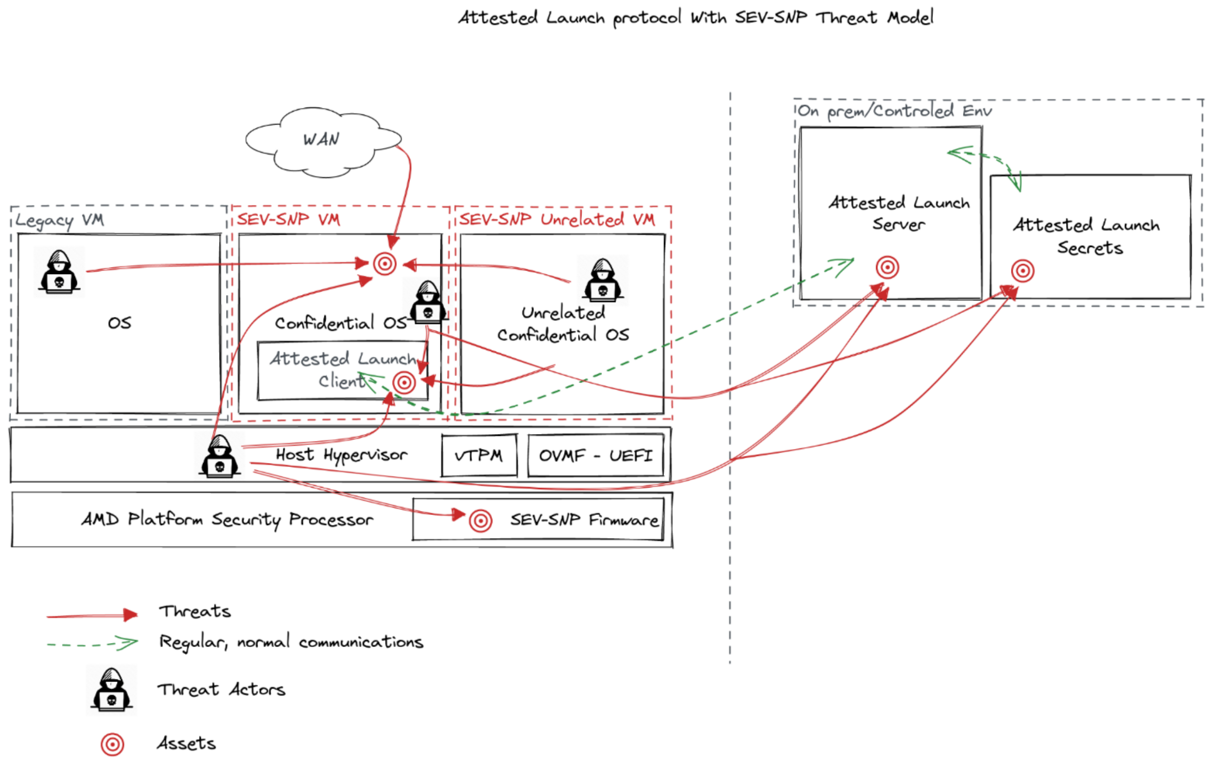


FIGURE 3.1: THREAT MODEL OF ATTESTED LAUNCH ON AMD SEV-SNP

Funded by Horizon Europe Framework Programme of the European Union

3.1.2.3 Protocol Design

The attested launch is achieved according to the following steps:

1. The guest VM connects to the Verifier through its *virtio-vsock* with the help of the host.
2. The guest VM sends the message "Begin Verification <ID> ", ID being a 32-byte VM-specific identifier provided by KVM at VM launch in the HostData SNP field (or baked into the OVMF image if KVM does not allow for HostData provisioning).
3. The Verifier checks that the ID value is known and sends a 32-byte nonce.
4. The guest VM attaches to the attestation report the nonce and the ID value. It then transmits the attestation report to the Verifier.
5. The Verifier then checks if the attestation report is correctly signed with AMD's keys, that the attached nonce matches the one provided during the communication, and that the ID value matches the one used to initiate the session. From there, the Verifier sends a payload containing any relevant cryptographic material

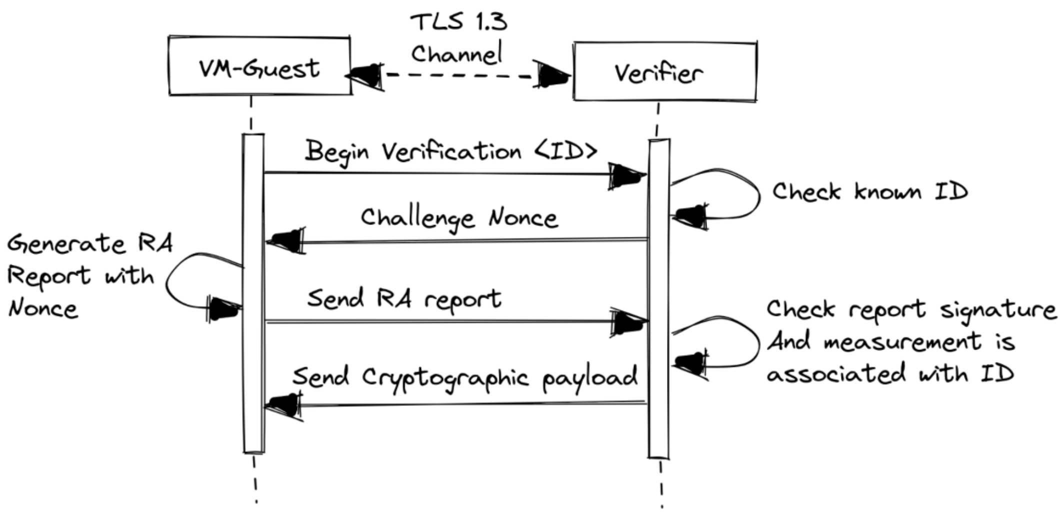


FIGURE 3.2: PROOF OF CONCEPT ATTESTED LAUNCH PROTOCOL

Figure 3.2 provides a description of the protocol.

Note that the protocol assumes the following:

- **Channel Security Requirements:** This protocol would rely upon a 1-way authenticated, confidential channel that is integrity-preserving. For this PoC, the channel has been established through TCP over TLS 1.3 between the guest VM and Verifier; the guest will be able to authenticate the Verifier through a provisioned certificate chain in the *initramfs* with the help of pinned trust anchors. The use of a 1-way authenticated channel instead of a 2-way authenticated channel is that this protocol itself does not care about who is communicating with the server. Any guest communicating with the server holding a valid identifier and its associated

Funded by Horizon Europe
Framework Programme of the European Union

launch measurements from the remote attestation mechanism will be given the secret associated with the identifier. As such, the task of ensuring that a received secret is verified in depth (i.e., that it properly unlocks the underlying disk) and safe aborts in case of failure is left to the measured code to implement. The use of 2-way authenticated channels would prove too complex to set up in a way that may not be manipulated by either the underlying host or guests themselves.

- Communications Between the VM and Verifier: The communication between the guest VM and the Verifier is somewhat outside the scope of the protocol itself. It is assumed that the underlying host and measured part of the guest will cooperate in such a way as to facilitate the opening of a TLS 1.3 channel to the Verifier through the time synchronization required for TLS 1.3 and networking necessary for cloud service provider contexts.

3.1.2.4 Security claims

The following are security claims that may be obtained from the protocol, assuming that the technology of the underlying Remote Attestation mechanism was not compromised and that the entire VM code was controlled by the VM developer :

- A VM may only start if its initial code was not tampered with. Indeed, through Remote Attestation measurement, we can check that the initial code was as expected.
- A VM may only start if it was launched inside authorized hardware (AMD SEV-SNP compatible CPU). Indeed, through Remote Attestation measurement, we can check that the system is running on a genuine CPU.
- Disk encryption keys may only be distributed to the VM if it is trusted. Indeed, through the first two security claims, we can be sure that we are running with untampered code within genuine, trusted hardware. As such, only a trusted instance may receive the encryption keys.
- Disk encryption keys may not be distributed by the protocol to any entity other than the VM or code running within the VM. Indeed, since the protocol runs within a TLS 1.3 channel that authenticates the Verifier, preventing a man-in-the-middle attack, the encryption keys may not be intercepted in transit. Due to the use of an inter-protocol nonce, captured reports may not be replayed at a later date to obtain the encryption keys.
  Due to the first three security claims, only a trusted VM may trigger the distribution of disk encryption keys. As such, only the VM or any code running within the VM may receive the encryption keys.
  NB: If the underlying protocol was to be Intel TDX, this claim could be pushed further such that "Only the VM may receive the encryption key, and only at boot time".
- A VM may only start if a distant system authorized its start (allows for permanent decommission of remote VMs). Indeed, Due to the disk encryption keys never being distributed to an unauthorized third party, if at any point the given VM instance were

Funded by Horizon Europe Framework Programme of the European Union

to be discontinued, this could be enforced through the distant system no longer provisioning the disk encryption keys.

- A VM may start securely even if the underlying host is untrustworthy or even malicious. Indeed, due to all the preview security claims and running exclusively trusted code within the VM, so long as the security of SEV-SNP remains uncompromised, the underlying host is unable to negatively impact the security of the virtual machine, aside from messing with data availability.

3.1.3 Attested Launch PoC Implementation

The implementation of the protocol has been done using Rust programming language whose emphasis on memory safety through its unique ownership system and borrow checker minimizes common vulnerabilities, such as buffer overflows and data races.

We also decided to implement the Verifier part of the protocol as modularly as possible. This is mostly due to a wish to not hinder the potential future adoption of alternative viable remote attestation mechanisms within FLUIDOS such as Intel TDX.

It was also decided to abstract away the TLS functionality from the Verifier and relegate it to a reverse proxy, such as to reduce the complexity of the Verifier codebase.

A simplistic proof of concept client was also written with code size considerations such that it would be able to run in a size-restricted early-boot environment.

Verifier Code the Verifier code includes the following information:

- *guest_id:* The instance identifier, allowing for multiple VMs with the same measurements to receive distinct secrets
- *expected_measurement:* The measurement we expect to find in an Attestation Report for a given instance identifier.
- *secret:* The secret associated with the identifier is to be released at the end of the attested launch protocol.

This persistence layer was accomplished through the rust "Diesel" ORM with the sqlite3 backend.

Remote Attestation Report Abstraction: Handling of Remote Attestation processing was relegated to a trait to ease the future implementation of other RA mechanisms. The code of this trait can be found in Figure 3.3.

Funded by Horizon Europe
Framework Programme of the European Union

```rust
use async_trait::async_trait;

// Only SEV-SNP reports are implemented, but other viable kinds have been listed
↪  for completeness
pub enum ReportKind {
    _Sev,
    SevSnp,
    _Tdx,
}

#[async_trait]
pub trait Report: Sync + Send {
    fn kind(&self) -> ReportKind;
    fn bytes(&self) -> &[u8];
    fn guest_id(&self) -> &[u8];
    fn guest_data(&self) -> &[u8];
    fn launch_measurement(&self) -> &[u8];
    // Must be async due to requesting certificates from online servers
    async fn verify_signature(&self) -> bool;
}
```

FIGURE 3.3: RUST TRAIT FOR INTERFACING WITH REMOTE ATTESTATION MECHANISMS

The implementation of this trait for the AMD SEV-SNP trait was mostly delegated to the "*sev_snp_utils*"[13], Rust crate, implementing all required methods for parsing and verifying the validity of an attestation report, along with all relevant network requests and certificate caching.

Client Code:

For this PoC, client code relies upon the "easy-tokio-rustls"[14] crate to provide simple TLS communication support with custom trust anchors, allowing for simple certificate pinning support.

This client code will have to be rewritten for FLUIDOS. Client code will connect to a distant Verifier server through a TLS channel, proceed to run the attested launch protocol, and on success, print the secret to standard output.

Testing:

Proper testing was mostly dedicated to the implementation of the Verifier aspect of the protocol. Most of the testing was around a subset of the failure modes identified previously.

Testing is done through a Python script that interfaced with the Verifier through a TCP client and interfaced with a SEV-SNP capable VM with relevant Remote Attestation helper scripts through SSH.

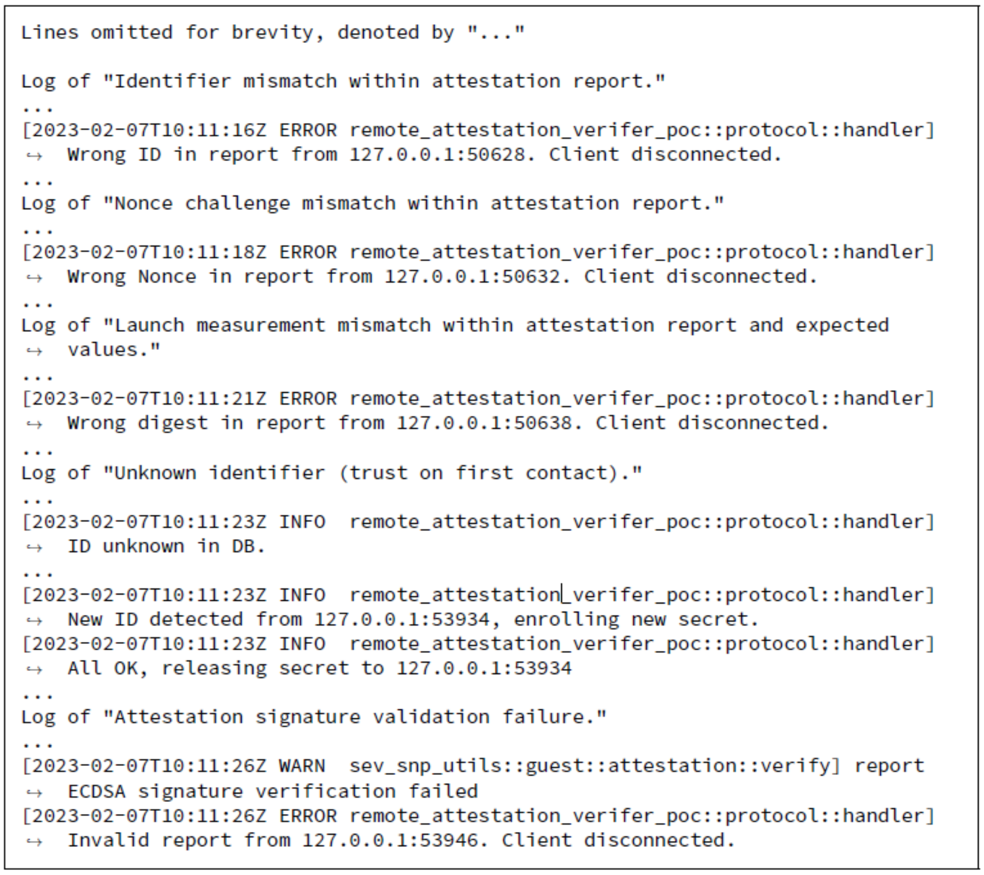
The following failure modes were tested successfully through this method:

- Unknown identifier (trust on first contact).

Funded by Horizon Europe Framework Programme of the European Union

- Attestation signature validation failure.
- Identifier mismatch within attestation report.
- Nonce challenge mismatch within attestation report.
- Launch measurement mismatch within attestation report and expected values

The results of such failure modes can be seen in Figure 3.4.



```
Lines omitted for brevity, denoted by "..."

Log of "Identifier mismatch within attestation report."
...
[2023-02-07T10:11:16Z ERROR remote_attestation_verifer_poc::protocol::handler]
↪  Wrong ID in report from 127.0.0.1:50628. Client disconnected.
...
Log of "Nonce challenge mismatch within attestation report."
...
[2023-02-07T10:11:18Z ERROR remote_attestation_verifer_poc::protocol::handler]
↪  Wrong Nonce in report from 127.0.0.1:50632. Client disconnected.
...
Log of "Launch measurement mismatch within attestation report and expected
↪  values."
...
[2023-02-07T10:11:21Z ERROR remote_attestation_verifer_poc::protocol::handler]
↪  Wrong digest in report from 127.0.0.1:50638. Client disconnected.
...
Log of "Unknown identifier (trust on first contact)."
...
[2023-02-07T10:11:23Z INFO  remote_attestation_verifer_poc::protocol::handler]
↪  ID unknown in DB.
...
[2023-02-07T10:11:23Z INFO  remote_attestation_verifer_poc::protocol::handler]
↪  New ID detected from 127.0.0.1:53934, enrolling new secret.
[2023-02-07T10:11:23Z INFO  remote_attestation_verifer_poc::protocol::handler]
↪  All OK, releasing secret to 127.0.0.1:53934
...
Log of "Attestation signature validation failure."
...
[2023-02-07T10:11:26Z WARN  sev_snp_utils::guest::attestation::verify] report
↪  ECDSA signature verification failed
[2023-02-07T10:11:26Z ERROR remote_attestation_verifer_poc::protocol::handler]
↪  Invalid report from 127.0.0.1:53946. Client disconnected.
```

FIGURE 3.4: OUTPUT LOGS OF THE VARIOUS FAILURE MODES TESTES

3.2 ISOLATION OF CONTAINERS

One of the main objectives of WP5 is to establish a reliable and isolated environment for running workloads on horizontally distributed FLUIDOS nodes. Achieving this goal involves the development of security mechanisms to protect FLUIDOS nodes from malicious users as well as to protect users from honest-but-curious FLUIDOS providers.

In this context, we, first, introduce Intent-based border protection, a policy-driven orchestration solution to automate the configuration of isolation primitives, e.g., Kubernetes Network Policies, across the entire orchestrated domain.

Funded by Horizon Europe Framework Programme of the European Union

Second, we present a tool designed to automatically determine the minimal privileges required for proper container functionality (as detailed in Section 3.2.1). This tool plays a crucial role in reducing the size of the container-kernel interface, thereby lowering the likelihood of adversaries with control over a container launching attacks against the underlying host.

3.2.1 Intent-based Border Protection

FLUIDOS is designed to work in a multi-cloud and multi-tenant environment, where each physical node could be shared by multiple and heterogeneous users, possibly belonging to different administrative domains. This approach has several advantages in terms of costs and scalability, but it also introduces some disadvantages, in particular an increased complexity in security management that necessitates the development of innovative methodologies.

The goal of ensuring a correct network isolation between containers in the continuum serves a dual purpose. First, it is imperative to protect the hosting cluster against potential harm resulting from guest applications. Second, it is equally important to protect guest applications against any potential stealing of data or code or unauthorized interference from the host. Another important need is to allow application owners to specify precisely what interactions their applications may have with their environments and what the hosting environments are willing to allow. Based on such specifications, communications according to the general least privilege principle are to be restricted. Furthermore, the concept of extended virtual cluster adopted by FLUIDOS dictates the need for novel solutions with respect to the current state of the art. The conventional notion of a physical boundary that must be protected has evolved into a dynamic and continuously shifting virtual boundary, which spans across various domains and evolves over time.

To achieve these objectives, the solution proposed for FLUIDOS is a security orchestrator to automate the configuration of isolation primitives, e.g., Kubernetes Network Policies, across the entire orchestrated domain. The solution is policy-driven, because desired and prohibited network connections are expressed through user-defined intents. In greater detail, the proposed solution allows the formulation and enforcement of finely grained isolation policies to ease the implementation of common security patterns such as zero trust and least privilege. The whole process revolves around different sets of intents defined by both hosting and hosted tenants which are exchanged during the peering process. Therefore, a harmonization is performed between the configuration of both tenants taking part in the peering to intelligently select the resulting set of approved intents, which are later translated and enforced in the appropriate locations. This document contains a study of the problem and the definition of the semantic of the intents, with an initial proposal for the refinement workflow. The actual implementation of the distinct sets of intents is still a work in progress, even if different enforcement solutions have already been explored, such as Kubernetes Network Policies and the security extension of Liqo implemented in FLUIDOS, which is presented in section 3.3.2.

Funded by Horizon Europe Framework Programme of the European Union

3.2.1.1 Network connections in the continuum

The border protection problem must be declined into the different communications that may occur in a cloud environment, and specifically into the different use cases foreseen in FLUIDOS. The network fabric envisioned in liquid computing could span across different clusters, which requires to protect not only the perimeter of single pods or clusters, but also the one of virtual clusters. First, it should be possible for the user to define connectivity policies for pods in the same virtual cluster but distributed over many physical clusters. This is the deployment scenario of the "elastic cluster", which could be adopted for example during a cloud migration or to absorb load spiked, i.e., cloud bursting, during which some pods are moved to a remote cluster due to the physical limitations of the one on premise. Second, it should be possible for the user to define connectivity policies for pods belonging to different virtual clusters but sharing the same physical one. This situation can be motivated by the common principle of data gravity, requiring that the processing is moved where the data is located for improved latency, or also to be compliant with regulation policies like GDPR, which requires that data cannot be moved outside a specific geographical location.

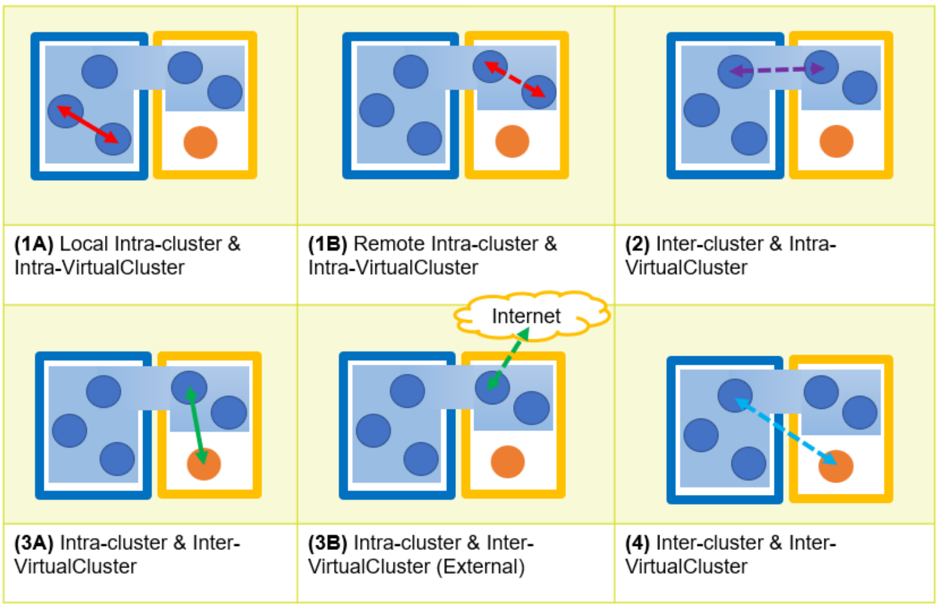

(1A) Local Intra-cluster & Intra-VirtualCluster
(1B) Remote Intra-cluster & Intra-VirtualCluster
(2) Inter-cluster & Intra-VirtualCluster
Internet
(3A) Intra-cluster & Inter-VirtualCluster
(3B) Intra-cluster & Inter-VirtualCluster (External)
(4) Inter-cluster & Inter-VirtualCluster

FIGURE 3.5: COMMUNICATION TYPES WITHIN FLUIDOS

Considering these specific interactions, along with the canonical ones, we have identified different types of communications interesting for enforcing network isolation and these are represented in Fig. 3.5, where each type of communication is presented in a different picture including two thick-bordered boxes representing two different physical clusters (one blue, owned by one tenant and one yellow, owned by another tenant), pods are represented by filled circles, and a virtual cluster owned by the blue tenant is represented by a region with blue background that spans the blue physical cluster (the home portion of the virtual cluster) and the yellow cluster (the offloaded portion of the virtual cluster). This virtual cluster includes 3 pods hosted in the home physical cluster of the blue tenant and other 2 pods offloaded to the yellow hosting physical cluster. Another pod, filled with yellow color, represents a pod

Funded by Horizon Europe Framework Programme of the European Union

that is in the yellow cluster but not part of the blue virtual cluster. The different communication types are represented by arrows. They differ for being intra-cluster or inter-cluster, and intra-virtual cluster or inter-virtual cluster.

Regarding the classified types of communications, type 1A corresponds to the base scenario for the case of controlling connectivity for pods in the same virtual cluster, involving pods within a single physical cluster which is owned by the same tenant that owns the physical cluster. This scenario can be addressed by state-of-the-art solutions based on Kubernetes Network Policies, a network isolation primitive designed to implement network isolation across pods within a cluster.

Communications of type 1B have similar characteristics with type 1A, targeting again the case of controlling connectivity for pods in the same virtual cluster, but introduce additional complexity, as the isolation primitives must be enforced on the remote physical cluster, which has control of the configuration of the remote physical infrastructure. This characteristic demands for a distributed approach to the orchestration of security intents, with a user owning the virtual cluster and another user owning the hosting physical cluster responsible for the enforcement of such intents.

Transitioning to the analysis of communication types 3A/B, which are considered jointly due to their similarities, they correspond to use cases where it is necessary to control communications involving pods belonging to different virtual clusters but sharing the same physical one or communications between offloaded pods and the internet. These cases can be handled in a manner not vastly different from types 1A/B. Specifically, from a technical point of view, both involve communications that involve a single physical cluster, where pods of a tenant are offloaded. These communications can be controlled, according to best practices, using Kubernetes Network Policy. However, the primary difference lies in the multi-tenant nature of types 3A/B, where different tenants define distinct isolation requirements that may potentially be in conflict. For this reason, a harmonization is necessary to orchestrate the different security intents, defined by different tenants, thereby ensuring a conflict free implementation. These kinds of communications are considered also by the solution of protected borders presented in section 3.3.2, which could be a base for implementing their enforcement.

Type 2 communications are another scenario referring to the case of controlling connectivity for pods in the same virtual cluster, but in the more complex situation when the communication is between pods hosted in different physical clusters. To operate traffic control for inter-cluster communications, the conventional approach is to implement a service mesh architecture, allowing the creation and denial of selected connections. However, being FLUIDOS designed to be deployed on edge devices with limited computation resources, the overhead introduced by a service mesh may be unacceptable. For this reason, we need to find a solution to intra-cluster traffic control which has a reduced footprint in terms of used resources, and which is more integrated into the FLUIDOS

Funded by Horizon Europe Framework Programme of the European Union

architecture. In this sense, the solution presented in the following section 3.3.2 can be used as a base for the enforcement of this type of communication too.

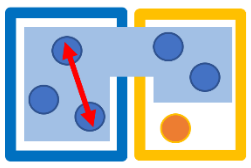
Finally, communications of type 4 could be also considered, but we consider them of limited relevance to FLUIDOS, based on the typical use cases that have been analysed.

3.2.1.2 Intents

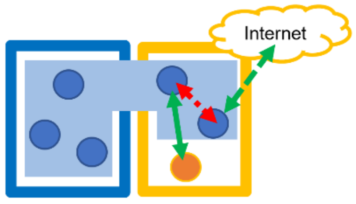
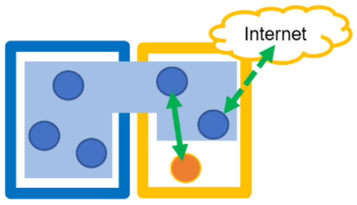
For what concerns the formulation of the security intents, users may define multiple sets of them, each tailored to achieve different objectives. We foresee three different sets of user's intents, plus a fourth one used for setup (e.g., to allow communications that are required by the FLUIDOS framework itself). When participating in FLUIDOS, users can play the roles of consumers (i.e., when they use resources of some remote cluster) and providers (i.e., when they supply resources to other clusters) in various peering scenarios. The "consumer" role entails the enforcement of network isolation policies for services offloaded to remote clusters, while the "provider" role grants the authority to impose restrictions on connections involving resources within their domain. For instance, a provider may only authorize access to selected services or force mandatory monitoring connections for all offloaded containers.

Depending on the role, a user could define different sets of intents with different goals. When users assume the consumer role, their primary network security objectives are safeguarding communications within their local cluster and protecting communications among resources offloaded to remote clusters. These objectives correspond to **Private** and **Request** intents, respectively. Conversely, whenever users assume the provider role, their primary network security goal is to limit communications between the hosted resources and local ones. For this reason, users may define a third set of intents, named **Authorization** intents. In this case, the hosting cluster can enforce its authority only to the communications crossing the border of the hosted virtual cluster, involving its own services or the usage of its connection towards the external network (i.e., the Internet). Conversely, no limitation is applied to communications happening within the border of the offloaded virtual cluster, between hosted resources. Additionally, a fourth set, automatically populated and enforced for every cluster participating in FLUIDOS, is the set of **Setup** intents. They are used to facilitate the configuration tasks, such as enabling the traffic necessary to maintain the network fabric created by the FLUIDOS architecture. Table 3.3 summarizes these intent types.

TABLE 3.3: INTENT TYPES IN FLUIDOS

Intent types	Description
PRIVATE INTENTS	These intents are related to communications happening within the local cluster (source, destination, or both, are entities of the original cluster). 

Funded by Horizon Europe Framework Programme of the European Union

REQUEST INTENTS	These intents are related to communications happening within the offloaded cluster (source, destination, or both, are offloaded entities). 
AUTHORIZATION INTENTS	These intents are related to communications from resources hosted in the cluster towards the external of their virtual cluster (e.g., deny internet access to all hosted pods, permit only traffic to some local services, maintain monitoring access to all hosted pods). 
SETUP INTENTS	These intents are related to communications required by the FLUIDOS framework. These intents are used for configuration purposes.

The definition of intents to control type 2 communications is left for future work for the moment. It will be added in a subsequent step of the work.

Regarding the format employed for defining intents, it should have a similar degree of expressiveness as the one achieved with selectors in Kubernetes Network Policy, so as to allow the specification of the information needed to select specific traffic. The envisioned structure is the following one:

"from SRC to DST, protocol [: port [- endPort]]"

- *SRC* and *DST* can be either a pod or a group of pods with the same label, or an address or a group of addresses defined through CIDR (at most one could be a CIDR address).
- *protocol* can be any transport protocol (TCP, UDP, SCTP, etc.) or "ALL".
- *port* can be a port, or a range of ports.

The information about the namespace could be added in the SRC and DST fields when specifying the selected entities. Note that, being the implementation based on Kubernetes Network Policy, allowing a unidirectional communication "from service A to service B" means that service A can start a communication with B, and B can send responses back over this connection, but service B cannot start a communication with A.

The Request, Private, and Setup intents can be expressed only in whitelisting, so as to be compliant with the default behaviour of Kubernetes Network policies, which allows to define the set of permitted communications and all the other ones are consequently blocked. Instead, The Authorization intents allow for more flexibility because they are used to authorize or deny the requested intents and they do not have a direct association with Network Policies. For this reason, Authorization intents can be expressed freely by the user

Funded by Horizon Europe Framework Programme of the European Union

according to the situation: the user could choose to express all allowed communications and to block the others (i.e., whitelisting), or to express all the denied communications and to allow the remaining ones (i.e., blacklisting).

3.2.1.3 Discordances

For the refinement of the defined intents, the first step is the analysis of the potential discordances that may arise when resource sharing occurs, and how these could be resolved with an adequate harmonization algorithm. In this context, a discordance arises when an intent defined by one user is not authorized or coherent with intents defined by another user with whom peering is requested.

These discordances can be classified in two main categories: first, the case in which a requested communication is not authorized by the hosting cluster, and second, the case in which the hosted user requests an interaction with the local resources of the hosting cluster which is not coherent with the isolation intents defined by the owner of those resources. These are represented in the following Fig. 3.6.

For the first case, a user performing the offloading is requesting that his offloaded entity A can contact a malicious website, but the authorization intents defined by the hosting user deny all connections to the internet for all offloaded pods, thus causing a discordance. Second, the consumer requests that the same offloaded entity A can contact entity B, which is part of the hosting cluster. However, the hosting user has not defined an intent allowing B to be contacted by A, thus resulting in another discordance.

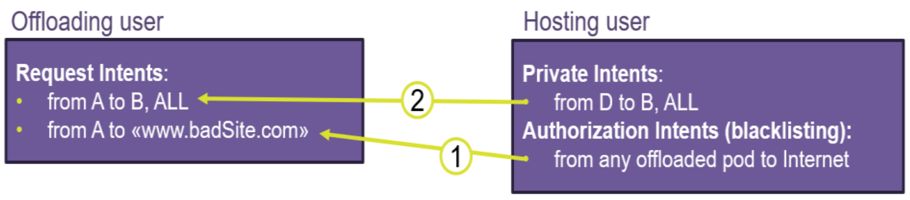

Offloading user
Request Intents:
from A to B, ALL
from A to «www.badSite.com»
2
1
Hosting user
Private Intents:
from D to B, ALL
Authorization Intents (blacklisting):
from any offloaded pod to Internet

FIGURE 3.6: EXAMPLES OF DISCORDANCES

3.2.1.4 Workflow

An algorithm has been designed to perform the harmonization between different sets of intents established by different users engaged in peering processes. This allows an intelligent resolution of the possible discordances between sets of intents, which are subsequently refined and enforced at the appropriate clusters. The general principle adopted in the harmonization algorithms is that the hosting cluster has the decision power: it chooses which request intents can or cannot be enforced while possibly forcing some new ones. The idea is that the one which is hosted must submit to the rules of the host.

The workflow envisioned for this orchestration of network isolation intents is that, during the peering process, the *Privacy and Security Managers (PSM)* of the offloading cluster and the

Funded by Horizon Europe Framework Programme of the European Union

hosting cluster interact exchanging the sets of intents and performing the harmonization, translation, and enforcement. The core process is the harmonization, which produces a new set of "*Harmonized intents*" that will be used in the peering process as an additional parameter for decisions. The peering strategy will choose which action should be taken according to the approved, forced, or denied intents (along with other parameters such as offered resources, latency, geographical position, etc.).

Fig. 3.7 graphically represents the workflow employed in the situation of a peering request performed by the offloading cluster C1, owned by user 1, towards the hosting cluster C2, owned by user 2.

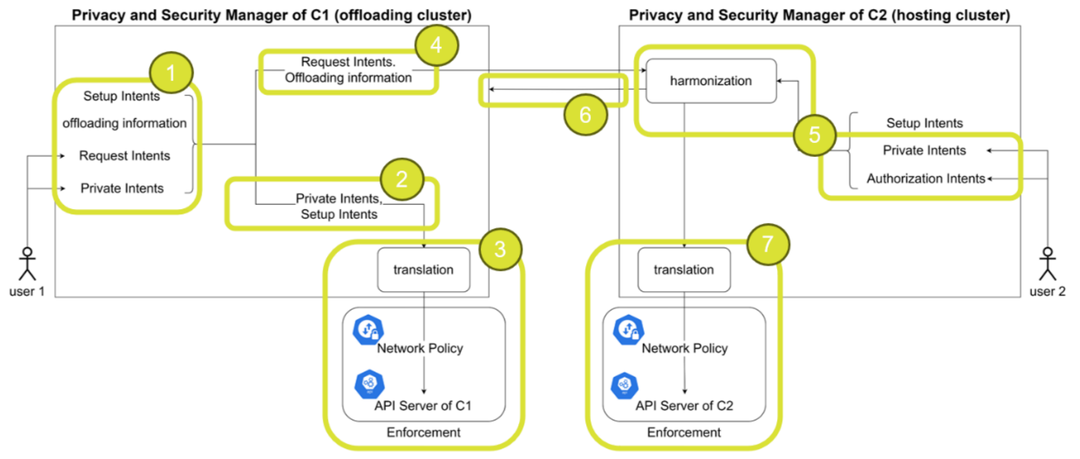


FIGURE 3.7: WORKFLOW DURING PEERING REQUEST

This process starts with (1) the PSM of the requesting cluster which retrieves the Setup intent, the information regarding the offloading (e.g., kind of resources, labels), and the Request and Private intents defined by the user requesting to offload some resources. The set of Private intents is relative to communications happening within the local cluster, thus (2-3) they are directly translated and enforced on the Kubernetes API server of the local cluster because no harmonization is needed. The Setup intents defined in cluster C1 follow the same path. On the other hand, (4) the Request intents and the offloading information are sent to the PSM of the hosting cluster C2 which performs the harmonization process. The harmonization module (5) processes the received Request intents along with the Authorization and Private intents defined by the hosting user. The goal is to detect any discordance and possibly correct them applying an adequate resolution strategy. The outcome of this process is the set of Harmonized intents (6) which is sent back to the PSM of the offloading cluster, which decides the action to take according to the adopted peering strategy. Finally (7), if the set of Harmonized intents are approved by the PSM of C1, these are enforced on the API server of the hosting cluster C2 being the one responsible for doing so. Notably, the complexity of the translation depends on the selected CNI, for example if there is no support from the CNI, intents that are global to the cluster cannot be simply translated "one-to-one" but need to be enriched with additional information gathered

Funded by Horizon Europe Framework Programme of the European Union

through the coordination of the different modules of FLUIDOS. Optimization has not yet been considered and its analysis is left for future work.

Example

We introduced an example to convey more clearly the proposed solution for FLUIDOS. The scenario includes two different clusters, each composed of multiple applications, some of which are offloaded to the other cluster. The need is to secure their interactions, which means to allow only some communications while others must be blocked. This situation is depicted in Fig. 3.8, where the arrows represent the allowed communications, and all the others must be blocked.

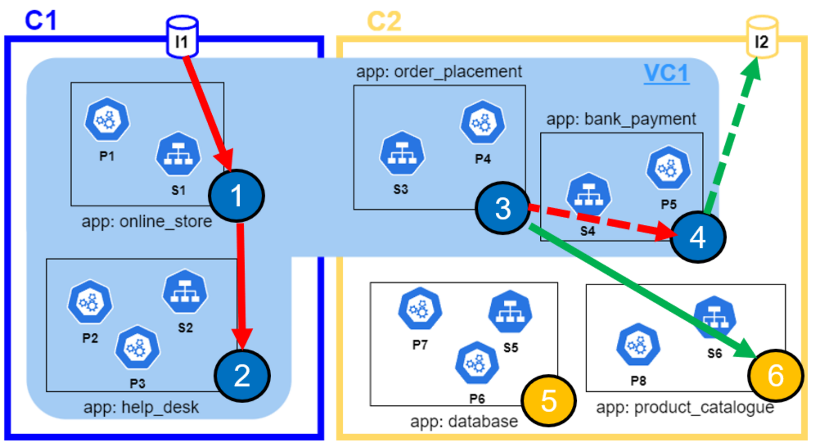


FIGURE 3.8: EXAMPLE OF WORKFLOW DURING PEERING REQUEST

The scenario presented is a simplified version of an online store built upon a microservices architecture. Within this setup, cluster C1 hosts the primary online store application, which collaborates with various other services to deliver a range of functionalities, mainly focusing on customer support and processing of incoming orders. On the other side, cluster C2 is designated to host applications that support the logistic service, a component that we can presume is outsourced to an external third party or another administrative domain within the same company. The concept underlying this configuration is to offload some applications to the logistic section, enabling them to access the provided data, such as the catalog and product availability information. Within this system, the "online store" is the only application being reachable from the internet, while the "bank payments" application is the only one allowed to connect with the internet, primarily for communication with the payment's network. Focusing on the interactions between applications within the physical clusters, we observe that the "online store" initiates email communications with the "help desk" for customer support. Additionally, the "order placement" application possesses the capability to establish connections with both the "bank payment" application for payment processing and the "product catalog" one to verify product availability, with the latter being responsible for the logistics aspect of supply management. It is important to note that while certain communications are evidently necessary, such as those between "product catalog" and "database" or between "online store" and "order placement", some interactions have not

Funded by Horizon Europe Framework Programme of the European Union

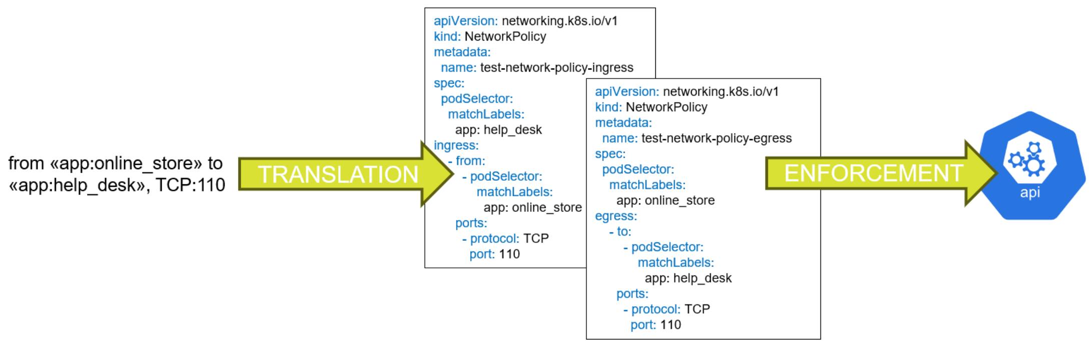
been included in this example to maintain brevity of the example, even if this omission results in a less comprehensive representation of a real-world scenario.

The desired communications are expressed through the different sets of intents. In this situation, for cluster C1, which is offloading its resources to C2, the defined sets are the Private and Request intents, being those needed in the offloading case. Whereas, for cluster C2 we have the sets of Private and Authorization intents, being the sets needed in the hosting case. These are detailed in Table 3.4.

TABLE 3.4: INTENTS OF C1 (OFFLOADING) VS C2 (HOSTING)

Intents Cluster 1 (offloading)	Intents Cluster 2 (hosting)
Private	**Private**
• from "app:online_store" to "app:help_desk", TCP:110 • from I1 to "app:online_store", TCP:80	• from any pod to "app:product_catalog", TCP:80
Request	**Authorization (whitelisting)**
• from "app:order_placement" to "app:bank_payment", ALL • from "app:order_placement" to "app:product_catalog", TCP:80 • from "app:bank_payment" to I2, ALL	Reachability: • from any offloaded pods to "app:product_catalog", TCP:80

As said in the workflow section, the private intents are processed inside the PSM of the local cluster C1. The operations performed for these can be, for example, just a translation (i.e., mapping an intent to Kubernetes Network Policies) and a subsequent enforcement (i.e., creating the Network Policy objects through the API server). The harmonization is not needed in this case. Taking as example the first private intent of C1, the process is depicted in Fig. 3.9.



```yaml
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: test-network-policy-ingress
spec:
  podSelector:
    matchLabels:
      app: help_desk
ingress:
  - from:
    - podSelector:
        matchLabels:
          app: online_store
      ports:
      - protocol: TCP
        port: 110
```

```yaml
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: test-network-policy-egress
spec:
  podSelector:
    matchLabels:
      app: online_store
egress:
  - to:
    - podSelector:
        matchLabels:
          app: help_desk
      ports:
      - protocol: TCP
        port: 110
```

FIGURE 3.9: WORKFLOW OF THE FIRST PRIVATE INTENT OF C1

Funded by Horizon Europe Framework Programme of the European Union

Instead, for what concerns the processing of the Request intents defined for C1, they are related to the communications happening in the offloaded cluster and are sent to the PSM of the remote cluster C2, where the preliminary operation which is performed is the harmonization. Translation and enforcement are performed too, but on the hosting cluster, because it is the one capable of implementing the proper network isolation. The objective of the harmonization is to detect any discordance and correct them. These could be:

1. discordances with the Authorization intents (of the hosting cluster);
2. discordances with the Private intents (of the hosting cluster).

In this specific situation, considering the set of Request intents, the first intent describes a communication internal to the virtual cluster, and for this reason no authorization is needed. For the remaining two, also considering the Authorization intents defined in C2, the second Request intent is allowed whereas the third one is denied not being an authorized communication. Considering the discordances with the Private intents of the hosting cluster C2, this requires that all pods in the cluster (also considering the offloaded ones) must allow a connection with "*app:product_catalog*". This results in an additional Harmonized intent added to the Request set of C1 to be compliant with the request. In the end, the set of harmonized intent includes the first and second Request intents (the third one is not authorized) and an additional harmonized intent to be coherent with the Private intents of the hosting cluster.

TABLE 3.5: EXAMPLE SET OF HARMONIZED INTENTS

Harmonized Intents
[*Request Intent 1*] from "app:order_placement" to "app:bank_payment", ALL
[*Request Intent 2*] from "app:order_placement" to "app:product_catalog", TCP:80
[*Harmonized Intent*] from "app:bank_payment" to "app:product_catalog", TCP:80

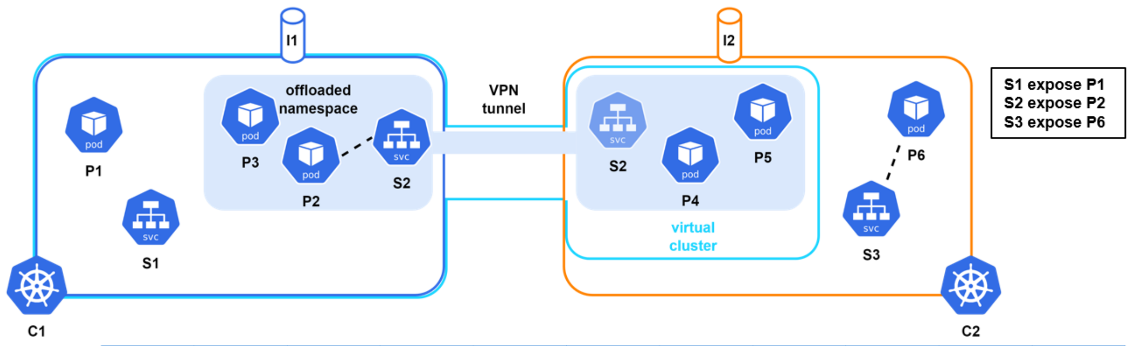
The PSM of C1 could then decide to either accept the Harmonized set of intents or to interrupt the peering process. If it decides to continue, they will be translated, e.g., to Kubernetes Network Policies and enforced on the API server of C2 or they could be implemented by using the solution proposed in section 3.3.2.

3.2.2 Liqo extension for border protection

The default mode of Liqo establishes full pod-to-pod connectivity between the pods of the two clusters involved in the peering. Given that this mode cannot provide any security perimeter to workloads running in a different cluster, we have identified some minimal features that have been pushed into the Liqo mainstream code [15], which can enable further experimentations in the FLUIDOS project.

Funded by Horizon Europe Framework Programme of the European Union

The first implementation of security in Liqo includes the segregation between intra-cluster and inter-cluster traffic. In this scenario, full pod-to-pod connectivity is enabled only within physical clusters. Regarding inter-cluster traffic, the offloading cluster can contact only its offloaded pods. However, the cluster hosting the offloaded pods can contact only the services offloaded on it, and not contact the offloaded pods directly. Figure 3.10 illustrates an example of a configuration with the related traffic matrix. The yellow highlights indicate the differences from the default mode of Liqo.



	P1	P2	P3	P4	P5	P6	S1	S2	S3	I1	I2
P1		YES	YES	YES	YES	NO	YES	YES	NO	YES	NO
P2	YES		YES	YES	YES	NO	YES	YES	NO	YES	NO
P3	YES	YES		YES	YES	NO	YES	YES	NO	YES	NO
P4	NO	YES	NO		YES	YES	NO	YES	YES	NO	YES
P5	NO	YES	NO	YES		YES	NO	YES	YES	NO	YES
P6	NO	YES	NO	YES	YES		NO	YES	YES	NO	YES

FIGURE 3.10: EXAMPLE OF A CONFIGURATION WITH THE RELATED TRAFFIC MATRIX

The second implementation, still under development, focuses on border protection, starting from the following assumption: in a physical cluster, the borders should be protected from entities outside those borders, similarly in the case of a virtual cluster that spans multiple clusters and (possibly) multiple domains. In this scenario, as shown in case 3A in the first picture, the intra-cluster traffic should be blocked and allowed only through authorized holes. This means that the offloading cluster permits communication between its pods and the pods scheduled on the hosting cluster only if necessary and not by default, to secure its borders.

Both implementations are based on Kubernetes-native data structures (i.e., Custom Resource Definitions, CRD), which facilitates future activities and extensions within the Liqo project.

Funded by Horizon Europe Framework Programme of the European Union

3.2.3 Minimizing Docker Container Privileges for Node Security Policy Enforcement

Over the past years, Docker containers have become a popular choice for developing applications in the cloud and at the edge due to their increased flexibility and lower costs. Unlike virtual machines (VMs), where each has its own operating system (OS), all containers on a host share the same underlying OS kernel allowing for faster boot times and greater efficiency. However, this comes at the expense of providing weaker isolation compared to that of VMs. Adversaries who control a container on a third-party server can exploit vulnerabilities in the kernel in order to compromise co-resident containers or, even worse, gain elevated privileges on the host. An example to illustrate the consequences of this design decision is the weakness found in the *waitid* syscall, which allowed adversaries to execute a privilege escalation attack in order to escape the container and gain access to the host [16]. This is not an isolated case since it is likely that there are many more kernel vulnerabilities to be discovered. Kernels are very complex components with a large code base that is regularly updated and extended, making them prone to vulnerabilities.

One of the main enablers of these attacks is that current containers typically run with more privileges than they need, violating the well-known principle of least privilege. By default, Docker containers can invoke any of the 300+ supported by the Linux kernel (except for 44 that are blocked) and are granted 14 capabilities (out of the 38 available) [17]. Importantly, even if the container does not utilize some of these syscalls or capabilities, all these privileges are available within the container and can therefore be abused by an adversary who gains access to the container. This has serious security implications because every syscall or capability allowed in a container becomes an entry point to the kernel, which increases the chances of adversaries finding vulnerabilities in the kernel [18].

According to the NIST container security guidelines, reducing the attack surface of the host OS kernel is a promising way to alleviate the fragile isolation containers provide [19]. There are several Linux kernel mechanisms, such as Seccomp and Capabilities, that are commonly used by cloud providers to mitigate attacks against the host OS kernel originating from malicious containers. However, these mechanisms do not provide any means for automatically discovering which are the privileges each container requires. Currently, this is a manual and time-consuming effort that typically leads to overestimating or underestimating the privileges required by containers.

To automate this process several papers have proposed solutions based on the usage of dynamic or static analysis (or a combination of the two). Dynamic analysis-based solutions require running the container and capturing the system events it invokes, while static analysis-based solutions extract this information by inspecting the container's source code or its binary. Unfortunately, both techniques have limitations. Standard dynamic analysis solutions are unable to capture syscalls that occur in container execution paths that are not observed in the container profiling phase. In contrast, the problem with static analysis solutions is that

Funded by Horizon Europe Framework Programme of the European Union

they do not take into account the container runtime and thus include all possible syscalls that can occur according to the container's source code or binary. Put differently, dynamic analysis-based solutions tend to underestimate the syscalls required by the container (which can lead to execution errors), while static analysis-based solutions typically overestimate privileges (which can lead to a larger attack surface for the host OS kernel). Given that static analysis-based solutions do not significantly reduce the attack surface and require access to the source code or binary of the container (which in some cases may not be available), we focus on dynamic analysis solutions and propose a way to mitigate their limitations in terms of coverage.

In this section, we propose BeaCon, a novel tool that enables cloud providers to automatically discover the syscalls and capabilities invoked by applications running inside containers. The goal of BeaCon is to create tighter policies than those generated when using static analysis techniques, while at the same time mitigating the coverage problems of existing dynamic analysis-based solutions. To achieve this, BeaCon relies on dynamic analysis, but unlike previous solutions based on dynamic analysis, it takes into account the possible environments that containers may be subjected to in practice while profiling them. This way, BeaCon is able to observe system events that would otherwise not be visible in the container profiling phase and that could later cause the container to fail. In addition, we design BeaCon such that application owners can adjust the security level of policies based on the desired level of security they wish to have in their applications.

3.2.3.1 Threat Model

For the threat Model, we consider adversaries who have gained access to a container. The way the adversary achieves this is beyond the scope of this document, but a serious possibility is that the adversary does so by exploiting a vulnerability in the container application. Recent studies have shown that most container images in public repositories (e.g Docker Hub), many of which are very popular and have been downloaded millions of times, contain serious vulnerabilities that take a long time to be fixed [22, 23]. After compromising the container, the adversary's goal is to escape the container in order to compromise other (co-resident) containers or gain elevated privileges on the host. The key to carrying out this attack successfully is for the adversary to find a vulnerability in the kernel [24], and to do so, the adversary can abuse any of the privileges granted to the container. From this observation, it becomes evident that the wider the container-kernel interface (i.e., the more syscalls are allowed within the container), the larger the attack surface and hence the more likely it is for the adversary to find a vulnerability in the kernel.

Assumptions

We assume that containers are run without root privileges and are properly isolated using standard Docker security mechanisms (e.g., *cgroups* and *namespaces*). These are standard practices in any container-based environment. The only assumption we make is that

Funded by Horizon Europe Framework Programme of the European Union

container images themselves are not compromised when pulling from the image repository. It is reasonable because cloud providers typically utilize security tools to detect vulnerabilities in container images before running them in their platforms [25]. This assumption guarantees containers during the profiling phase are not compromised, thus they do not request more privileges than those they need.

3.2.3.2 BeaCon

The goal of BeaCon is to provide a method for cloud providers to automatically identify the system calls and capabilities that should be allowed in the seccomp and capabilities profiles they create for their containers. BeaCon comprises three main modules: (1) the Emulation Agent (2) the Monitoring Agent, and the (3) Decision Agent.

1. Emulation agent: The Emulation Agent is responsible for generating a set of environments that are likely to be exposed to containers, and running containerized applications with each environment (one at a time). BeaCon takes a proactive approach by emulating execution environments for containers. This emulation involves accurately replicating the external factors that influence the container's behaviour. In particular, BeaCon places a strong emphasis on two external factors: (i) Docker command options provided to containers during initialization and (ii) workloads applied to containers at runtime. The decision to prioritize Docker options and workloads in the emulation process is rooted in the understanding that these factors can strongly impact the way containers are executed.
2. Monitoring agent: The monitoring agent runs the container multiple times while applying each of the environments generated by the Emulation Agent one at a time. For each environment, the monitoring agent collects the syscalls invoked and the capabilities requested.
3. Decision agent: Beacon proposes the use of a security and functionality score in the generation of policies. These scores allow application owners to quantitatively express their desired security and functionality goals. The security score, denoted as $Ssecurity$, and the functionality score, denoted as $Sfunctionality$, of policy pc for container c are calculated using Equation (1) and (2), respectively. In these equations, $CVSS(e)$ represents the highest CVSS value among the CVEs associated with the system event (system call or capability) e, and $Ec(env)$ denotes the set of system events triggered when the external environment env is applied to the container c.

$$S_{security}(p_c) = 1 - max_{e \in p_c}(CVSS(e))/10 \qquad \text{Equation (1)}$$

$$S_{functionality}(p_c) = P(E_c(env) \subseteq p_c) \qquad \text{Equation (2)}$$

At first glance, BeaCon could be designed to run entirely online. However, the Beacon profiling phase requires running the container multiple times, each time with a different environment, and collecting all system events sent over a few minutes. Consider, for example, a simple scenario where cloud providers want to consider 10 different

Funded by Horizon Europe Framework Programme of the European Union

environments, capturing for each environment all system events sent for say 5 minutes. (Note that, in practice, the monitoring time may be longer and that the number of containers to be profiled simultaneously may be much higher). The problem with this approach is that the time required for the container profiling phase can break quick-start containers, dwarfing the benefits of the container environment. To avoid this problem, BeaCon uses a two-phase analysis, which includes an offline phase conducted before containers are run, and an online phase where container profiles are created and used in production environments.

In the offline phase, BeaCon exposes containers to both generated and user-provided environments, and collects and stores all the information in order to speed up policy decision-making in the online phase. However, a potential problem with this approach is that all possible combinations of containers with different environments would have to be considered, which would have significant computational and storage costs. We show that it is possible to achieve this while considerably reducing the number of times the container has to be monitored and the information that has to be saved about it.

3.2.3.3 Evaluation

We evaluate the effectiveness of BeaCon in generating policies that are accurate, secure and lightweight. For this purpose, we rely on a dissimilarity score D that resembles the widely known Jaccard Similarity [26] and is defined in Equation (3). Its goal is to measure the difference between multiple sets Si each containing a sequence of system events (i.e., syscalls or capabilities) collected from the i-th observation. The dissimilarity score (D) can take any real value between 0 to 1, where 1 means that there are no common elements in the sets and 0 represents that they are identical.

$$D(S_1, \ldots, S_n) := 1 - \frac{|S_1 \cap \ldots \cap S_n|}{|S_1 \cup \ldots \cup S_n|}$$

Equation (3)

Effect of different options passed to containers

We started by investigating whether the Docker command options are likely to influence the privileges the container needs to run. To that end, we executed each of the container images first without any option then with nine popular Docker runtime options one at a time, collecting the system events sent by the containers each time. Subsequently, for each container image, we computed the dissimilarity score considering the sets of system events observed when applying each of the options separately as follows: $Doptions = D\,(Sno\_option$, $Sopt1$ …, $Sopt9$). Figure 3.11 illustrates the impact of different options on the dissimilarity of system events in 161 containers. A high dissimilarity score indicates the generation of more distinct system events. Our results show that at least 5% distinct system calls in 53% of the containers (85 images). On the other hand, we saw that 6% of the containers (10 images) required 20% more capabilities when certain Docker options were used.

Funded by Horizon Europe Framework Programme of the European Union

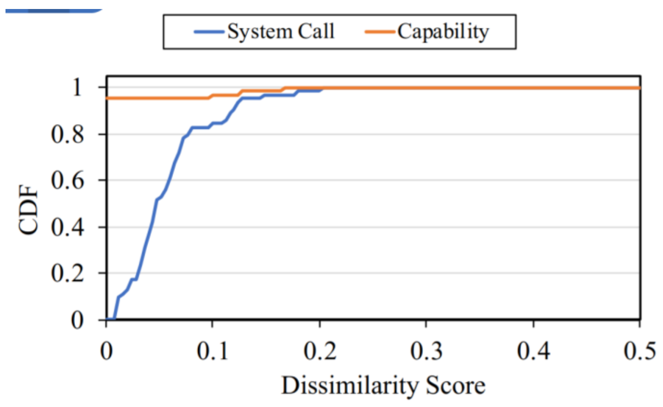


FIGURE 3.11: CUMULATIVE DISTRIBUTION OF A NUMBER OF SYSCALLS, WHEN APPLYING DIFFERENT OPTIONS

Effect of different workloads passed to containers

Afterwards, we ran a similar experiment, but this time to quantify the effect of different workloads on system events generated by containers during their execution. For this experiment, we selected 71 official images that can be used with YCSB. As before, we first run the container and collect the system events it requests when no workload is applied and then repeat the same process while applying 8 different workloads one at a time. Finally, we calculate the dissimilarity score between the different sets of system events observed for a given container and the environments applied to it, i.e. $Dworkload = D(Sno\_workload\ , SWa\ ,$ $SWb, SWc\ , SWd\ )$. Figure 3.12 shows the dissimilarity score of system events when different workloads are applied. Our results indicate that 39% of containers (28 out of 71 images) exhibit at least 5% of distinct syscalls under varying workloads. Moreover, for 10% of containers (7 images), we observed 20% of distinct syscalls. In terms of capabilities, we observed that 4% of containers (3 images) required an additional 10% of capabilities.

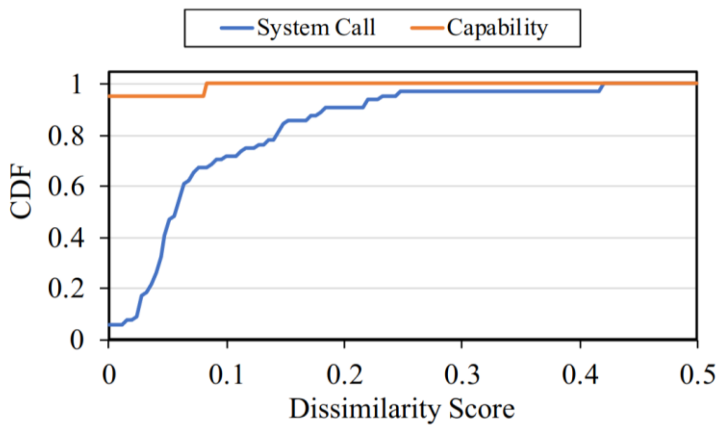


FIGURE 3.12: CUMULATIVE DISTRIBUTION OF A NUMBER OF SYSCALLS, WHEN APPLYING DIFFERENT WORKLOADS INTO CONTAINER

Independence of container's behavior under multiple environments

BeaCon requires executing and capturing the system calls invoked by the container while potentially applying it to a significant number of environments. This could create a large

Funded by Horizon Europe Framework Programme of the European Union

overhead, both in terms of the number of computations to perform and the information to store. Next, we investigate if once we have executed the container with different environments (one at a time) it is possible to merge the results obtained with several environments to know what the behaviour of the container would be if several of those environments were applied at the same time. This way cloud providers could achieve the desired result without needing to run the container for all possible combinations of environments.

To study the feasibility of this optimization technique, we select two options (i.e. -it and --publish-all) and two workloads that perform read and update, respectively. We then take all the combinations between them to create six more complex environments that include only options, only workloads as well as options and workloads simultaneously. At this point, we proceeded in the same way for all cases. We collect the system events requested by the container first by applying each of the environments separately, and then by applying them simultaneously. As a final step, for all containers and their sets of system events, we compute the dissimilarity score as follows: $Dind = D\ (Senvi \cup Senvj\ ,\ Senvi\ j\ )$. Figure 3.13 shows that up to 62.5 % of container images-environment pair have an identical set. Moreover, the largest value of syscall dissimilarity is 0.02, meaning that only 2% of syscalls are the difference between the two sets. Meanwhile, there is no difference between the two sets independent of the type of container images and given environment regarding capabilities.

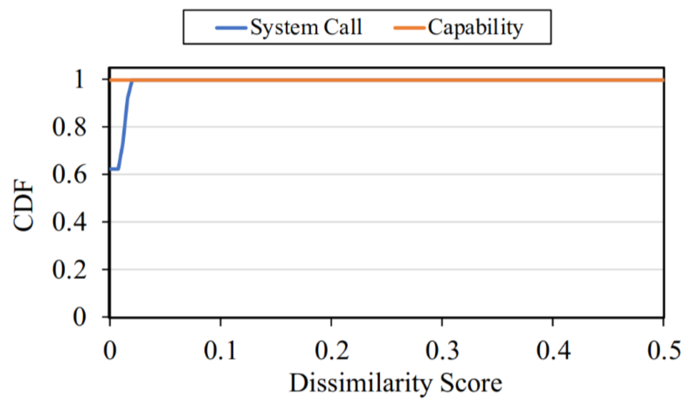


FIGURE 3.13 :CUMULATIVE DISTRIBUTION OF A NUMBER OF SYSCALLS, ONE SET BY MERGING TWO SETS FROM SINGLE ENVIRONMENTS,

3.2.3.4 Conclusion

This work presents the design and implementation of a novel approach for the automated generation of intention awareness policies suitable for Docker containers. It is based on running containers and capturing the system events they request using dynamic analysis coupled with realistic environments. BeaCon is lightweight, non-intrusive, and fully compatible with any type of container image. Our security use-cases demonstrate that BeaCon can efficiently reduce the attack surface, thus preventing attacks from adversaries who exploit excessively granted privileges. Additionally, our evaluation results indicate that it is crucial to consider the environments under which containers are executed in order to assign only the minimum privileges required to containers.

Funded by Horizon Europe Framework Programme of the European Union

3.3 WORKLOAD CONFIDENTIALITY

Workload confidentiality is a very important target when securing FLUIDOS nodes from malicious users as well as to protect users from FLUIDOS providers.

In this context, we present a novel attack against confidential containers, where honest-but-curious FLUIDOS providers, offering users the option to run their workloads within a Trusted Execution Environment (TEE) to enhance security and privacy, may potentially identify the application within the TEE by exploiting information leakage at the container-kernel interface in the form of syscalls. Furthermore, we demonstrate the practical feasibility of this attack and introduce a countermeasure to mitigate its impact.

3.3.1 A new side channel attack against confidential computing

Until recently, cloud providers have relied on a security model that focuses on keeping their customers' applications encrypted at rest and in transit, but not while in use. Unfortunately, this security model falls short for many of the applications running in the cloud today. This is because every time the code and data of an application are decrypted and moved in clear-text from secondary storage to system memory for use, they are at serious risk of being accessed or manipulated by vulnerable or malicious software at the system level (e.g., the operating system, hypervisor or the BIOS), or by a malicious cloud operator with administrator or physical access to the cloud provider's infrastructure. In response to this threat, confidential computing has emerged to protect the customers' applications not only at rest and in transit, but also while in use. Since then, major cloud providers, such as Amazon [28], Google [29] and Microsoft [30], have begun to include confidential computing as part of their infrastructure offerings, in order to provide greater security and privacy guarantees to their customers.

At the core of trusted computing are Trusted Execution Environments (TEEs), which allow storing data and executing arbitrary code in a secure memory area (also known as enclave) on an untrusted server. Importantly, TEEs allow to significantly reduce the Trusted Computing Base (TCB), and provide strong protection even against attacks from adversaries who have root privileges on the server. Leveraging TEEs, confidential computing allows customers to run their applications inside containers as usual, but without cloud providers (or any compromised or malicious software in their servers) being able to infer any information about the currently running applications. Indeed, failure to deliver on this promise would jeopardize the widespread adoption of confidential computing among software developers and enterprises.
In our work, we uncover a new side-channel attack targeting confidential computing that enables adversaries to discover sensitive information about an application running inside a container backed by a TEE, by leveraging its interactions with the "outside world". As shown in Figure 3.14, containers can interact with the "outside world" in two ways: (i) over the network with other containers and cloud services, and, (ii) with the host OS kernel. In the former case, the information exchanged is usually encrypted. While traffic analysis attacks that exploit any information that is preserved when traffic is encrypted are possible (e.g., using the packet length or the number of packets exchanged in each direction), these attacks have already been studied both for TEEs (e.g., [31]), and other domains (e.g., [32]). Yet, the

container-kernel interface – which is particularly relevant in the context of containers due to the fact that it is a shared resource – remains totally unexplored. Our work targets this interface and is motivated by the fact that system calls sent via this interface could constitute an application fingerprint. Such knowledge could then be used by adversaries to launch more effective, efficient and stealthy attacks. Moreover, it could allow adversaries to detect co-location, an important prerequisite to mount a wide range of attacks including Spectre [33] or Meltdown [34].

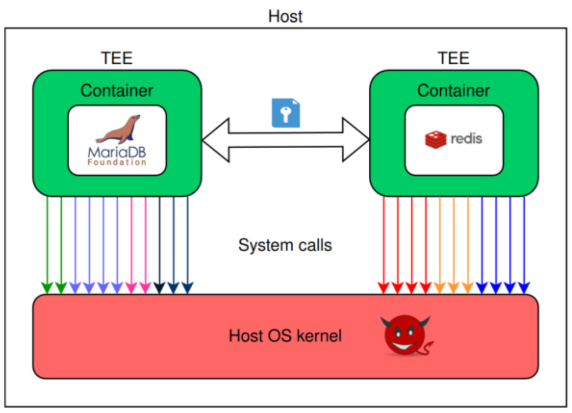


FIGURE 3.14: SYSTEM CALL PATTERNS FOR TWO CONTAINERS HOSTING TWO DISTINCT APPLICATIONS. EACH ARROW REPRESENTS A SYSCALL, AND ARROWS OF THE SAME COLOUR REFLECT SYSCALLS OF A GIVEN TYPE (E.G., THE WRITE SYSCALL)

In this work, we design a system to detect, quantify and reduce information leakage arising at the container-kernel interface in the context of confidential computing. To begin with, we propose a novel fingerprinting attack that allows adversaries to infer sensitive information about containerised applications running inside a TEE solely by observing the system calls they invoke. A key enabler for this new class of attack is that the system calls invoked by containerised applications running inside the TEE can be captured accurately from outside the container without affecting the container's performance, thus, leaving no trace from the adversary. We then go one step further and analyse the feasibility of these attacks from the perspective of a "weak adversary" who can only monitor the container for the first 2 minutes from the time the container is started (i.e., its booting time). Motivated by our findings, we introduce a countermeasure that aims to considerably reduce information leakage in the container-kernel interface. Our devised solution aligns with the tenets of differential privacy, and strategically injects a small number of (fake) syscalls in order to achieve greater uniformity in the containers' syscall patterns.

3.3.1.1 Motivation

Our work builds on the key observation that many TEEs, such as Intel SGX, require applications to leave the enclave to handle the system calls they invoke [35-37]. Through experiments, we confirmed, in line with what is mentioned in the SCONE paper [36], that when SCONE is used all syscalls are handled outside the enclave (and are thus visible to the adversary). Without loss of generality, we decided to run the Docker images without SCONE. This is due to two reasons. Firstly, the freely available version of SCONE does not support

Funded by Horizon Europe Framework Programme of the European Union

some types of Docker images. And secondly, but not less important, we wanted to quantify the information leakage that comes directly from the application itself.

As mentioned above, with confidential computing it should not be possible for adversaries to obtain any information whatsoever about the applications running inside confidential containers. This includes information such as what application is running within the TEE (e.g., a MariaDB instance), what type of application it is (e.g., if it is a database or one that performs security checks), or which version of the application is used. It is worth noting that all of this information is considered sensitive in the context of confidential computing, since it could give adversaries valuable information for them to perform security attacks more efficiently, effectively, and stealthily. For example, if adversaries can discover that a confidential container implements a MariaDB image, they can design more targeted attacks, by taking advantage of known weaknesses and attacks commonly used against this type of database. Knowing the version of the application within the confidential container can further help adversaries choose which attack is most effective and efficient – especially in the case where there are known vulnerabilities against specific versions of the application. For example, if adversaries could know that MariaDB v10.2 is used, they could try to exploit the remote code execution vulnerability reported in CVE-2021-27928 [38], while if the version of the MariaDB container is 5.5 (or older), a highly successful option for adversaries would be to try to exploit the vulnerability reported in CVE-2016-6662 [39].

3.3.1.2 Threat Model

We consider a cloud environment that offers confidential computing services to their customers. Container applications are protected using the standard container security mechanisms, and additionally are executed inside a TEE (e.g., Intel SGX). This allows application owners to upload the (encrypted) application directly into the TEE, after successfully attesting and establishing a shared key with the TEE (not known to the cloud providers). Here, we adopt the standard TEE adversary model [40], which considers adversaries who have root privileges, and hence full control over all software running outside of the TEE's hardware-protected memory region, including privileged software such as the operating system, BIOS or hypervisor.

The goal of the adversary is to discover sensitive information about applications running inside confidential containers solely from the information sent through the container-kernel interface. Adversaries view the confidential container (i.e., the victim) as a black box, and hence cannot distinguish between processes running inside the TEE or associate system calls with the processes from which they originate. The only information available to them is the list of syscalls invoked by the confidential container, namely what we use for our attack. To carry out the attack, the adversary first builds an Machine Learning (ML) model offline using information they gather from container images they own, or find in public repositories like Docker Hub [41]. As the most popular TEEs are open sourced, and the information about which TEE is used by each cloud provider is public, adversaries can easily recreate and study

Funded by Horizon Europe Framework Programme of the European Union

any of the expected environments. At runtime, the adversary captures system calls invoked by a confidential container (the victim). Then, the adversary uses the previously trained ML model to infer sensitive information about the containerised application such as type, and even version. This type of attack can be carried out by the cloud providers themselves, or by malicious tenants who manage to escape the container and gain control of the underlying host. In the latter case, after carrying out the attack, the adversary has the same privileges as the cloud provider on that host.

3.3.1.3 Technical challenges

Next, we detail the main technical challenges we had to overcome to perform fingerprinting attacks against applications running inside confidential containers (C1, C2 and C3) and to mitigate them (C4).

- C1. How to capture the application's system calls. Containers have their own namespace. Thus, the visibility of the processes and their numbering are different inside and outside the container. Also, containers run applications that can spawn multiple processes during their execution. Here, the challenge is how to identify and monitor both the parent process of the container as well as all children processes spawned within the container in order to capture all system calls invoked by the application. It should be noted that this must be done transparently from the host side, i.e., without having to modify the application, the container runtime or the TEE. In addition, the proposed method must be able to group the system calls of each container together since there can be more than one container running on the server at the same time.
- C2. How to distinguish between application system call patterns. Once the system calls have been captured, the second challenge is how to accurately associate the system call patterns to a Docker image (or application class). Both are very challenging tasks for a number of reasons. Firstly, Docker images are made up of layers, many of which are shared with many Docker images. In fact, it is common for software developers to download Docker images from public repositories, such as Docker Hub [41] or RedHat Quay [42] and use them as a starting point to build their own applications. Secondly, some container images require other containers to be created to work correctly, e.g., a database container to store the data they handle. For example, matomo – one of the leading open-source analytic platforms [43] – requires running a second container containing a MySQL database. These factors make it more difficult to distinguish container images solely from their system call patterns.
- C3. How to perform these attacks in practice. After clearing the first two challenges, adversaries must think about how best to carry out their attacks in practice. The third challenge is related to this – more specifically on how to perform these fingerprinting attacks in an efficient, effective and stealthy manner. On the one hand, due to the large number of containers running in the cloud simultaneously, it would be undesirable for adversaries to have to monitor each containerised application for a

Funded by Horizon Europe Framework Programme of the European Union

long time in order to carry out the attack successfully. It is important to note that the goal of adversaries is always to conduct attacks that cause significant damage, while minimizing the cost of mounting them. On the other hand, adversaries must develop their fingerprinting attacks by taking advantage of container information that is stable and that cannot be easily hidden by application owners. For example, adversaries could fingerprint containers based on the name of the processes created within the container, or the hashes of the container layers. However, application owners could easily hide this information and thwart these attacks, for example, by renaming processes so that their names do not leak information, or making very small changes to the container layers so that the resulting hashes are completely different.

- C4. How to protect against such fingerprinting attacks. To counter potential fingerprinting attacks, a viable defence strategy involves introducing controlled noise by injecting (fake) system calls during the container's execution. This generation of noise is non-trivial, for several reasons. First of all, generating Differential Privacy (DP) noise, for example using a Gaussian distribution, requires information about distribution of syscall to be available. This is information that is not generally available, but it is a strong requirement to be able to generate noise in a robust manner. This is also crucial to avoid naïve implementations that would mask the signal at the cost of significantly degrade system degradation, or by not providing a real protection in terms of indistinguishability of the executed container image. Secondly, DP assumes that the noise could be both positive and negative. This is not possible in the current scenario, as preventing the execution of syscalls would cause disruption in the running container. Third, the way system calls are generated will need to follow realistic patterns, which means the syscall added needs to follow the logic relationship between syscalls. Randomly-generated and inserted system calls could lead to unrealistic system call patterns that could open the door for attackers to obtain the real system call patterns from the unrealistic ones. For example, by observing a close before fsopen, or a munlock before a mlock2. Finally, the proposed countermeasure needs to be able to handle both long and short running applications. This means that the noise needs to be generated in an approximation of a continuous, streaming, fashion.

3.3.1.4 Methodology

To assess the feasibility of our attack, we conducted our study using a substantial number of official images retrieved from Docker Hub, currently the largest and most widely used public repository for Docker images. Our initial step involved the development of a custom web scraper, leveraging the Docker Hub API [44], to obtain essential information required for running Docker Hub images. We then employed a monitoring agent, which we designed, to capture system calls generated by the containerized application during the first two minutes following its startup. For each observed system call, we collected critical data, including the timestamp (indicating when the system call occurred), process ID (identifying the originating process), and event name (detailing the syscall's name). Subsequently, we ran each container

Funded by Horizon Europe Framework Programme of the European Union

multiple times while continuously gathering system call data generated by the applications running within them, thereby accumulating multiple traces for each container.

Following the collection of application system calls, we conducted further data processing to extract valuable fields suitable for deriving features. We proposed and analysed two types of syscall features. The first type involves reflecting the presence or absence of different syscall types within the container trace, which we denote as binary vectors. In the second type, we counted the occurrences of each syscall in the container trace, creating frequency vectors. Each Docker image is represented by a syscall vector (binary or frequency), accompanied by its class label. This syscall vector serves as the feature set for analysis by the machine learning model. Initially, we considered 353 features, corresponding to the number of system calls supported by the Linux kernel. However, not all features hold equal importance. To enhance the model's generalizability and reduce noise, we conducted a feature importance analysis to identify which features contribute significantly to the machine learning model.

Having extracted the necessary features from the collected data, the subsequent phase focused on assessing the feasibility of attacks designed to deduce the type of application running within a confidential container. This challenge was framed as a multi-class classification problem, where the adversary initially trains an offline machine learning model using their own Docker images, with each Docker image serving as a distinct class. We conducted this study using a 3-fold cross-validation approach and explored various hyperparameters and values across five distinct machine learning classifiers, including Random Forest, Neural Network, Support Vector Machine, Naive Bayes, and XGBoost. We evaluated the performance of each classifier by considering both types of features (binary vs. frequency). Although all models offer a fairly high accuracy, the Random Forest classifier we built provides the best performance.

To mitigate information leakage, we propose a countermeasure based on the principles behind Differential Privacy (DP) [45, 46] that involves carefully injecting fake syscalls during the container's execution in order to generate more uniform system call patterns, hiding the "identity" of the application executed within a TEE . The proposed mechanism involves an application running alongside the containerised one to generate a fictitious sequence of events to be captured by the adversary. Our technique leverages the fact that the adversary is not able to distinguish between the processes executed within a TEE. Note that this is not "free". Adding fictitious syscalls to a container running within a TEE has a performance overhead, i.e., it will introduce some form of delay because of the additional operations required to execute the added syscalls. Because of that, a key point in designing the defense mechanism is to minimize this overhead while providing sufficient privacy protection.

In the following sections, we begin by showcasing a series of experiments that demonstrate the viability of executing the proposed attack. Subsequently, we introduce the

Funded by Horizon Europe Framework Programme of the European Union

countermeasure we have developed to alleviate the information leakage stemming from the syscalls invoked by containerized applications.

3.3.1.5 ML Modelling of Individual Containers

Next, we detail the comprehensive set of experiments we conduct to assess various essential aspects, including the choice between binary and frequency system call vectors, the relationship between monitoring duration and ML accuracy, the influence of container execution parameters on the ML accuracy, and the robustness of container fingerprints when considering multiple versions of the containers

1. Using Binary VS. Frequency Syscall Vectors. The next step is to assess the suitability of the two types of features we use in our work. To that end, we train a Random Forest ML model and compare the accuracy it provides when the Docker images are executed without parameters and monitored for 2 minutes, first when the binary syscall vectors are applied (denoted as 'binary') then when the frequency syscall vectors are used (denoted as 'frequency'). Table 3.6 shows that with the frequency vector, the trained ML model achieves considerably higher accuracy.

TABLE 3.6: ACCURACY, PRECISION AND F1-SCORE OF OUR TRAINED RANDOM FOREST ML MODEL TO FINGERPRINT APPLICATIONS RUNNING INSIDE CONFIDENTIAL CONTAINERS. WE SHOW THE OBTAINED RESULTS WHEN RUNNING THE DOCKER IMAGES WITH AND WITHOUT PARAMETERS, AND WHEN USIN

	Random Forest (without parameters)			Random Forest (with parameters)		
	Accuracy	Precision	F1-score	Accuracy	Precision	F1-score
Binary syscall vectors	0.74	0.68	0.70	0.80	0.75	0.76
Frequency syscall vectors	**0.95**	**0.93**	**0.94**	**0.96**	**0.96**	**0.95**

2. Monitoring Time VS. ML Test Accuracy. We also carry out an experiment in which we analyse how different container monitoring times affect the ML model's accuracy for the two types of feature we consider. For this experiment, we leverage the timestamp value on each system call in order to split our 2-minute container traces into smaller traces, starting with just 10 seconds and adding 10 seconds each time until we reach 2 minutes. The results of the experiment, shown in Figure 3.15, demonstrate that in both cases the accuracy of the ML model evolves in a more or less linear way with respect to the containers' monitoring time. From the results obtained, it is also easy to see that the frequency syscall vectors provide much better results. Remarkably, the accuracy of the trained ML model when using the frequency syscall vectors and the containers are monitored for just 10 seconds is higher than that achieved after 2 minutes when using the binary system call vectors.

Funded by Horizon Europe Framework Programme of the European Union

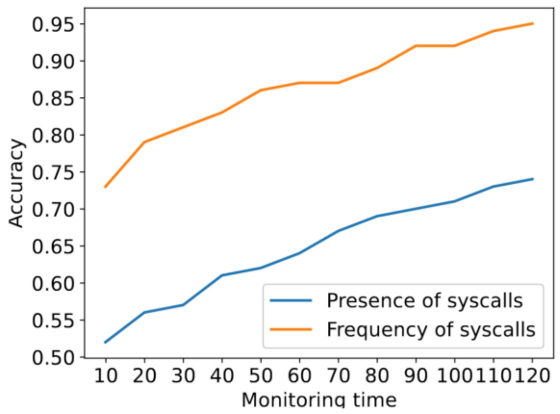


FIGURE 3.15: ACCURACY OF THE TRAINED ML MODEL FOR DIFFERENT CONTAINER MONITORING TIMES (IN SECONDS)

3. Running With VS. Without Parameters. Moving further, we investigate the possibility of fingerprinting attacks for the case where the containers are executed without parameters (i.e., by simply executing "Docker run container_name") as well as the case where the containers are run with their corresponding Docker parameters (e.g., with an open port to send/receive network traffic to/from users). With the former, we examine whether we can extract a unique container fingerprint from container logic that is always executed regardless of the Docker parameters the container runs with. Conversely, running containers with their parameters helps us quantify the effect of parameters in the container's fingerprint. To know which Docker parameters each container uses, we look at their descriptions in Docker Hub. In the case that more than one "Docker run" command is shown in the description, we take the one with the least number of parameters to consider what is probably the worst case for adversaries (our hypothesis is that the Docker parameters make the container's fingerprint more unique). Table 3.6 shows the accuracy, precision and F1-score of the Random Forest model when the Docker images are run with and without parameters, for the two types of features we consider (i.e., binary and frequency). We can see that in both cases it is slightly easier for adversaries to fingerprint applications running inside confidential containers when these are run with their corresponding Docker parameters, thus confirming our hypothesis

4. Running Different Application Versions. In addition to the previous experiments, we analyse the similarity of system call vectors across various versions of the same Docker image. To perform the experiment, we manually extract the three most recent tags based on their order of appearance in Docker Hub (or version information) from each of the 149 official Docker images in our dataset. Next, we run each Docker image with its respective tags for 2 minutes, while collecting the system calls invoked in each case. From these data, we proceed to create the corresponding system call vectors for each Docker image and tag combination. During this process, we find certain combinations of Docker images and tags that do not produce satisfactory results, leading us to exclude those containers from further analysis. After excluding the problematic containers, we obtained a new dataset with 129 Docker official images, and used these containers in our subsequent experiment.

Funded by Horizon Europe Framework Programme of the European Union

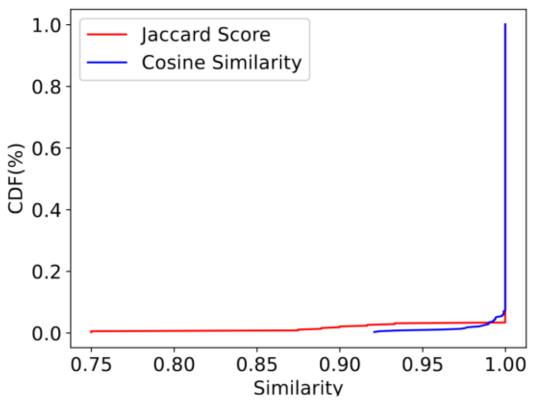


FIGURE 3.16: SIMILARITY BETWEEN DISTINCT VERSIONS OF DOCKER IMAGES

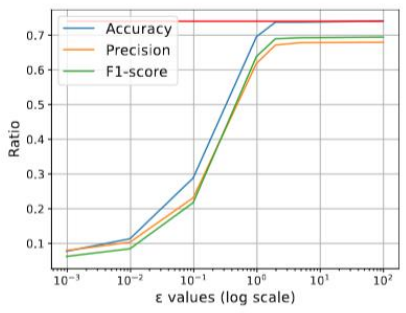
We conduct a careful assessment of the system call vectors for each pair of Docker images and its corresponding tags, considering both the binary and frequency system call vectors. (Note that the binary syscall vectors are used to compute the Jaccard score and that the Cosine similarity is computed using the binary syscall vectors). Figure 3.16 shows the Cumulative Distribution Function (CDF) of the similarity scores for multiple instances of the same container (each with a distinct tag). The Jaccard similarity reveals that a significant number of container instances exhibit a perfect match, as indicated by the abundance of similarity scores equal to 1. This suggests that, despite new versions appearing, containers tend to use a fairly well-defined set of system calls. Similarly, the Cosine similarity vector also shows a high degree of similarity between container instances, with the vast majority of scores being close to 1. Nevertheless, it is worth noting that some instances show slightly lower scores, indicating subtle variations in their system call patterns.

In general, both similarity measures reinforce the observation that different versions of the same container tend to possess very similar system call vectors, i.e., that the container fingerprint tends to be preserved. This means that samples collected from a container with an older version may be enough to identify instances of this same container running a newer version. This, on the one hand, can be positive since it would not be necessary for adversaries to retrain the originally created ML model using a dataset that includes new samples. However, we also see cases of images whose system call vectors can reveal the version of the container that is being used. In these cases, adversaries can extract a fingerprint that provides information not only what Docker image it is but also what version it is using. However, this would require adversaries to train the ML with samples from different versions of each of the containers in their dataset.

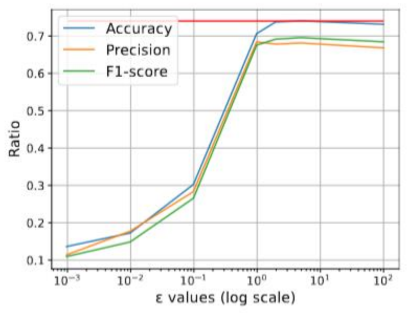
3.3.1.6 Proposed countermeasure

Generating a source of noise capable of injecting fictitious syscalls to hide the genuine system call pattern for a specific application presents key challenges. First, we can only add positive noise to establish indistinguishability among TEEs while preserving performance and functionality of the protected applications. Second, the execution of syscalls is not completely independent. Last but not least, the noise generator needs to operate over a fixed time window to be more generally applicable across containers with variable execution times. Choice of noise generation and addition mechanism are central to DP.

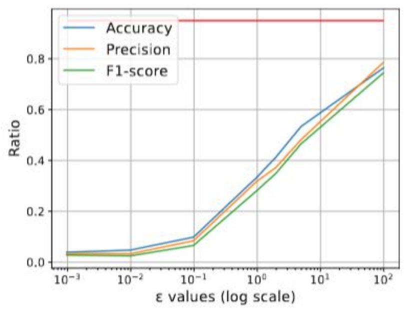
Funded by Horizon Europe Framework Programme of the European Union



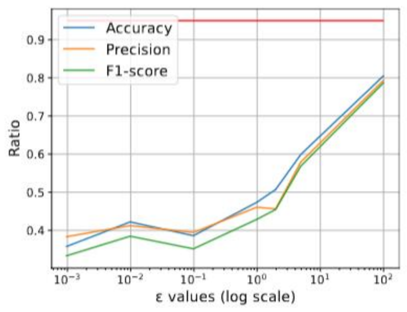
(a) Scenario 1: model trained on clean traces, binary features



(b) Scenario 2: model trained on noisy data, binary features



(c) Scenario 1: model trained on clean traces, frequency features



(d) Scenario 2: model trained on noisy data, frequency features

FIGURE 3.17: IMPACT OF THE DEFENCE STRATEGY

We performed several experiments to evaluate the effectiveness of the defence mechanism. Specifically, we tested the approach in two scenarios using the dataset collected for the attack. The first one, shown in Figures 3.17a and 3.17c, assumes that the adversary collects information about executions of container images without the defence mechanism enabled. Thus, the adversary is able to collect information and create a model for image classification not affected by the noise generation. The second scenario, shown in Figures 3.17b and 3.17d, assumes that the adversary is aware of the defence and of its impact on the data that can be observed from running containers, perhaps by executing container images in a TEE with the proposed defence mechanism enabled. In the latter case, the adversary trains the model to fingerprint the container images on noisy traces, hence allowing the model to be more robust to the fictitious syscalls added to the container.

We evaluate both scenarios using ϵ-values among the following values: 0.001, 0.01, 0.1, 1, 2, 5, 100. These values have been selected according to theoretical [47] and practical [48] best practices, thus, ranging from high privacy values (any value less than 1), to the ones that provide less noise to be added to the system (any value more than 1). The results are shown to be in line with our expectations. Comparing the plots of Figure 3.17 with the baseline from Table 3.6, we see that as more noise is added, smaller ϵ values, the model perform less accurately. Moreover, Figures 3.17b-3.17d show how training the model using a dataset that has been created observing containers protected by the defence mechanism provides no significant advantages to the adversary. Hence, the similarity between both scenarios demonstrate that knowledge of the defence mechanism configuration provides no advantage to the adversary.

Funded by Horizon Europe Framework Programme of the European Union

3.4 WORKLOAD PROTECTION

3.4.1 Threats and Intrusion Detection

Detecting threats and network intrusions within FLUIDOS-enabled federations necessitates careful consideration of two primary challenges:

1. Resource Cost Management: Managing the expenditure associated with the computational resources employed by an Intrusion Detection System (IDS) is paramount. Efficient resource allocation and utilization are critical to ensure cost-effectiveness while maintaining robust security.
2. Training data: Equally significant is the scarcity of training data that accurately represents the attack surface of the federation. This deficiency in representative data poses a significant obstacle in training effective intrusion detection models.

Within a FLUIDOS federation, an administrative domain might opt to lease computing resources from external domains for the execution of specific tasks. However, it is important to acknowledge that these resources come at a cost. Consequently, in the realm of security, an important challenge emerges: finding the optimal balance between the complexity of the detection algorithms that protect such tasks from cyber threats, which directly impact resource utilization, and the precision of threat detection.

On the other hand, in terms of safeguarding the FLUIDOS federation against novel and unidentified threats, it becomes imperative to establish a mechanism enabling a member to share newly discovered security incidents with other members without the need to share sensitive data (e.g., portions of the network traffic or logs of the computing nodes). To address these challenges, in this Section, we first present a methodology designed to evaluate the overall performance of the entire pipeline of an ML-based IDS. This methodology enables a deeper understanding of how various configurations and settings influence the balance between detection accuracy and operational efficiency.

Additionally, we introduce an enhanced version of Federated Learning, a technique for collaborative training ML models with privacy guarantees. The new approach, called FLAD for Federated Learning Approach to DDoS attack detection, has been specifically tailored to accommodate the unique constraints and demands of the cybersecurity domain. This adaptation is essential for optimizing the collaborative learning process within a federated cybersecurity context such as FLUIDOS.

3.4.1.1 A methodology for Online Performance Analysis of Network Intrusion Detection Systems

A FLUIDOS-enabled infrastructure defines a dynamic environment, in which it is possible to dynamically acquire and release resources from a node. As a result, a pay-per-use cost

Funded by Horizon Europe Framework Programme of the European Union

defined by the *Cost Manager* and *Resource Acquisition Manager* is assigned to the allocation of resources. In this context, even after the deployment of a user-specified service/application, the user can adjust the amount of resources reserved based on criteria such as the monetary budget. Therefore, both the performance and the workload of the service could change throughout its execution.

To harden the security of business applications running in FLUIDOS nodes, a user can couple the main service with an anomaly detection system that monitors the system to detect potential cyberattacks or malfunctions. However, the disposal of these algorithms requires computing power and memory, resulting in a higher overall monetary cost. Therefore, these algorithms need to be meticulously tuned based on the volume and level of detail of the data that should be gathered on the specific application, which is strictly influenced by the available budget and specifics of the underlying FLUIDOS node. Hence, two research questions arise:

1) Is it always necessary to use all available traffic features to achieve a satisfactory level of detection accuracy?
2) If we reduce the number of features to conserve computing resources (hence the cost of the service) in the FLUIDOS node, how would this impact the performance of the anomaly detection system, particularly in terms of accuracy and throughput?

To shed light on this multifaceted challenge, we propose a novel methodology for evaluating the performance of a Machine Learning-enabled Network Intrusion Detection System. ML-based NIDS are becoming more popular due to their effectiveness, flexibility, and high level of automation, but their “offline” assessment, which typically involves testing a trained model against a portion of historical data, is limited to mere classification of the flows. Instead, we introduce an "online" phase that simulates the complete process, from real-time feature extraction to network flow classification and filtering, leveraging different configurations.

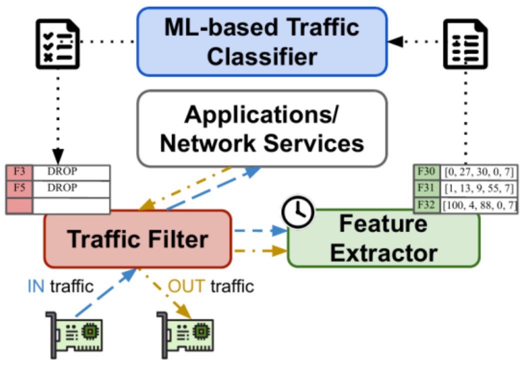


FIGURE 3.18: HIGH-LEVEL COMPONENTS AND INTERACTIONS

The architecture illustrated in Figure 3.18 offers an overview of the complete traffic processing pipeline of a user service (one or more applications) coupled with a NIDS. Initially, incoming network traffic entering the container and traffic generated by local applications

Funded by Horizon Europe Framework Programme of the European Union

and services undergo filtration by the *Traffic Filter*. This filter relies on a blacklist to block packets identified as malicious. Subsequently, the traffic proceeds to the *Feature Extractor*, where relevant attributes are extracted for use by the *Traffic Classifier*. These attributes are gathered within a defined observation time window, ensuring sufficient data is available for the classification of network traffic.

Regarding the Traffic Classifier, we operate under the assumption of employing an ML-based binary classification system, which categorises network traffic as either legitimate or malicious. The output from this classifier consists of a list of identifiers for traffic flows that are deemed malicious and need to be intercepted by the Traffic Filter. On the other hand, the union of the input layer and the feature extraction process defines the configuration space of the NIDS in our analysis.

To explore possible setups in terms of performance and resource consumption of the NIDS, we adopt the traffic representation framework introduced in our previous work, known as LUCID [49]. This framework organizes network traffic into flows, with each flow consisting of packets sharing common attributes, namely source and destination IP addresses, transport ports, and protocol. Each flow is presented as an array of features, which serves as input for the ML model during the classification process. In this array, rows represent packets in chronological order, and columns represent packet-level features like timestamps, IP Flags, and more. A visual representation of a flow is provided in Figure 3.19.

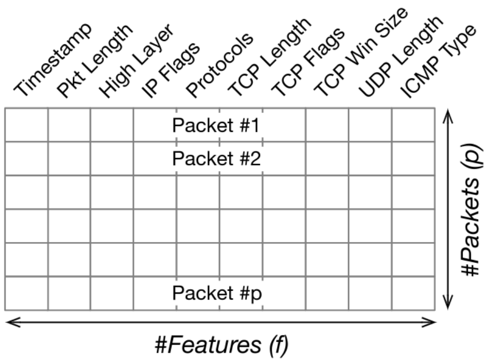


FIGURE 3.19: NETWORK TRAFFIC FEATURES REPRESENTATION

The assessment of the NIDS's performance in the FLUIDOS node involves iteratively reducing the two dimensions of the feature array. This reduction entails decreasing the number of features ***f*** extracted from each packet and limiting the maximum number of packets ***p*** collected for each flow. The fundamental premise here is that by extracting less information from network traffic, a user can reduce the computational demands on the CPU and memory resources of the NIDS. During each iteration, the ML model's input layer shape is adjusted to align with the current array configuration, and the least important packet features are removed using a greedy-based approach named Recursive Feature Elimination. Our proposed methodology assumes the availability of a labelled dataset containing pre-

Funded by Horizon Europe Framework Programme of the European Union

recorded traffic traces, encompassing both benign and malicious traffic flows. A portion of these traces is designated for constructing training and validation sets, while another portion is reserved for forming the test set and enabling online evaluations. The training and validation sets are exclusively used for training and validating the ML models and conducting feature ranking. Conversely, we assess the offline accuracy of the models using unseen data from the test set. As a reference dataset for the experiments, we used the Intrusion Detection Evaluation Dataset *CIC-IDS2017* dataset [50], which contains both legitimate and DDoS flows.

Moreover, we considered three distinct ML model architectures for the task of traffic classification: a Convolutional Neural Network (CNN), a Multi-Layer Perceptron (MLP), and a Recurrent Neural Network (RNN). All these models take as input a representation of a traffic flow, as previously described. They share a common final classification layer which returns the probability of a flow being classified as benign or malicious, rounded using a classification threshold of 0.5.

Notably, all three models demonstrate exceptional F1 scores on the test set, irrespective of the chosen combinations of ***p*** and ***f*** values, as detailed in Figure 3.20. Even when ***p***=*1* and ***f***=*1*, the accuracy consistently surpasses 0.8. It is worth noting that employing only four features results in achieving nearly maximum accuracy and minimal False Negative Rate (FNR) across all models, regardless of the ***p***.

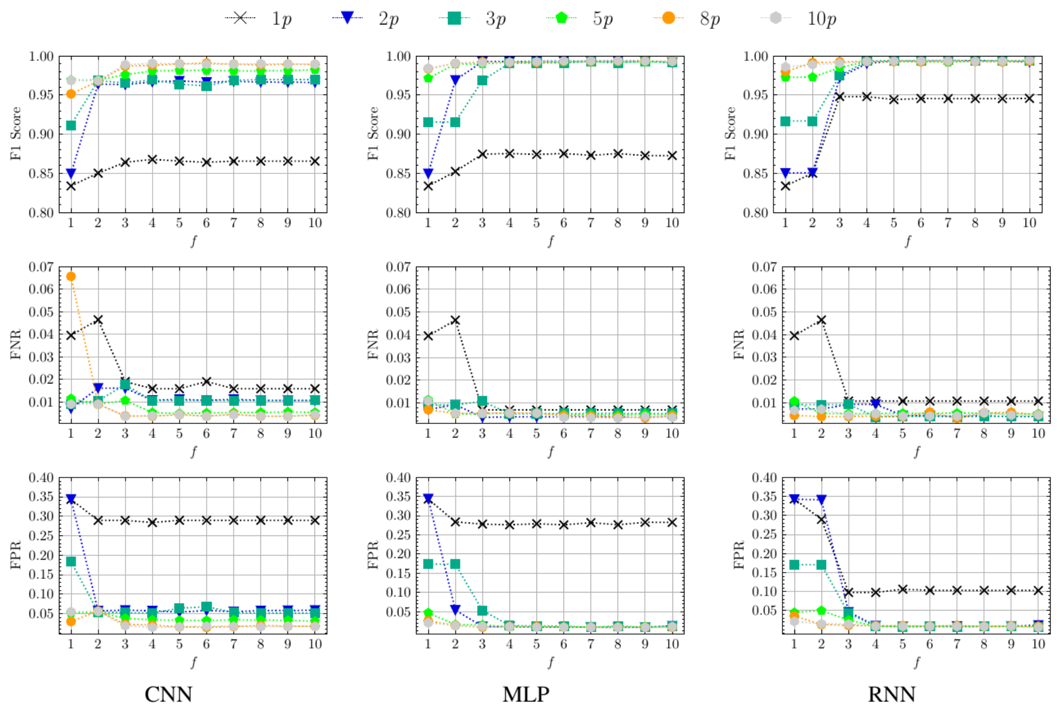


FIGURE 3.20: OFFLINE ACCURACY OF THE THREE ALGORITHMS WHEN FED WITH DIFFERENT FEATURES "F" AND PACKETS "P".

Funded by Horizon Europe Framework Programme of the European Union

To be precise, the RFE feature ranking algorithm consistently ranks four out of the following five features in the top four positions, albeit in varying orders: *Highest Layer*, *TCP Window*, *TCP Length*, *IP Length*, and *Protocols*. Our offline experimentation strongly indicates that extracting just five features from packets can yield remarkable accuracy while conserving computational resources.

To evaluate the overall pipeline's performance, we inject the NIDS with the test portion of the traffic traces. The end-to-end throughput of the NIDS is measured in terms of *packets/s* and *flows/s*, considering the time spent by the system on various tasks, including blacklist lookup, feature extraction, feature pre-processing, and traffic classification. In this context, we want to understand how the throughput is impacted by *p*, *f*, as well as by the computational complexity of the ML model being used for traffic classification.

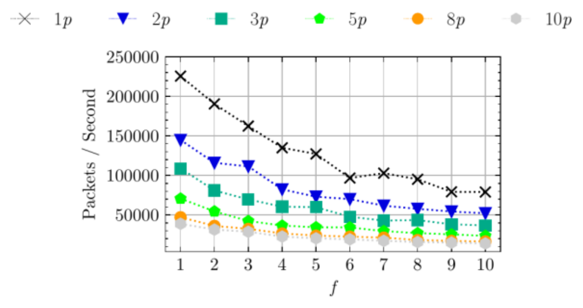


FIGURE 3.21: PACKET RATE OF THE CNN WHEN USING DIFFERENT COMBINATIONS OF FEATURES F AND PACKETS P.

Figure 3.21 displays the throughput of the CNN-based NIDS at varying values of *p* and *f* measured in *packets/s*. The throughput of the other two versions of the NIDS (MLP and RNN) follows similar trends. Overall, the *1p×1f* configuration with the CNN is the fastest, achieving around 20 500 *flows/s* and 225 500 *packets/s*, while the *10p×10f* configuration with the RNN is the slowest, with a throughput of approximately 1100 *flows/s* and 12 000 *packets/s*. These results reflect the findings of recent studies on the complexity of common ML architectures [51].

We also examine how well the NIDS updates the blacklist to block malicious traffic promptly in two setups of the CNN-based NIDS: 1) the *Time Window* approach, where features are collected by the Feature Extractor for a predefined time interval of 10 seconds before transmitting arrays to the classifier, and 2) an alternative method called *Early Mitigation*, implemented in the Feature Extractor, which sends a flow representation in array format to the classifier as soon as the array is filled, without waiting the end of the time window. Results are reported in Figure 3.22.

Funded by Horizon Europe Framework Programme of the European Union

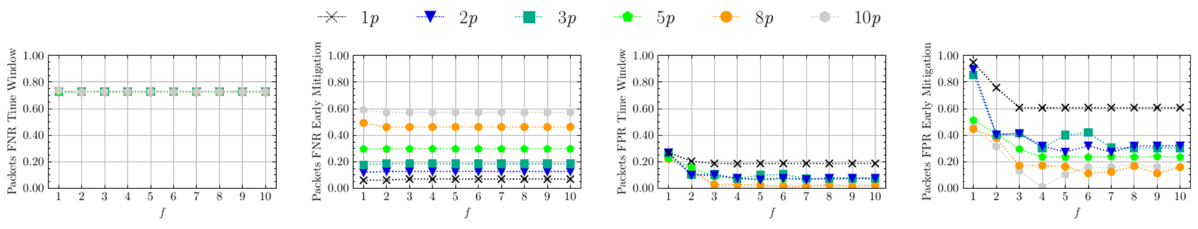


FIGURE 3.22: ONLINE ACCURACY OF THE CNN WITH DIFFERENT COMBINATIONS OF FEATURES F AND PACKETS P.

In cases of short flows (such as in CICIDS2017), around 73% of packets are not blocked by the Traffic Filter because the blacklist is not timely updated, as shown in the first plot. On the other hand, in the second plot, we notice a relevant improvement with the Early Mitigation strategy. Of course, with $p=1$ the FNR is very low because a flow is classified using only one packet and the blacklist is populated more frequently. On the other hand, using $p=1$ results in poor detection accuracy, with most of the flows classified as malicious (*1p* curve in the fourth plot). The plot depicting the FPR with the Time Window shows that the trend in the online test closely mirrors that of the offline test, while the Early Mitigation has a negative impact as the Traffic Filter initiates earlier blocking of misclassified benign flows, resulting in a higher FPR.

In conclusion, this activity outlines a novel methodology for evaluating the effectiveness of the NIDS feature extraction, detection, and mitigation pipeline. Through a sensitivity analysis, we provide insight into how different settings impact the balance between accuracy and efficiency, enabling better-informed decisions for optimizing the performance of the system and the potential costs for allocating resources in a FLUIDOS node. Moreover, the methodology outlines the importance of undergoing an online validation phase, whose results indicate that accuracy scores obtained from conventional offline tests do not always reflect the full impact of misclassifications and implementation decisions on the overall performance of the NIDS. Our analysis suggests that the potential adoption of a NIDS in a FLUIDOS-enabled computing environment can be effective, and tuning its configuration would help save computational resources (hence monetary budget) while providing a near-optimal detection performance.

3.4.1.2 Training a ML-based NIDS with distributed data

Federated Learning (FL) [52] is a recent approach for training Machine Learning (ML) algorithms with decentralized data. FL is a promising approach for enabling collaborative training and updating of ML models in a FLUIDOS-enabled computing environment, without the need to share private network data. This is important, as it allows multiple administrative domains to work together to improve the accuracy and security of their IDSs.

In this Section, we present a novel adaptive Federated Learning Approach to DDoS attack detection (FLAD), which enhances Federated Averaging (FedAvg), the algorithm behind the original FL algorithm, to make it suitable for network security problems.

FL relies on a set of participants called clients (the FLUIDOS administrative domains) that train the model on their local data, and on a central server that aggregates ANN model parameters collected from clients and distributes the aggregated model back to clients for

Funded by Horizon Europe Framework Programme of the European Union

further training sessions. This sequence of operations is executed multiple times (federated training rounds) with no exchange of clients' private training data, until a target convergence level is reached.

The application of FL in cyber security for intrusion detection has been explored in previous research [49, 50, 53]. However, previous works rely on FedAvg, the FL mechanism introduced by McMahan et al., which necessitates a representative test set available at the server side to control the training process. We argue that this approach poses a data privacy issue and may restrict the applicability of FL in scenarios where only a subset of data classes can be tested by the server. It is reasonable to assume that network data containing recent cyber incidents against one or more clients may include sensitive information that cannot be shared with the server for testing purposes. Consequently, in such cases, the server would not have the ability to assess the performance of the aggregated model using the latest attack traffic. Furthermore, achieving convergence in the FL process can present challenges due to several factors. These include the presence of non-independent and identically distributed (non-i.i.d.) data across clients, as well as unbalanced datasets, which are common in network anomaly detection. Slow convergence can hinder the ability to promptly update the IDS service in response to attacks targeted at specific clients within the federation. While some of these issues have been addressed to some extent in previous works, their effectiveness remains uncertain, as outlined in the subsequent sections.

To tackle the above challenges, we propose FLAD, in which the server verifies the classification accuracy of the global model on clients' validation sets with no exchange of training or validation data, granting that the model is learning from all clients' data and allowing it to implement an effective early-stopping regularisation strategy. FLAD is conceived to apply FL in the cybersecurity domain, where we assume that no attack data will be shared at any time between the FLUIDOS administrative domains and the server. We tackle the convergence of the federated learning process in the context of Distributed Denial of Service (DDoS) attack detection, with a focus on the trade-off between convergence time and accuracy of the merged model in segregating benign network traffic from a range of different DDoS attack types. We consider a dynamic scenario, where clients are targeted by zero-day DDoS attacks, and where the global model must be updated with new information as soon as possible to empower all participants with the latest detection features.

Problem formulation

McMahan et al. have evaluated the FedAvg algorithm for image classification and language modelling problems. However, we argue that it does not satisfy two basic requirements for effective DDoS attack detection:

1 Short convergence time to reach the target attack detection accuracy, especially in emergency threat situations in which the global model must be quickly distributed to clients upon retraining with recent DDoS attack information. Indeed, FedAvg assigns the same amount of computation to all the clients selected for a round of training, irrespective of the accuracy level reached by the global model on specific clients' data. This inefficient management can lead to long FL training sessions with no substantial gain in accuracy.

2 Accurate detection of all attack types in realistic conditions, where the detection system must learn from unbalanced and non-i.i.d. data obtained from heterogeneous DDoS

Funded by Horizon Europe Framework Programme of the European Union

attack types characterised by different traffic rates and feature distributions. The weighted average of FedAvg gives more importance to the weights of the clients with large local training sets, to the detriment of the smallest ones. We argue that this strategy could hinder FedAvg's ability to detect attacks characterised by out-of-distribution features that are available only in small local training sets.

Furthermore, it should be noted that FedAvg operates under the assumption that some test data is accessible at the server site to verify that a target accuracy of the global model is achieved and stop the training process. We argue that this assumption rarely holds in the cybersecurity domain. For instance, let us consider a scenario where one client contributes with updates related to zero-day attack traffic that is not public at training time. In this case, the only solution for the server to verify that the model has learned the new attack would be to use the client's test set. However, even if we discount the willingness of the client to provide such information, this would require data cleaning (anonymisation) from the client's sensitive information, with the risk of losing IP, transport and application layer features that could be critical for model validation.

Threat model

We consider a scenario in which the federation is composed of a set of clients (FLUIDOS administrative domains) that might belong to different organisations, plus an additional entity that manages the FL process (the central server). We assume that no one in the federation has the willingness/permission to share network traffic data with others. On the other hand, the federation's goal is to enhance the DDoS detection capabilities of each client' IDSs with attack profiles owned by other members.

In such a scenario, the clients are vulnerable to zero-day DDoS attacks at any given moment. To ensure the highest level of security, our system requires the global model to be updated promptly with the latest information, empowering all participants with the most recent detection features available. However, it is important to note that the central server may not always have access to network traffic profiles associated with these new and evolving threats. As a result, verifying the effectiveness of the global model in classifying such attacks becomes a challenge.

In this context, the adversary does not belong to the federation and does not have the knowledge to generate adversarial evasion attacks against the global ANN model [51]. However, it knows the IP addresses of the victims and how to generate DDoS attacks using spoofed network packets with the source IP address of the victims.

Methodology

The high-level idea behind FLAD is to involve in a training round only those clients that do not obtain sufficiently good results on their local validation sets with the current global model. For such clients, the amount of computation (number of training epochs and gradient descent steps/epoch) is determined based on their relative accuracy on their validation sets. Note that the accuracy score is computed by clients on their validation sets and communicated to the server upon request. Hence, no exchange of sensitive data between server and clients is involved. Compared to FedAvg, FLAD introduces a negligible traffic overhead between clients and server, without disclosing clients' sensitive data, even for testing purposes.

Funded by Horizon Europe
Framework Programme of the European Union

The details of FLAD are presented in Algorithms 1, 2 and 3.

The pseudocode in Algorithm 1 describes the main process executed by the server, which orchestrates the operations of the clients. The algorithm takes as input a global model (w_0) and the set of clients involved in the FL process (C). It runs indefinitely until convergence is reached, as controlled by parameter patience, which is the number of rounds to continue before exit if no progress is made. The federated learning starts with the initialisation of the variables that are used to record the best global model along the process (max accuracy score a_{max}) and to implement the early stopping strategy (counter sc keeps track of the rounds with no improvements in average accuracy score a^{μ}). At line 5 of Algorithm 1, the amount of computation for the clients is set to the maximum values of training epochs and MBGD steps. The loop at lines 8-10 triggers the ClientUpdate methods (Algorithm 3) for a subset of selected clients C_{t-1}. Note that at round t = 1, $C_{t-1} = C_0 = C$, i.e., the input set of clients (line 4).

Algorithm 1 Adaptive federated learning process.

```
Input: Global model ($w_0$), set of clients ($C$)
 1: procedure AdaptiveFederatedTraining
 2:     $a_{max} \leftarrow 0$                                      ▷ Max accuracy score
 3:     $sc \leftarrow 0$                                           ▷ Early stop counter
 4:     $C_0 \leftarrow C$
 5:     $c_e = e_{max}, c_s = s_{max}\ \forall c \in C_0$           ▷ Epochs and steps
 6:     $c \leftarrow$ InitClients$(w_0, c_e, c_s)\ \forall c \in C_0$
 7:     for round $t = 1, 2, 3, \ldots$ do                         ▷ Federated training loop
 8:         for all $c \in C_{t-1}$ do                              ▷ In parallel
 9:             $w_t^c \leftarrow$ ClientUpdate$(w_{t-1}, c_e, c_s)$
10:         end for
11:         $w_t = \frac{1}{|C|} \sum_{c=1}^{|C|} w_t^c$             ▷ Arithmetic mean
12:         $a^{\mu} \leftarrow [a^c]_{c \in C} \leftarrow$ SendModel$(w_t, C)$
13:         if $a^{\mu} > a_{max}$ then
14:             $\bar{w} \leftarrow w_t$                            ▷ Save best model
15:             $a_{max} \leftarrow a^{\mu}$                        ▷ Save max accuracy score
16:             $sc \leftarrow 0$                                   ▷ Reset early stop counter
17:         else
18:             $sc \leftarrow sc + 1$
19:         end if
20:         if $sc >$ PATIENCE then
21:             SendModel$(\bar{w}, C)$                             ▷ Send final model
22:             return                                              ▷ End of the process
23:         else
24:             $C_t \leftarrow$ SelectClients$(C, [a^c]_{c \in C}, a^{\mu})$
25:         end if
26:     end for
27: end procedure
```

At each round, the server computes the average of the parameters from all clients, regardless of whether they were involved in the previous round of training (line 11). The new global model is sent to all clients, which return the accuracy scores $[a_c]\ c \in C$ obtained on their local validation sets with the new global model (line 12). The server computes the mean accuracy score value a^{μ}, which is used to evaluate the progress of the federated training (lines 13-19). If $a^{\mu} > a_{max}$, the new global model is saved and the stopping counter sc is set to 0. Otherwise, sc is increased by one to record no improvements. When $sc >$ PATIENCE (in our experiments we set PATIENCE = 25 rounds), the process stops and the best model is sent to all the clients for integration in their IDSs (line 21). Otherwise, the server calls Algorithm 2 to determine which clients will participate in the next round and to assign the number of epochs and MBGD steps to each of them.

```
Algorithm 2 Select the clients for the next round of training.
Input: Clients ($C$), accuracy scores ($[a^c]_{c \in C}$), average accuracy score ($a^\mu$)
Output: List of selected clients ($C'$)
 1: procedure SELECTCLIENTS($C$, $[a^c]_{c \in C}$, $a^\mu$)
 2:     $C' \leftarrow \{c \in C \mid a^c \leq a^\mu\}$
 3:     $\underline{a} = min_{c \in C'}(a^c)$
 4:     $\overline{a} = max_{c \in C'}(a^c)$
 5:     for all $c \in C'$ do
 6:         $\sigma = \frac{\overline{a} - a^c}{\overline{a} - \underline{a}}$                ▷ Scaling factor
 7:         $c_e = e_{min} + (e_{max} - e_{min}) \cdot \sigma$
 8:         $c_s = s_{min} + (s_{max} - s_{min}) \cdot \sigma$
 9:     end for
10:     return $C'$
11: end procedure
```

Algorithm 2 starts with selecting the subset of clients C' that will execute the local training in the next round. C' is the set of $c \in C$ whose accuracy score a^c obtained on their local validation set is lower than the mean value a^μ (line 2). The number of epochs and steps assigned to each client $c \in C'$ depends on the value of a^c. The rationale is that the higher a^c, the lower the amount of computation needed from the client (thus, fewer epochs and MBGD steps/epoch, as explained at the beginning of this section). This is formalised in the equations within the loop at lines 5-9, where each client $c \in C'$ is assigned a minimum number of epochs/steps plus an additional amount that is inversely proportional to the accuracy score a^c. The scale factor σ ranges over $[0,1]$, assuming value 0 when $a^c = (a^c)$ (hence $c_e = e_{min}$ and $c_s = s_{min}$) and value 1 when $a^c = (a^c)$ (hence $c_e = e_{max}$ and $c_s = s_{max}$). Algorithm 2 returns the set of clients C' that will perform computation during the next round, each assigned with a specific number of epochs and MBGD steps.

The pseudo-code provided in Algorithm 3 outlines the local training procedure carried out by clients. This process starts from the weights and biases of the current global model w received from the server, and is executed for a number of epochs c_e and MBGD steps c_s assigned by the server. The first operation is the computation of the batch size c_b using c_s (line 4). It ensures that $c_b \geq 1$, for the cases in which the number of samples in the local training set is smaller than c_s. Once the batch size is computed, the algorithm continues with $c_e \cdot c_b$ steps of gradient descent (lines 7-11) and finally returns the updated model to the server.

Funded by Horizon Europe Framework Programme of the European Union

Algorithm 3 Local training procedure at client c.

```
Input: Global parameters $w$, epochs ($c_e$), MBGD steps ($c_s$)
Output: Updated parameters ($w$)
1: procedure ClientUpdate($w$, $c_e$, $c_s$)
2:     $X, y \leftarrow$ LoadDataset()
3:     if $c_s > 0$ then
4:         $c_b \leftarrow max(|X_{train}|/c_s, 1)$        ▷ Compute batch size
5:     end if
6:     $\mathcal{B} \leftarrow$ split $X_{train}$ into batches of size $c_b$
7:     for epoch $e$ from 1 to $c_e$ do
8:         for all batch $b \in \mathcal{B}$ do
9:             $w \leftarrow w - \eta \nabla L(w, b)$
10:        end for
11:    end for
12:    return $w$        ▷ Return updated parameters to server
13: end procedure
```

Experimental evaluation

We compare FLAD against FedAvg, the original FL algorithm proposed by McMahan et al. [52] and against a recent FL-based solution for DDoS attack detection called FLDDoS [53]. Both FedAvg and FLDDoS adopt a randomised client selection strategy, while also employing fixed batch sizes and local training epochs across all clients. The goal of this evaluation is to expose the limitations of such design choices in a cybersecurity scenario, where the server does not possess a test set (for the reasons discussed earlier in this paper) to measure the performance of the global model on different attack types.

We train the global model with FLAD until convergence, i.e., waiting for patience=25 rounds with no progress in the average F1 Score across the clients. Following this, we evaluate the performance of the original FedAvg algorithm and FLDDoS by subjecting them to the same number of training rounds as FLAD.

We perform the convergence analysis in a worst-case scenario, i.e., with a federation of 13 clients and a one-to-one mapping between clients and DDoS attack types. We replicate the same experiment by employing a federation of 50 clients, each containing two attack types in their local dataset. The latter settings align with Lv et al.'s evaluation of FLDDoS in their study [53].

Each experiment is repeated 10 times and the average metrics are reported in this section. As TensorFlow relies on a pseudo-random number generator to initialise the global model, and both FedAvg and FLDDoS perform a random selection of clients at each FL round, each experiment is initiated with a unique random seed to ensure diverse testing conditions.

The results obtained in the worst-case scenario are summarised in Table 3.7, which reports average metrics across the 10 iterations of this experiment. As introduced in Section 7.3, FLAD is configured with adaptive tuning of epochs and MBGD steps of local training (E=A,S=A). FLAD is compared against two configurations of FedAvg, with E=1 and E=5 epochs/round of local training, and against FLDDoS configured with E=10.

Funded by Horizon Europe Framework Programme of the European Union

TABLE 3.7: AVERAGE METRICS OVER 10 EXPERIMENTS IN THE 13-CLIENT SCENARIO, WITH ONE-TO-ONE CLIENTS/ATTACKS MAPPING.

Metric	FLAD E=A,S=A	FedAvg E=1,B=50	FedAvg E=5,B=50	FLDDoS E=10,B=100
FL Rounds	68	68	68	68
Round Time (sec)	9.08	34.19	179.48	205.39
Total Time (sec)	617	2325	12205	13967
F1 Score	0.9667	0.8577	0.9157	0.9091
F1 StdDev	0.0369	0.2714	0.1597	0.1605
F1 WebDDoS	0.8990	0.0815	0.8148	0.7376
F1 Syn	0.9877	0.4563	0.4613	0.5094

The table shows the advantages of FLAD over FedAvg and FLDDoS: higher accuracy within a shorter time frame. These improvements can be attributed to the dynamic client selection strategy implemented by FLAD. At each round of the federated training process, clients are chosen based on the performance of the current aggregated model on their local datasets. Consequently, FLAD prioritizes attacks that are more challenging to learn, specifically the o.o.d. attacks WebDDoS and Syn Flood. The clients with these attacks are selected more frequently for local training, with an average of approximately 44 and 46 rounds respectively out of a total of 68 rounds, compared to an average of around 18 rounds for the clients with the other attacks.

In contrast, both FedAvg and FLDDoS rely on random client selection, where each client is involved in approximately 77% of the training rounds (around 52 rounds on average out of a total of 68 rounds), considering the client fraction F = 0.8 used in our experiments. This results in longer rounds due to the frequent inclusion of clients with large local datasets, even when their contribution is not essential for improving the accuracy of the aggregated model. Furthermore, FLAD dynamically tunes the amount of computation assigned to the selected clients at each round of training, resulting in a significant reduction in the average local training time per round. Comparatively, FLAD achieves an average local training time of around 9 seconds per round, while the two configurations of FedAvg require 34 and 179 seconds per round, respectively. The FLDDoS configuration, on the other hand, takes more than 200 seconds per round. Consequently, FLAD's adaptive allocation strategy not only decreases the per-round training time but also effectively reduces the overall duration of the federated training process.

It is also worth noting that the overall performance of FLDDoS and FedAvg with E=5 is similar, as they assign approximately the same amount of computation to the clients. Specifically, FLDDoS is configured with E=10 epochs local training (as in the original paper by Lv et al. [53]), while FedAvg uses E=5 epochs with twice the number of MBGD steps/epochs due to the smaller batch size. Additionally, the strategy employed by FLDDoS to handle non-i.i.d. data does not yield significant improvements compared to FedAvg in

Funded by Horizon Europe Framework Programme of the European Union

our evaluation scenario. In fact, the local models maintained by clients with o.o.d. data, such as WebDDoS and Syn Flood attack traffic, do not contribute to improving the accuracy of the global model on such attacks but, instead, increase the total training time.

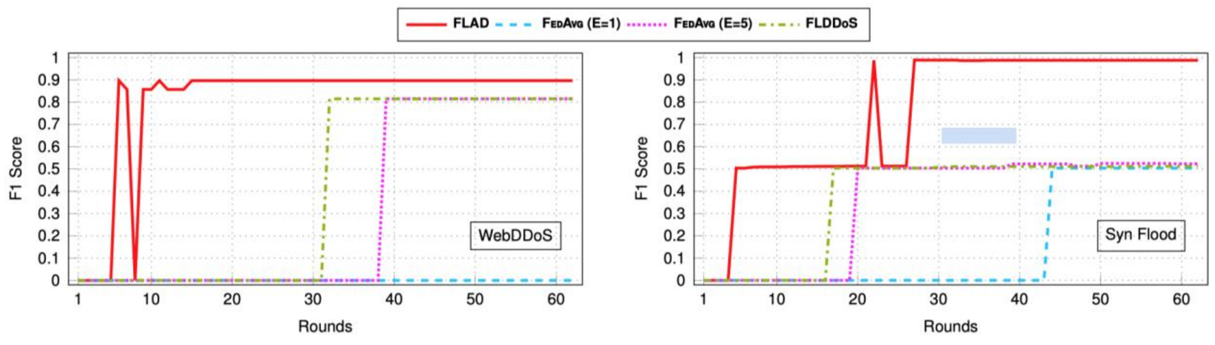


FIGURE 3.23: PERFORMANCE ON OUT-OF-DISTRIBUTION DATA, NAMELY ON THE WEB-DDOS AND SYN FLOOD ATTACKS.

This is clearly shown in Figure 3.23, which shows the performance trend of FLAD, FedAvg and FLDDoS on the WebDDoS and Syn Flood attack traffic during the first of the 10 iterations of the experiment. The two plots clearly demonstrate that FLAD achieves faster learning and higher accuracy for both of these attacks, while FLDDoS and FedAvg with E=5 exhibit similar trends.

In these plots, we can also observe the adaptive mechanism of FLAD in action. Once the global model has learnt a client's data profile, FLAD excludes such client from the next round of federated training. In the case of clients with o.o.d. data, such as WebDDoS and Syn attack data, this behaviour might cause the model to forget what it has previously learnt on such attacks, as can be seen on both plots in the figure. However, this prompts FLAD to reintegrate such clients in the subsequent rounds of the training process, ultimately leading to global convergence.

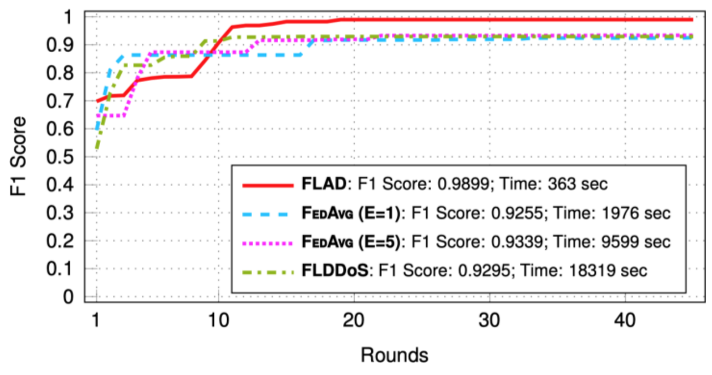


FIGURE 3.24: PERFORMANCE COMPARISON ON A FEDERATION OF 50 CLIENTS, TWO ATTACKS/CLIENT.

Finally, Figure 3.24 presents the performance trend of FLAD, FedAvg, and FLDDoS in a scenario with 50 clients, where each client's dataset consists of two attacks along with benign traffic. Also in this case, we repeated the experiment 10 times. However, due to limited space, only the test results of the first iteration are displayed. Nevertheless, a similar pattern was observed throughout the remaining nine iterations. The local datasets of the 50 clients

Funded by Horizon Europe Framework Programme of the European Union

are generated by randomly combining pairs of the 13 datasets. For each test iteration, a different random seed is utilised to generate a distinct federation.

It is worth mentioning that, given these test settings, the two o.o.d. attacks are present in the datasets of multiple clients. Due to this, we observe a higher average F1 score on the clients' validation sets compared to the 13-client scenario (approximately 0.99 with FLAD and 0.94 with FLDDoS and the two configurations of FedAvg) and lower standard deviation across the 50 local validation sets (approximately 0.01 with FLAD and 0.1 with FLDDoS and FedAvg). These results demonstrate the advantages of the adaptive mechanism implemented by FLAD, even in scenarios with more uniformly distributed and less imbalanced data.

Discussion

FLAD has been validated using an unbalanced dataset of non-i.i.d. DDoS attacks. However, we see the potential of the FLAD's approach in other cybersecurity applications that are relevant to a FLUIDOS-enabled federation. For instance, we believe that FLAD can be effectively used to train generic network intrusion detection systems (IDS) in the presence of unknown network attack types, and can be adapted to train host-based IDSs in contexts where zero-day vulnerabilities are exploited to compromise computing infrastructure.

3.4.2 Cyber Deception

The highly-distributed nature of the FLUIDOS environment can undermine the security posture of the multitude of cloud-native applications running across different FLUIDOS nodes. Due to the variety of technologies employed to implement the FLUIDOS software stack, malicious actors can leverage misconfiguration and/or vulnerabilities among the different modules of the FLUIDOS architectures to penetrate the microservices of a service provider in order to steal data and/or cause service disruption. As a consequence, traditional cyber defense mechanisms, such as intrusion detection systems (IDSs) and firewalls, may not be sufficient to ensure a dependable security perimeter, thus increasing the risk of insider threats.

Cyber deception can be a valuable tool to enhance the security of the FLUIDOS ecosystem. This proactive defense strategy consists in the allocation of decoys, resembling legitimate system components, within the defender infrastructure to lure malicious actors into interacting with them. Decoys must appear realistic to engage the attacker and increase the likelihood of interaction. In this context, containerization techniques and cloud-native technologies like Kubernetes offer the means to craft high-fidelity and high-interaction decoys by replicating microservices from deployed applications with minimal complexity. By seamlessly integrating this deception mechanism into the existing microservices, defenders can better identify and mitigate attackers lateral movements between microservices, all while gathering valuable insights of their tactics, techniques, and procedures (TTPs).

According to this intuition, we propose an algorithmic solution to address the decoy allocation problem (in other words, which and how many microservices should be cloned)

Funded by Horizon Europe Framework Programme of the European Union

and an implementation solution to integrate such deception strategy within a real cloud-native application deployment. In detail, we resume our contribution as follows:

- We design a scalable resource-aware decoy placement strategy that selects which legitimate microservices of an application should be cloned as decoys in order to maximize the chance of the attacker interaction. Our solution can be integrated within the production environment of cloud-native applications thanks to the flexibility offered by container-based technologies.
- We propose a first implementation of such a deception mechanism by developing a controller, extending the default K8s API, that automatizes the decoy creation process as well as it integrates the instantiated decoys within the legitimate application data flow.

3.4.2.1 State of the art

Modern cyber deception has evolved towards the design of flexible and adaptive deception techniques tailored to the considered defense scenario in order to maximize the deception effectiveness and reduce the management cost complexity. A more detailed discussion of cyber deception principles and related challenges can be found in [54]. The authors of [55] develop a sandbox network that misdirects attacks from the production network towards deceptive applications that are one-to-one copies of legitimate ones. We instead limit the orchestration complexity by directly disseminating deceptive microservices replicas within the production infrastructure according to a given resource budget. In [56], the authors formulate a honeypot placement scheme given a limited budget of resources. Differently, we aggregate the exploit difficulty of the various microservices in order to place decoys on critical microservices whose violation would allow the attacker to spread more effectively among the infrastructure. The authors of [57, 58, 59] leverages game theory and deep reinforcement learning approaches to adapt the decoy allocation according to the attacker activity. Beside the omission of the decoy resource consumption, these suffer from a high computational complexity as the training procedure requires many iterations to converge toward a stable allocation policy. Differently, we tackle the design of a resource-aware decoy allocation scheme using an optimization-based approach that allows to extend the solution for a high number active microservices by leveraging a low-complexity heuristic.

3.4.2.2 System Model

We discuss the system model that we consider to design the decoy allocation strategy. We provide a comprehensive representation of the main system features in Figure 3.25.

Funded by Horizon Europe Framework Programme of the European Union

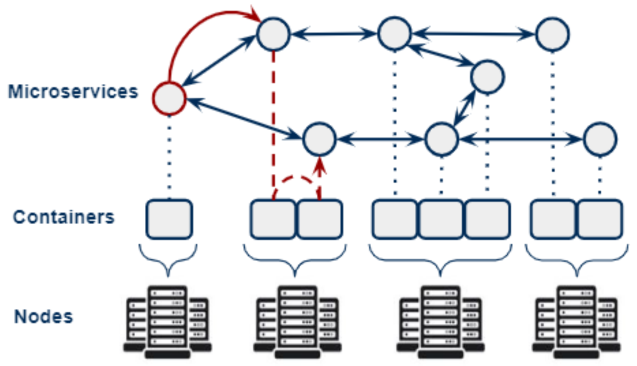


FIGURE 3.25: OVERVIEW OF THE CONSIDERED SYSTEM MODEL. MICROSERVICES CAN BE VIOLATED BY MEANS OF REMOTE CODE EXECUTION TECHNIQUES (SOLID RED ARROW) OR BY CONTAINER ESCAPE TECHNIQUES (DASHED RED ARROW).

3.4.2.3 Decoy configuration and placement model

We assume that a service provider, referred to as the defender, deploys a cluster of M microservices over N nodes. Each microservices requires an amount of $\bar{r}_m^{(cpu)}$ CPU cycles and $\bar{r}_m^{(ram)}$ memory. To enhance the security posture of its applications, the defender employs microservices as decoys in order to intercept ongoing cyber-attacks targeting the deployed microservices. Each decoy can be considered as a clone of production microservices replicating the related functionalities as well as the interfaces towards adjacent communicating microservices. We indicate the number of decoys cloning microservice m as x_m. Every decoy is configured to provide the following features:

- Attack detection reliability: any data traffic intercepted by a decoy is considered as malicious since the legal data traffic is exclusively forwarded to production microservices.
- High interactivity: each decoy reacts to the attacker input like a production microservice. However, any data extracted by the attacker is fake and its content is configured to resemble the structure of production data
- TTPs monitoring: each decoy implementation is augmented by monitoring functionalities that allow the defender to gather cyber threat intelligence information on the attacker TTPs employed to violate the decoy.

3.4.2.4 Attacker Model

We consider an attack scenario where a malicious user, referred to as the attacker, has managed to bypass the defender's security measures (such as IDSs and firewalls) and has gained unauthorized access to its cloud services by compromising one of the deployed microservices with the intent to steal important data.

Funded by Horizon Europe Framework Programme of the European Union

The attacker utilizes any compromised microservice as a foothold to further violate other deployed microservices In detail, the attacker lateral movements can be executed by means of two techniques:

- Remote Code Execution (RCE): the attacker can violate a microservice by injecting malicious code throughout its interfaces. The compromised microservice is used as a foothold to violate other communicating microservices.
- Container Escape: the attacker exploits any non-root privilege configuration available in each container to interact with other containers that do not belong to the same name-space. In other words, we assume that the attacker can break the microservices logical isolation in order to compromise any microservice deployed on the same node.

Regardless of the employed technique, we assume that the attacker cannot accomplish a privilege escalation to gain access to the whole cluster (i.e. the full set of nodes running the containerized microservices). This attack scenario would bypass any deployed defense mechanism and it is mostly due to a poor configuration of the system privilege levels performed by the defender, which is beyond the deception scope.

3.4.2.5 Threat Model

We model the potential threats of the considered scenario using graph theory in order to better tailor the design of an effective decoy allocation strategy. In particular, we introduce the attack graph (AG) associated to the current microservice deployment configuration as a directed acyclic graph *G=(V,E,W)* where:

- *V* is the set of vertices corresponding to the active microservices and decoys.
- *G* is the set of edges that express whether an attacker can move laterally from one microservice to another microservice by either leveraging RCE or container-escape techniques.
- *W* is the weight associated with each in-ward edge and reflects the vulnerability level of the corresponding microservice (the lower the value, the higher is the vulnerability level). We formalized this quantity by leveraging Exploitability Metrics (EM) and Exploit Code Maturity (ECM) indicators.

In this context, we formally define an *attack path (AP)* between a source vertex s (i.e. the attacker entry-point) and target vertex t (i.e. the target microservice whose violation makes it possible for the attacker to access some organization assets) as the shortest path in AG. Note that this scenario represents the worst-case attack scenario where a malicious user employs the most efficient sequence of techniques to penetrate the system.

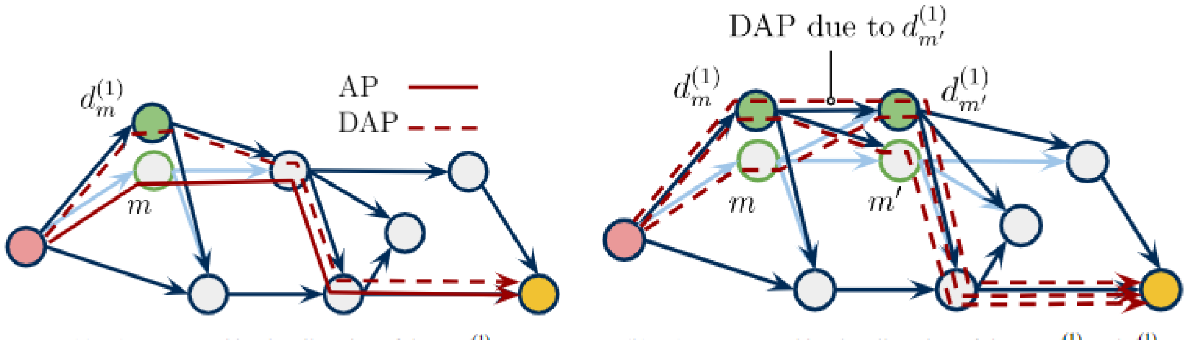
Furthermore, if an AP contains at least one decoy between v and t (i.e. one of the vertices along the path represents a decoy) we denote this path as a *deceptive attack path (DAP)*.

Funded by Horizon Europe Framework Programme of the European Union

From the above definitions, we remark that a DAP is also an AP, hence we can generally refer to any path in the AG as an AP if needed.

3.4.2.6 Problem definition

The amount of DAPs in AG measures the deception quality of the decoy placement configuration since the higher is this value, the higher is the number of attacks that are likely to be intercepted by the decoys. Therefore, an effective decoy allocation maximizes the number of generated DAPs between any vertex pair in AG according to the resource availability.



(a) DAP generated by the allocation of decoy $d_m^{(1)}$. (b) DAPs generated by the allocation of decoys $d_m^{(1)}$ and $d_{m'}^{(1)}$.

FIGURE 3.26: EXAMPLE OF DAPS GENERATED BY A DIFFERENT NUMBER OF DECOYS ALONG A SPECIFIC AP IN AG. THE RED AND YELLOW MICROSERVICES REPRESENT THE AP SOURCE AND TARGET, RESPECTIVELY.

We illustrate this intuition in Fig. 3.26, where each decoy generates a number of DAPs equals to the number of APs traversing the cloned microservice. According to this dynamic, the allocation of decoys on microservices sharing multiple APs can produce a high number of DAPs and thus increases the decoys likelihood to misdirect and intercept possible attacks. Following this idea, we propose an optimization problem that maximizes the number of generated DAPs by computing a suitable decoy allocation. Moreover, to overcome the prohibitive computational complexity of such formulation, we also propose a heuristic algorithm to approximate the optimal solution. We present the mathematical details of both schemes in the Appendix B section.

3.4.2.7 Performance evaluation

We evaluate the performance of the presented deception mechanism under different configurations of deployment sizes ranging from 100 to 500 microservices with a decoy resource ratio of 0.3. We randomly sampled 100 configurations of synthetically microservice architectures and we plotted the average results within the 95% confidence intervals for each considered simulation instance. We compare the performance of the *Optimal* decoy allocation defined in equations (1)-(6) of Appendix B, which provides the performance

Funded by Horizon Europe Framework Programme of the European Union

upper-bound of the considered metrics, and the related _Heuristic_ allocation presented in Algorithm 1 with the following three schemes:

- _Linear_: his scheme provides a decoy allocation that maximizes the number of DAPs without accounting for the additional paths introduced by the decoys. We assess the performance of this approach in order to analyse the performance gain provided by the optimal solution which accounts for every DAPs introduced by each decoy.
- _Sidecar_: this scheme is a simple decoy placement strategy that allocates decoys on the most vulnerable assets.
- _Random_: this scheme randomly allocates the decoys and it is used as a performance lower bound.

Metrics

We define the following metrics to compare the security performance achieved by the aforementioned decoy allocation schemes:

- _Decoy interaction probability_: this is the probability that an attacker interacts with a decoy when moving laterally between the microservices composing an AP. In other words, this is the probability that an attacker follows a DAP in order to reach its target. We define this metric in order to provide a more practical insight about the deception performance of the considered schemes as it measures the decoys likelihood to intercept attacks.
- _Average number of decoy per AP_: this metric measures the average ratio of decoys over the number of legitimate microservices contained in each AP. We define this indicator to evaluate the efficiency of the various schemes in condensing the allocation of decoys on microservices that are traversed by an AP. Intuitively, the higher is this value, the higher is the frequency that an attacker can interact with more than one decoy before reaching its target.

Results

In Figure 3.27 we present the decoy interaction probability. The optimal solution achieves the best performance as it distributes the decoys more effectively compared to other schemes thanks to the exact computation of the DAPs.

The heuristic scheme provides some notable optimality gap since its greedy approach prioritizes the allocation of decoys along the same AP instead of diversifying the allocation on other APs. However, it still generates a higher number of DAPs compared to the linear scheme, which underestimates the number of generated DAPs as it neglects the impact of the allocated decoy on the AG topology. The sidecar is outperformed by all schemes since it does not consider the sequence of microservices that must be violated by the attacker in order to reach the target.

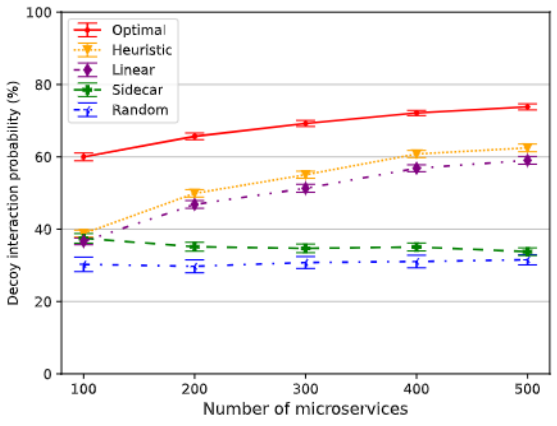


FIGURE 3.27: DECOY INTERACTION PROBABILITY

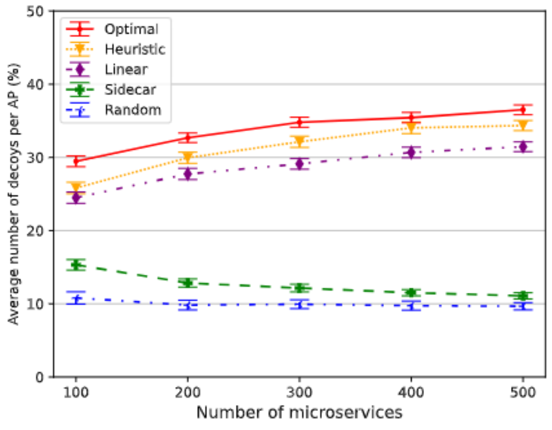


FIGURE 3.28: AVERAGE NUMBER OF DECOYS PER AP

In Fig. 3.28 we show the average number of decoys per AP. In general, the discussion of the previous plot also applies for this scenario. The optimal scheme ensures the highest number of decoys per AP as it places the decoys along microservices contained in multiple APs. The heuristic scheme achieves similar performance and provides better gain compared to the linear and sidecar schemes. In particular, the greedy nature of the heuristic scheme encourages the allocation of decoys on microservices within the same AP. This strategy generates a high amount of DAPs at each new algorithm iteration thanks to the already allocated decoys along that AP in the previous steps. In Figure 3.29 we show the total number of allocated decoys. The number of decoys increases as more computing nodes are activated to accommodate the required microservices. Generally, excluding the random scheme which is resource-agnostic, the various schemes roughly deploy the same number of decoys in a given microservice deployment configuration. This behaviour confirms that the performance gain provided by the optimal and heuristic schemes derive from a decoy allocation that is tailored to the vulnerabilities of the current microservice deployment and it is not due to a higher number of allocated decoys that artificially inflates the number of generated DAPs. In Figure 3.30 we show the convergence performance of the considered schemes. The non-linearity of equation (1) of Appendix B makes the optimal scheme unsuitable for large-scale microservice deployments that need to be frequently updated due to the high convergence time. Conversely, the heuristic decoy allocation requires a considerably lower computational complexity, thus it is preferable in the aforementioned scenarios. Moreover, the proposed heuristic also outperforms the linear approach, which suffers from a similar trend as the optimal scheme when the number of microservices is higher than 300. The sidecar scheme is characterized by a very low complexity due to the simplicity of its approach, which however achieves poor results in terms of DAP generation as previously discussed. As a consequence, the optimal and heuristic schemes provide the most effective solutions to ensure an effective deception strategy and a scalable decoy placement configuration, respectively.

Funded by Horizon Europe Framework Programme of the European Union

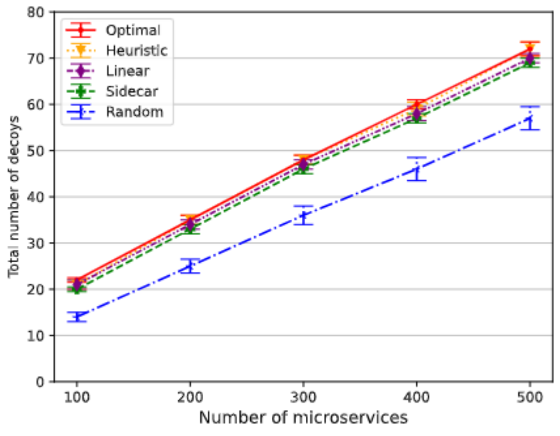


FIGURE 3.30: TOTAL NUMBER OF ALLOCATED DECOYS

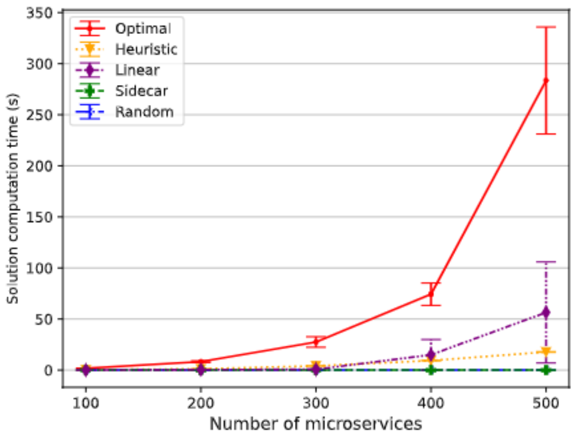


FIGURE 3.29: COMPUTATIONAL COMPLEXITY

3.4.2.8 System implementation approach

As anticipated above, a first implementation of the deception mechanism has been developed and it will be tested experimentally in the next reporting period. During the design and implementation phases we took into consideration modularity and extensibility, for this reason two main components with different responsibilities are proposed:

Decepto

Decepto is the component responsible to automatize the decoy creation process integrating decoys within the legitimate application data flow. Moreover it offers notification and monitoring mechanisms to identify the behaviours of an attacker. It targets Kubernetes environments by extending its default API using CRD (Custom Resource Definitions). In more detail it offers the following main features:

- Cloning of a generic microservice into a decoy: the ability to clone a microservice at Pod level taking into consideration the resource-aware algorithm directives. The new decoy Pod is instrumented to control alerting and monitoring features.
- Isolating communication flows across the application microservices: the ability to programmatically control the communications flows across legitimate microservices and/or decoys. Implementation through activation/deactivation of proper network rules and service discovery entries.
- Monitoring the adversaries behaviours: the ability to collect all relevant data in order to identify as much as possible the attackers' behaviour patterns. Collects system-calls, cluster audits, application logs and microservices in/out traffic.
- Alerting when a decoy receives unwanted traffic: the ability to discover potential malicious communications and notify them to start other relevant actions. A background process listens in promiscuous mode to the connections to the decoy which should never receive incoming traffic.

Funded by Horizon Europe Framework Programme of the European Union

Decepto software artifact and relative documentation are available at the following Git repository: https://gitlab.fbk.eu/cyber-deception/decepto.

Decoy placer

Decoy placer is the component responsible for providing a given placement strategy. It is designed on top of a standard interface that receives inputs, such as the application on which it should compute the given strategy and returns as output the list of microservices that should be cloned as decoys. This design makes it an extremely extensible and scalable component, indeed it comes with four strategy implementations: the first three are used for testing purposes while the latter is the implementation of the strategy described above.

Decepto and Decoy placer processes communicate through shared and public interfaces and data models, publicly available in the above repositories, that permit anyone to extend or customize the placement strategy.

3.4.2.9 Design

Decepto Custom Resource Definition

In order to manage the decoys of a given application, Decepto needs a way to represent it by means of a set of microservices and dataflows. Microservices are either legitimate services of the application or clones of them while dataflows represent the allowed communications across microservices which could be legitimate when used by users of the application which follow the application logics or illegitimate when used by an attacker and potentially does not follow the application logics.

We define the model of a generic application, which is also used to extend the default k8s API via a CRD, using the Application Graph that is depicted in detail on Figure 3.31.

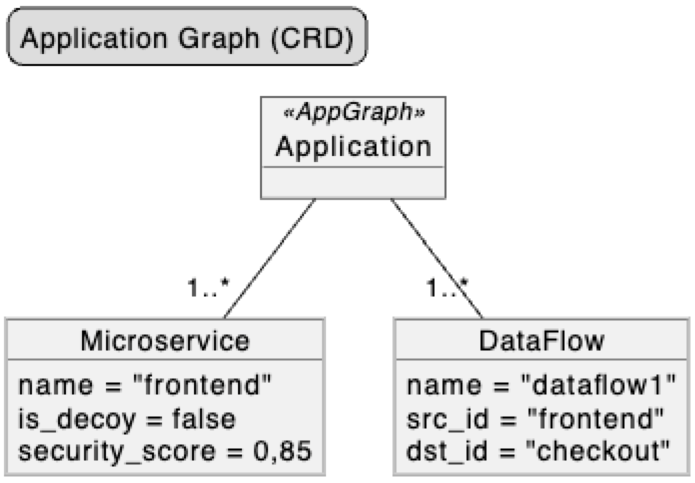


FIGURE 3.31: APPLICATION GRAPH (CRD)

Funded by Horizon Europe Framework Programme of the European Union

System components and modules

The implementation of the system relies on two main components: the Decoy placer and Decepto. While the first is essentially a collection of algorithms that implement different strategies on top of a standard interface, the second is made up of 6 modules, each of which offers certain functions to the overall system. Figure 3.32 depicts all the modules and how they interact with each other.

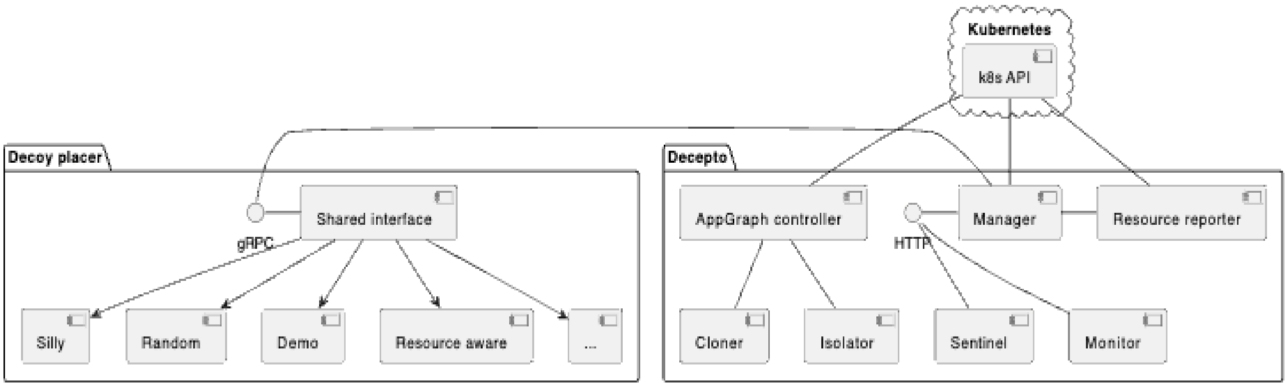


FIGURE 3.32: SYSTEM COMPONENTS AND MODULES

3.4.2.10 Decepto module implementation

We propose hereunder a deep dive on the implementation details for every module of the system, highlighting main features and default behaviours.

Manager

It is a non-terminating control-loop with various key responsibilities, indeed it performs a set of actions at a regular intervals and listens for system events via HTTP API.

This component collects infrastructure and workload information at a regular interval by querying the Resource reporter and sends it to the decoy-placer in order to get instructions for any new decoy to install/delete in/from the system. Then it programmatically modifies the AppGraph resource by updating its desired state through the Kubernetes API.

This component exposes an HTTP interface used to receive updated information from the monitor and from the sentinel components. For instance it is notified with an alert when a decoy is contacted because a potential breach is discovered.

Finally, in the upcoming releases, the manager will be responsible for analysing logs, metrics and traces in order to better study lateral movements and reconstruct the behaviour of the attacker.

Funded by Horizon Europe Framework Programme of the European Union

AppGraph controller

AppGraph controller is a Kubernetes controller inspired from the Kubernetes Sample Controller project which is responsible for managing CRUD operations on the AppGraph CRD. As any standard Kubernetes controller does, it controls the desired and actual state of the AppGraph resources which describes the graph of all microservices, decoys and their communication.

It monitors the actual state of the managed applications in the cluster, by constantly looking at its decoys as well as the communications across them and intercepts any inconsistency with the desired state specified in the CRD. As soon as the CRD is modified by a user or by the Decepto manager, it reacts by invoking relevant action in the system, for example activating the Cloner or the Isolator relevant actions.

Resource reporter

As its name suggests, the Resource reporter component is used to report the status of the cluster in terms of Infrastructure: detailed information about Kubernetes node resource usage and Workload: detailed information on microservices and data flows for any activated AppGraph, hence it relies on the data model presented in the previous sections but performing filtering operations across all the Pods running in the cluster to select only those which are part of any application controller by Decepto. To complete this mapping operation between the CRD and the application workload it uses two labels: *appgraph* and *app*.

Cloner

This component is responsible for creating decoys starting from legitimate microservices by installing in the system a slightly modified copy of the legitimate Pod.

The cloning process involves the following steps:

- Create a new Pod by cloning the original Pod and instrumenting it with monitor and sentinel sidecar containers
- For every cloned Pod, it resolves and fixes some dependencies the microservice has with other resources in the system. On the service discovery, for example it:
    - Creates new services by copying from those pointing to the original Pod, allowing the decoy to be reachable and discoverable
    - Removing the value of every environment variables used to contact other microservices, denying the discovery of other legitimate components
- Install the decoy in the cluster and properly modifies the Network Policies by making it available in the application network domain

Funded by Horizon Europe Framework Programme of the European Union

Isolator

This component grants isolation at application level on the communications among the application microservices and it is able to control it in a programmatic way by means of i) rules that allow or deny traffic between the application microservices depending on the topology of the communication graph the system wants to achieve and by means of ii) proper addition/removal of k8s Service resources in the k8s service discovery.

For instance, at first it allows only the legitimate traffic between the microservices of the application, later on, when a decoy has been placed, it also allows the traffic towards it, meaning that all the containers in the same k8s namespace will be able to reach and discover the newly installed decoys.

In the final version it should grant the following:

- From legitimate microservices to legitimate microservices:
    - All communications are possible and discoverable
    - All other communications (e.g.: to microservices part of other AppGraph) are denied by default
    - Service discovery of allowed destinations is possible
- From legitimate microservice to decoys:
    - A communication is allowed if it:
        - Is initiated by a legitimate micro-service
        - Respect the original legitimate flow specified in the AppGraph CRD
    - Communications from a given legitimate microservice to its decoy/clone are denied
    - Service discovery of allowed destinations is possible
- From decoys to legitimate microservices:
    - All communications are denied if initiated from the decoy
    - Service discovery is denied
- From decoys to decoys:
    - A communication is allowed if it:
        - Respect the original legitimate flow specified in the AppGraph CRD
        - Service discovery of allowed destinations is possible

Sentinel

Sentinel is a tiny and simple service which is injected as a sidecar into any decoys created by Decepto with the sole objective to raise an alert if any unwanted communication reaches it. Of course the decoys should not receive any traffic by its nature so it considers possibly harmful any connection that reaches any decoy Pod. Once discovered, Sentinel collects useful information such as the source IP, the timestamp and other packet low level details,

Funded by Horizon Europe Framework Programme of the European Union

which are crafted into a message and sent to the Decepto manager via the HTTP API notifying a possible lateral movement in action.

Monitor

The monitor is responsible for collecting traces about all the relevant communications across applications components and decoys with the main objective to study and reconstruct the behaviour of an attacker interacting with the deceptive environment.

This component will be implemented with a client/server architecture, hence the client part is injected as a sidecar into the decoy and it is responsible for collecting all incoming and outgoing communication, while the server is installed in the system backend as its purpose is not only to collect but also to analyse traces, like application log streams, container system-calls, k8s audit logs and node system logs, looking for correlations.
In the first Decepto release the monitor relies on tcpdump to collect network traffic only at client side, but later on it could be enriched with the integration of Sysdig and/or Prometheus.

3.4.2.11 System phases and life-cycle

We identified three main phases in which actors interact in different ways with the overall system. The first two are characterized by the fact that an attack is not yet discovered while in the former a possible threat has been identified by the system. These phases are described below and respective diagrams are presented in Appendix B.

Setup phase

This first phase collects preparatory steps the developer must perform in order to first make its application compliant with the Decepto system then to activate the Decepto features for a given application. In this phase are also included the interactions between components during the initial configuration of the system. Steps can be summarized as follow:

- Developer defines the Application Graph data model for a given application and writes it in a yaml format by respecting its CRD definition.
- Developer modifies the manifest of the given application in order to include the two labels used by Decepto to perform the mapping between the CRD and the application workload: *appgraph* and *app*.
- Developer installs a given application in the cluster following the deployment method of his choice.
- Developer inputs the AppGraph custom resource in the cluster, thus activating Decepto for a target application
- A set of Network Policy rules is set up by the isolator in the cluster to permit only legitimate application flows.

Funded by Horizon Europe Framework Programme of the European Union

Cloning phase

This phase aims at summarizing the honeypot setup by cloning microservices under the directive of the decoy placement strategy. Following steps are included in this phase:

- At a regular interval Decepto polls the Resource reporter in order to retrieve application and cluster information.
- Decepto manager reports Workload and Infrastructure to the Decoy placer asking for decoy placements
- Decoy placer returns a list of instructions for cloning microservices into decoys and Decepto updates the AppGraph CRD.
- If desired state is different with respect to the actual state, the AppGraph controller takes corrective actions by querying the cloner and the isolator
- If needed, cloner start the cloning process for the given microservices
- If needed, isolator add or update relevant Network Policies

Manage threat phase

In this phase we describe the interactions between components after the honeypot setup and when a potential threat has been discovered in the system. Following steps are included in this phase:

- Sentinel sidecar monitor the decoy for incoming connections
- If unwanted lateral movement is intercepted, it alerts Decepto for a possible threat
- Decepto intercept the alert and start the monitoring process which permits to grab more information of the attacker and study his behaviour

Funded by Horizon Europe
Framework Programme of the European Union

4 DISSEMINATION ACTIVITIES

This section highlights our efforts to share our research within the scientific community. Through conferences, papers, and proceedings, we contribute to the collective knowledge in the field of secure computing in FLUIDOS.

In this regard, the following scientific dissemination activities were performed by the partners.

4.1 CONFERENCES

4.1.1 The 5th International Workshop on Cyber-Security in Software-defined and Virtualized Infrastructures (SecSoft)

5th International Workshop on Cyber-Security in Software-defined and Virtualized Infrastructures (SecSoft) is a joint initiative from EU-funded projects (including FLUIDOS, PALANTIR, SIFIS-Home, ELECTRON, RIGOUROUS, and CATRIN) with WP5 members in the steering and organising committees, as well as in the TPC, and co-located with the 9th IEEE International Conference on Network Softwarisation.

This edition of the workshop has welcomed contributions from members of the projects within the organization, as well as from contributors and practitioners working on the theme of "Security, Safety, Trust, and Privacy support in virtualized environments". The project FLUIDOS and its security-related technical challenges were presented and discussed with active audience participation. Additionally, two papers from the project were presented.

4.1.2 The 3rd Eclipse Security, AI, Architecture and Modelling Conference on Cloud to Edge Continuum

Polito participated to the 3rd Eclipse Security, AI, Architecture and Modelling Conference on Cloud to Edge Continuum, held in Ludwigsburg (Germany), October 17, 2023 with a presentation entitled "*Enabling Compute and Data Sovereignty with Infrastructure-Level Data Spaces*".

The Liqo Protected Borders Extension introduces infrastructure-level data spaces, facilitating secure data exchange and resource sharing while aligning with the data gravity concept. It automatically enforces privacy, security, and access policies, enabling data providers to share data confidently. The International Data Spaces (IDS) Connector is a vital component for secure data exchange, handling entity authentication and policy enforcement. It also acts as an application-level gateway, offering uniform access to data and ensuring security

Funded by Horizon Europe Framework Programme of the European Union

measures. However, this architecture lacks support for computing components, does not address data gravity concerns, and requires publicly reachable endpoints. To address these issues, the proposed architecture can be integrated with application-level data spaces, providing flexibility and scalability.

4.2PAPERS AND PROCEEDINGS

In this section, we describe the various activities we have been involved in sharing our research and findings. This includes our participation in conferences, where we have both presented our work and helped in organizing these events. We also list the papers we have authored and submitted to conferences and journals, some of which have been published while others are still being reviewed.

4.2.1 Conference and journal papers published

- (TID) E. Marin, D. Perino, R Di Pietro, "Serverless computing: a security perspective", Journal of Cloud Computing, 11, 69 (2022), doi: https://doi.org/10.1186/s13677-022-00347-w.
  In this article we review the current serverless architectures, abstract and categorise their founding principles, and provide an in-depth security analysis. In particular, we: show the security shortcomings of the analysed serverless architectural paradigms; point to possible countermeasures; and, highlight several research directions for practitioners, Industry, and Academia.

- (FBK) S. Magnani, R. Doriguzzi-Corin and D. Siracusa, "Enhancing Network Intrusion Detection: An Online Methodology for Performance Analysis," 2023 IEEE 9th International Conference on Network Softwarization (NetSoft), Madrid, Spain, 2023, pp. 510-515, doi: 10.1109/NetSoft57336.2023.10175465.
  This paper proposes a methodology for evaluating the effectiveness of a Network Intrusion Detection System (NIDS) by placing the model evaluation test alongside an online test that simulates the entire monitoring-detection-mitigation pipeline. Besides that, the paper shows that in resource-constrained environment (like the cloud-to-edge continuum proposed in FLUIDOS), it is possible to reduce the amount of data monitored and fed to the anomaly detection module without severely affecting its accuracy and with a moderate impact on the mitigation capabilities of the system.
- (UMU) J. M. B. Murcia, J. F. P. Zarca, A. M. Zarca and A. Skármeta, "By-default Security Orchestration on distributed Edge/Cloud Computing Framework," 2023 IEEE 9th International Conference on Network Softwarization (NetSoft), Madrid, Spain, 2023, pp. 504-509, doi: 10.1109/NetSoft57336.2023.10175478.

Funded by Horizon Europe Framework Programme of the European Union

This paper provides a by-default security orchestrator approach to mitigate the above mentioned challenges in distributed edge/cloud computing frameworks. We use an Intent-based/policy-based orchestration paradigm for dealing with heterogeneity, allowing users to request service deployments securely without requiring knowledge about the underlying distributed infrastructure. By-default security orchestration will decide how to provide the requested services, ensuring that they are compliant with the security requirements provided by the user and the ones gathered by the system, locally and from reliable external sources. We provide design and use-cases based workflows for managing by-default security orchestration in proactive and reactive ways. In the future, it is expected to perform the implementation and validation of the proposed approach inside the scope of the FLUIDOS EU project.

4.2.2 Conference and journal papers under review

- (TID) E. Marin, N. Kourtellis, D. Perino, S. Braghin, A. Rawat, N. Holohan, "Detecting and Mitigating Information Leakage at the Container-Kernel Interface", *under review at ASIACCS 2024*.
  In this paper, we introduce a novel side-channel attack targeting confidential computing, that exploits information leakage in the interactions between TEE-backed containers and the host OS kernel. By observing system calls, adversaries can infer previously unknown sensitive information about containerised applications (e.g., the exact images, their version, or their class), that can allow them to conduct more efficient, effective and stealthy attacks. We demonstrate the effectiveness of the attack and assess its feasibility for weak adversaries with limited monitoring time. To mitigate information leakage, we propose a countermeasure based on the principles behind differential privacy that involves carefully injecting fake syscalls during the container's execution in order to generate more uniform system call patterns.
- (TID) H. Kang, E. Marin, M. You, J. Kim, D. Perino, and S. Shin,. "BeaCon: Automatic Container Policy Generation using Environment-aware Dynamic Analysis", *under review at EuroSys 2024*.
  This paper presents BeaCon, a novel approach for the automated generation of Docker container security policies that are adjustable based on the level of security application owners wish to have. Unlike previous works, we develop BeaCon based on dynamic analysis which applies realistic environments to trigger container execution paths that would otherwise not be visible during the container's profiling phase. In addition, we propose the use of a security and functionality score that determines the importance of each system call and capability to the security of the host OS kernel and the functionality of the containerized application, respectively. From these scores, BeaCon gives application owners the ability to fine-tune the policies depending on the requirements of their application. We implement a full-fledged prototype of

Funded by Horizon Europe
Framework Programme of the European Union

BeaCon utilizing eBPF kernel technology and conduct an extensive set of evaluations to demonstrate its efficiency in mitigating the limitations of previous studies. Finally, we present two widely-known, serious security vulnerabilities that can reduce risk if policies are generated using BeaCon.

- (FBK) R. Doriguzzi-Corin, D. Siracusa, "FLAD: Adaptive Federated Learning for DDoS Attack Detection", under review at Elsevier's Computers & Security journal.
  The Federated Averaging algorithm at the core of the FL concept requires the availability of test data to control the FL process. Although this might be feasible in some domains, test network traffic of newly discovered attacks cannot be always shared without disclosing sensitive information. This is specifically the case of FLUIDOS. In this paper, we address the convergence of the FL process in dynamic cybersecurity scenarios, where the trained model must be frequently updated with new recent attack profiles to empower all members of the federation with the latest detection features. To this aim, we propose FLAD (adaptive Federated Learning Approach to DDoS attack detection), an FL solution for cybersecurity applications based on an adaptive mechanism that orchestrates the FL process by dynamically assigning more computation to those members whose attacks profiles are harder to learn, without the need of sharing any test data to monitor the performance of the trained model.

Funded by Horizon Europe Framework Programme of the European Union

5 CONCLUSION AND NEXT STEPS

In this report, we have investigated the security considerations within the FLUIDOS project. We began by gathering security requirements from FLUIDOS' various components and partners, ensuring alignment with project use cases. We also conducted a high-level threat analysis to identify vulnerabilities within the FLUIDOS architecture. This process revealed potential security threats, underscoring the importance of robust security measures.

In response to the identified threats, we propose several mitigation services and mechanisms showcased during the different phases, from the discovery of peering candidates to resource acquisition and utilization. Our work has encompassed trust establishment, resource segregation and control, system attestation, workload confidentiality, and advanced threat detection and mitigation.

Note that this work will remain dynamic to ensure that we effectively address evolving security challenges. We plan to update the threat analysis to align with FLUIDOS's Work Packages (WPs) and incorporate feedback from open calls.

Our ongoing research and development efforts will focus on key topics from the first reporting period, including authentication, authorization, system attestation, privacy and confidentiality preserving, border protection, and the provision of cyber deception as a service.

In addition to that, we also plan to address some threats that were not considered so far, by assessing trustworthiness of repositories from which images are being deployed in FLUIDOS providers' nodes, as well as by investigating potential attacks that a malicious actor could perform against the orchestrator.

Finally, a selected set of the proposed solutions will be validated through the use cases proposed in FLUIDOS.

Funded by Horizon Europe Framework Programme of the European Union

REFERENCES

[1] Kubernetes. "Kubernetes." Accessed October 2023. https://kubernetes.io/.
[2] Liqo. "Liqo." Accessed October 2023. https://liqo.io/.
[3] Smart Grid Security, Recommendations for Europe and Member States, ENISA, 2012
[4] Guidelines for Smart Grid Cybersecurity, NIST, published September 25, 2014.
[5] OWASP Top Ten. Accessed October 2023. https://owasp.org/www-project-top-ten/
[6] Decentralized Identifiers (DIDs) v1.0, W3C Recommendation. https://www.w3.org/TR/did-core/
[7] Verifiable Credentials Data Model v1.1, W3C Recommendation. https://www.w3.org/TR/vc-data-model
[8] SPIFFE, Universal identity control plane for distributed systems, the Cloud Native Computing Foundation, Accessed October 2023, https://spiffe.io/
[9] ARM TrustZone Technology for Security in Mobile Devices, A. Prakash, V. Kumar, Apress, 2018, ISBN-13: 978-1484238910
[10] PARSEC, Platform AbstRaction for SECurity. Accessed October 2023, https://parsec.community/
[11] Henk Birkholz et al. Remote Attestation Procedures Architecture. Internet-Draft draft-ietf-rats-architecture-22. Work in Progress. Internet Engineering Task Force,. 57 pp. URL: https://datatracker.ietf.org/doc/draftietf- rats-architecture/22/.
[12] William A. Johnson, Sheikh Ghafoor, and Stacy Prowell. “A Taxonomy and Review of Remote Attestation Schemes in Embedded Systems”. In: IEEE Access 9 (2021), pp. 142390–142410. DOI: 10.1109/ACCESS.2021.3119220
[13] AMD SEV-SNP rust utils and primitives, Accessed October 2023. https://github.com/LIT-Protocol/sev-snp-utils
[14] Easy Tokio Rustls. Accessed October 2023. https://github.com/KennethWilke/easy-tokio-rustls
[15] Security Mode: traffic segregation, issue #1695. Available at https://github.com/liqotech/liqo/pull/1695 .
[16] 2017. epanaroma: Escape Docker Container Using waitid() | CVE-2017-5123 | Twistlock. https://www.epanorama.net/blog/2018/01/05/escapedocker-container-using-waitid-cve-2017-5123-twistlock-19/
[17]2023. Docker run reference. https://docs.docker.com/engine/reference/run/
[18]Anil Kurmus, Reinhard Tartler, Daniela Dorneanu, Bernhard Heinloth, Valentin Rothberg, Andreas Ruprecht, Wolfgang Schröder-Preikschat, Daniel Lohmann, and Rüdiger Kapitza. 2013. Attack Surface Metrics and Automated Compile-Time OS Kernel Tailoring.. In Network and Distributed System Security Symposium (NDSS). San Diego, California.
[19]Murugiah Souppaya Karen Scarfone and John Morello. 2017. Application Container Security Guide. https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-190.pdf.
[20]Abusing DAC_OVERRIDE capability. https://attackdefense.com/challengedetailsnoauth?cid=1459.
[21]Abusing SYS_MODULE capability. https://attackdefense.com/challengedetailsnoauth?cid=1199.
[22]Peiyu Liu, Shouling Ji, Lirong Fu, Kangjie Lu, Xuhong Zhang, Wei-Han Lee, Tao Lu, Wenzhi Chen, and Raheem Beyah. 2020. Understanding the Security Risks of Docker Hub. In European Symposium on Research in Computer Security (ESORICS). Guildford, UK, 257–276.
[23]Rui Shu, Xiaohui Gu, and William Enck. 2017. A Study of Security Vulnerabilities on Docker Hub. In ACM on Conference on Data and Application Security and Privacy (CODASPY). Scottsdale Arizona USA, 269–280.

Funded by Horizon Europe Framework Programme of the European Union

[24] Xin Lin, Lingguang Lei, Yuewu Wang, Jiwu Jing, Kun Sun, and QuanZhou. 2018. A Measurement Study on Linux Container Security: Attacks and Countermeasures. In Annual Computer Security Applications Conference (ACSAC). San Juan, Puerto Rico, USA, 418–429

[25] 2018. 10+ top open-source tools for Docker security. https://techbeacon.com/security/10-top-open-source-tools-docker-security

[26] Paul. Jaccard. 1912. The distribution of the flora in the alpine zone. In New phytologist 11.2. 37–50.

[27] Lingguang Lei, Jianhua Sun, Kun Sun, Chris Shenefiel, Rui Ma, Yuewu Wang, and Qi Li. 2017. SPEAKER: Split-phase execution of application containers. In International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment (DIMVA). Bonn, Germany, 230–251

[28] "Confidential computing: an AWS perspective," https://aws.amazon.com/en/blogs/security/confidential-computing-an-aws-perspective/, 2021.

[29] "Confidential computing in Google," https://cloud.google.com/confidentialcomputing?hl=en, 2023

[30] platform for Azure confidential computing," https://azuremarketplace.microsoft.com/en-us/marketplace/apps/scontainug1595751515785.scone?tab=overview, 2023

[31] S. Tople, K. Grover, S. Shinde, R. Bhagwan, and R. Ramjee, "Privado: Practical and Secure DNN Inference," CoRR, 2018. [Online]. Available: http://arxiv.org/abs/1810.00602

[32] M. Seo, J. Kim, E. Marin, M. You, T. Park, S. Lee, S. Shin, and J. Kim, "Heimdallr: Fingerprinting SD-WAN Control-Plane Architecture via Encrypted Control Traffic," in Annual Computer Security Applications Conference (ACSAC), 2022, p. 949–963

[33] P. Kocher, J. Horn, A. Fogh, D. Genkin, D. Gruss, W. Haas, M. Hamburg, M. Lipp, S. Mangard, T. Prescher, M. Schwarz, and Y. Yarom, "Spectre attacks: Exploiting speculative execution," Commun. ACM, p. 93–101, 2020

[34] M. Lipp, M. Schwarz, D. Gruss, T. Prescher, W. Haas, A. Fogh, J. Horn, S. Mangard, P. Kocher, D. Genkin, Y. Yarom, and M. Hamburg, "Meltdown: Reading Kernel Memory from User Space," in USENIX Security Symposium (USENIX), 2018, pp. 973–990

[35] "What does it imply to disable syscalls in Intel SGX?" https://stackoverflow.com/questions/28114746/what-does-it-implies-to-disable-syscall-in-intel-sgx, 2023

[36] S. Arnautov, B. Trach, F. Gregor, T. Knauth, A. Martin, C. Priebe, J. Lind, D. Muthukumaran, D. O'Keeffe, M. L. Stillwell, D. Goltzsche, D. Eyers, R. Kapitza, P. Pietzuch, and C. Fetzer, "SCONE: Secure linux containers with intel SGX," in USENIX Symposium on Operating Systems Design and Implementation (OSDI 16), 2016, pp. 689–703

[37] V. Costan and S. Devadas, "Intel SGX Explained," Cryptology ePrint Archive, Paper 2016/086, 2016. [Online]. Available: https://eprint.iacr.org/2016/086

[38] "A remote code execution issue was discovered in MariaDB 10.2," https://www.cvedetails.com/cve/CVE-2021-27928/, 2021.

[39] "Vulnerability to create arbitrary configurations and bypass certain protection mechanisms," https://www.cvedetails.com/cve/CVE-2016-6662/, 2016

[40] P. Maene, J. Götzfried, R. de Clercq, T. Müller, F. Freiling, and I. Verbauwhede, "Hardware-based trusted computing architectures for isolation and attestation," IEEE Transactions on Computers, vol. 67, no. 3, pp. 361–374, 2018.

[41] Docker Hub," https://hub.docker.com/search?q=, 2023.

Funded by Horizon Europe Framework Programme of the European Union

[42] Red Hat, "Quay," https://quay.io/.

[43] Matomo, https://matomo.org/.

[44] Docker Hub API," https://docs.docker.com/docker-hub/api/latest/, 2023.

[45] I. Dinur and K. Nissim, "Revealing information while preserving privacy," in ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems, 2003, pp. 202–210.

[46] C. Dwork, "Differential privacy," in International Colloquium on Automata, Languages and Programming (ICALP), Proceedings, Part II 33, 2006, pp. 1–12.

[47] "A Firm Foundation for Private Data Analysis," Commun. ACM, p. 86–95, 2011.

[48] J. Tang, A. Korolova, X. Bai, X. Wang, and X. Wang, "Privacy loss in apple's implementation of differential privacy on macos 10.12," arXiv preprint arXiv:1709.02753, 2017.

[49] J. Zhang, P. Yu, L. Qi, S. Liu, H. Zhang, and J. Zhang, "FLDDoS: DDoS Attack Detection Model based on Federated Learning," in Proc. of IEEE TrustCom, 2021

[50] Q. Tian, C. Guang, C. Wenchao, and W. Si, "A lightweight residual networks framework for ddos attack classification based on federated learning," in Proc. of IEEE INFOCOM Workshops, 2021.

[51] I. Rosenberg, A. Shabtai, Y. Elovici, and L. Rokach, "Adversarial machine learning attacks and defense methods in the cyber security domain," ACM Computing Surveys, vol. 54, pp. 1–36, 2021.

[52] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-efficient learning of deep networks from decentralized data," in Artificial intelligence and statistics, 2017.

[53] D. Lv, X. Cheng, J. Zhang, W. Zhang, W. Zhao, and H. Xu, "Ddos attack detection based on cnn and federated learning," in 2021 Ninth International Conference on Advanced Cloud and Big Data (CBD). IEEE, 2022, pp. 236–241

[54] M. Zhu, A. H. Anwar, Z. Wan, J.-H. Cho, C. A. Kamhoua, and M. P. Singh, "A survey of defensive deception: Approaches using game theory and machine learning," IEEE Communications Surveys & Tutorials, vol. 23, no. 4, pp. 2460–2493, 2021.

[55] A. Osman, P. Bruckner, H. Salah, F. H. Fitzek, T. Strufe, and M. Fischer, "Sandnet: towards high quality of deception in container-based microservice architectures," in ICC 2019-2019 IEEE International Conference on Communications (ICC). IEEE, 2019, pp. 1–7

[56] A. H. Anwar and C. Kamhoua, "Game theory on attack graph for cyber deception," in International Conference on Decision and Game Theory for Security. Springer, 2020, pp. 445–456.

[57] H. Li, Y. Guo, P. Sun, Y. Wang, and S. Huo, "An optimal defensive deception framework for the container-based cloud with deep reinforcement learning," IET Information Security, vol. 16, no. 3, pp. 178–192, 2022.

[58] H. Li, Y. Guo, S. Huo, H. Hu, and P. Sun, "Defensive deception framework against reconnaissance attacks in the cloud with deep reinforcement learning," Science China Information Sciences, vol. 65, no. 7, pp. 1–19, 2022.

[59] A. H. Anwar and C. A. Kamhoua, "Cyber deception using honeypot allocation and diversity: A game theoretic approach," in 2022 IEEE 19th Annual Consumer Communications & Networking Conference (CCNC). IEEE, 2022, pp. 543–549.

Funded by Horizon Europe Framework Programme of the European Union

APPENDIX A – FLUIDOS WORKFLOWS

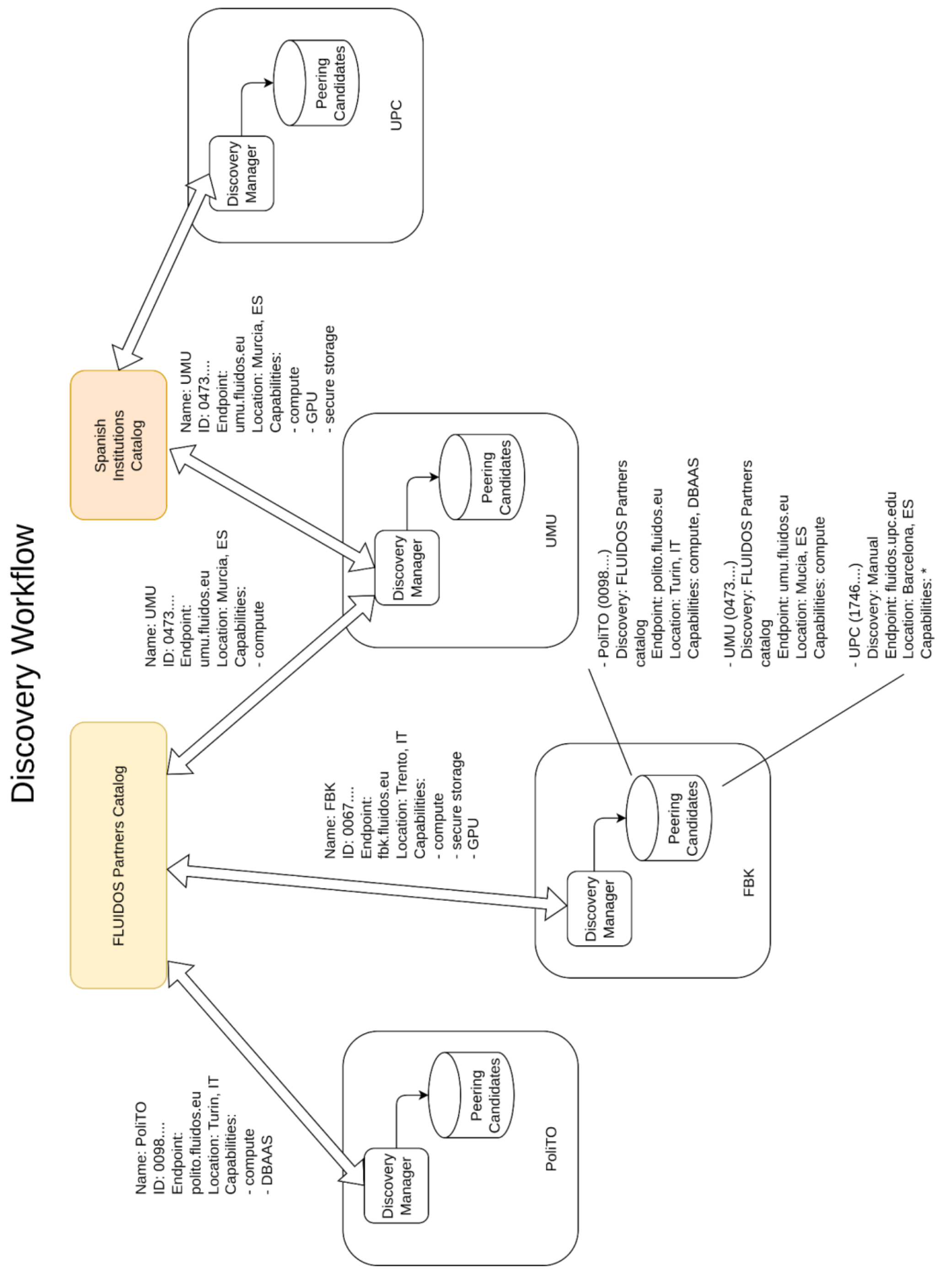


Figure A. FLUIDOS Discovery Workflow

Funded by Horizon Europe Framework Programme of the European Union

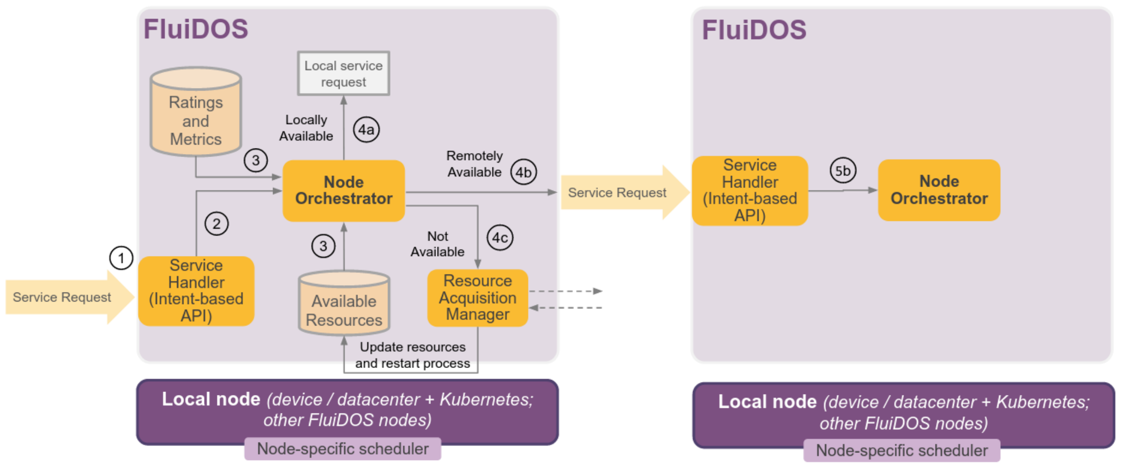

FluiDOS
Ratings and Metrics
Local service request
Locally Available
4a
3
Node Orchestrator
Remotely Available
4b
2
1
Service Request
Service Handler (Intent-based API)
3
Available Resources
Not Available
4c
Resource Acquisition Manager
Update resources and restart process
Local node (device / datacenter + Kubernetes; other FluiDOS nodes)
Node-specific scheduler
Service Request
FluiDOS
Service Handler (Intent-based API)
5b
Node Orchestrator
Local node (device / datacenter + Kubernetes; other FluiDOS nodes)
Node-specific scheduler

Figure B. FLUIDOS service requests Process

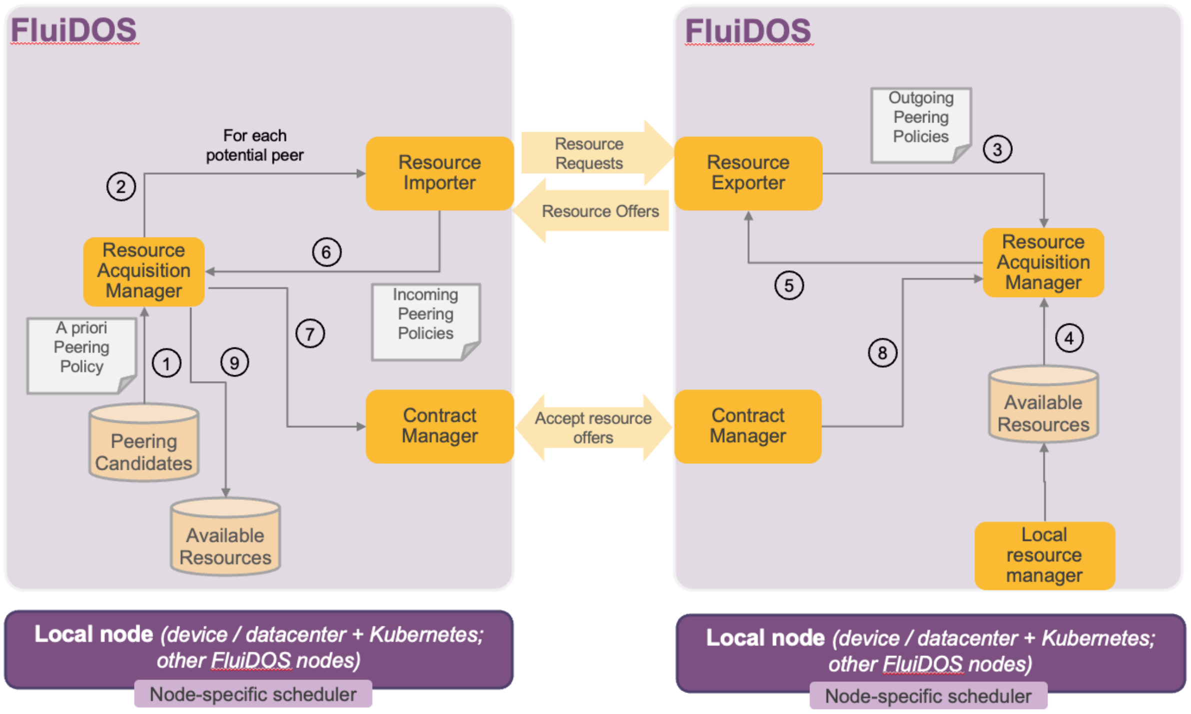


Figure C. FLUIDOS resource acquisition process

Funded by Horizon Europe Framework Programme of the European Union

APPENDIX B – CYBER DECEPTION

CYBER DECEPTION DESIGN

We report hereafter the mathematical formulation of the proposed optimization problem as well as the related heuristic algorithm approximating the optimal solution.

Optimal decoy allocation

We formalize the decoy allocation dynamic by leveraging the concept of Betweenness Centrality (BC) of a vertex. This metric measures the number of shortest paths traversing each graph vertex and offers an analytic approach to tackle the decoy allocation problem. The higher this value is, the more "central" is the related vertex compared to the surrounding vertices since its location allows it to reach multiple destinations with minimum cost. In our scenario, we can consider the BC as a security metric indicating the likelihood that a microservice may eventually interact with an attacker. Consequently, a suitable decoy allocation strategy should select those microservices with high BC values in order to increase the decoys chance to intercept possible attacks.

Following this idea, we formulate the following optimization problem:

$$\max_{\mathbf{x}} \sum_{m \in M} x_m \cdot \bar{b}(v_m) \qquad (1)$$

subject to

$$\sum_{m \in M} x_m r_{m,n}^{(cpu)} \leq \delta \cdot \Delta C_n^{(cpu)} \quad \forall n \in N \qquad (2)$$

$$\sum_{m \in M} x_m r_{m,n}^{(ram)} \leq \delta \cdot \Delta C_n^{(ram)} \quad \forall n \in N \qquad (3)$$

Funded by Horizon Europe Framework Programme of the European Union

$$x_m \cdot (r_{m,n}^{(cpu)} - \bar{r}_m^{(cpu)}) \geq 0 \quad \forall m \in M, \forall n \in N \tag{4}$$

$$x_m \cdot (r_{m,n}^{(ram)} - \bar{r}_m^{(ram)}) \geq 0 \quad \forall m \in M, \forall n \in N \tag{5}$$

- $x_m \in \mathbb{N}, \quad \forall m \in M$ (6)

Each addend in the objective function (1) provides the number of DAPs generated by the x_m decoys replicating microservice m. In particular, $\bar{b}(v_m)$ is the BC value associated to vertex v_m that accounts for DAPs generated by already deployed decoys. We remark that we proposed a closed-form formulation $\bar{b}(v_m)$ as a function of x_m in order to analytically compute the total number of generated DAP. Constraints (2)-(3) ensure that the allocated decoys do not exceed the dedicated CPU and RAM resources defined by the decoy resource ratio δ, respectively. The latter indicates the fraction of available resources dedicated to the deception mechanism. Constraints (4)-(5) enforce that each decoy must be allocated on the same node of the cloned microservice. Finally, constraint (6) expresses the integer nature of the considered allocation problem. The non-linear objective function as well as the integrity constraint render the problem complexity NP-Hard. For this reason, this problem formulation is not practical for a highly dynamic ecosystem composed of thousands of microservices deployed across multiple FLUIDOS nodes. Their resource occupation might be often reconfigured, which would trigger a possible deployment re-orchestration with a consequential need to recompute the decoy allocation as well. To overcome this limitation, we approximate the optimal allocation by designing an heuristic decoy allocation scheme to reduce the solution computational complexity.

Heuristic decoy allocation

The proposed heuristic consists of a greedy algorithm that prioritizes the allocation of decoys on microservices that require a low amount of resources and, at the same time, that are traversed by a high number of APs (in other words, it employs the BC values associated to each microservice in the original AG). In detail, the algorithm allocates one decoy at every iteration and updates the DAPs generation according to the previously allocated decoys. This procedure makes it possible to progressively enumerate every DAP introduced by the decoys and thus can help approximating the optimal allocation. We present the pseudo-code for this scheme in Algorithm 1.

Funded by Horizon Europe Framework Programme of the European Union

Algorithm 1 Heuristic decoy allocation

```
1:  **Input:** AG, $\bar{\sigma}$, $b$, $\mathbf{R}^{(.)}$, $\Delta C^{(.)}$
2:  **Output:** $\mathbf{x}$
3:  Initialize $x_m = 0$ for each $m \in M$
4:  Initialize the priority queue $Q$ to sort microservices
5:  Compute $\hat{d}_m = \lfloor \delta\Delta C_n^{(.)} / \bar{r}_m^{(.)} \rfloor$ for each $m \in M, n \in N$
6:  **for** each m $\in M$ **do**
7:      **if** $\hat{d}_m \geq 1$ **then**
8:          $p \leftarrow$ -$b(v_m) \cdot \hat{d}_m$
9:          Insert $m$ into $Q$ with priority $p$
10:     **end if**
11: **end for**
12: **while** $Q$ is not empty **do**
13:     Extract the first $m$ from $Q$
14:     Allocate a decoy on $m$ and update the AG topology
15:     Update $x_m \leftarrow x_m + 1$
16:     Update resource availability $\delta\Delta C_n^{(.)} \leftarrow \delta\Delta C_n^{(.)} - \bar{r}_m^{(.)}$
17:     **for** each $i \in M$ **do**
18:         **for** each $t \in M$ **do**
19:             $b(v_i) \leftarrow b(v_i) + (\bar{\sigma}_{AP(v_m,v_t)}^{(v_i)} + \bar{\sigma}_{AP(v_t,v_m)}^{(v_i)}) \cdot (x_t + 1)$
20:             $\theta_{v_t}^{(v_i)} \leftarrow \theta_{v_t}^{(v_i)} + (\bar{\sigma}_{AP(v_t,v_m)}^{(v_i)} + \bar{\sigma}_{AP(v_m,v_t)}^{(v_i)}) \cdot (x_m + 1)$
21:         **end for**
22:     **end for**
23:     Clear $Q$
24:     Recompute $\hat{d}_m$ for each microservice
25:     **for** each $i \in M$ **do**
26:         $\Delta b(v_i) \leftarrow b(v_i) \cdot (x_i + 1) \,/\, (x_i + 2)$
27:         **for** each $t \in M$ **do**
28:             $\Delta b(v_i) \leftarrow \Delta b(v_i) + [\sigma_{AP(v_m,v_t)}^{(v_i)} \cdot (x_t / x_t + 1)]$
29:         **end for**
30:         $p \leftarrow -\Delta b(v_i) \cdot \hat{d}_i$
31:         **if** $\hat{d}_i \geq 1$ **then**
32:             Insert $i$ into $Q$ with priority $p$
33:         **end if**
34:     **end for**
35: **end while**
```

Line 5 computes the number of deployable decoys $\hat{d}_m$ on each microservice defined as the available resources on the assigned node divided by the microservice resource consumption. This value is used to weight each microservices in the priority queue Q according to the number of APs going through them in lines 8-9.

We allocate a decoy on the microservice with the highest priority in Q (i.e. the one in the first position) by updating the AG topology in line 15 and we recompute the available resources for decoys in line 16. We iteratively calculate the number of generated DAPs in lines 17-22. In detail, line 19 updates the BC values for other microservices in AG given the new decoy allocation, while line 20 takes into consideration the total DAPs, denoted as θ_{v_t} , generated

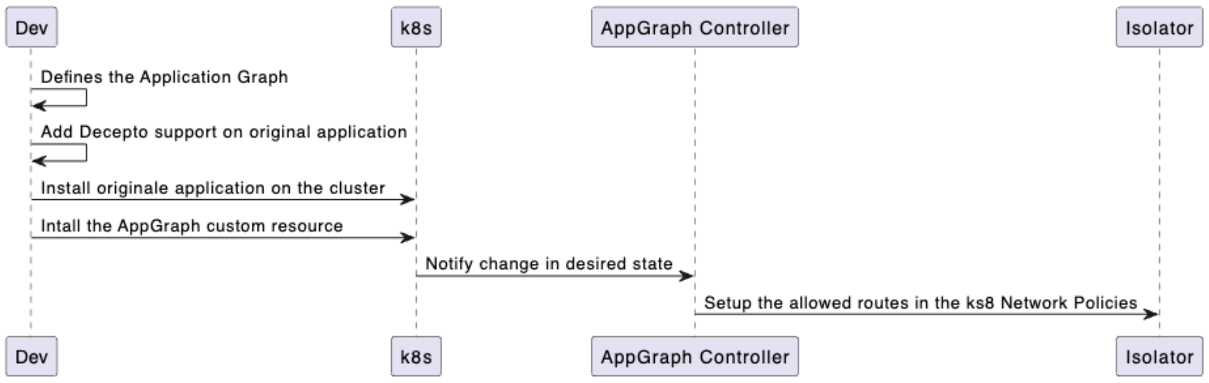
Funded by Horizon Europe Framework Programme of the European Union

by the interaction with the newly allocated decoy and the previously allocated ones. Lines 23-24 clean the queue and recompute the number of deployable decoys. These steps are needed to discard microservices that cannot be cloned as decoys, thus they are no longer considered as possible candidates. Lines 26-28 weight the microservices priority according to the potential number of DAPs, expressed as the BC increment $\Delta b(i)$, that can be generated by allocating an additional decoy on the corresponding microservice. These steps guide the decoy allocation computation in the next iteration as the algorithm is incentivized to select microservices with the highest BC gain. Lines 31-33 insert the microservices into the queue with the updated priority only if the resources available are sufficient. The algorithm convergence is completed when Q is empty, which indicates that the resources reserved for the decoys are exhausted. The overall algorithm time complexity is polynomial and can be quantified as $O(DM^2)$, where D is the number of allocated decoys and M is the number of active microservices.

CYBER DECEPTION IMPLEMENTATION

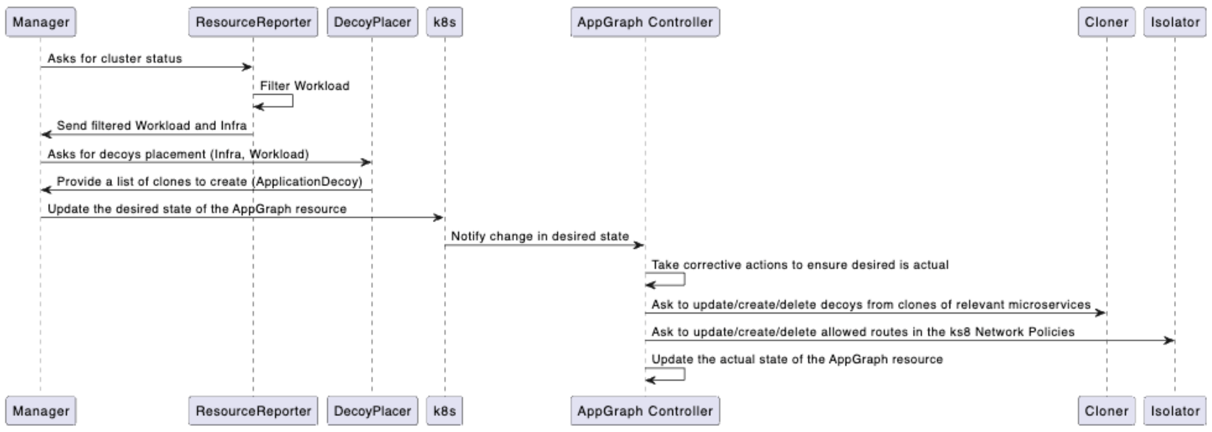
We report hereafter activity diagrams of the three main phases managed in the cyber-deception implementation.

Setup phase

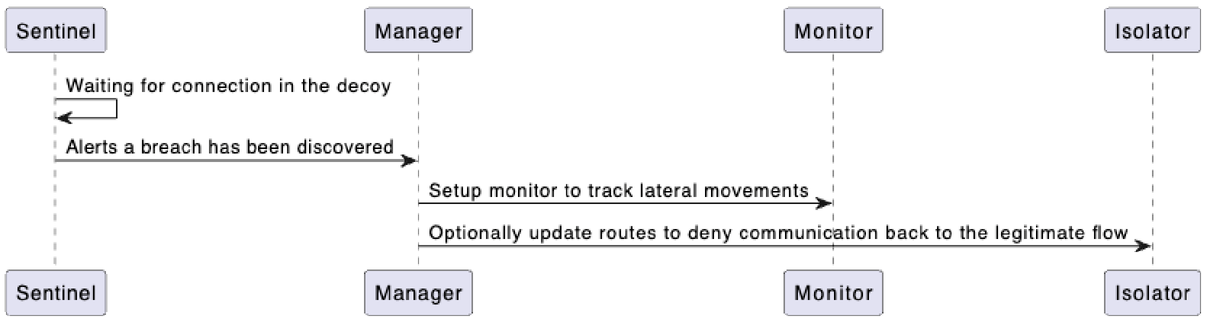

Dev
k8s
AppGraph Controller
Isolator
Defines the Application Graph
Add Decepto support on original application
Install originale application on the cluster
Intall the AppGraph custom resource
Notify change in desired state
Setup the allowed routes in the ks8 Network Policies
Dev
k8s
AppGraph Controller
Isolator

Funded by Horizon Europe Framework Programme of the European Union

Cloning phase


Manager
ResourceReporter
DecoyPlacer
k8s
AppGraph Controller
Cloner
Isolator
Asks for cluster status
Filter Workload
Send filtered Workload and Infra
Asks for decoys placement (Infra, Workload)
Provide a list of clones to create (ApplicationDecoy)
Update the desired state of the AppGraph resource
Notify change in desired state
Take corrective actions to ensure desired is actual
Ask to update/create/delete decoys from clones of relevant microservices
Ask to update/create/delete allowed routes in the ks8 Network Policies
Update the actual state of the AppGraph resource
Manager
ResourceReporter
DecoyPlacer
k8s
AppGraph Controller
Cloner
Isolator

Manage threat phase


Sentinel
Manager
Monitor
Isolator
Waiting for connection in the decoy
Alerts a breach has been discovered
Setup monitor to track lateral movements
Optionally update routes to deny communication back to the legitimate flow
Sentinel
Manager
Monitor
Isolator

Funded by Horizon Europe Framework Programme of the European Union