SUCCESS-6G: DEVISE

WP5 Deliverable E14

Project Title:	SUCCESS-6G: DEVISE
Title of Deliverable:	Final testing and validation of service KPIs
Status-Version:	v1.0
Delivery Date:	30/04/2025
Contributors:	Allen Abishek, Ricard Vilalta, Raul Muñoz (CTTC), Miguel Fornell, Francisco Paredes (Idneo), Angelos Antonopoulos (Nearby Computing)
Lead editor:	CTTC
Reviewers:	Charalampos Kalalas (CTTC), Francisco Paredes (Idneo)
Keywords:	Real-time location awareness; end-to-end latency; closest node selection

Document revision history

Version	Date	Description of change
v0.1	20/02/25	Table of Contents (ToC)
v0.2	28/02/25	Content added
v0.3	24/03/25	Additional inputs
v0.4	16/04/25	Final inputs and revision
v1.0	30/04/25	Final version uploaded to the website

Disclaimer

Acknowledgment

The research conducted by SUCCESS-6G - TSI-063000-2021-39/40/41 receives funding from the Ministerio de Asuntos Económicos y Transformación Digital and the European Union-NextGenerationEU under the framework of the “Plan de Recuperación, Transformación y Resiliencia” and the “Mecanismo de Recuperación y Resiliencia”.

[1] http://creativecommons.org/licenses/by-nc-nd/3.0/deed.en_US

Executive Summary

Security remains a paramount concern in the deployment of over-the-air (OTA) software updates, as vulnerabilities in vehicle software can expose systems to cyber threats. SUCCESS-6G-DEVISE introduces a security-first approach by integrating Security as a Service (SECaaS), AI-enhanced threat detection, and blockchain-based integrity verification. These mechanisms ensure that software updates are authenticated, encrypted, and delivered securely, preventing unauthorized modifications and cyberattacks. Experimental results on the ADRENALINE testbed demonstrate enhanced security enforcement, reduced threat detection times, and improved compliance with cybersecurity standards. The adoption of AI-driven security policies and network slicing for isolated update distribution solidifies SUCCESS-6G-DEVISE as a robust framework for secure vehicular software updates.

Table of Contents

List of Figures

1 Introduction

The increasing reliance on software-driven functionalities in modern vehicles necessitates a robust and secure method for delivering over-the-air (OTA) software updates. As connected and autonomous vehicle ecosystems continue to evolve, manufacturers and service providers must ensure that software updates are not only timely but also protected from cyber threats. Vehicle-to-Everything (V2X) communication is a key enabler of this transformation, facilitating seamless and reliable OTA updates while incorporating advanced security measures. However, ensuring data integrity, preventing unauthorized access, and safeguarding against cyberattacks remain critical challenges. The SUCCESS-6G-DEVISE project aims to address these security concerns through the integration of Software-Defined Networking (SDN), Security as a Service (SECaaS), and AI-driven threat detection.

A fundamental requirement for secure OTA software updates is ensuring end-to-end encryption and authentication mechanisms to prevent unauthorized modifications. Cellular V2X (C-V2X) technology, enabled by 5G and edge computing, enhances secure communication channels by enabling encrypted, tamper-proof data exchanges between vehicles and update servers. The SUCCESS-6G-DEVISE framework incorporates threat detection to continuously monitor update traffic for anomalies and potential cyber threats. Additionally, the use of network slicing ensures that OTA updates are transmitted over dedicated, isolated channels to prevent unauthorized interception and manipulation.

Beyond connectivity, cybersecurity remains a primary concern for OTA software updates. Outdated vehicle software is a prime target for cyberattacks, necessitating stringent security protocols to authenticate update sources and verify software integrity. SUCCESS-6G-DEVISE integrates Security as a Service (SECaaS) mechanisms, incorporating AI-enhanced threat detection, real-time anomaly detection, and blockchain-backed update verification. These security measures ensure that only authorized updates are deployed, mitigating risks such as firmware tampering, data breaches, and ransomware attacks. Efficient security management is another key factor in optimizing OTA software updates. MEC-based security processing reduces the computational burden on centralized cloud infrastructure by distributing security monitoring and threat mitigation to edge nodes. This allows for real-time security assessments, rapid threat response, and proactive risk mitigation. SUCCESS-6G-DEVISE employs threat prediction models to dynamically adjust security policies based on evolving cyber threats, vehicle density, and update criticality. By leveraging federated learning and distributed intelligence, the system ensures proactive security enforcement while minimizing processing delays.

This deliverable presents results on the implementation and validation of secure OTA software updates within a V2X connectivity framework. Experimental evaluations conducted on the ADRENALINE testbed demonstrate significant improvements in security enforcement, threat mitigation efficiency, and update integrity verification. The integration of SDN, SECaaS, and AI-driven security mechanisms has resulted in a scalable and adaptive solution capable of addressing the evolving cybersecurity demands of connected vehicle ecosystems. The findings from this research highlight the potential of SUCCESS-6G-DEVISE in revolutionizing vehicular software update security methodologies. By leveraging cutting-edge networking and security technologies, the proposed framework ensures that vehicles remain protected against cyber threats while maintaining seamless software update deployment. As the automotive industry continues to transition towards fully connected and autonomous systems, the implementation of secure and efficient OTA update mechanisms will be instrumental in enhancing vehicle safety, data protection, and regulatory compliance.

The subsequent sections present the specific methodologies employed, experimental setup, and detailed performance evaluations of the proposed secure OTA update system, providing a comprehensive analysis of its benefits and potential industry applications.

2 Use case 2: Automated software updates for vehicles

2.1 General description and overall objectives

Over-the-air software updates are delivered remotely from a cloud-based server, through a cellular connection, to the connected vehicle with the aim of providing new features and updates to the vehicle's software systems. Such software updates may include changes to any software that controls the vehicle's physical parts or electronic signal processing system. In practice, the updates often tend to apply more to user interfaces like infotainment screens and navigation (i.e., vehicle maps). The update procedure, when performed over the air, enables a vehicle's performance and features to be continuously up-to-date and improved. The integration of advanced data analytics, automated and remote service delivery eliminates the need for visiting repair/service centres, while technological advancements in these updates give vehicle manufacturers the freedom to constantly "freshen up" finished products remotely. C-V2X technology plays a crucial role in the update process, enabling efficient, scalable, and seamless wireless communication between vehicles and software management platforms. Figure 1 illustrates the implementation phases for this use case.

Figure 1 Implementation phases for the automated software updates

The overall **objectives** of this use case can be summarized as follows:

- Safer and more entertaining driving experience.
- Hardware and software components maintained and updated regularly during a vehicle's lifespan, implying a slower rate of depreciation.
- Prevention of cyberattacks targeting outdated software.
- Compliance to new rules and standards.
- Lower repair costs and elimination of labour charges.
- Lower warranty costs for manufacturers and lower downtime for customers

The key **stakeholders** involved in the use case are:

- The ***Mobile Network Operator (MNO)***, providing wireless connectivity between the vehicle, the edge computing infrastructure, and the vehicular software management system. The MNO is interested in optimizing the network operation by enhancing its energy efficiency and coverage, while offering novel services to accommodate more users.
- The ***edge infrastructure provider***, offering and managing computational resources at the edge and supporting real-time services as well as virtualized network functions and AI-empowered algorithms for advanced computational tasks.
- The ***equipment provider,*** providing in-vehicle embedded devices, e.g., hardware components and sensor devices, that can be remotely reconfigured and updated.
- The ***vehicular software management system,*** operated by the equipment provider or vehicle manufacturer, is responsible for issuing periodically new software updates.
- The ***software developers***, devising and applying data-processing modules for automated update of vehicular components' software.
- The ***cloud providers*** can optionally be involved, offering additional computational resources to host the service.

Note that, without loss of generality, some stakeholders may assume multiple roles or, equally, some roles may be assumed by multiple stakeholders. For instance, the MNO could also be the owner of the

edge infrastructure, or an equipment provider may also be responsible for the operation of the vehicular software management system or outsource it to a third party.

2.2 User story 2.2: Over-the-air vehicular software updates with security guarantees

Over-the-air software updates deliver critical information to onboard vehicular devices. As vehicles introduce new functionalities (such as advanced driver-assist features like self-parking) and the number of connected vehicles keeps growing, automakers need to handle the regular software updates required in a secure and trustworthy way. Thus, the integration of intelligent security enforcement solutions and effective prediction/mitigation of security threats is deemed essential for the secure operation of the OTA update service and to preserve trustworthiness. Additionally, by instantiating virtual security functions and by exploiting secure edge provisioning empowered by AI-driven capabilities, the threat risk for software updates can be further minimized.

2.3 Overall UC2 architecture and network deployments

The elaboration of Figure 2 details a system architecture specifically designed for OTA software updates, integral to the SUCCESS-6G framework. This architecture addresses the complex requirements of Use Case 2. Figure 2 Proposed overall UC2 system architectureprovides a high-level system architecture for OTA vehicular software updates within a robust V2X connectivity framework, leveraging ETSI TeraFlowSDN for network automation and control. The figure illustrates the key components enabling software update dissemination to connected vehicles via 5G mobile edge computing (MEC) nodes.

At the core of this system is the ETSI TeraFlowSDN Controller, which manages the network infrastructure, including the gNBs (5G base stations) and Transport Network. The NFV Orchestrator (NFV-O) enables dynamic deployment and scaling of virtualized network functions, such as Distributed User Plane Functions (D-UPF) within MEC nodes.

Each edge node (Edge Node 1 & Edge Node 2) hosts a Software Update Server, responsible for caching and distributing updates to C-V2X On-Board Units (OBU) in connected vehicles. These updates are delivered via the 5G network, passing through the transport network, controlled by the TeraFlowSDN controller.

To ensure security and integrity, the system integrates a Security-as-a-Service module, providing firewall protection and secure communications for software updates. The updates originate from local cloud infrastructure, which includes 5G Core Control Plane components such as SMF (Session Management Function), AMF (Access and Mobility Management Function), and UPF (User Plane Function).

The software update client within the vehicle's C-V2X OBU interacts with the Software Update Servers over the network, ensuring efficient and timely delivery of critical updates for vehicle applications.

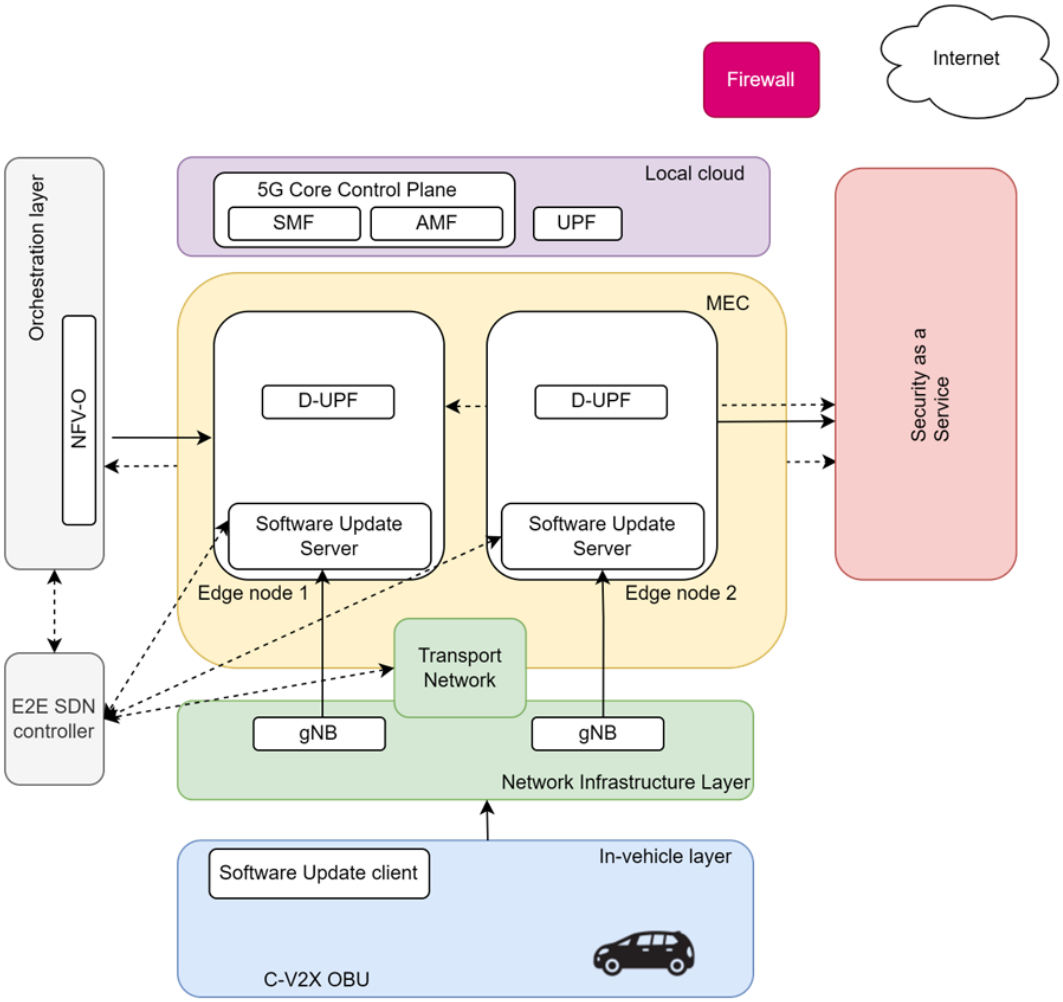

Figure 2 Proposed overall UC2 system architecture

This architecture highlights the interplay between 5G, MEC, SDN, and V2X technologies to facilitate secure and efficient OTA software updates, enabling reliable vehicle connectivity and automation.

2.4 Facilities for Use Case 2: ADRENALINE Testbed

The ADRENALINE testbed® is an open and disaggregated SDN/NFV-enabled packet/optical transport network and edge/core cloud infrastructure for 6G, IoT/V2X and AI/ML services, constantly evolving since its creation in 2002, and reproducing operators' networks from an End to End (E2E) perspective and Data Centre Interconnect (DCI). The figure below summarizes the networking scenario of ADRENALINE testbed, to be used for the execution of SUCCESS-6G.

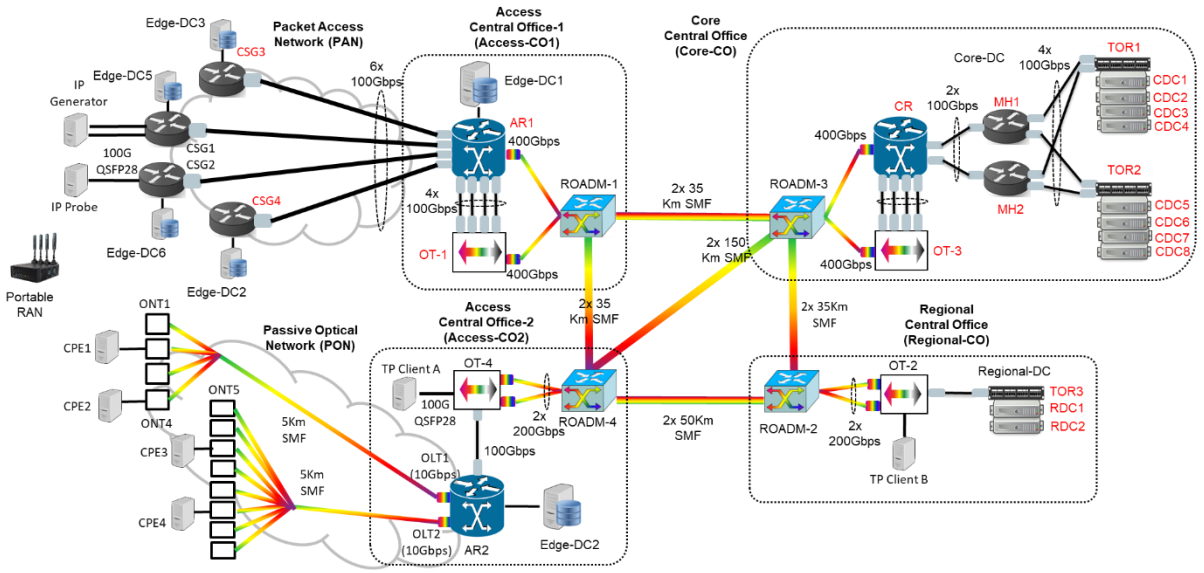

Figure 3 ADRENALINE testbed to be used for Use Case 2

ADRENALINE spans the access, aggregation-metro and core segments, and includes distributed Data Centres (DCs) geographically disperse and located at the edge or in central locations. As depicted in the figure, the key elements are: (1) an SDN-controlled optical network (flexi-grid DWDM photonic mesh), with 4 ROADM nodes and over 600km of amplified DWDM links. Currently, all the links of the mesh are based on amplified C-band transmission, but one of them also supports amplified flexible L-band transmission; (2) packet-optical nodes with optical pluggable transceivers, providing aggregated 400G data rates (muxponders) for transporting traffic flows between the access networks and the core central offices or data centers; (3) programmable SDN-enabled S-BVTs able to transmit multiple flows at variable data rate/reach up to 1 Tb/s; (4) a Packet Access Network (PAN) connected to the metro infrastructure with IP Cell Site Gateways (CSGs); (5) a PON tree formed by disaggregated Optical Network Terminals (ONTs), offering connectivity to several Customer Premises Equipment (CPEs). ADRENALINE also includes a Portable 5G RAN platform for testing and validation of 5G and beyond use cases. The different access networks (i.e., PON) and the photonic mesh are managed by dedicated orchestrators and controllers (e.g., CTTC FlexOpt Optical Controller) to automatically handle the connectivity services entailing the de-/allocation of heterogeneous network resources (i.e., packet and optical devices). The *domain-specific* controllers and orchestrators are coordinated hierarchically by the ETSI TeraFlowSDN controller, which exposes a North Bound Interface to allow interaction of resources to request network connectivity services. This service platform orchestrates the transport (optical/packet) and computing:

i) Multi-VIM (virtualized infrastructure managers) combining OpenStack and K8s controllers for virtual machines and containers;

ii) TeraFlowSDN controller for E2E connectivity among virtual machines, containers, and end-points. The service platform is also in charge of managing the life-cycle of network services and network slices: i) a network service is composed of chained NFs;

iii) a network slice is composed of one or several concatenated network services that deploy a set of NFs.

3 Over-the-air vehicular software updates with security guarantees: Implementation at the ADRENALINE testbed

3.1 Security as a Service

Figure 4 illustrates a sophisticated network architecture designed for service management through Software Defined Networking (SDN) and Network Function Virtualization (NFV) technologies. At the apex of this architecture is the API (NBI), the Northbound Interface, which functions as a pivotal point of access for service management, facilitating communication between the service management layers and the underlying network infrastructure.

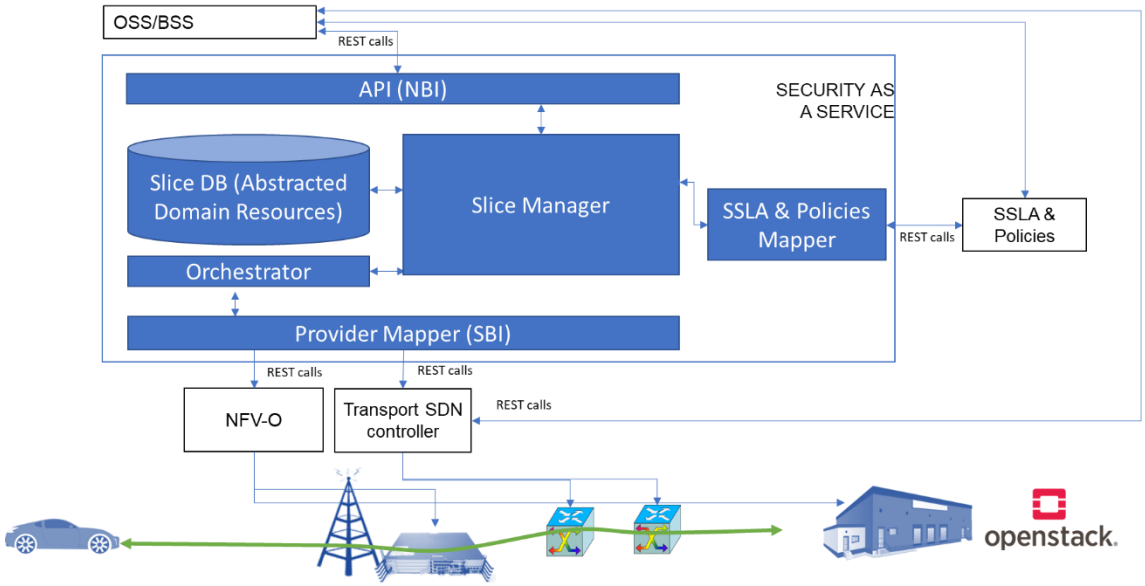

Figure 4 Proposed architecture for OTA vehicular software updates with security guarantees

Central to this architecture is the Security as a Service component. It includes a Slice Manager, a component that manages network slices—distinct segments of the network tailored for specific operational requirements, each with its own set of performance parameters and policies. This management is intricately connected to the SSLA & Policies Mapper, which translates service level agreements and policies into practical, enforceable rules for each network slice.

These slices are cataloged in the Slice DB (Abstracted Domain Resources), a comprehensive repository that maintains details about network slices and their corresponding resources. The Orchestrator operates in concert with the Slice DB, orchestrating the deployment and lifecycle of network services across various network segments.

The Provider Mapper (SBI), or the Southbound Interface, communicates with physical and virtual network functions, translating the orchestrated service management directives into actionable tasks within the network fabric. This includes the NFV Orchestrator (NFV-O), which is responsible for the overall management of virtualized network functions, ensuring their proper instantiation, scaling, and termination.

At the foundation of the network control plane is the Transport SDN controller, which governs data plane devices such as routers and switches, enabling efficient and dynamic routing of traffic within the network.

The infrastructure components, including vehicles, cellular towers, and data centers, are depicted at the bottom of the figure, highlighting the endpoints of this network architecture. These components represent the tangible elements where data and services are consumed and delivered, completing the ecosystem of this advanced network architecture.

3.2 Requirements and KPIs

The rapid evolution of connected vehicle technology has heightened the need for stringent security, reliability, and performance standards. As modern vehicles increasingly depend on software to manage critical functionalities—ranging from engine control and braking systems to autonomous navigation—ensuring robust mechanisms for software updates and network management is paramount. To address these concerns, this section outlines the key requirements that safeguard automotive software ecosystems against a broad spectrum of cyber threats while maintaining operational efficiency.

Each requirement detailed in the following subsections underscores a unique aspect of secure and reliable software lifecycle management. From Secure Software Update Management that focuses on authentication, authorization, and integrity checks, to Security as a Service (SECaaS) offering scalable and automated policy enforcement, these requirements collectively establish a comprehensive defense strategy. Further, Software-Defined Networking (SDN) & Network Function Virtualization (NFV) Integration and Slice-Based Service Management enable agile, resource-efficient network operations tailored to the varying needs of different vehicles and update types. Finally, Northbound and Southbound Interface Support ensures seamless interoperability across multiple platforms and vendors, facilitating smooth update distribution and real-time monitoring.

Together, these requirements form a holistic framework designed to protect and enhance the software-driven functionalities of connected vehicles. By emphasizing security, scalability, and interoperability, they serve as the foundational building blocks for maintaining trust, minimizing risks, and supporting the advanced features of next-generation automotive systems.

3.2.1 Requirements

3.2.1.1 Secure Software Update Management

Ensuring that software updates are securely distributed is critical to preventing unauthorized modifications, malware insertion, or other cybersecurity threats. The system must authenticate and authorize update sources before distribution, verifying the integrity and authenticity of software packages before deployment to vehicles. This process helps mitigate security risks associated with malicious attacks or accidental corruption of updates, ensuring that only trusted software is installed on connected vehicles.

The rationale behind this requirement is to enhance the safety and reliability of vehicular software systems. Modern vehicles rely heavily on software to control essential functionalities, including braking, navigation, and autonomous driving. Any compromise in the update process could lead to severe security breaches or operational failures. By implementing stringent authentication mechanisms and encryption methods, the system can maintain trust in over-the-air updates while complying with automotive cybersecurity standards.

3.2.1.2 Security as a Service (SECaaS)

The security framework should be designed as a scalable and adaptable service that automates security policy enforcement across different network slices. This ensures a consistent security posture while accommodating various levels of security requirements based on the specific needs of each vehicle or update type. Security policies should be dynamically adjustable to respond to evolving cyber threats and regulatory requirements without requiring significant infrastructure changes.

The rationale for integrating Security as a Service is to provide a centralized and efficient approach to managing security controls across multiple network domains. Instead of applying security configurations manually to individual network components, SECaaS enables automation, monitoring, and policy enforcement in a streamlined manner. This approach reduces operational overhead,

improves compliance with cybersecurity regulations, and enhances the ability to respond quickly to emerging threats in the automotive ecosystem.

3.2.1.3 Software-Defined Networking (SDN) & Network Function Virtualization (NFV) Integration

The system must leverage SDN and NFV to optimize network resource allocation, enhance scalability, and ensure efficient update distribution. SDN enables centralized network control, while NFV provides flexible network functions that can be deployed dynamically as needed. Together, these technologies support real-time monitoring and adaptation of network conditions, ensuring that software updates reach their intended destinations with minimal latency and high reliability.

The rationale behind integrating SDN and NFV is to address the growing complexity of vehicular networks, where software updates must be transmitted efficiently over heterogeneous network infrastructures. By decoupling network control and function deployment from traditional hardware dependencies, SDN and NFV enable more agile, responsive, and cost-effective network management. This improves update delivery speeds and enhances the resilience of the network infrastructure, preventing bottlenecks and service disruptions.

3.2.1.4 Slice-Based Service Management

Network slicing must be implemented to create isolated virtual networks tailored to different operational needs. Each slice should be assigned specific security policies and Quality of Service (QoS) parameters to ensure that critical updates receive the highest priority. The slice database should store metadata about network slices, facilitating efficient resource allocation and policy enforcement.

The rationale for using network slicing is to ensure that essential updates, such as security patches or firmware upgrades, are delivered with guaranteed performance and minimal risk of interference. Vehicles operating in different regions or under varying network conditions may have different connectivity requirements. Slicing enables customized resource allocation, allowing for differentiated services while maintaining strong security and performance guarantees.

3.2.1.5 Northbound and Southbound Interface Support

The system must expose well-defined interfaces for seamless integration with external platforms. The Northbound Interface (NBI) should facilitate communication with service management applications, while the Southbound Interface (SBI) should interact with physical and virtual network functions. These interfaces should support standardized protocols to ensure compatibility across different vendors and infrastructure providers.

The rationale for exposing standardized interfaces is to promote interoperability and simplify integration with third-party applications, including automotive manufacturers, cybersecurity monitoring tools, and cloud-based orchestration platforms. By enabling flexible API interactions, the system can support automated update workflows, real-time security monitoring, and policy-driven update deployments, enhancing the overall efficiency of over-the-air update management.

3.2.2 Key Performance Indicators (KPIs)

3.2.2.1 End-to-end encryption success rate (%)

This KPI measures the effectiveness of encryption mechanisms in securing data transmission by assessing the percentage of successful encrypted transmissions without breaches.

Encryption is crucial for ensuring the confidentiality and integrity of software updates during transmission. A high success rate indicates a robust security framework, reducing the risk of data tampering or interception by malicious actors.

Target Value: At least 99.9% of transmitted updates should be successfully encrypted without any detected breaches.

3.2.2.2 Software update delivery latency (ms)

This KPI measures the time taken for an update to reach its destination from the source.

Low latency is essential for delivering time-sensitive security patches and feature updates to vehicles. Delays in updates could leave vehicles vulnerable to exploits or operational inefficiencies.

Target Value: The system should deliver critical updates within 500 ms in optimal network conditions.

3.2.2.3 SLA compliance rate (% adherence to defined policies)

This KPI evaluates how well the system adheres to predefined service level agreements (SLAs) regarding security, reliability, and performance.

Ensuring SLA compliance helps maintain trust with stakeholders and guarantees that vehicles receive updates as promised within set parameters.

Target Value: At least 98% adherence to SLAs across all update transactions.

3.2.2.4 Mean Time to Detect (MTTD)

MTTD is the time elapsed from the moment a security threat or attack occurs to the point where it is successfully detected by the security system. It represents the efficiency of threat detection mechanisms, including AI-driven inference models, log monitoring, and real-time anomaly detection.

Importance: A lower MTTD is critical for early threat identification, reducing the window of exposure and limiting potential damage. Faster detection enables security teams to respond proactively before an attacker can exploit vulnerabilities. Optimizing MTTD is particularly important in cloud-native environments, 5G networks, and real-time cybersecurity frameworks, where high-speed attack detection is required to protect critical infrastructure and services.

Target Values/Thresholds: Ideal: Less than 1 minute for AI-enhanced cybersecurity systems with real-time monitoring. Acceptable: Between 5–10 minutes in standard SOC (Security Operations Center) environments with human-assisted analysis. Critical Risk: Above 30 minutes, as this increases the likelihood of widespread damage, extended data breaches, and regulatory non-compliance.

3.2.2.5 Mean Time to Respond (MTTR)

MTTR is the time elapsed from the moment a security threat is detected to the complete mitigation or resolution of the issue. It includes incident triage, threat analysis, containment, mitigation, system restoration, and validation to ensure the threat has been neutralized.

Importance: A lower MTTR reduces the impact of cyberattacks by minimizing system downtime, data breaches, and business disruptions. In environments like telco networks, IoT ecosystems, and cloud-native microservices, rapid response ensures business continuity and compliance with cybersecurity policies. Automated orchestration and AI-driven remediation significantly improve MTTR by enabling real-time threat containment and recovery.

Target Values/Thresholds: Ideal: Under 5 minutes for automated threat response mechanisms (e.g., SDN-controlled ACL updates, automated patching). Acceptable: 30–60 minutes in cases where manual intervention or forensic analysis is required. Critical Risk: Above 2 hours, as prolonged response times increase financial losses, system disruptions, and potential data exfiltration risks.

3.3 UC2 Architecture and network deployment

Figure 5 illustrates a comprehensive architecture for enabling software updates in a Cellular Vehicle-to-Everything (C-V2X) environment using 5G, Multi-access Edge Computing (MEC), Network Function Virtualization (NFV), and Software-Defined Networking (SDN). The system is structured into multiple layers to ensure efficient data processing, low-latency communication, and secure software updates for in-vehicle systems. These layers include the Orchestration Layer, 5G Core Control Plane (Local

Cloud), MEC Layer (Edge Nodes), Network Infrastructure Layer, and In-Vehicle Layer. Among these, the dark blue elements play a crucial role in orchestrating network functions, ensuring security, and enabling seamless data transmission.

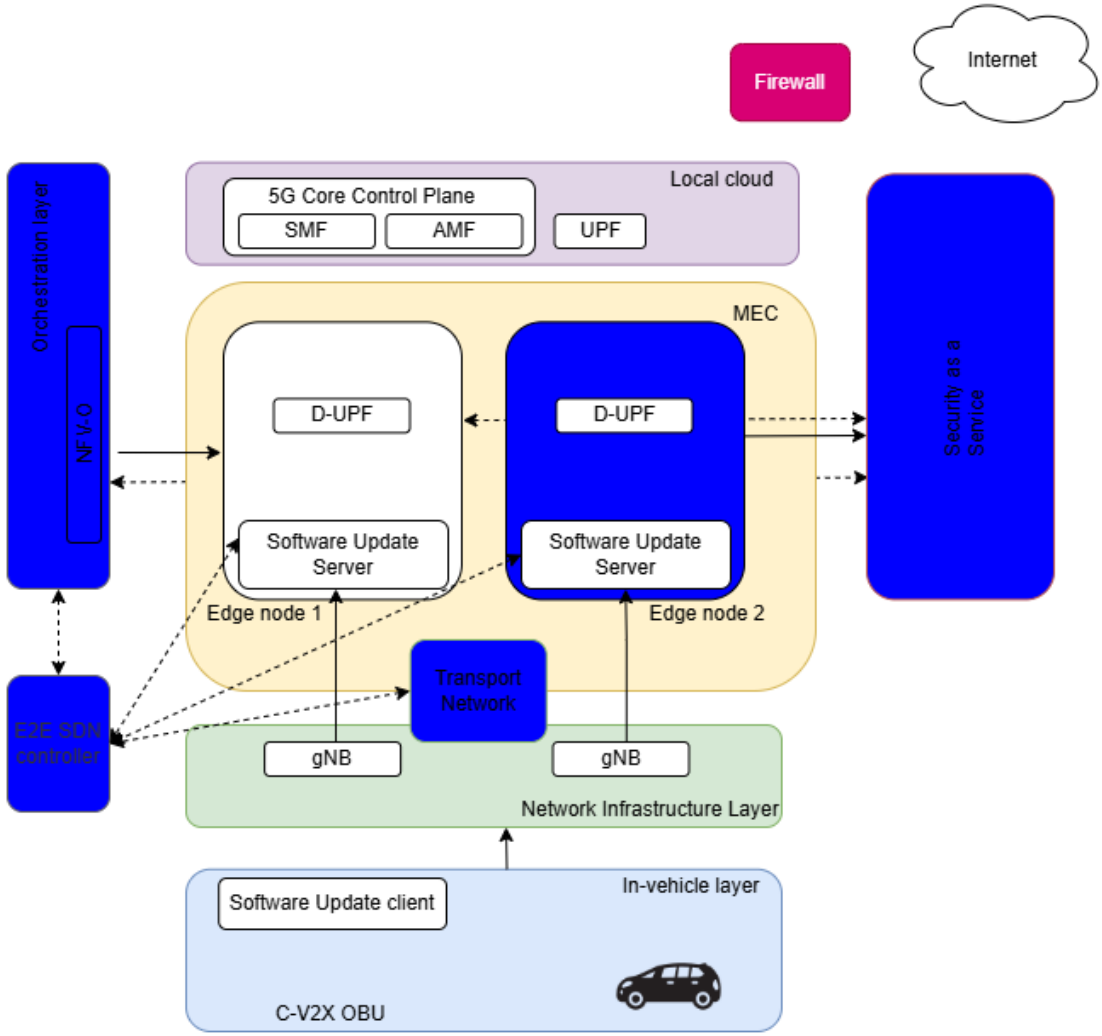

Figure 5 Instantiation of SUCCESS-6G architecture for OTA vehicular software updates with security guarantees

At the top of the architecture, the Orchestration Layer is responsible for managing the deployment and lifecycle of network functions using NFV and SDN technologies. The NFV Orchestrator (NFV-O) oversees the instantiation, scaling, and management of Distributed User Plane Functions (D-UPFs), which are deployed closer to the edge to reduce latency. Alongside NFV-O, the End-to-End SDN Controller provides centralized control over the transport network, ensuring efficient data routing and dynamic policy enforcement. This integration of NFV and SDN allows for adaptive network resource management, dynamic traffic steering, and enhanced service reliability, which are essential for delivering real-time software updates to connected vehicles.

The Multi-access Edge Computing (MEC) layer is a critical part of the architecture, hosting D-UPFs and Software Update Servers at edge nodes. The D-UPF (Distributed User Plane Function) is responsible for offloading data traffic from the 5G core, allowing localized data processing and reducing dependency on central cloud infrastructure. This offloading mechanism enhances the responsiveness of V2X applications, including OTA software updates. The Software Update Servers deployed at the edge nodes ensure that vehicles receive software patches and security updates efficiently, minimizing downtime and optimizing network resource utilization. By leveraging MEC, the system reduces latency, improves update delivery speed, and enhances overall network performance.

To protect the integrity and confidentiality of software updates and vehicular communications, the architecture incorporates a Security as a Service framework. This module ensures that all software

updates are encrypted, verified, and transmitted securely to prevent cybersecurity threats such as man-in-the-middle attacks, data tampering, and unauthorized access. The security framework is tightly integrated with the MEC layer and orchestration systems, ensuring continuous threat monitoring, anomaly detection, and compliance with cybersecurity policies. By implementing security measures at both the orchestration and edge levels, the system guarantees the safety and reliability of C-V2X applications.

At the transport network level, gNBs (Next-Generation Base Stations) facilitate communication between vehicles and the edge computing infrastructure. These base stations provide high-speed, low-latency connectivity to ensure that software update clients in vehicles (C-V2X OBUs) receive real-time updates. The transport network is dynamically managed by the E2E SDN controller, which optimizes routing paths based on network congestion, traffic demand, and security policies. Within the In-Vehicle Layer, the Software Update Client ensures that received updates are properly installed, validated, and synchronized with the vehicle's onboard systems. This integration ensures that connected vehicles remain updated with the latest firmware, security patches, and feature enhancements without requiring physical intervention.

3.4 Exposed interfaces

We have developed two different interfaces, dedicated to the attack detector and attack mitigator.

3.4.1 Attack Detector

The provided Protobuf definition outlines an AttackDetector service within the "attack_detector" package, leveraging proto3, the latest version of the Protobuf syntax. This service is responsible for configuring and managing an attack detection system that integrates with Elasticsearch, Kafka, and machine learning-based threat detection.

The AttackDetector service is defined as a gRPC-based service that enables clients to configure and retrieve attack detection settings remotely. It consists of six Remote Procedure Call (RPC) methods, each of which facilitates specific operations. The ConfigureDetector method allows clients to configure the attack detector using a DetectorConfig message, which includes Elasticsearch and Kafka settings along with a minimum machine learning confidence level for attack detection. The GetDetectorConfiguration method enables clients to retrieve the current configuration settings, ensuring they can verify or modify existing parameters when necessary. Additionally, the ConfigureAttack method allows users to define attack specifications, while ListConfiguredAttacks provides a list of all active attack configurations. To retrieve details of a specific attack, clients can use the GetConfiguredAttack method, which takes an AttackId as input and returns the corresponding AttackSpecs. Lastly, the DeconfigureAttack method allows for the removal of an attack configuration, ensuring that outdated or irrelevant attack profiles do not persist in the system.

Each RPC function follows a request-response model, meaning a client sends a request message and expects a structured response. This approach is particularly useful for gRPC-based microservices, where real-time communication is essential. The service relies on various Protobuf messages to facilitate structured data exchange. The Empty message serves as a placeholder for functions that do not require parameters or return values, similar to an empty JSON object ({}). The ElasticSearch message defines the configuration for an Elasticsearch database, which the attack detector uses for storing and querying attack data. It includes a list of hosts, authentication details, and an index name, ensuring secure and efficient interactions with the database. Similarly, the Kafka message encapsulates settings for Kafka-based event streaming, including broker hosts, authentication credentials, consumer group identifiers, and topic names for sending and receiving messages. This configuration enables the attack detector to process real-time threat intelligence streams efficiently.

The DetectorConfig message acts as a central configuration entity, combining Elasticsearch and Kafka settings with a minimum machine learning confidence level. This confidence level determines the threshold at which the system considers a potential attack to be valid, balancing false positives and

false negatives effectively. The attack identification process relies on the AttackId message, which contains a unique attack UUID and a probability threshold. This allows the system to manage and track individual attack instances with a predefined level of certainty. Furthermore, the AttackSpecs message extends AttackId by introducing an additional parameter: the minimum confidence level for detection. This ensures that the attack detection model can be fine-tuned to only trigger alerts for high-confidence threats, reducing noise in the system. To facilitate bulk retrieval of attack configurations, the ListAttackSpecs message aggregates multiple AttackSpecs entries into a structured list.

The use of Protobuf in this architecture provides several key advantages. Firstly, Protobuf's binary format ensures efficient serialization, reducing both the size and transmission time of messages compared to traditional text-based formats. This efficiency is particularly beneficial in network security and attack detection, where large volumes of streaming data must be processed in real time. Secondly, Protobuf allows for schema evolution without breaking compatibility. Fields can be added or deprecated without disrupting existing services, making it easier to introduce new attack detection parameters as threats evolve. Thirdly, strongly typed structures in Protobuf reduce parsing errors and improve data integrity, ensuring that attack configurations remain accurate and consistent across different components of the system.

Furthermore, Protobuf is optimized for gRPC, which enables efficient and secure remote procedure calls over HTTP/2. This allows the AttackDetector service to scale effectively, leveraging features such as bidirectional streaming, load balancing, and authentication. The combination of Protobuf, Kafka, and Elasticsearch in this architecture creates a robust framework for real-time attack detection and response. By enabling efficient configuration, retrieval, and deconfiguration of attack profiles, this system ensures that security measures remain adaptive and responsive to emerging threats.

```
syntax = "proto3";
package attack_detector;

service AttackDetector {
   rpc ConfigureDetector      (DetectorConfig) returns (      Empty        ) {}
   rpc GetDetectorConfiguration(Empty        ) returns (      DetectorConfig ) {}
   rpc ConfigureAttack        (AttackSpecs   ) returns (      Empty        ) {}
   rpc ListConfiguredAttacks  (Empty        ) returns (      ListAttackSpecs) {}
   rpc GetConfiguredAttack    (AttackId     ) returns (      AttackSpecs    ) {}
   rpc DeconfigureAttack      (AttackId     ) returns (      Empty        ) {}
}

message Empty {}

message ElasticSearch {
   repeated string hosts = 1;
   string authentication = 2; // example: "api_key:<API_KEY>"
   string index          = 3;
}

message Kafka {
   repeated string hosts    = 1;
   string authentication    = 2; // example: "api_key:<API_KEY>"
   string consumer_group_id = 3;
   string consumer_topic    = 4;
   string producer_topic    = 5;
}

message DetectorConfig {
```

```
    ElasticSearch elasticsearch = 1;
    Kafka         kafka         = 2;
    float min_ml_confidence_level = 3;
}

message AttackId {
    string attack_uuid = 1;
    float probability_threshold = 2;
}

message AttackSpecs {
    AttackId attack_id          = 1;
    float    min_confidence_level = 2;
}

message ListAttackSpecs {
    repeated AttackSpecs attack_specs_list = 1;
}
```

3.4.2 Attack Mitigator

RFC 8519, titled "YANG Data Model for Network Access Control Lists (ACLs)," defines a standardized YANG 1.1 data model for configuring and managing ACLs on network devices. An Access Control List (ACL) is an ordered set of rules, known as Access Control Entries (ACEs), that determine how packets are processed based on specific match criteria and corresponding actions. The YANG model provides a structured and vendor-neutral way to configure ACLs, ensuring interoperability across different network platforms.

The ACL model is structured as a hierarchical YANG schema that allows network administrators to define ACLs in a standardized manner. At the top level, the acls container holds multiple ACL configurations. Each ACL is uniquely identified by a name and a type, which specifies whether it applies to IPv4, IPv6, or Ethernet traffic.

Within each ACL, the aces container holds a list of Access Control Entries (ACEs). Each ACE contains:

- A unique name for identification.
- Match conditions that define criteria based on packet headers, organized into:
- Layer 2 (l2): Matches Ethernet fields, such as source and destination MAC addresses.
- Layer 3 (l3): Matches IP packet fields, such as source and destination IP addresses.
- Layer 4 (l4): Matches transport-layer fields, such as TCP/UDP ports.
- Actions specifying how matching packets should be handled, such as permit, deny, or log.

One of the most critical aspects of the YANG ACL model is its ability to bind ACLs to network interfaces. The model defines an acl-interfaces structure that enables administrators to attach ACLs to specific network interfaces, including physical and logical interfaces. This ensures that ACLs can be applied in a structured and policy-driven manner to control both ingress (incoming) and egress (outgoing) traffic. The interface attachments can apply to various network devices, such as routers, switches, and firewalls.

The YANG model supports different types of interfaces where ACLs can be enforced:

- Physical Interfaces – Ethernet, fiber, or any other hardware network interface.
- Logical Interfaces – VLAN interfaces, virtual tunnel interfaces, or sub-interfaces.
- Software-Defined Interfaces – Interfaces used in SDN-based environments where ACL policies are dynamically enforced.

- By defining ACLs within the YANG model, administrators can programmatically attach them to these interfaces, ensuring centralized and automated security policies across network environments.

The YANG ACL model also includes mechanisms to collect and report statistics. Each ACE can have associated counters tracking packet matches, drops, and forwarding actions. These statistics allow administrators to monitor ACL effectiveness, optimize rule sets, and troubleshoot network security policies efficiently.

The YANG-based ACL model is designed to be extensible, allowing vendors to augment the base model with additional capabilities. For instance, vendors can introduce custom match conditions, logging mechanisms, or policy-based ACL configurations while still maintaining compatibility with the standard model. Additionally, the model includes feature statements that enable devices to advertise the specific ACL capabilities they support.

3.5 Workflow

The entire architecture of the CLA platform can be seen in Figure 6. The functionality of the CLA platform begins with the presumption that a connected car has already established a Protocol Data Unit (PDU) session between itself and the OTA server via the Telecom Network. Then, the user plane packets from the vertical service (e.g, IP audio calls, music, video-streaming, OTA updates, etc.) are transmitted from the connected car through the gNB to the UPF located in the Mobile Edge Compute (MEC). Then, the UPF forwards the data plane packets to the CLA platform. The components in the CLA platform determine if incoming packets are an attack or not, if the CLA platform determines the Network is being exposed to a PoD attack, the CLA platform creates the necessary ACL which is sent to the TFS and finally the ACL is pushed into the Network Element (switch or router or firewall or server) that will block the malicious packets.

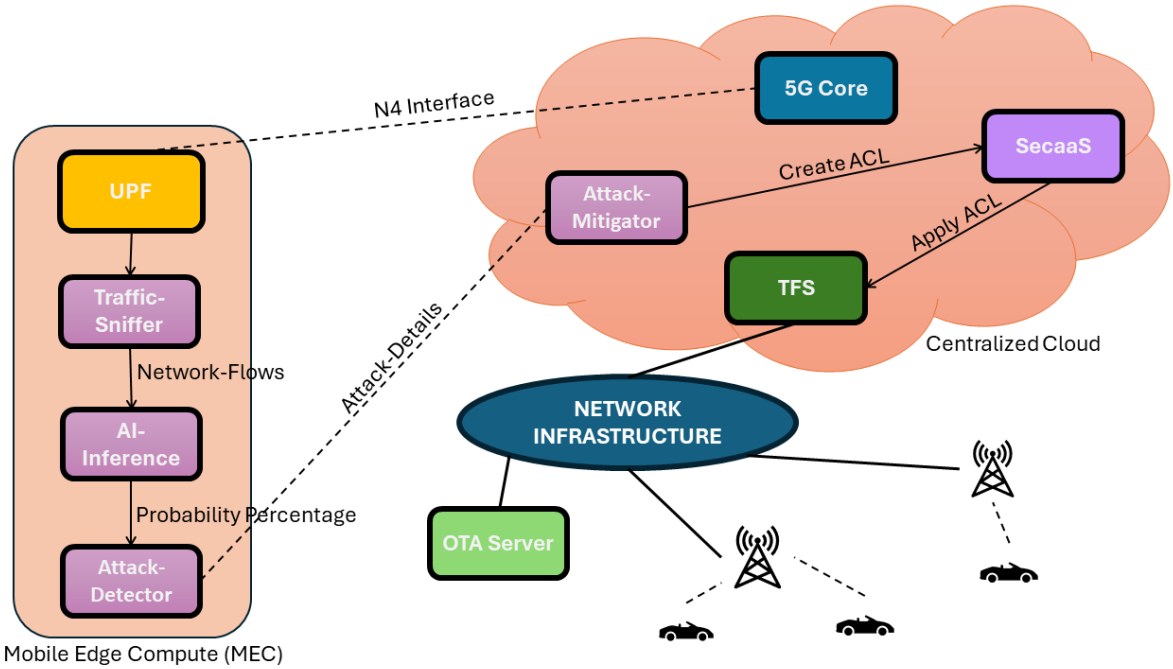

Figure 6 Specific architecture of the SECaaS platform

Figure 7 illustrates a sequence diagram representing the process of network traffic analysis, attack detection, and mitigation using Elasticsearch, Kafka, AI inference, and multiple security components. The process begins with Traffic Sniffer capturing network flows and writing them into an Elasticsearch database. This step ensures that network data is logged and stored for further processing. Once the

network flow is created and written, the Traffic Sniffer publishes the document ID of the network flow to Kafka, a real-time event streaming platform. This enables subsequent components in the pipeline to access and analyze the network data efficiently.

Following this, the AI Inference module extracts the document ID of the network flow from Kafka and processes it to determine the probability of an attack. Once the inference process is completed, the AI system publishes the attack probability back to Kafka, making it available for Attack Detector to retrieve. The Attack Detector listens to Kafka for incoming probabilities and assesses the likelihood of an attack occurring. If a potential attack is detected, the Attack Mitigator takes action by publishing the necessary information to create an Access Control List (ACL). This ACL is intended to restrict or control access to affected network entities based on the detected attack patterns.

The Attack Mitigator listens to Kafka for further information required to generate the ACL in JSON format. Once all required data is gathered, the TFS (presumably a security enforcement or traffic filtering system) creates and pushes the ACL. The final step involves TFS pushing the ACL into the Device, ensuring that security policies are enforced at the endpoint level. This process establishes a closed-loop security framework, where threats are detected, evaluated, and mitigated in an automated manner.

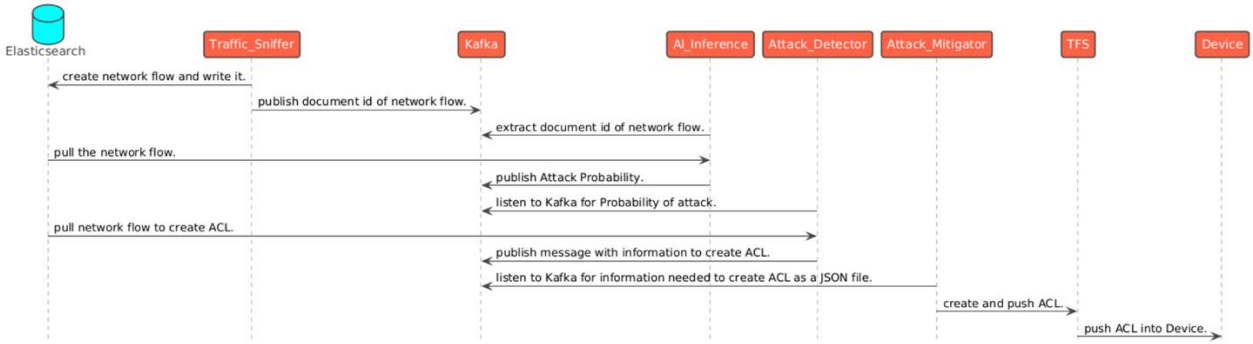

Figure 7 Sequence diagram

3.6 Preliminary experimental validation of the functionalities

Upon running the evaluation of the CLA platform, the results are shown in what follows. The Traffic Sniffer publishes into Kafka the document id of the network flow it creates, as shown in Figure 8.

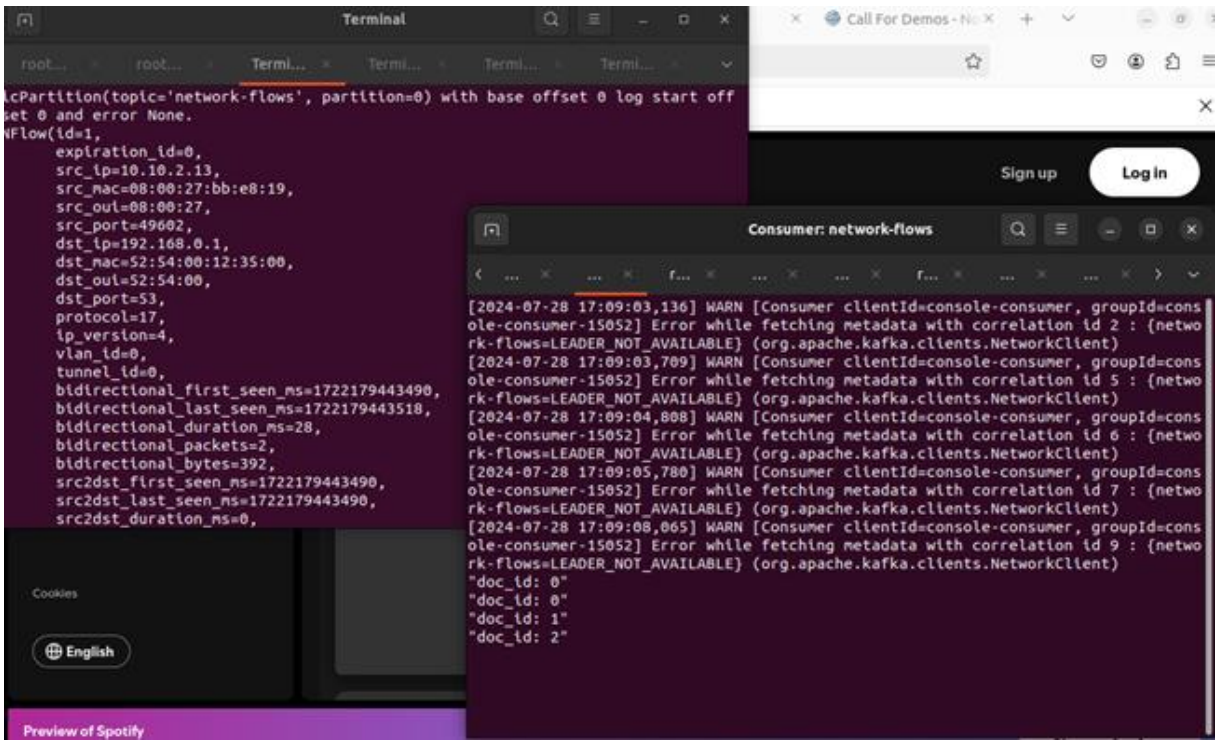

Figure 8 Creating network flows and publishing into Kafka

The NBI is used to extract network flows from the Elasticsearch database, as shown in Figure 9.

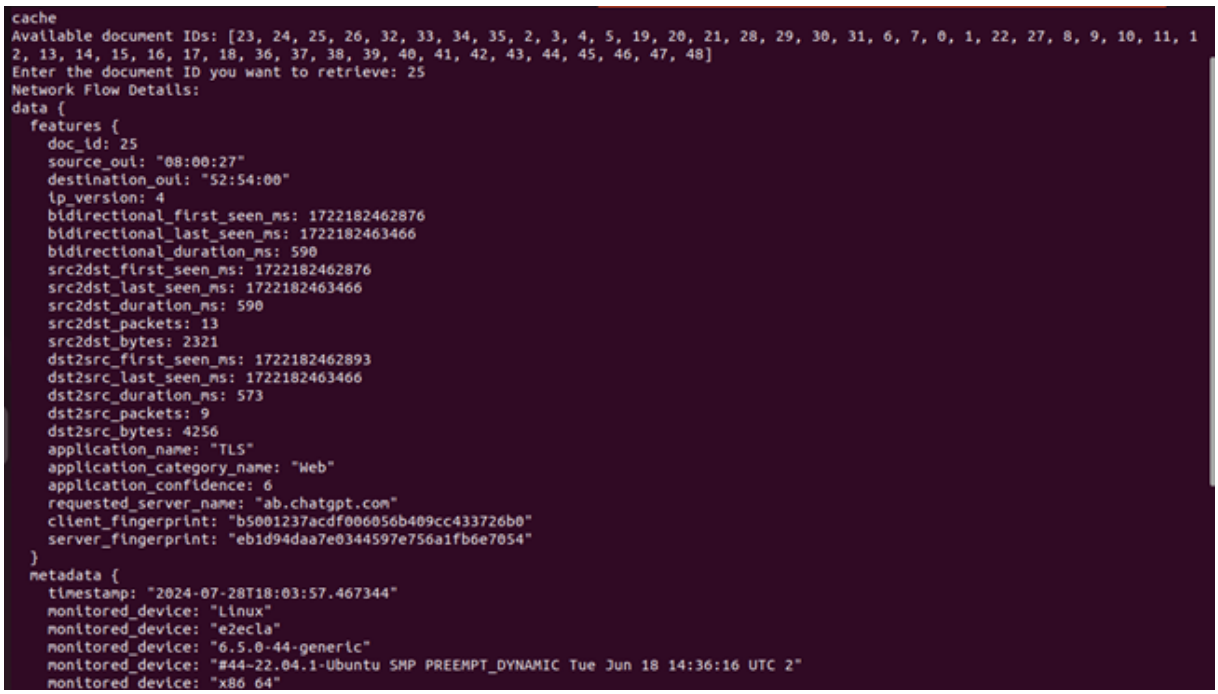

Figure 9 Extract Network Flow from Traffic Sniffer

The next step in the CLA process is the AI-inference, which extracts network flow from elasticsearch and calculates the probability of it being a PoD attack using a Random Forest Classifier machine learning algorithm. This demonstration can be seen in Figure 10. The calculated probability is published as a message in Kafka.

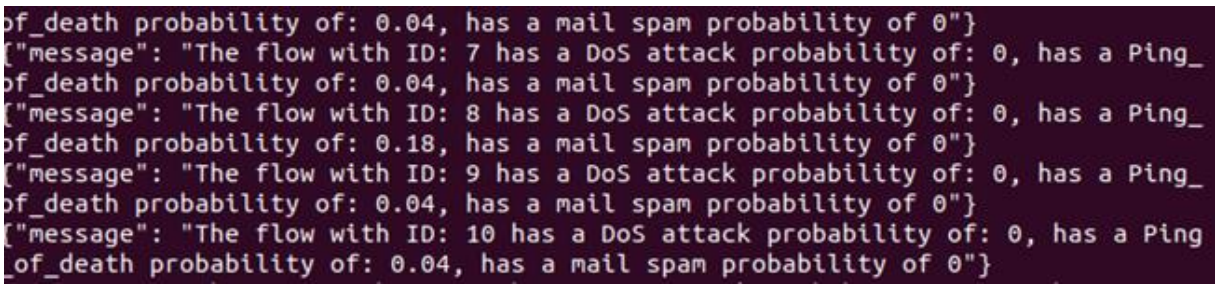

Figure 10 AI Inference message

The Attack-Detector steps in and is configured remotely if the probability published by the AI-Inference is an attack or not, as shown in Figure 11.

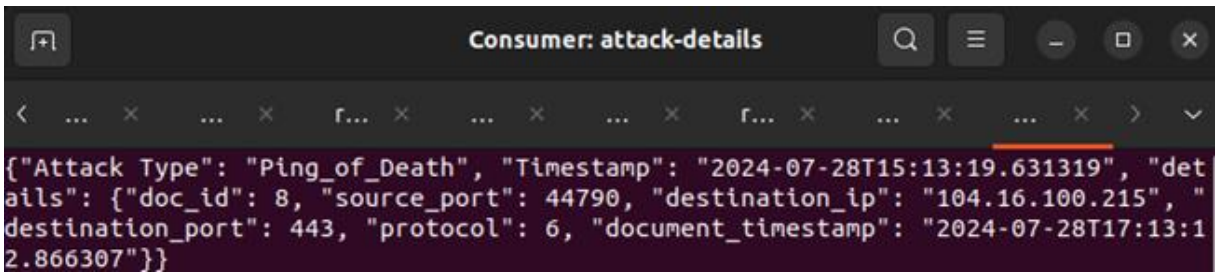

Figure 11 Attack Details Message

The user can set a threshold limit dynamically such that any probability value above the threshold is classified as an attack and published as an attack into Kafka, as shown in Figure 12.

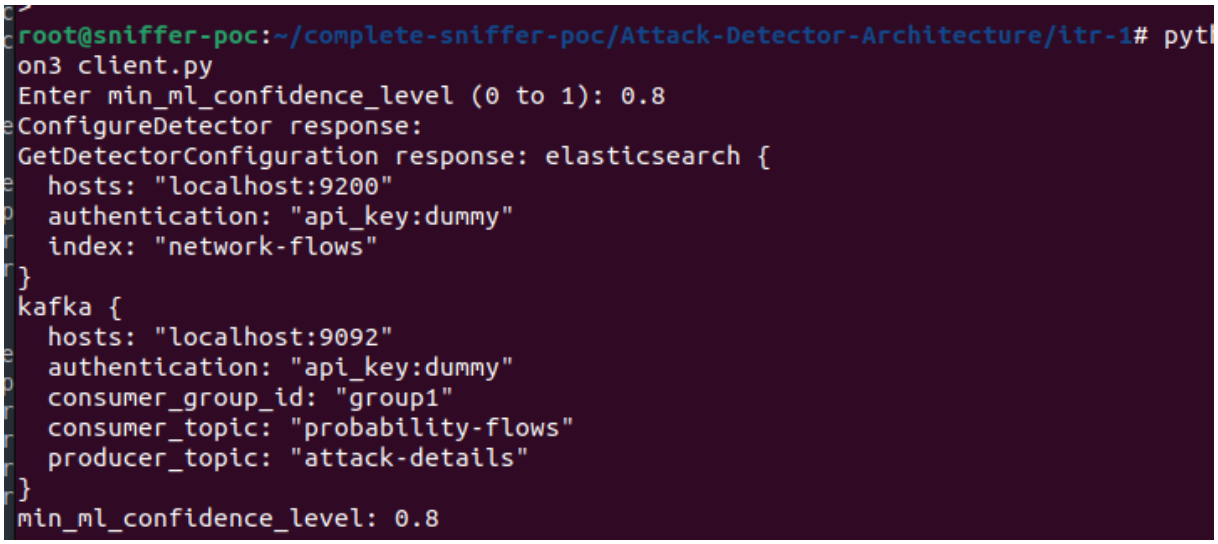

Figure 12 Attack Detector Configuration

The next step is for the Attack-Mitigator to create the ACL and post it into the TFS controller, which can be seen in Figure 13 and Figure 14.

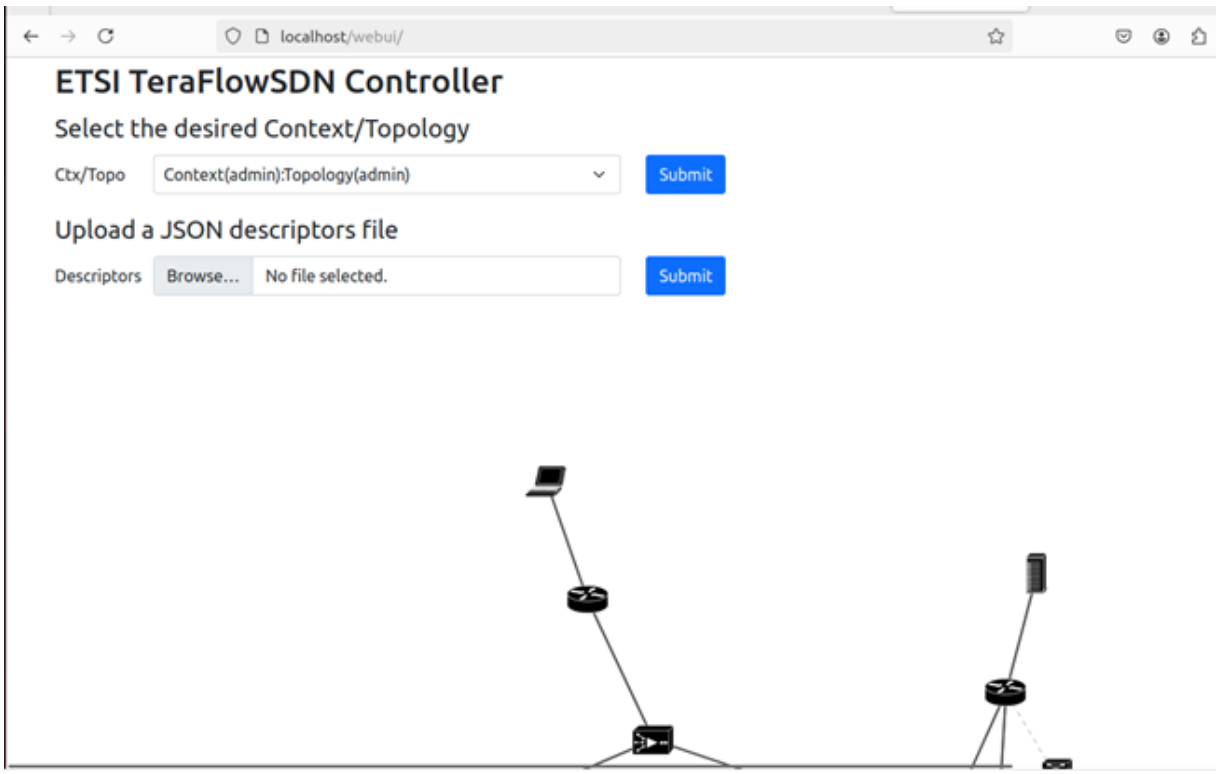

Figure 13 TFS Web UI

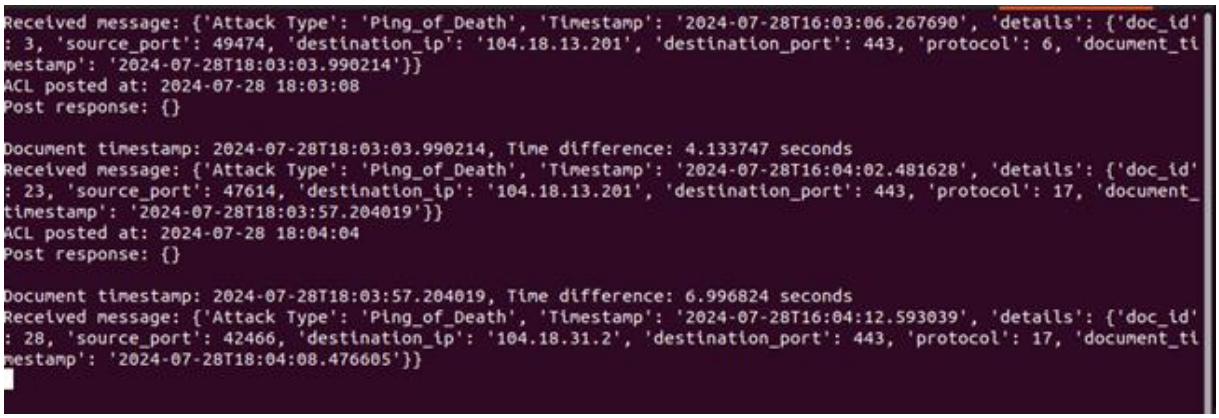

Figure 14 POST ACL into TFS

3.7 Final testing and validation

The CLA platform components are run on a single node along with Elasticsearch database installed in the Linux Machine and a single-node Kafka messaging service instead of a multi-node distributed Kafka service. On every occurrence of a network flow being produced by the Traffic Sniffer, a timestamp is added to it, and every event of AI-Inference assigning a probability of PoD attack to the network flow, a timestamp is attached to the message published in Kafka. When the Attack-Detector detects an attack a timestamp is recorded at the moment of determination of an attack, these timestamps are written as log files in the Linux machine and are later used to calculate the time taken to detect each attack, similarly a timestamp is recorded when the Attack-Mitigator posts an ACL into TFS to block malicious packets and it is written into a log file which is used to calculate the attack's MTTD and MTTR.

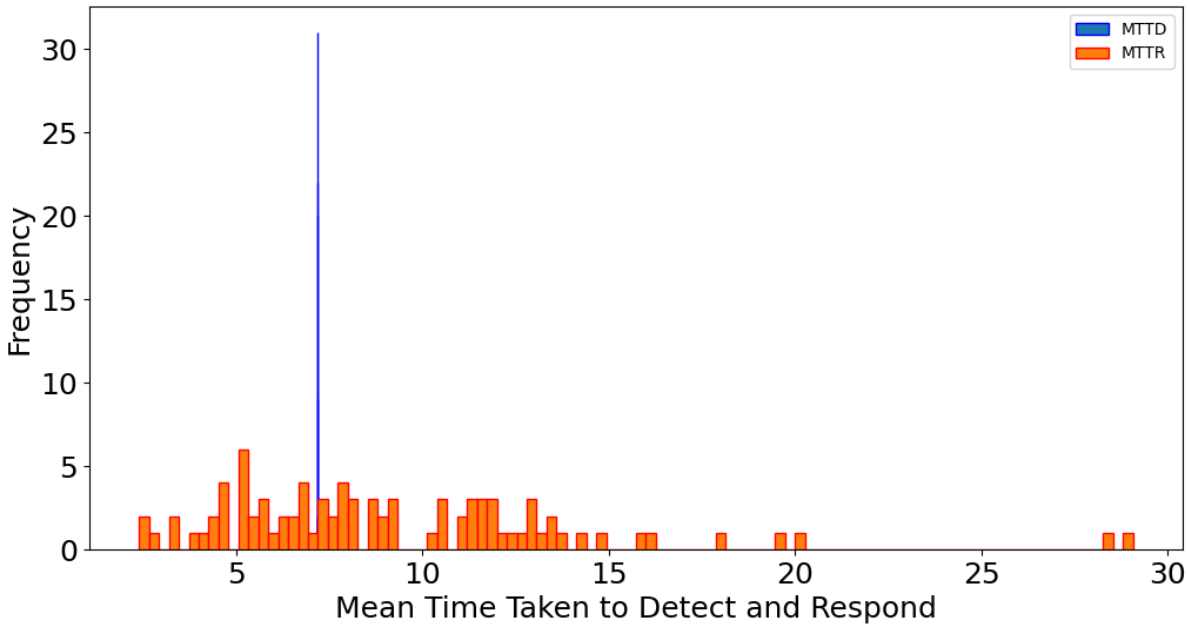

Figure 15 The Frequency of the mean time taken to detect an attack

From 2675 Network Flows, it has been observed that 88 are Ping-of-death (PoD) attacks on the Network Infrastructure. The Mean Time To Detect (MTTD) and the Mean Time To Respond (MTTR) for these attacks are calculated. The Histogram plot in Figure 15 shows the frequency of the PoD attacks and the mean time to detect these attacks and respond to them as well.

The MTTD of an attack is 7.194 seconds with a variance of 2.3e-05 seconds, and the MTTR to an attack is 9.265 seconds with a variance of 23.71 seconds. This means the MTTD for all PoD attacks was around 7.2 seconds; there were 2 instances where MTTR to a PoD attack was 28 seconds. It can be seen that MTTD is a thin bar graph at a single point, while MTTR is spread from 2.4 to 28 seconds, which aligns with the consistency of MTTD attacks as observed in Figure 15.

The placeholder AI-Inference component is trained on a dataset using the Random Forest Classifier algorithm, and the learning curve of the ML algorithm is shown in Figure 16. The precision value of the ML Model is 0.556, and the Recall value is 0.54. It is important to note that this evaluation is not about the AI-Inference component but about the novel CLA platform. A placeholder AI-Inference component that has been trained to detect PoD attacks is used to demonstrate the successful integration of the AI-Inference component in the CLA platform.

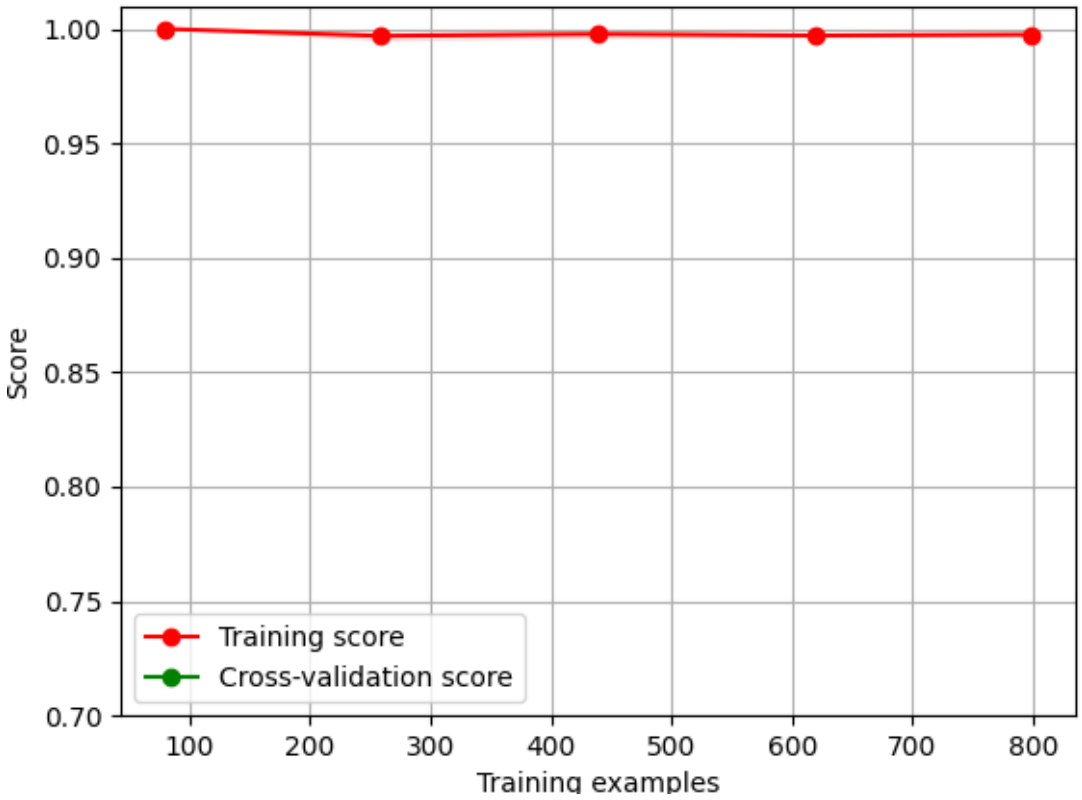

Figure 16 Random Forest Classifier Learning Curve

Finally, the cumulative distribution function (CDF) of MTTD and MTTR is shown in Figure 17. It can be seen that the cumulative mean time to detect attacks is roughly 7 seconds, as expected due to previous observations in MTTD. As seen in Figure 18, the entire spread of CDF distributions ranges from 7.175 to 7.2 seconds and around 28 seconds for MTTR, as seen in Figure 17.

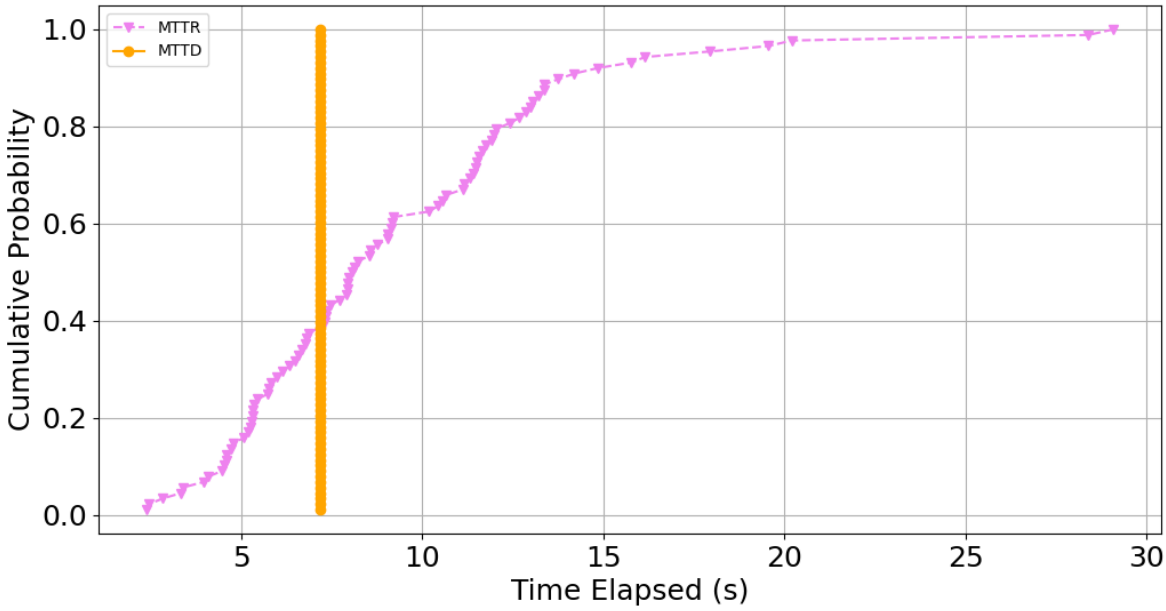

Figure 17 Cumulative Mean Time taken to detect all attacks on the system

The components in CLA that affect the MTTR are Attack-Mitigator and TFS. The performance of the Attack-Mitigator is relatively consistent, thus it can be concluded that TFS is responsible for the variance in MTTR, the time taken by TFS to post an ACL to deny/block malicious packets vary to a high degree, this can be due to numerous reasons such as overload of 'POST' ACL requests, time taken to

identify the appropriate network element to POST the ACL (generated by Attack-Mitigator), interoperability efficiency between TFS and underlay network elements. A detailed description of the CDF for MTTD is shown in Figure 18.

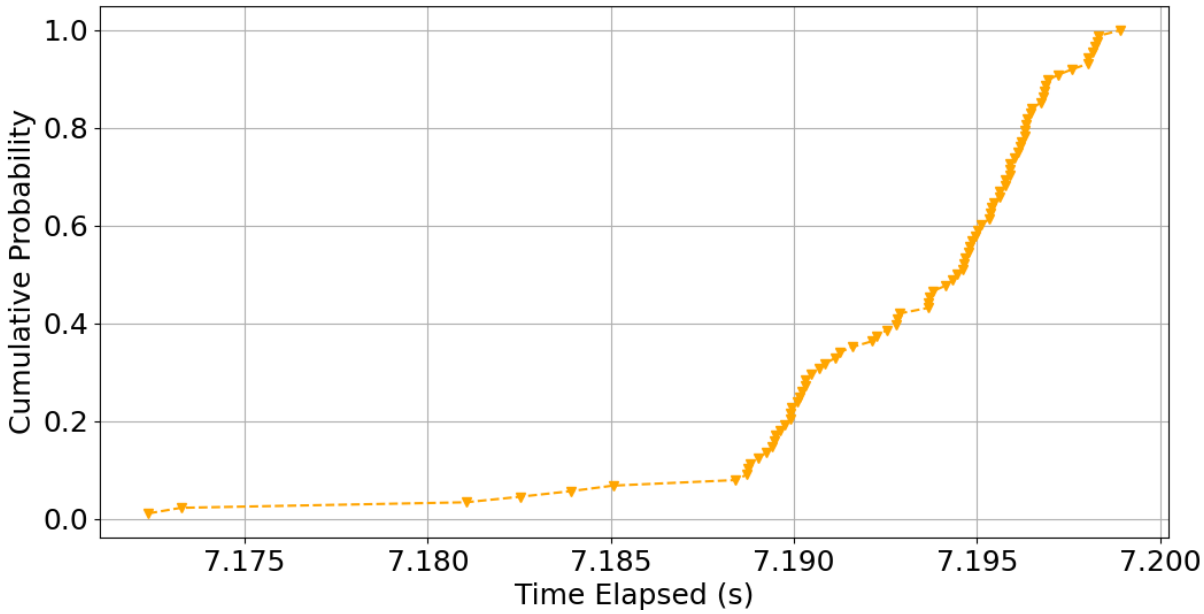

Figure 18 Cumulative Mean Time taken to detect all attacks on the system

4 Use case 2 proof-of-concept (PoC)

For the proof-of-concept of OTA vehicular software updates at the Circuit Parcmotor Castellolí, all software, both server-side and on-board unit (OBU) software, has been migrated to their final locations: the Castellolí server and the IDNEO-developed OBU installed in the test vehicle (Figure 19).

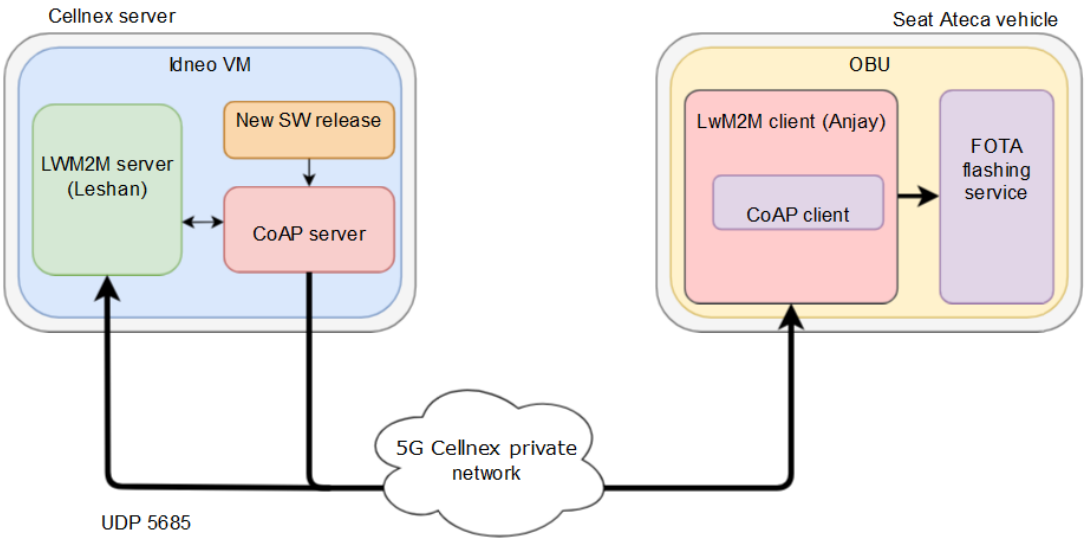

Figure 19 PoC architecture for OTA vehicular software updates at Castelloli

During the third round of tests, the focus was on evaluating network connectivity and performance in communications between the OBU and the Firmware Over-the-Air (FOTA) server. The assessment included analyzing persistent connections, reconnection processes, failure scenarios, download times, and network events that could potentially impact communication between the OBU application and the FOTA server. Additionally, the use case involving software updates from the management application was tested to ensure its functionality and reliability.

Below is the procedure followed to evaluate the use case. Figure 20 shows the dashboard used to manage the devices. There were no devices registered on the dashboard initially. This dashboard is generated by an application that, in addition to having a communications port, also has a web management port. This application was containerized and deployed at the edge of the network.

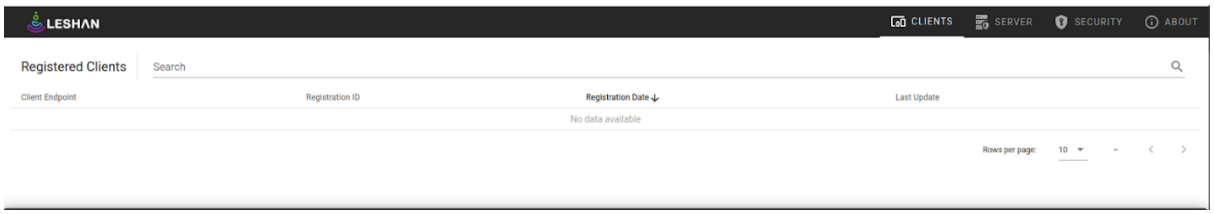

Figure 20 LWM2M server deployed on the virtual machine waiting for connections

Once the OBU was powered on, the device registration process with the server was observed, displaying the assigned endpoint name on the dashboard. Additionally, real-time logs from the client application running on the device were monitored during the test. Simultaneously, network communication packets were captured to facilitate further analysis of system performance and connectivity.

Figure 21 Top: Client connected to Server, Bottom: TCU console showing server logs

Within the management platform, relevant data can be obtained from the OBU. This information is communicated between the OBU and the server via COAP communication encrypted with DTLS. A relevant piece of information, which we can see in Figure 22, is the version of the release deployed on the OBU.

Figure 22 TCU and Server interaction, device data observation

Next, the server is instructed on the location of the file containing the new software version to be deployed. For the server, this file is considered a resource that must be transmitted as a configurable item. Prior to this step, the file was uploaded to a repository accessible via HTTPS, ensuring secure and efficient retrieval during the update process.

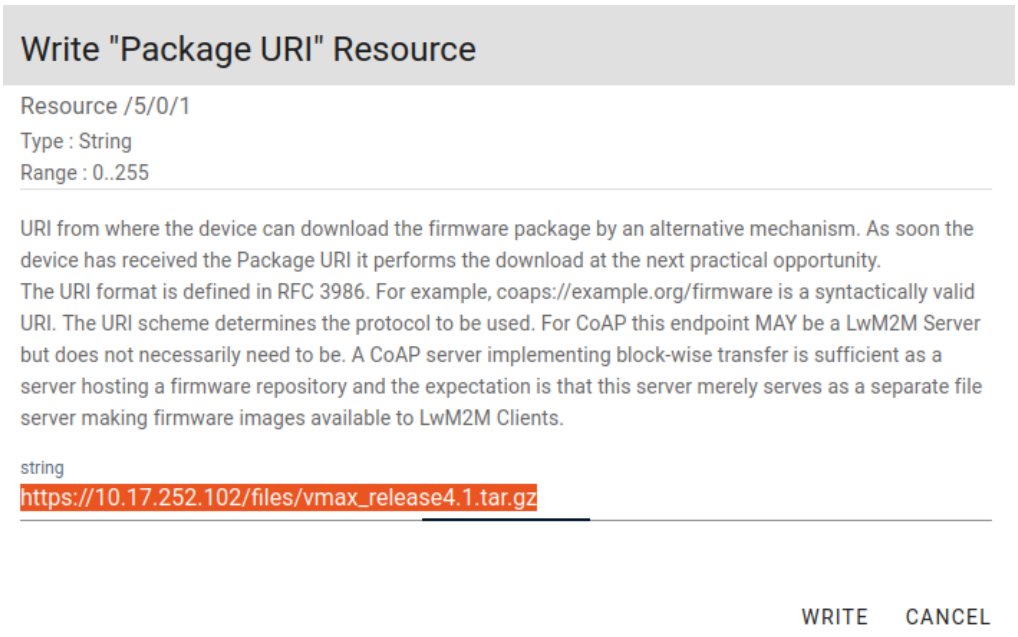

Figure 23 Top: Software Release configuration for update, Bottom: Software Releases in HTTPS repository

The file is deployed to the OBU, which downloads it once it has been instructed where to retrieve it. Figure 24 shows the *"firmware.fota"* file, which contains the updated software and is stored on the OBU's internal SD card. At this point, the Update button will be used to begin deploying the new release, since in a real-world environment, this new release will be deployed when the system is rebooted during an engine shutdown.

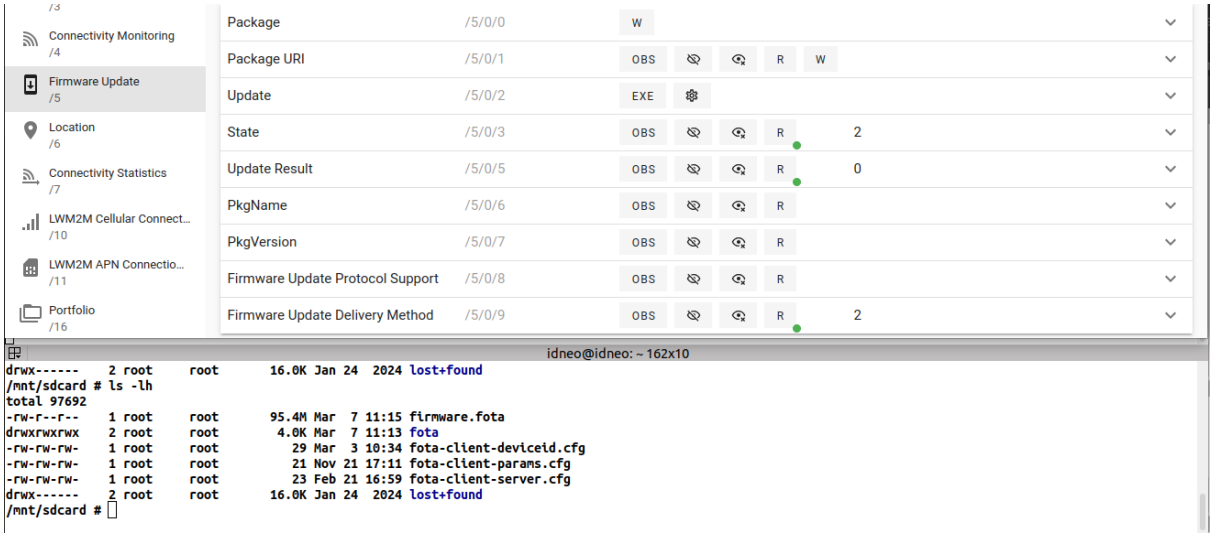

Figure 24 Top: Software Update User Interface, Bottom: Release Download to the TCU File System

Upon system reboot, the OBU executes a startup script that verifies whether a firmware update is required. If an update is necessary, the script initiates the update process and ensures its completion. Once the update is successfully applied, the OBU is expected to automatically reconnect to the edge server, as illustrated in Figure 25.

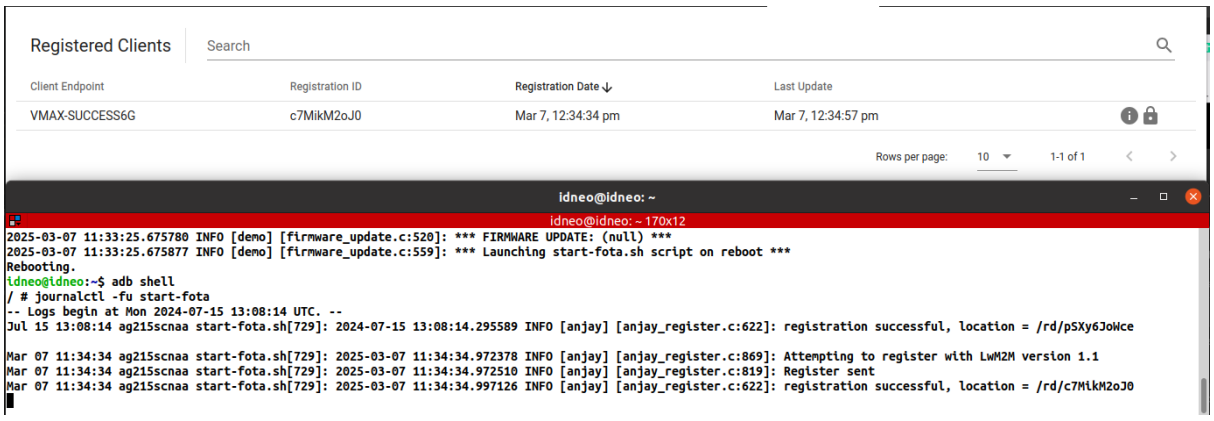

Figure 25 Reboot and automatic reconnection to the LWM2M server

A further analysis is performed based on the captured network traffic. As illustrated in Figure 26, the OBU, assigned the IP address 10.17.201.240, receives a reset command from the server at IP address 10.17.252.102. Following this, the OBU initiates a DTLS handshake to re-register with the network. This event marks the completion of the update process.

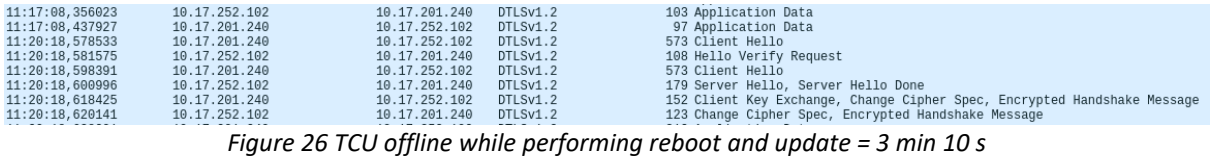

Figure 26 TCU offline while performing reboot and update = 3 min 10 s

5 Conclusions

Ensuring the security of OTA vehicular software updates is essential to protect vehicles from cyber threats and unauthorized modifications. The SUCCESS-6G-DEVISE framework demonstrates the effectiveness of AI-driven security policies, network slicing, and SECaaS in mitigating risks associated with OTA updates. The validation results show a significant reduction in security vulnerabilities, enhancing overall trust in the software update process. As cybersecurity threats continue to evolve, ongoing research into AI-driven threat detection, blockchain-based verification, and real-time anomaly monitoring will be critical in maintaining the integrity and resilience of vehicular software updates.

One of the key challenges in secure OTA updates is maintaining end-to-end encryption while minimizing performance trade-offs. The use of blockchain-based verification techniques ensures that software packages remain tamper-proof throughout the update process. By leveraging distributed ledger technology, updates can be validated across multiple nodes, reducing the risk of data breaches and unauthorized alterations.

Additionally, integrating AI-driven threat detection mechanisms enhances the real-time security posture of the update ecosystem. By continuously monitoring network activity and analyzing behavioral anomalies, the system can proactively identify and mitigate potential cyber threats. Future advancements should focus on expanding adaptive security frameworks that dynamically respond to emerging attack vectors, ensuring robust protection against evolving cyber risks.

6 References

Abishek A, Vilalta R, Gifre L, Alemany P, Manso C, Casellas R, Martínez R, Muñoz R. Network Extensions to Support Robust Secured and Efficient Connectivity Services for V2X Scenario. In2024 24th International Conference on Transparent Optical Networks (ICTON) 2024 Jul 14 (pp. 1-4). IEEE.

Vilalta R, Vía S, Mira F, Casellas R, Muñoz R, Alonso-Zarate J, Kousaridas A, Dillinger M. Control and management of a connected car using sdn/nfv, fog computing and yang data models. In2018 4th IEEE Conference on Network Softwarization and Workshops (NetSoft) 2018 Jun 25 (pp. 378-383). IEEE.

Muñoz R, Vilalta R, Yoshikane N, Casellas R, Martínez R, Tsuritani T, Morita I. Integration of IoT, transport SDN, and edge/cloud computing for dynamic distribution of IoT analytics and efficient use of network resources. Journal of Lightwave Technology. 2018 Apr 1;36(7):1420-8.

Asensio-Garriga, R. *et al*., "ZSM-Based E2E Security Slice Management for DDoS Attack Protection in MEC-Enabled V2X Environments," in *IEEE Open Journal of Vehicular Technology*, vol. 5, pp. 485-495, 2024

Fallgren M, Dillinger M, Alonso-Zarate J, Boban M, Abbas T, Manolakis K, Mahmoodi T, Svensson T, Laya A, Vilalta R. Fifth-generation technologies for the connected car: Capable systems for vehicle-to-anything communications. IEEE vehicular technology magazine. 2018 Jul 17;13(3):28-38.

Garg S, Kaur K, Kaddoum G, Ahmed SH, Jayakody DN. SDN-based secure and privacy-preserving scheme for vehicular networks: A 5G perspective. IEEE Transactions on Vehicular Technology. 2019 May 20;68(9):8421-34.

Varma IM, Kumar N. A comprehensive survey on SDN and blockchain-based secure vehicular networks. Vehicular Communications. 2023 Aug 22:100663.

Meyer P, Hackel T, Langer F, Stahlbock L, Decker J, Eckhardt SA, Korf F, Schmidt TC, Schüppel F. A security infrastructure for vehicular information using sdn, intrusion detection, and a defense center in the cloud. In2020 IEEE Vehicular Networking Conference (VNC) 2020 Dec 16 (pp. 1-2). IEEE.