x O-RAN.WG11.Security-Near-RT-RIC-xApps-TR.0-R003-v05.00

Technical Report

O-RAN Work Group 11 (Security Work Group)

Study on Security for Near Real Time RIC and xApps

Contents

Foreword

This Technical Report (TR) has been produced by O-RAN Alliance.

Modal verbs terminology

In the present document "**shall**", "**shall not**", "**should**", "**should not**", "**may**", "**need not**", "**will**", "**will not**", "**can**" and "**cannot**" are to be interpreted as described in clause 3.2 of the O-RAN Drafting Rules (Verbal forms for the expression of provisions).

"**must**" and "**must not**" are **NOT** allowed in O-RAN deliverables except when used in direct citation.

Introduction

The Near-Real Time RAN Intelligent Controller (Near-RT RIC) is a new virtualized function that adds RAN programmability to existing or new RAN networks, e.g., self-organizing networks (SON) type of functions. It is a platform designed to use AI/ML-based tools with open interfaces to the RAN (E2) and the management and orchestration system (A1, O1).

The Near-RT RIC enables near real time control and optimization of RAN elements and resources via fine-grained data (e.g., UE basis, Cell basis) collection and actions over E2 interface, and includes open interfaces (A1, O1) to the management and orchestration system (SMO).

The Near-RT RIC platform is a software-based near-real-time microservice-based platform for hosting microservice-based applications – the xApps – that run on the Near-RT RIC platform. A xApp may consist of one or more microservices and at the point of onboarding it identifies which data it consumes and which data it provides. It is independent of Near-RT RIC and may be provided by any third party. The E2 interface enables a direct association between xApp and RAN functionalities.

1 Scope

The contents of the present document are subject to continuing work within O-RAN and may change following formal O-RAN approval. Should the O-RAN Alliance modify the contents of the present document, it will be re-released by O-RAN with an identifying change of version date and an increase in version number as follows:

version xx.yy.zz

where:

xx: the first digit-group is incremented for all changes of substance, i.e. technical enhancements, corrections, updates, etc. (the initial approved document will have xx=01). Always 2 digits with leading zero if needed.

Yy: the second digit-group is incremented when editorial only changes have been incorporated in the document. Always 2 digits with leading zero if needed.

Zz: the third digit-group included only in working versions of the document indicating incremental changes during the editing process. External versions never include the third digit-group. Always 2 digits with leading zero if needed.

The present document considers the security aspects of the Near-RT RIC platform and xApps, as well as the associated network and management interfaces (E2, A1, O1) and APIs.

Taking into consideration the threats and risks identified in O-RAN Threat Model specification [1], the study analyses a list of key issues in the Near-RT RIC overall architecture and proposes solutions in order to derive security requirements, intended to be part of O-RAN Security Requirements Specifications [2].

2 References

2.1 Informative references

References are either specific (identified by date of publication and/or edition number or version number) or non-specific. For specific references, only the cited version applies. For non-specific references, the latest version of the referenced document (including any amendments) applies.

NOTE: While any hyperlinks included in this clause were valid at the time of publication, O-RAN cannot guarantee their long term validity.

The following referenced documents are not necessary for the application of the present document but they assist the user with regard to a particular subject area.

[1] O-RAN ALLIANCE TR: “O-RAN Security Threat Modeling and Risk Assessment”

[2] O-RAN ALLIANCE TS: “Security Requirements and Controls Specification”

[3] O-RAN ALLIANCE TS: “Near-RT RIC Architecture ”

[4] O-RAN ALLIANCE TS: “Non-RT RIC and A1 Interface Use Cases and Requirements ”

[5] O-RAN ALLIANCE TS: “E2 General Aspects and Principles (E2GAP)”

[6] O-RAN ALLIANCE TS: “O1 Interface specification for O-DU”

[7] O-RAN ALLIANCE TS: “O1 Interface specification for O-CU-UP and O-CU-CP”

[8] O-RAN ALLIANCE TS: “O-RAN Operations and Maintenance Interface Specification (O1)”

[9] O-RAN ALLIANCE TS: “A1 interface: General Aspects and Principles”

[10] OWASP API Security Top 10 2019. https://owasp.org/www-project-api-security

[11] RFC 6749: The Oauth 2.0 Authentication framework.
https://datatracker.ietf.org/doc/html/rfc6749

[12] RFC 6819: OAuth 2.0 threat model and security considerations.
https://datatracker.ietf.org/doc/html/rfc6819

[13] 3GPP TS 33.501: “Security architecture and procedures for 5G system”

[14] ETSI NFV SEC022: “Network Functions Virtualization (NFV) Release 2; Security; Access Token Specification for API Access”.

[15] ETSI GS MEC 009 V3.1.1 (2021-06): Multi-access Edge Computing (MEC); General principles, patterns and common aspects of MEC Service APIs.
https://www.etsi.org/deliver/etsi_gs/MEC/001_099/009/03.01.01_60/gs_MEC009v030101p.pdf

[16] 3GPP TR 33.845 v16.1.0 (2020-09): Study on security aspects of the 5G Service Based Architecture (SBA)

[17] RFC 7519: JSON Web Token. https://datatracker.ietf.org/doc/html/rfc7519

[18] RFC 7515: JSON Web Signature. https://datatracker.ietf.org/doc/html/rfc7515

[19] O-RAN ALLIANCE TS: “Security Protocols Specifications”

[20] 3GPP TS 33.310: “Network Domain Security (NDS); Authentication Framework (AF)”.

[21] 3GPP TS 33.210: “3G security; Network Domain Security (NDS); IP network layer security”.

[22] O-RAN ALLIANCE TS: “O-RAN Operations and Maintenance Architecture”

[23] O-RAN ALLIANCE TS, “O-RAN Orchestration Use Cases and Requirements for O-RAN Virtualized RAN”

[24] Executive Order 14028, Improving the Nation’s Cybersecurity: Security Measures for EO-Critical Software Use. https://www.nist.gov/itl/executive-order-improving-nations-cybersecurity/security-measures-eo-critical-software-use

[25] OpenSSF Best Practices. https://bestpractices.coreinfrastructure.org/en/projects?q=O-RAN

[26] OpenSSF Software Development Fundamentals Link: https://openssf.org/training/courses/

[27] OpenSSF Best Practices Badge Program: https://bestpractices.coreinfrastructure.org/en

[28] CISA NSA 5G Security guideline https://media.defense.gov/2021/Oct/28/2002881720/-1/-1/0/SECURITY_GUIDANCE_FOR_5G_CLOUD_INFRASTRUCTURES_PART_I_20211028.PDF

[29] Opensource Software "Security 5G Security" whitepaper. https://www.5gamericas.org/security-for-5g/

[30] NIST Special Publication (SP) 800-40, Guide to Enterprise Patch Management Technologies. https://csrc.nist.gov/publications/detail/sp/800-40/rev-4/draft

[31] 3GPP TS 33.122: "Security aspects of Common API Framework (CAPIF) for 3GPP northbound APIs"

[32] O-RAN ALLIANCE TS: "O-RAN Architecture Description"

[33] 3GPP TS 23.501: "System Architecture for the 5G System (5GS); Stage 2".

[34] RAIE Impacts on Near-RT RIC Architecture; discussion document, CMCC-2022.12.12-WG3-C-RAIE Impacts on RICARCH-v2.pptx

[35] Solving the Bottom Turtle — a SPIFFE Way to Establish Trust in Your Infrastructure via Universal Identity: https://spiffe.io/pdf/Solving-the-bottom-turtle-SPIFFE-SPIRE-Book.pdf

[36] SPIRE Concepts: An overview of SPIRE's architecture and fundamentals: https://spiffe.io/docs/latest/spire-about/spire-concepts/

[37] Kubernetes Documentation: PKI certificates and requirements: https://kubernetes.io/docs/setup/best-practices/certificates/

[38] Plugin_agent_workloadattestor_k8s: https://github.com/spiffe/spire/blob/v1.7.0/doc/plugin_agent_workloadattestor_k8s.md

[39] xApp_Writer_s_Guide_v2.pdf: https://wiki.o-ran-sc.org/download/attachments/17269011/xApp_Writer_s_Guide_v2.pdf?version=4&modificationDate=1625642899082&api=v2

[40] SPIFFE-ID: https://github.com/spiffe/spiffe/blob/master/standards/SPIFFE-ID.md

[41] NIST Special Publication 800-207A: https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-207A.pdf

[42] Secure Software Development Fundamentals Courses: https://openssf.org/training/courses/

[43] O-RAN Core Infrastructure Initiative (CII) Badging: https://wiki.o-ran-sc.org/display/ORAN/Core+Infrastructure+Initiative+%28CII%29+Badging

[44] O-RAN Software Community Projects: https://wiki.o-ran-sc.org/display/ORAN/Projects

3 Definition of terms, symbols and abbreviations

3.1 Terms

For the purposes of the present document, the following terms apply:

Near-RT RIC: O-RAN Near-Real-Time RAN Intelligent Controller: A logical function that enables near-real-time control and optimization of RAN elements and resources via fine-grained data collection and actions over E2 interface.

Non-RT RIC: O-RAN Non-Real-Time RAN Intelligent Controller: A logical function within SMO that drives the content carried across the A1 interface. It is comprised of the Non-RT RIC Framework and the Non-RT RIC Applications.

O-CU-CP: O-RAN Central Unit – Control Plane: a logical node hosting RRC and the control plane part of PDCP protocol.

O-CU-UP: O-RAN Central Unit – User Plane: a logical node hosting the user plane part of PDCP protocol and SDAP protocol.

O-DU: O-RAN Distributed Unit: a logical node hosting RLC/MAC/High-PHY layers based on a lower layer functional split.

O-RU: O-RAN Radio Unit: a logical node hosting Low-PHY layer and RF processing based on a lower layer functional split. This is similar to 3GPP's "TRP" or "RRH" but more specific in including the Low-PHY layer (FFT/iFFT, PRACH extraction).

O-eNB: An eNB or ng-eNB that supports E2 interface.

O1: An interface between SMO and O-RAN managed elements, for operation and management, by which FCAPS management, PNF (Physical Network Function) Software management, File management shall be achieved.

SMO: A Service Management and Orchestration system.

A1: An interface between Non-RT RIC and Near-RT RIC to enable policy-driven guidance of Near-RT RIC applications/functions, and support AI/ML workflow.

E2: An interface connecting Near-RT RIC and one or more O-CU-CPs, one or more O-CU-Ups, or one or more O-Dus.

Y1: An interface between Near-RT RIC and Y1 consumers [32]. The interface enables RAN analytics information exposure from Near-RT RIC.

E2 Node: A logical node terminating E2 interface. In this version of the specification, O-RAN nodes terminating E2 interface are:

- for NR access: O-CU-CP, O-CU-UP, O-DU
- for E-UTRA access: O-eNB.

xApp: An application designed to run on Near-RT RIC. It may consist of one or more microservices and at the point of on-boarding it identifies which data it consumes and which data it provides. It is independent of Near-RT RIC and may be provided by any third party. The E2 interface enables a direct association between xApp and RAN functionalities.

O-Cloud: O-Cloud is a cloud computing platform comprising a collection of physical infrastructure nodes that meet O-RAN requirements to host the relevant O-RAN functions (such as Near-RT RIC, O-CU-CP, O-CU-UP, and O-DU), the supporting software components (such as Operating System, Virtual Machine Monitor, Container Runtime, etc.) and the appropriate management and orchestration functions.

Solution Provider: An application developer who delivers applications to Service Providers. [22]

Service Provider: A network provider who is planning to deploy applications into their network. [22]

3.2 Symbols

For the purposes of the present document, the following symbols apply:

3.3 Abbreviations

For the purposes of the present document, the following abbreviations apply:

API	Application Programming Interface
AI/ML	Artificial Intelligence/Machine Learning
CP	Control Plane
DoS	Denial of Service
DTLS	Datagram Transport Layer Security
eNB	e NodeB (applies to LTE)
E2SM	E2 Service Model
FCAPS	Fault, Configuration, Accounting, Performance, Security
FM	Fault Management

gNB	g NodeB (applies to NR)
IPsec	Internet Procotol Security
JWS	JSON Web Signature
JWT	JSON Web Token
LCM	Life-Cycle Management
MEC	Multi-access Edge Computing
MIMO	Multiple Input Multiple Output
Near-RT RIC	Near-real-time RAN Intelligent Controller
NIB	Network Information Base
Non-RT RIC	Non-real-time RAN Intelligent Controller
O-CU	O-RAN Central Unit
O-DU	O-RAN Distributed Unit
O-RU	O-RAN Radio Unit
PM	Performance Management
QoS	Quality of Service
QoE	Quality of Experience
RAIE	RAN Analytic Information Exposure
RAI API	RAN Analytics Information API, internal to the Near-RT RIC
RAN	Radio Access Network
RIC	RAN Intelligent Controller
R-NIB	Radio-Network Information Base
RRM	Radio Resource Management
SBA	Service Based Architecture
SDL	Share Data Layer
SMO	Service Management and Orchestration
TLS	Transport Layer Security
UE	User Equipment
UE-NIB	UE-Network Information Base
UP	User Plane
Y1 API	RAN Analytics Information API, for consumers outside of the Near-RT RIC

Y1T	Y1 API Termination, i.e. Near-RT RIC platform function implementing a Y1 API provider

4 RIC background, concepts and assumptions

4.1 Introduction

The RIC architecture consists of the Non-RT RIC, which deals with control loops of more than 1 second, and the Near-RT RIC, with control loops in the order of 10 ms to 1s [3]. The Non-RT RIC is embedded in the Service and Management Orchestration function and hosts rApps to provide policy-based guidance, machine learning model management and enrichment information to the Near-RT RIC function for the purpose of RAN optimization [4]. The Near-RT RIC is a logical function that enables near real-time control and optimization of E2 Nodes functions and resources, steered via the policies and enrichment data provided from the Non-RT RIC [3].

4.2 Near-RT RIC architecture

The following diagram shows the building blocks of the Near-RT RIC internal architecture. For more detailed descriptions of the architecture and platform functions, see WG3 Near-RT RIC Architecture specification [3].

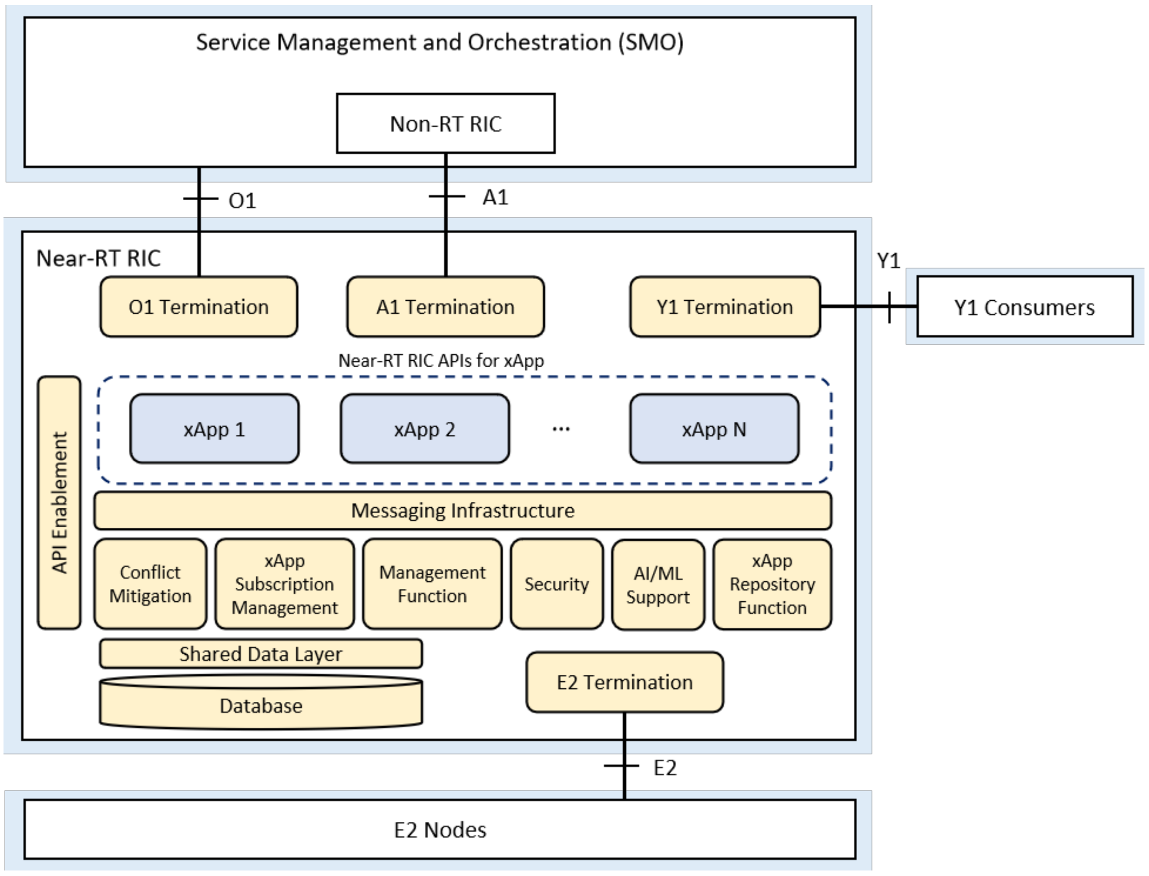


Figure 4.2-1: Near-RT RIC Internal Architecture

- **Interface Termination** for E2, A1, O1, Y1
- **xApp:** application that runs on the Near-RT RIC to provide desired RAN functionality
- **Messaging infrastructure:** provides low-latency message delivery service b/t Near-RT RIC internal endpoints
- **Conflict mitigation:** resolves potentially overlapping or conflicting requests from multiple xApps
- **Subscription management:** manages subscriptions from xApps to E2 Nodes and enforces authorization of policies
- **Management services:** life-cycle management of xApps (onboarding, deployment, resource management, and termination) and FCAPS management of Near-RT RIC (logging, tracing, metrics collection)
- **Security:** provides the security scheme for the xApps. One of the targets of this function is to prevent malicious xApps from abusing radio network information (e.g. exporting to unauthorized external systems) and/or control capabilities over RAN functions

  Note. The Security function is a placeholder in Near-RT RIC architecture and has not been specified yet. The requirements and solutions proposed in this study will support the specification of this function.

- **AI/ML support:** provides data ingestion and preparation for xApps and AI/ML training for xApps
- **xApp Repository Function:** performs selection of xApps for A1 message routing and provides policy types supported in Near-RT RIC to the A1 termination function
- **Shared Data Layer (SDL):** API for accessing shared data storage
- **Database:** UE-NIB database contains a list of Ues and associated data, and Radio-NIB database stores configurations and information relating to the connected E2 Nodes
- **API Enablement:** support for registration, discovery and consumption of Near-RT RIC APIs
- **Y1 Consumers:** communicate via Y1 interface and consume RAN analytics information service(s) from Near-RT RIC

4.3 Interfaces

E2 : Interface between the Near-RT RIC and the E2 Nodes (O-DU, O-CU-UP, O-CU-CP, O-eNB) [5]

O1 : Interface between the SMO and O-RAN Managed Functions (Near-RT RIC, O-RU, O-DU, O-CU-UP, O-CU-CP, O-eNB) [6][7][8]

A1 : Interface between the Non-RT RIC at the SMO and the Near-RT RIC [9]

Y1: communicates with Y1 consumers via Y1 interface and exposes RAN analytics information service(s) from Near-RT RIC.

4.4 xApps

As per definition in [3], an xApp is an application designed to run on Near-RT RIC. It may consist of one or more microservices and at the point of on-boarding it identifies which data it consumes and which data it provides. It is independent of Near-RT RIC and may be provided by any third party. The E2 interface enables a direct association between xApp and RAN functionalities.

From security viewpoint the following requirements [3] are key aspects to take into consideration in this study, and will be developed in subsequent key issues and solutions:

- xApps shall communicate with Near-RT RIC platform via Near-RT RIC APIs.
- xApps shall use Near-RT RIC APIs to make use of the Information Elements (Ies) **<u>only</u>** of the E2 Service Models (E2SM) that are associated with it.

  Note. E2SM describes the functions in the E2 Node which may be controlled by the Near-RT RIC and corresponding procedures.
- xApp shall register the APIs it produces.

  Note. Services for which the API producer is the xApp are FFS
- xApp shall be capable of discovering the Near-RT RIC APIs they consume
- The platform will consider the level of trust related to the xApp, e.g. 3rd party xApp, RIC-owned xApp (i.e. xApp provided by the RIC vendor)

4.5 Near-RT RIC APIs

The following diagram [3] provides a representation of the APIs involved in Near-RT RIC.

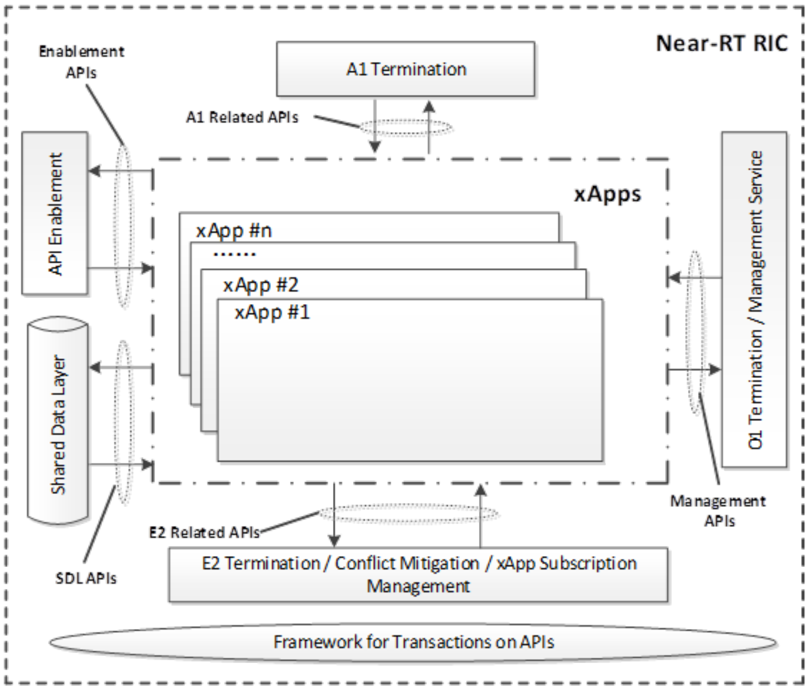


Figure 4.5-1: Overview of Near-RT RIC APIs

- **A1 related APIs**: APIs allowing to access A1 related functionality.
    - **Policy Enforcement API** (transactional)
    - **Enrichment information API** (transactional)
- **E2 related APIs**: APIs allowing to access E2 related functionality and associated xApp Subscription Management and Conflict Mitigation functionality.
    - **E2 Subscription API** (transactional)
    - **E2 Guidance** (transactional)
    - **E2 Indication** (time critical, uni-streaming)
    - **E2 Control** (time critical, bi-streaming)
- **Management APIs**: APIs allowing to access management related functionality. They are used for services such as O1 Termination, management services and logging, tracing, metrics collection.
    - **ML model related APIs** (FFS)
    - **FCAPS related APIs** (transactional):
        - **xApp Registration API**
        - **xApp Deregistration API**
        - **Configuration API**

        - **PM** (Performance Management) **API**
        - **FM** (Fault Management) **API**
- **SDL API** (transactional + Uni-streaming): API allowing to access Shared Data Layer related functionalities (store/fetch/notify/modify/push/…) over R-NIB, UE-NIB, other data in the database.
- **Enablement APIs** (transactional): APIs between xApps and API enablement functionality.
    - **Registration API.** Enabling the API producer to register the APIs it produces
    - **Discovery API.** Enabling the discovery of the Near-RT RIC APIs by the xApps
    - **Subscription API.** Enabling the subscription/un-subscription to API-related events and event notification.

5 Key Issues

5.1 Introduction

This clause details the key issues identified for security aspects related to the Near-RT RIC architecture, relevant interfaces, xApps and APIs. Each key issue defines the background to the issue, defines the threats related to the issue and proposes requirements that resolve or mitigate the key issue.

5.2 Key Issue #1: Malicious xApps deployed on the Near-RT RIC

5.2.1 Key issue detail

xApps on the Near-RT RIC can collect near real-time information from E2 Nodes and influence behavior of E2 Nodes thereby impacting cell performance and delivery of services to a group of Ues or a single UE. The security requirements for onboarding and hosting xApps should be defined to reduce the risk of unauthorized, untrusted, or malicious xApps to be deployed into the Near-RT RIC.

Malicious xApps could be onboarded onto the Near-RT RIC with the intent of manipulating or negatively impacting the delivery of services to subscribers, the RAN, or the core network. Malicious xApps could also be onboarded onto the Near-RT RIC with the intent of accessing, manipulating or negatively impacting the privacy of subscribers, the RAN or the core network.

This threat has been identified in O-RAN Threat Model [1] as Threat ID T-NEAR-RT-01.

5.2.2 Security threats

The security threats associated with the onboarding and deployment of malicious xApps include:

- Malicious xApps attaining unauthorized access to the Near-RT RIC and E2 Nodes
- Malicious xApps abusing radio network information and control capabilities over RAN functions
- Malicious xApp impacting service for a subscriber or a dedicated area
- Malicious xApp exploiting UE identification, tracking UE location and changing UE slice priority

5.2.3 Potential security requirements

xApps shall be authenticated during onboarding using a signature that is generated by the xApp Solution Provider and validated by the Service Provider.

xApps shall be validated during Registration to the Near-RT RIC platform using signatures from both the Service Provider and the xApp Solution Provider.

5.3 Key Issue #2: Compromised xApp

5.3.1 Key issue detail

Vulnerabilities can potentially exist in any xApp. Those can be e.g.:

- Security misconfiguration, such as enabled but unnecessary protocols
- Weak or misconfigured authentication and authorization
- A malicious xApp an untrusted source may intentionally provide
- A backdoor intentionally inserted in the xApp by a trusted source
- An attacker submitting requests without prior authentication and authorization by executing an injection attack to manipulate configurations, access logs, perform remote code execution, etc

If attackers can find an exploitable xApp, they can disrupt the offered network service and potentially take over another xApp or the Near-RT RIC. The actual consequences may vary. For example, an attacker may gain the ability to alter data transmitted by any accessible API, extract sensitive information, etc.

5.3.2 Security Threats

The security threats associated with compromised xApps include:

- Security misconfiguration
- Weak authentication and authorization

- Malicious xApps from an untrusted source
- Backdoors in trusted xApps
- Vulnerabilities in trusted apps

5.3.3 Potential security requirements

Requirement: xApps images shall be authenticated during onboarding using a signature that is generated by the xApp Solution Provider and validated by the Service/Cloud Provider and telco operator.

Requirement: xApps instances shall be validated during Registration to the Near-RT RIC platform using signatures from both the Service/Cloud Provider and the xApp Solution Provider. Signatures made from certificates reaching the end of their lifetime shall be renewed before the certificate times out (signatures provided by the xApp Solution Provider might be ignored if the signature of the Service/Cloud Provider is valid).

Recommendation: xApps should be hardened by the vendor, and this hardening should be tested by the service provider, using a well-defined standard, e.g., using penetration tests

Requirement: xApp APIs shall be provided with proper Authentication/Authorization schemas.

Recommendation: prevent supply chain attacks, e.g., as defined in [24]

5.4 Key Issue #3: Conflicting xApps

5.4.1 Key issue detail

xApps can change RAN parameters to optimize a specific network metric, which could result in conflicting actions between xApps. Conflicting xApps can unintentionally or maliciously impact O-RAN system functions such as mobility management, admission controls, radio resource control, and load balancing to degrade the performance or availability of the network.

The functional split between the Near-RT RIC and the E2-Node depends on the available xApps and the capabilities exposed by the O-gNB. This can create possible conflicts between the decisions taken by the Near-RT RIC and the O-gNB that could lead to instability in the network, which introduces vulnerabilities that could be exploited by threat actors. [1]

This threat has been identified in O-RAN Threat Model [1] as Threat ID T-xApp-02.

5.4.2 Security threats

The security threats associated with conflicting xApps include:

- DoS from an xApp intentionally creating RRM decisions that conflict with gNB internal decisions

- Degradation of performance from an xApp intentionally creating RRM decisions that conflict with other xApp decisions

5.4.3 Potential security requirements

The solution should make use of the existing, but currently optional, E2 Guidance Request Procedure during xApp Registration to mitigate conflicts arising from new xApps. The purpose of the E2 Guidance procedures is to allow xApps to obtain guidance from the Conflict Mitigation function in the Near-RT RIC. The Conflict Mitigation function is intended to resolve potentially overlapping or conflicting requests from multiple xApps, but some conflict mitigation messages and procedures are FFS [3].

5.5 Key Issue #4: Isolation between xApps

5.5.1 Key issue detail

The Near-RT RIC enables near real time control and optimization of RAN elements and resources via fine-grained data collection and actions over E2 interface, and includes open interfaces (A1, O1) to the management and orchestration system (SMO). This will be achieved using a collection of xApps, with each xApp providing a specific functionality by consuming a specific subset of data or producing data for other xApps or Near-RT RIC platform. To provide its desired functionality, the xApps would interact with other Near-RT RIC platform components.

In a multi-vendor environment, where 3rd party xApps and those provided by the RIC vendor are deployed together, an xApp can be used to impact the other co-located xApps/Near-RT RIC platform thus creating a noisy neighbour effect where an xApp can overconsume the underlying shared resources provided by the Near-RT RIC platform like CPU, memory, I/O bandwidth, etc. and can negatively affect the performance of other co-located xApps and potentially other CNFs sharing the same underlying platform.

It’s also possible that xApps have different levels of access to the data and resources available on Near-RT RIC. Appropriate isolation is needed for xAPPs that have different threat profiles.

5.5.2 Security threats

Without proper isolation, an xApp can be used to:

- overconsume the platform resources shared by all xApps, creating a noisy neighbour effect by leaving other co-located xApps with a resource deficit.
- administer DoS attacks on other co-located xApps and impacting service offered by the xApp under attacks

- administer DoS attacks on the Near-RT RIC platform components, impacting the availability of the Near-RT RIC
- administer unauthorised access to sensitive real-time information from RAN, persistent configuration used by the Near-RT RIC to control the RAN
- probe for vulnerabilities in the underlying system hosting the xApps application and use them to administer attacks on another xApps and the platform

5.5.3 Potential security requirements

Requirement: Based on xAPPs threat profile, the Near-RT RIC platform shall categorise and isolate xApps into different security groups and define security boundaries. Near-RT RIC platform shall cap the resources allocated to xApps make sure that xApps trying to overconsume the resources are quarantined.

Requirement: Access controls shall be established and enforced for accessing critical data in Near-RT RIC

Requirement: Communication between security zones shall require mutual authentication and authorization between communicating xApps

5.6 Key Issue #5: Compromise of ML data pipelines used in Near-RT RIC

5.6.1 Key issue detail

The Near-RT RIC contains database(s) which contain historic data collected from UE and RAN information. This historic data can be used to train various ML models used in Near-RT RIC. Also, streaming data may be collected from E2 nodes by some ML based xApps, and used either for training ML models or deriving inferences.

ML data pipelines can consist of several ML models, using one or more sources of data for training as well as inference derivation. xApps can have ML pipelines implemented within, or can be using ML pipelines from another xApps that provide analytics services.

ML data pipelines can be compromised due to:

- A potential vulnerability existing in the xApp that implements the ML pipeline
- Poisoning of ML training data due to compromised E2 node(s)
- Poisoning of ML training data due to compromised UE(s)
- Compromised database(s) which store historic data used for ML training
- Compromised interfaces of Near-RT RIC allowing a malicious intruder to alter some of the training data that may be used to train ML models used by xApps

5.6.2 Security threats

T_ML_01 from ORAN Threat Modelling [1] is applicable here.

5.6.3 Potential security requirements

The following security requirements should be considered:

- Secure data storage, and secure data transfer mechanisms shall be implemented in Near-RT RIC.
- Authenticated and authorized access to data stores shall be implemented in Near-RT RIC.
- Near-RT RIC deployment shall have provision for regular security vulnerability patch updates for xApps.
- Integrity of xApps shall be monitored after installation as well as during runtime on a periodic or need basis.
- O-Cloud security requirements shall be adhered to, in order to ensure that the interfaces feeding data to ML data pipelines are secured.
- Detection of compromised E2 nodes, and filtering ML data pipelines to exclude inputs from such nodes.
- Detection of compromised UE, and filtering ML data pipelines to exclude inputs from such Ues.

5.7 Key Issue #6: Altering the ML learning models used in Near-RT RIC

5.7.1 Key issue detail

Near-RT RIC hosts a set of xApps, some of which may contain ML based solutions performing inference and/or training processes.

- ML enabled xApps can hold any required ML models and would be deployed as an integrated function.
  - In such cases, if ML models use online training, or transfer learning, or reinforcement learning, or other such mechanisms, it is likely that during runtime, the binary footprint of the models can change for good reasons. However, this makes it a challenge to distinguish between maliciously modified ML models and genuine model changes.
- xApps can also use ML models stored in databases or separate filesystems/storage
  - xApps can perform online training of ML models by updating the models in DB, while ensuring that the binary footprint of xApp remains un-changed

For xApps having embedded ML models, a compromise of ML models is likely due to following reasons:

- A potential vulnerability existing in xApp
- Security Weakness in APIs exposed by xApps
  - For example, APIs allowing modifications of ML models using transfer learning can allow changes of models by a malicious source, if authentication and authorization for such interfaces are not strong

For xApps using ML models stored in DB/separate storage, a compromise of ML models is likely due to following reasons:

- A potential vulnerability existing in xApp leading to compromised ML model updated in the DB
- A potential vulnerability existing in DB security
  - Missing or weak authentication and authorization for DB

Transfer Learning can be implemented in a manner that Non-RT RIC can provide updated ML models to Near-RT RIC via A1 interface. If this interface is compromised, or, if an un-authenticated or un-authorized source is allowed to update ML models in Near-RT RIC, it can lead to unexpected behaviour of ML models.

5.7.2 Security threats

T_ML_02 from ORAN Threat Modelling [1] is applicable here.

An attacker can illegally access a machine learning model and alter its parameters and thereby influence how it produces results. This can lead to wrong prediction and might result in catastrophic decisions if the results of the predictions were being used to make key business decisions.

Also, an attacker can extract sensitive or confidential data that, through training, are built right into the ML model.

5.7.3 Potential security requirements

Following security requirements should be considered:

- Integrity of xApps shall be monitored after installation as well as during runtime on a periodic or need basis.
- Secure data storage, and secure data transfer mechanisms shall be implemented in Near-RT RIC.

- Authenticated and authorized access to data stores shall be implemented in Near-RT RIC.Security measures shall be ensured to allow only authenticated and authorized applications providing ML model updates over the A1 interface.

5.8 Key Issue #7: Data Protection and Privacy in Near-RT RIC Database

5.8.1 Key issue detail

The Near-RT RIC contains database(s) which contain sensitive UE and RAN information, including history of the network state, configurations related to E2 Nodes, cells, bearers, flows, UE identities, and the mappings between them. Overlayed on top of these databases is the shared data layer (SDL) which enables xApp access. There is currently no model for authentication or authorization for the database(s). The Near-RT RIC requires security mechanisms in place to ensure xApps have authenticated access to the database(s) and may access only parts of the database(s) in which they have permission to access. Adverse effects may result if no security mechanisms are in place, such as manipulation of database information.

5.8.2 Security threats

xApps that have gained unauthorized access to Near-RT RIC database(s) have the ability to:

- Perform unauthorized database operations and manipulate database information to impact network service and performance
- Export sensitive data from the Near-RT RIC database(s) (data or user privacy threat)
- Monitor the database for specific information and take an adverse action

5.8.3 Potential security requirements

Near-RT RIC shall authenticate xApp access to the Near-RT RIC database(s) during SDL registration.

Near-RT RIC shall provide authorized access to Near-RT RIC database(s).

5.9 Key Issue #8: Onboarding untrusted xApps

5.9.1 Key issue detail

As per definition in [3], an xApp is an application designed to run on Near-RT RIC. It may consist of one or more microservices. It is independent of Near-RT RIC and may be provided by any third party. As part of offering its service to Near RT-RIC, some of the xApp's functionalities include

- communicating with Near-RT RIC platform via Near-RT RIC API,

- providing the APIs it produces to other xApps and Near-RT RIC,
- collecting near real-time information from E2 Nodes and influence behaviour of E2 Nodes and
- accessing Near-RT RIC Databases (via SDL) which gives access to sensitive information related to UE and RAN

Onboarding an untrusted third party xApp onto Near-RT RIC can create serious security issues which can compromise the Near-RT RIC functionality, impact the availability of other xApps and have cascading impact on overall performance of ORAN.

5.9.2 Security threats

Onboarding an untrusted third party xApp onto Near-RT RIC can cause similar or same security threats as a Malicious or Badly configured xApp. Some well-known and easily administrated attacks with untrusted xApps are:

- If the xApp is not from a trusted source or is not maintained by a trusted source, it could contain security vulnerabilities or back doors which could be exploited by the attackers and impact Near-RT RIC functions
- The untrusted/unverified xApp could provide APIs with weak security (weak or no authentication and authorizaton) and such xApps could be easy target for attackers to break in and propagate lateral movement of attack in the system (towards other xApps).
- The untrusted/unverified xApp could expose more services than required and such xApps could expose a large attack surface to the attackers to break in and gain access, manipulate configurations, access logs, insert back doors
- The untrusted/unverified xApp can be used to sniff sensitive UE information which can be used by attackers or the untrusted/unverified xApp itself for rogue purposes

5.9.3 Potential security requirements

Requirement: The xApp shall be compliant to industry standards for API security (eg : OWASP Top 10)

Requirement: The xApp shall be free of any known software vulnerabilities.

Requirement: The xApp shall be authenticated before the app is onboarded to the cloud platform and registered onto the Near-RT RIC platform

Requirement: The xApp shall support appropriate authorization schemes to make sure that a user accessing its service shall not be able to execute any actions that the user is not authorized for.

Requirement: The xApp shall harden its service offering by making sure that only required services should be exposed to its users.

5.10Key Issue #9: xApp exploitation via malicious A1 policies

5.10.1 Key issue detail

The Non-RT RIC defines policies that are provided to the Near-RT RIC over the A1 interface. The purpose of the A1 policies is to guide and optimize the RAN, and the policies inform behavior of xApps in the Near-RT RIC. [9]

Unauthorized access to the Non-RT RIC enables the creation of ‘false policies’ that can be issued to the Near-RT RIC for enforcement. Existing Near-RT RIC policies could also be modified to achieve a false policy. False policies passed to the Near-RT RIC would be persistent until they were modified or deleted by the Non-RT RIC or the Near-RT RIC power cycles.

5.10.2 Security threats

False policies can be created to have numerous impacts to the normal performance of the RAN. A single false A1 policy can target a specific UE, groups of Ues, or an entire cell. A false policy could influence the Near-RT RIC to configure the O-DU and O-RU functions to support Denial of Service (DoS) attacks by simply using feedback data to degrade RAN performance.

False policies could also be used for the purpose of locating a subscriber or group of subscribers. In this case, the false policy would cause the Near-RT RIC to isolate a subscriber in the O-CU. The Near-RT RIC could also use MIMO beamforming in the O-DU and O-RU to isolate a user onto a single beam. The data feedback from the RAN can include UE location or trajectory information from GPS data. The subscriber location would be attained from access to the Non-RT RIC in the SMO function.

The Near-RT RIC is capable of steering traffic to achieve optimal QoS or QoE performance. A false policy could notionally cause the Near-RT RIC to steer user data to isolate the data in order to facilitate a cyber-attack.

5.10.3 Potential security requirements

Security protections are defined for the A1 interface and security work is in process for A1 related APIs and the Non-RT RIC. For example, A1 peering should be established with only trusted sources using TLS 1.2, or higher, and PKI X.509 certificates, as specified in [19]. TLS also provides confidentiality and integrity protection. Additional Near-RT RIC authentication of the A1 policy may be a consideration later, after the above security protections are defined and evaluated.

A perimeter security approach may enable an Internal threat actor to conduct a MitM attack on the A1 interface. Security for the Near-RT should take a zero-trust approach in which security controls are implemented at the Near-RT RIC without assumption of a secure perimeter. The Near-RT RIC could implement security controls consistent with a zero-trust architecture in which there is no assumption of perimeter defences being provided by other O-RAN functions. For example, the Near-RT RIC could provide checking and validation of policies received from the Non-RT RIC on the A1 interface.

5.11Key Issue #10: Near-RT RIC APIs authentication

5.11.1 Key issue detail

Like any software, APIs can be compromised, and data can be intercepted. APIs can be manipulated causing service disruptions and can be subject as well to DoS attacks. Authentication, an essential part of secure communication, ensures that an attribute claimed by a given entity (e.g., its identity) is actually correct. In the context of secure communication between xApps and Near-RT RIC platform, the platform shall authenticate all xApps instances and only provide them with the information and services for which they are authorized through the use of the Near-RT RIC APIs, as a key objective to guarantee the validity of the transferred messages. xApps can also act as service producers, exposing APIs to be consumed by other xApps for example. In that case, the xApps ‘producers’ shall authenticate the instances interested in consuming those xApp services. Authentication is also a prerequisite for conducting authorization which itself is another key issue for API Security.

Note that using the xApp registration procedure Near-RT platform performs authentication and validation of the xApp, addressed in solution #1, however xApps, as consumers of the services exposed by Near-RT RIC APIs, are not authenticated as per current O-RAN specifications at the time of requesting those services.

5.11.2 Security threats

Not mutually authenticating xApps and Near-RT RIC platform APIs could potentially allow attackers to perform the following type of attacks [17]:

- Operating malicious xApp claiming to be genuine in order to request certain services (theft of services) or information (data leakage)
- Man in the middle attacks between a genuine xApp and a Near-RT RIC platform API
- Querying network or UE information from a compromised xApp to Near-RT RIC platform (e.g. database via SDL API), thereby leaking potentially sensitive data about network and/or UE (potential privacy issues)
- Subscribing a malicious xApp to services provided by the Near-RT RIC platform, such as API-related events notifications, discovery of APIs, E2SM, etc.

The use of weak credentials in the process of API authentication can compromise the overall system. The user/password combination isn’t considered safe, not only for password related attacks (e.g., brute-force), but also it would represent a high risk to allow xApps, especially 3rd party xApps, to store the user/password combo. This approach would extend the attack surface into xApps side.

As a reference, OWASP API Security Top 10 report [10] indicates that authentication mechanisms are often implemented incorrectly, allowing attackers to compromise authentication tokens or to exploit implementation flaws to assume other use''s identities temporarily or permanently. Compromising syste''s ability to identify the client/user, compromises API security overall.

5.11.3 Potential security requirements

The communication between xApps and Near-RT RIC platform APIs shall be authenticated at least at the transport layer (for e.g., with mutual TLS).

At application layer complementary end to end authentication mechanisms based on JSON Web Tokens (JWTs) may be considered as well.

5.12Key Issue #11: Near-RT RIC APIs authorization

5.12.1 Key issue detail

Authorization is the function of specifying and enforcing access rights or privileges to certain resources or services. In the actual context of Near-RT RIC, the platform as API producer shall be responsible to specify those rights/privileges for the platform services as resources to the xApps as consumers. In general, an xApp should only have the required set of permissions to perform the actions for which they are authorized, and no more.

Note. The investigation of services for which the API producer is the xApp is FFS [3].

Authorization mechanisms are to be enforced by the Near-RT RIC platform among other in the following key API procedures [3]:

- **Discovery of Near-RT RIC APIs**: The Near-RT RIC platform shall provide means to restrict xApps from discovery of some published APIs based on configuration policies.
- **E2 Subscription API procedure:** The subscription management shall be based on operator's policies. An xApp may be restricted to interface with only a subset of E2 Nodes by such policies. This procedure establishes a set of preconditions that assume authorization processes:
    - xApp has been authorized to issue E2 Subscription API requests
    - xApp has been authorized to request guidance from Conflict mitigation
    - xApp Subscription Management has been configured to permit E2 Subscription API requests only from specific list of xApps.
- **E2 Control API procedure:** Only authorized xApps may initiate RIC control request messages issued by the Near-RT RIC over the E2 interface to the E2 Nodes, for a specific scope (e.g, E2 Node list, RAN function, etc.).
- **E2 Guidance API procedure:** The purpose of this procedure only authorized xApp to obtain guidance from the conflict mitigation platform function prior to initiating an action.

- **SDL API procedures:** The precondition to consume the services exposed by SDL API (e.g, client registration, fetch data, notification, store, etc.) is that xApps have been successfully registered and authorized.

5.12.2 Security threats

If the API consumers are not authorized by the API producers, attackers (e.g, malicious xApps) would potentially be able to perform the following types of attacks:

- Abuse and/or theft of services or information (data leakage), requesting and successfully obtaining them from the platform, e.g, in order to extract potentially sensitive information from the network and/or uEs
- Negatively impacting the network performance due to malicious policies over E2 Nodes
- Flooding the platform with resource demanding operations that may lead to a Denial of Service attack

As a reference, OWASP API Security Top 10 report [10] indicates that 'Broken Object Level Authorization' has been the most common and impactful attack on APIs. Even if the application implements a proper infrastructure for authorization checks, developers might forget to use these checks before accessing a sensitive object. Unauthorized access can result in data disclosure to unauthorized parties, data loss, or data manipulation.

5.12.3 Potential security requirements

Near-RT RIC architecture shall provide an authorization framework for the consumption of the services exposed in the platform APIs by the xApps, that takes operator policies into consideration. The framework should be used by the specified API procedures in [3].

The authorization framework may need to consider the level of trust related to the xApp (e.g. 3rd party xApp, RIC-owned xApp, etc.)

5.13 Key Issue #12: xAppID abuse

5.13.1 Key issue detail

As part of the xApp registration procedure into the Near-RT RIC platform, the platform assigns an identity (ID) to the xApp, so called xApp ID. This xApp ID is assumed to be used in xApps API request messages to the Near-RT RIC platform to enable the Near-RT RIC platform to identify the xApp (API service consumer). For the platform to trust the xAppID in the xApps request messages it is necessary to bind this xAppID to some trust anchor. Also, the xAppID needs to be uniquely identifying a single xApp instance.

5.13.2 Security threats

Not uniquely identifying xApps using a trusted xAppID potentially entails certain threats and potential attacks:

- A non-unique xAppID might cause misidentification of an xApp, possibly allowing a potentially malicious xApp to request certain services (theft of services), information (data leakage), or alter existing information
- A malicious xApp might use the xAppID assigned to a legitimate xApp to request services or information from Near-RT RIC platform
- A non-unique xApp ID could make it impossible to accurately assign actions to the correct xApp
- A non-unique xApp ID could make it difficult to recognize that a malicious xApp is in the environment

5.13.3 Potential security requirements

The Near-RT RIC shall provide a procedure to embed the xAppID into a trust anchor (e.g., X.509 certificate) assigned to the xApp used for authentication (mTLS) and for authorization (OAuth – client authentication).

The Near-RT RIC shall create each xAppID using procedures that avoid duplicates, such as UUID as described in IETF RFC 4122.

5.14 Key Issue #13: Malicious Y1 Consumer

5.14.1 Key issue detail

Near-RT RIC provides RAN analytics information services via Y1 service interface. These services can be consumed by Y1 consumers by subscribing to or requesting the RAN analytics information via the Y1 service interface. Y1 consumers may be Application Functions (AFs) which are within an O-RAN trusted domain. AFs outside the O-RAN trusted domain may use Y1 services, too.

Malicious Y1 consumer functions may use their access through the Y1 interface with the intent of accessing, manipulating or negatively impacting the privacy of subscribers, the RAN or the core network.

5.14.2 Security threats

An attacker may use a malicious Y1 consumer function to

- attain unauthorized access to the Near-RT RIC
- abuse radio network information
- exploit UE identification or otherwise impact the privacy of subscribers

In addition, an attacker may perform a man in the middle attack between a genuine Y1 consumer and the Near-RT RIC platform.

A malicious attacker may also use access to the Y1 interface to mount volumetric and non-volumetric DoS attacks on the Near-RT RIC in order to impact its overall performance.

5.14.3 Potential security requirements

All Y1 consumers shall be properly authenticated and authorized before access to the Y1 functions get enabled for them at least at the transport layer (for e.g., with mutual TLS and OAuth). All communication through the interface shall be confidentiality and integrity protected. Y1 consumers may be within a PLMN trusted domain. Y1 consumers located outside of the PLMN trusted domain should only be able to use the Y1 services via an exposure function acting as a Y1 Consumer, e.g. as specified in 3GPP TS 23.501, Clause 5.20 [33].

Additional security measures should be defined such as

- the Near-RT RIC should provide checking and validation of the data received
- the Near-RT RIC should implement ingress message rate limiting on its Y1 interface

5.15 Key Issue #14: Malicious E2 Node

5.15.1 Key issue detail

E2 Nodes communicate with the Near -RT RIC through the E2 interface. The Near -RT RIC uses the E2 interface to collect near real-time information and provide value added services. The Near-RT RIC Architecture specification [2] clause 8.1 provides further details on the E2 interface.

A malicious actor may use unauthorized access to E2 Nodes or the E2 interface to mount malicious attacks against the Near-RT RIC. A malicious actor may also abuse compromised E2 Nodes properly authenticating to the Near-RT RIC.

5.15.2 Security threats

Altered and misleading data sent from a compromised E2 Node to the Near-RT RIC may cause the Near-RT RIC to draw erroneous conclusions on the RANs status which may impact the normal performance of the RAN. An attacker may also obtain information on the status of the RAN and details on the UEs.

A malicious attacker may also use access to the E2 interface to mount volumetric and non-volumetric DoS attacks on the Near-RT RIC in order to impact its overall performance.

5.15.3 Potential security requirements

Security protections are defined for the E2 interface as explained in clause 6.9 of the present document.

Additional security measures should be defined such as

- the Near-RT RIC should provide checking and validation of the data received
- the Near-RT RIC should implement ingress message rate limiting on its E2 interface termination

6 Mitigations and Solutions

6.1 Introduction

This section will contain any potential Mitigations or Solutions that may be used to address or reduce the risk associated with Key Issues identified in section 5.

6.2 Solution #1: Secure onboarding and deployment of xApps in Near-RT RIC

6.2.1 Introduction

xApps are applications that are onboarded to the SMO, deployed on the Near-RT RIC, and then registered in the Near-RT RIC. At each of these steps in the xApp life cycle, security measures can be applied to ensure that the xApp comes from a valid solution provider and is integrity protected.

6.2.2 Solution details

Onboarding to the SMO is the first step of the xApp life cycle. This solution details the flow from the xApp solution provider to the SMO in Figure 6.2-1.

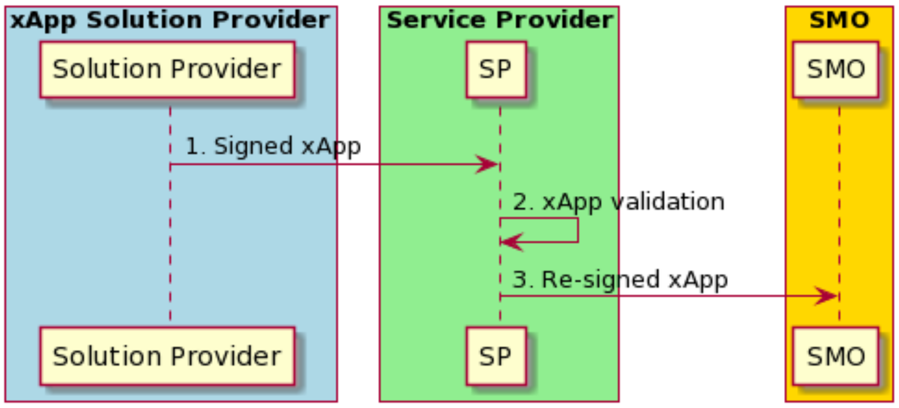


Figure 6.2-1: Onboarding xApp to SMO

1. Solution provider digitally signs the xApp and sends to the Service Provider.
2. The Service Provider validates the digital signature for code integrity, performs a schema check and any operational testing, and then adds their signature to the package.
3. The xApp is published into the catalogue for visibility by the SMO.

The next step of the xApp life cycle is deployment on the Near-RT RIC, which starts with a request to the SMO from the NF Install Project Manager to deploy new xApps on the RIC, per [23]. The SMO maps the RIC variables and instructs the O-Cloud DMS to create the xAPP Deployment and allocate the resources necessary on the Near-RT RIC O-Cloud. The DMS notifies the SMO that the xAPP Deployment has been created and the DMS assigns a Deployment ID which the SMO uses to update its xApp inventory. At the end of the deployment process, the xApp starts the registration process with the Near-RT RIC by sending a registration request to Near-RT RIC Management Services function, per [3].

This solution proposes a security validation request sent from the Management Services to the Security Function in the Near-RT RIC, as shown in Figure 6.2-2.

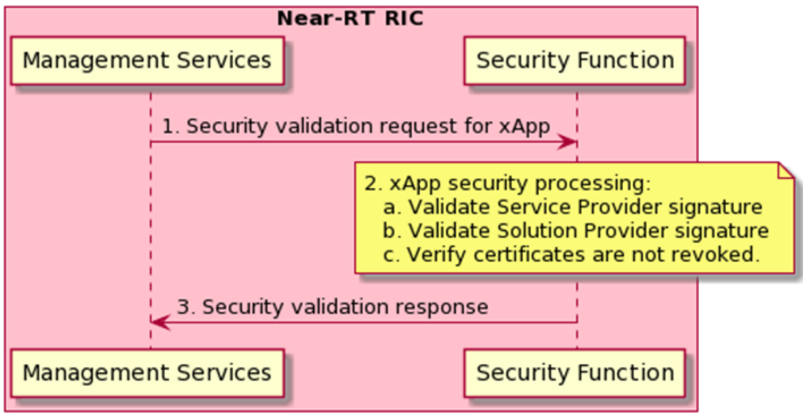

Near-RT RIC
Management Services
Security Function
1. Security validation request for xApp
2. xApp security processing:
a. Validate Service Provider signature
b. Validate Solution Provider signature
c. Verify certificates are not revoked.
3. Security validation response
Management Services
Security Function

Figure 6.2-2–- Authentication during Deployment Process

Once the xApp is deployed and has sent a registration request to Management Services:

1. In the Near-RT RIC, a xApp validation request is sent from Management Services to the Security Function.
2. The Security Function performs the following checks:
    a. Validate the signature of the Service Provider. Confirm the signing Service Provider is the trusted Service Provider. This proves that the valid Service Provider approved the xApp, and that the xApp has not been modified since it was approved for use by the Service Provider.
        i. Note – Consideration on different security validation processes for 3rd party xApps, as compared to xApps from the Near-RT RIC platform provider, is FFS after the impacts of all key issues and solutions have been studied.
    b. Validate signature of the xApp Solution Provider. Confirm that the signing xApp Solution Provider is a member of a trusted list of providers. This proves that the xApp came from a trusted provider and has not been modified since the Solution Provider released it.
    c. Verify that neither the Service Provider nor Solution Provider certificates are revoked. This ensures that nothing has changed to imply either signature can no longer be trusted.
3. A security validation response is sent back to Management Services, who can then continue with the xApp registration process if the checks are successful.

In the context of WG3 Near-RT RIC Architecture specification [3], this solution would be added to Figure 9.3.1-1 to fill in details of the authentication and validity checks during registration processing.

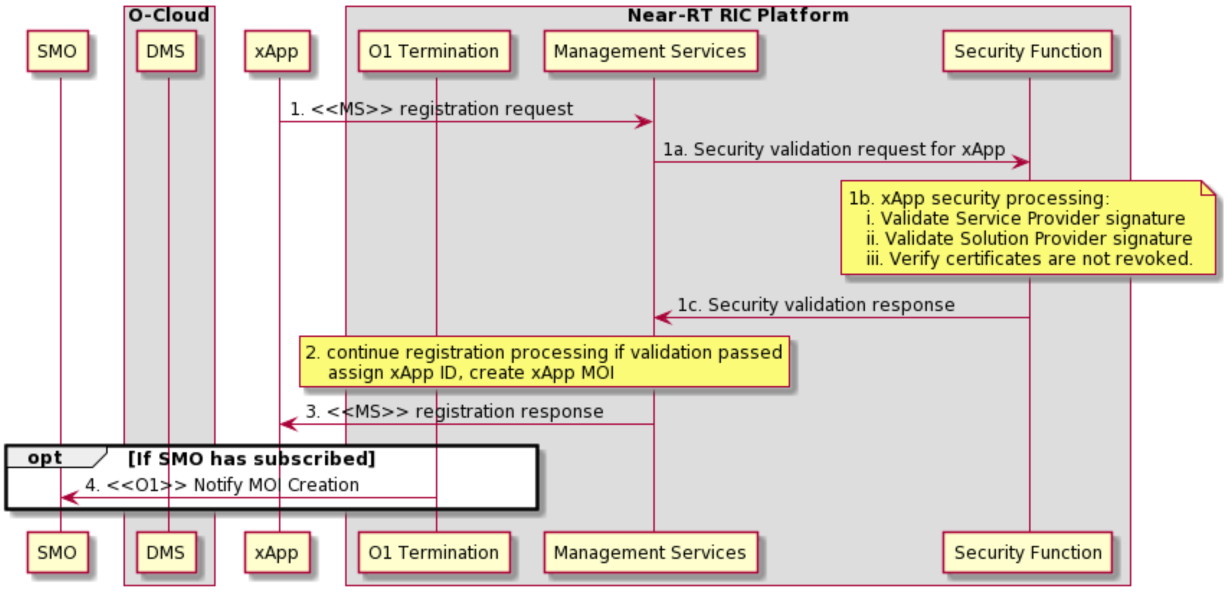

O-Cloud
Near-RT RIC Platform
SMO
DMS
xApp
O1 Termination
Management Services
Security Function
1. <<MS>> registration request
1a. Security validation request for xApp
1b. xApp security processing:
i. Validate Service Provider signature
ii. Validate Solution Provider signature
iii. Verify certificates are not revoked.
1c. Security validation response
2. continue registration processing if validation passed
assign xApp ID, create xApp MOI
3. <<MS>> registration response
opt
[If SMO has subscribed]
4. <<O1>> Notify MOI Creation

Figure 6.2-3 – Extended xApp Registration procedure from [3]

6.2.3 Evaluation

By requiring a secure onboarding process with authentication for all xApps, this solution addresses key issues #1, #2, #8.

6.3 Solution #2: Security management of risks in open-source components of Near-RT RIC architecture

6.3.1 Introduction

O-RAN Near-RT RIC and xApps software components will hugely depend on open-source software and open-source libraries, because of agility, flexibility, community support, cost-effectiveness and many other benefits open-source community provides to meet modern software development. However, on the flip side, there are a number of risks associated with open-source software which needs to be managed.

Key risks with Near-RT RIC and xApps open-source components:

- Open-source software provides opportunity for attackers to exploit any known vulnerabilities. A bad actor can use publicly available information to target Near-RT RIC open-source components that have not implemented a patch for known vulnerabilities.

- Non-adaptation of secure software development practices results in unintentional insertion of vulnerable code, propagation of vulnerable code and libraries.
- Intentional backdoors can be inserted by malicious developers.
- Attackers can review the code to identify vulnerabilities.
- Non-compliance of Near-RT RIC open-source license could lead to legal action, affecting business operations and incurring financial damage to consumers. I''s critical for consumers to establish processes to manage open-source software license compliance.

6.3.2 Solution details

Recommended Security best practices to manage open-source software risks listed in section 6.3.1

Maintain open-source Near-RT RIC components and xApps Inventory

- Recommended to have Software Composition Analysis tooling to automatically scan and identify all open-source components, version(s) in use, all package dependencies, libraries, and create an accurate bill of materials (BOM), checks for policy and license compliance, security risks, protect against threats to intellectual property (IP) and version updates.

Track and analyse open-source Near-RT RIC components and xApps vulnerabilities

- Track known vulnerabilities against the inventory of open-source software and libraries.
- Run automated static application security testing to identify known vulnerabilities in open-source software components.
- Create SBOM to facilitate identification of a potential vulnerability in the software product when a CVD is released for an open-source binary, library, or package.
- Perform frequent scanning for known vulnerability or misconfiguration on open-source container image during build phase, within the registry, before deployment and during run time. Recommended to use automated tools integrated into the CI/CD pipeline.
- Perform dynamic application security testing (DAST) which can be tuned and trained to detect run-time vulnerabilities.
- Continuously monitor open-source workload during runtime for any security risks and vulnerabilities.
- Conduct Penetration testing to identify and resolve critical security defects.

Remediate open-source vulnerabilities

Software vulnerabilities fall into three categories: publicly known with a patch available for the software; publicly known without a patch (n-day); and not publicly known (0-day). Although measures should be taken to mitigate the risk of n-day and 0-day vulnerabilities, patching publicly known vulnerabilities as quickly as possible significantly reduces the risk of exploitation.

- Patch critical vulnerabilities in the operational environment within [policy defined, suggested: < 15] days and other vulnerabilities within [policy-defined, suggested: <60] days.
- Refer to [30]
- Refer to [28]

Continuously monitor for new open-source risks

- Perform frequent scanning for known vulnerabilities during run time.
- Enable security audit logs for monitoring security events.
- Dynamic application security testing (DAST) which can be tuned and trained to detect run-time vulnerabilities.

Secure Software development practice

Security Best practices based on openssf secure software development practices

- The project MUST have at least one primary developer who knows how to design secure software.
- At least one of the project’s primary developers MUST know of common kinds of errors that lead to vulnerabilities in this kind of software, as well as at least one method to counter or mitigate each of them.
- Use basic good cryptographic practices:
  - The software produced by the project MUST use, by default, only cryptographic protocols and algorithms that are publicly published and reviewed by experts (if cryptographic protocols and algorithms are used).
  - If the software produced by the project is an application or library, and its primary purpose is not to implement cryptography, then it SHOULD only call on software specifically designed to implement cryptographic functions; it SHOULD NOT re-implement its own.
  - All functionality in the software produced by the project that depends on cryptography MUST be implementable using FLOSS.
  - The default security mechanisms within the software produced by the project MUST NOT depend on broken cryptographic algorithms (e.g., MD4, MD5, single DES, RC4, Dual_EC_DRBG), or use cipher modes that are inappropriate to the context, unless they are necessary to implement an interoperable protocol (where the protocol implemented is the most recent version of that standard broadly supported by the network ecosystem, that ecosystem requires the use of such an algorithm or mode, and that ecosystem does not offer any more secure alternative). The documentation MUST describe any relevant security risks and any known mitigations if these broken algorithms or modes are necessary for an interoperable protocol.
  - The default security mechanisms within the software produced by the project SHOULD NOT depend on cryptographic algorithms or modes with known serious weaknesses (e.g., the SHA-1 cryptographic hash algorithm or the CBC mode in SSH).
  - The security mechanisms within the software produced by the project SHOULD implement perfect forward secrecy for key agreement protocols so a session key derived from a set of long-term keys cannot be compromised if one of the long-term keys is compromised in the future.
  - If the software produced by the project causes the storing of passwords for authentication of external users, the passwords MUST be stored as iterated hashes with a per-user salt by using a key stretching (iterated) algorithm (e.g., Argon2id, Bcrypt, Scrypt, or PBKDF2).

- The security mechanisms within the software produced by the project MUST generate all cryptographic keys and nonces using a cryptographically secure random number generator and MUST NOT do so using generators that are cryptographically insecure.

- Secured delivery against man-in-the-middle (MITM) attacks
    - The project MUST use a delivery mechanism that counters MITM attacks. Using https or ssh+scp is acceptable.
    - A cryptographic hash (e.g., a sha1sum) MUST NOT be retrieved over http and used without checking for a cryptographic signature.
- Publicly known vulnerabilities fixed
    - There MUST be no unpatched vulnerabilities of medium or higher severity that have been publicly known for more than 60 days.
    - Projects SHOULD fix all critical vulnerabilities rapidly after they are reported.
- Static code analysis
    - At least one static code analysis tool (beyond compiler warnings and“"saf”" language modes) MUST be applied to any proposed major production release of the software before its release, if there is at least one FLOSS tool that implements this criterion in the selected language.
    - It is SUGGESTED that at least one of the static analysis tools used for the static_analysis criterion include rules or approaches to look for common vulnerabilities in the analyzed language or environment.
    - All medium and higher severity exploitable vulnerabilities discovered with static code analysis MUST be fixed in a timely way after they are confirmed.
    - It is SUGGESTED that static source code analysis occur on every commit or at least daily.
- Dynamic code analysis
    - It is SUGGESTED that at least one dynamic analysis tool be applied to any proposed major production release of the software before its release.
    - All medium and higher severity exploitable vulnerabilities discovered with dynamic code analysis MUST be fixed in a timely way after they are confirmed.

Refer to open-source software best practices: [27], [42], [43] and [44].

6.3.3 Evaluation

Recommended open-source best practices mitigates Near-RT RIC components and xApps open-source Risks. This solution addresses key issues #1, #2, #5, #6, #8.

6.4 Solution #3: Authentication schema for APIs

6.4.1 Introduction

REST APIs and gRPC APIs are being considered as valid network solutions for Near-RT RIC API. REST APIs use HTTP and support Transport Layer Security (TLS). REST APIs utilizes JavaScript Object

Notation (JSON) as encoding protocol. gRPC APIs use HTTP/2 and support as well TLS. In contrast with REST APIs, gRPC used Protobuf as encoding protocol.

Time critical E2 related APIs are currently specified to run over SCTP/Protobuf network protocol stack. The security network layer specified for E2 interface is IPSec (SEC-CTL-E2 [2]). DTLS is proposed as optional mechanism.

The solution proposes a basic network authentication schema for transactional APIs (REST and gRPC) based on mutual TLS (mTLS) authentication via X.509 certificates. The use of mTLS serves a dual purpose:

- Mutual authentication
- Confidentiality protection for the authorization framework (solution #4)

mTLS is fully aligned with Zero Trust principles of O-RAN security architecture.

Additionally, the network authentication schema can be complemented by a client authentication schema at application layer, with the use of secure tokens (e.g. JWT, secured with digital signatures based on JWS). Authentication is a prerequisite of the authorization framework, applicable to those APIs.

For time critical E2 related APIs, secured by IPsec, the solution proposes IKEv2 certificate-based authentication implementation according to O-RAN Security Protocols specification [19], which is aligned with 3GPP TS 33.310 specification [20] in terms of certificates and IKEv2 profiles.

6.4.2 Solution details

6.4.2.1 mTLS authentication

In mTLS both the client (e.g, xApp as API consumer) and the server (e.g, platform as API producer) require a certificate, and both sides authenticate each other using their public/private key pair. The authentication process includes the following steps:

- the client connects to server and requests access
- the server presents a server certificate
- the client verifies the server certificate
- the client presents a client certificate
- the server verifies the client certificate
- the server grants access
- client and server can exchange information over encrypted TLS connection

6.4.2.2 Application layer client authentication

The purpose of the solution is to perform an additional consumer authentication at application layer by the platform, which acts as authorization server and producer. The process is as follows:

- The API consumer (e.g, xApp) shall attach a signed token in the service request towards the receiving end point, i.e, authorization server, producer.
- The signed token includes the consumer instance ID (e.g, xApp ID), that can be checked against the certificate by the producer
- The signed token includes a timestamp as basis for restriction of its lifetime. It could be even one per service request, this is up to the configuration.

6.4.2.3 IPsec authentication

Please refer to O-RAN Security Protocols Specification [19], chapter 2.5 for supported IPsec capabilities in O-RAN.

In summary, the following options for IPSec authentication have been specified in [19]:

- Data origin authentication, always enabled
- ESP authentication transforms
- IKE endpoint Identification
- X.509v3 digital certificates provided by a Certificate Authority solution shall be supported
- Pre-shared Keys may be supported
- Key exchange shall be supported via IKEv2, profile as described in [21].

6.4.3 Evaluation

The proposed authentication schema addresses the key issues #1, #2, #4, #5, #6, #7, #8, #9, #10, #11

Note. The authentication mechanisms described in the solution may be applicable to other O-RAN parts of the architecture.

6.5 Solution #4: Authorization framework for APIs

6.5.1 Introduction

The proposed authorization framework is based on OAuth 2.0 as specified in RFC 6749 [11]. The OAuth 2.0 authorization framework enables a third-party application (e.g, xApp) to obtain limited access to a service (HTTP), either on behalf of a resource owner by orchestrating an approval interaction between the resource owner and the HTTP service, or by allowing the third-party application to obtain access on its own behalf.

OAuth 2.0 is the framework adopted in 3GPP 5G Service Based Architecture (SBA) [13], where the NRF (Network Resource Function) acts as authorization server.

MEC specifications [15] mandate the use of OAuth 2.0 for authorization of access to RESTful MEC service APIs, and ETSI NFV [14] also specifies the same framework for ETSI NFV-MANO APIs.

The proposed solution implementation in Near-RT RIC architecture of O-RAN is based on the options selected for OAuth 2.0 in both standards developing organizations, 3GPP and ETSI.

6.5.2 Solution details

The roles defined in OAuth 2.0 has been assigned as follows:

- Resource owner / Resource server (producer): Near-RT RIC platform modules providing services via APIs.
- Client (consumer): xApp

  Note.- xApp as API producer is FFS in current RIC Architecture specification.
- Authorization server: Security function in Near-RT RIC platform

Grants shall be of the type Client Credentials Grant, as described in clause 4.4 of RFC 6749 [11], and represented in the following diagram:

```
+---------+                                  +---------------+
|         |                                  |               |
|         |>--(A)- Client Authentication --->| Authorization |
| Client  |                                  |     Server    |
|         |<--(B)---- Access Token ---------<|               |
|         |                                  |               |
+---------+                                  +---------------+
```

Figure 6.5-1–- from RFC 6749

3GPP SBA specifies in [13] that access tokens shall be JSON Web Tokens as described in RFC 7519 [18], and are secured with digital signatures or Message Authentication Cores (MAC) based on JSON Web Signature (JWS) as described in RFC 7515 [18]. ETSI MEC can also use JWT, that may include scopes of roles to restrict the access to MEC services. They may be signed to become tamper proof and encrypted. In general, the use of JWTs as access tokens in Near-RT RIC will depend on the final specification for network API solutions by O-RAN WG3.

The authorization process entails two verifications:

- Check whether the consumer is permitted to discover the requested Near-RT RIC API via the Enablement Discovery API during the discovery procedure. This is performed on per consumer granularity by the authorization server
- Check whether the consumer is permitted to access the requested service producer for consuming the service (e.g. SDL fetch data), on a per request type and corresponding subscription. The granularity is determined by the service logic of the producer.

Management module of Near-RT RIC platform maintains the information of available producers and their supported services, as each of them needs to register the list of services it supports.

The following schema represents the main steps of the entire authorization process:

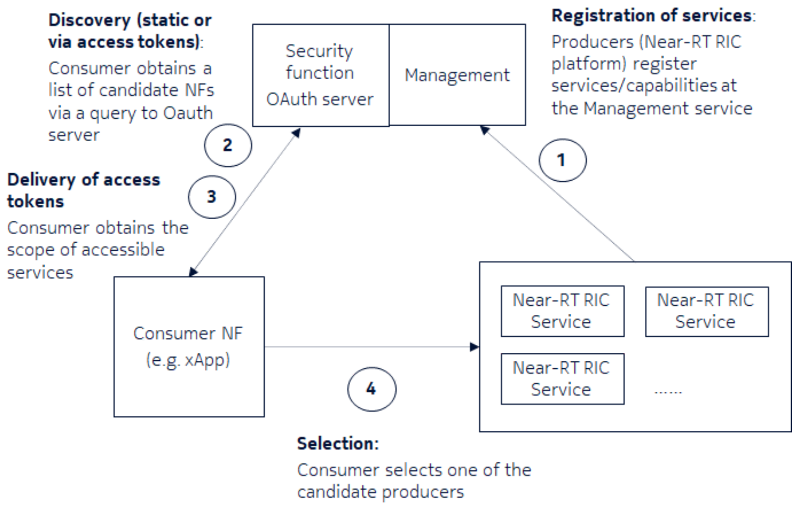

Discovery (static or via access tokens): Consumer obtains a list of candidate NFs via a query to Oauth server
Security function OAuth server
Management
Registration of services: Producers (Near-RT RIC platform) register services/capabilities at the Management service
1
2
3
Delivery of access tokens Consumer obtains the scope of accessible services
Consumer NF (e.g. xApp)
Near-RT RIC Service
Near-RT RIC Service
Near-RT RIC Service
......
4
Selection: Consumer selects one of the candidate producers

Figure 6.5-2–- OAuth 2.0 process in Near-RT RIC

The overall OAuth 2.0 process is foreseen to take place after xApp registration and API discovery procedures specified in Near-RT RIC. The main steps would be:

- Registration procedure in Management service of Near-RT RIC platform
  - xApp registration

    Note 1. Solution #1 adds a security validation feature in this process.

    Note 2. During the proposed registration/validation process the platform may check whether the xApp is provided by 3rd party or is RIC-owned application.
  - APIs registration by the producer via Enablement API
- Enablement API related procedures
  - Discovery
    - Basic: Static restriction based on the registration information in database
    - Optional: Protection of Discovery services by OAuth 2.0, e.g. in case of 3rd party xApps
- OAuth 2.0 based authorization
  - Services offered by the Near-RT RIC platform are described in the scope of the access token provided to the xApp, required to use the specified APIs.

6.5.3 Evaluation

The proposed authorization framework addresses the key issues #1, #2, #4, #5, #6, #7, #8, #9, #10, #11

Note. The authorization mechanisms described in the solution may be applicable to other O-RAN parts of the architecture.

6.6 Solution #5: Near-RT RIC database and SDL access control mechanisms

6.6.1 Introduction

OAuth 2.0 as described in Solution 4 is used to ensure authorized xApp access to SDL API. Additional steps need to be implemented to control granularity for the database access and that is considered in this solution.

This solution implements Role-Based Access Control (RBAC) to control and restrict xApp access to the information stored in the Near-RT RIC database(s) and accessible via SDL APIs. Under RBAC, xApps would access the database(s) according to their assigned role, and the assigned role determines what operations the xApp can perform on what database.

6.6.2 Solution details

The solution is comprised of separate steps during the Onboarding to SMO process and during the SDL Registration process. RBAC roles are assigned during the onboarding process where privileges are defined by the xApp Solution Provider by a xApp database "token" and vetted by the Service Provider. The token needs granularity for data and operations in order to control access.

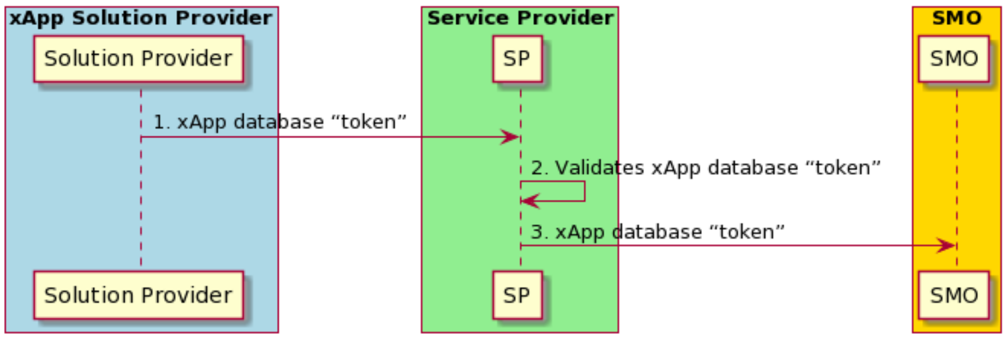


Figure 6.6-1–- xApp database token during onboarding

As part of the xApp onboarding to SMO process:

1. The xApp Solution Provider generates a database "token" that declares what kind of database access and operations the xApp would need. This is sent to the Service Provider along with the xApp.

2. The Service Provider vets the proposed database access and operations during xApp validation by agreeing to the access rights requested by the xApp.
3. The database "token" is passed to the SMO when the xApp is onboarded.

Before accessing the database, the xApp must register with the SDL using the SDL Client Registration procedure. This solution adds in an access authorization step before the xApp is allowed to register with the SDL. The Security Function uses the xApp database token to identify which roles the xApp is authorized for.

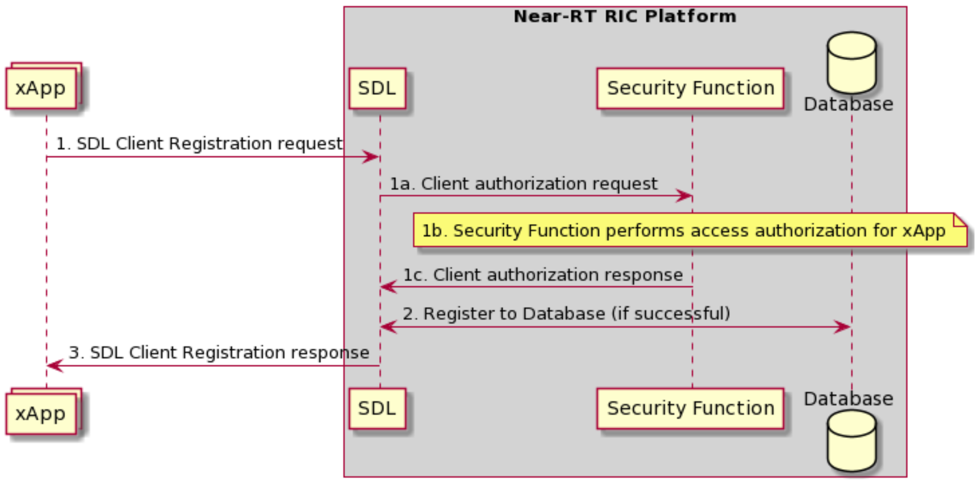

Near-RT RIC Platform
xApp
SDL
Security Function
Database
1. SDL Client Registration request
1a. Client authorization request
1b. Security Function performs access authorization for xApp
1c. Client authorization response
2. Register to Database (if successful)
3. SDL Client Registration response
xApp
SDL
Security Function
Database

Figure 6.6-2 – Modified SDL Client Registration Procedure from [3]

1. The xApp sends SDL Client Registration request to the SDL to access the database in the Near-RT RIC Platform.
    a. After the receipt of the request, the SDL sends a client authorization request to the Security Function.
    b. The Security Function performs access authorization for the xApp by comparing the database "token" from the xApp with the one validated by the Service Provider. After verification, the security function will assign the xApp a role based on the token.
    c. The Security Function sends a client authorization response to SDL.
2. SDL arranges with the database the permission for the xApp to access the database (if authentication is successful).
3. The SDL sends the response of successful or failed registration to the xApp.

6.6.3 Evaluation

This solution addresses key issue #7 Data protection and privacy in Near-RT RIC Database(s), and key issues #4, #5, #6.

6.7 Solution #6: Security Logging (part of FCAPS)

NOTE: Security Logging will be tackled in dedicated Work Item study to Security Log Management within O-RAN Security Focus Group

6.7.1 Introduction

Void

6.7.2 Solution details

Void

6.7.3 Evaluation

Void

6.8 Solution #7: Access control mechanism for API repository/registry

6.8.1 Introduction

Near-RT RIC architecture specification [3] provides the following requirement in clause 5.1.3: '*Near-RT RIC shall provide means to restrict xApps from discovery some published APIs based on configured policies.*'

The same specification in clause 6.2.8 describes the API Enablement function, which provides the discovery service as well as repository/registry services for the Near-RT RIC APIs.

'API Enablement provides support for registration, discovery and consumption of Near-RT RIC APIs within the Near-RT RIC scope. In particular, the API enablement services include:

- *Repository / Registry services for the Near-RT RIC APIs;*
- *Services that allow discovery of the registered Near-RT RIC APIs;*
- *Services to authenticate xApps for use of the Near-RT RIC APIs;*
- *Services that enable generic subscription and event notification;*
- *Means to avoid compatibility clashes between xApps and the services they access.*

The API enablement services can be accessed by the xApps via one or more enablement APIs.

NOTE: The provided enablement APIs may need to consider the level of trust related to the xApp (e.g. ^{3r}d party xApp, RIC-owned xApp, etc.)'

The proposed solution addresses:

- The access control to API enablement function providing API authentication and authorization mechanisms.
- The level of trust related to the xApp via a secure onboarding and deployment of xApps in Near-RT RIC platform.

6.8.2 Solution details

API authentication (mutual TLS) and authorization (OAuth 2.0) mechanisms, providing the necessary access control to API Enablement function, are described in solutions #3 and #4 of the present document respectively.

Secure onboarding and deployment procedure of xApps in Near-RT RIC, considers the level of trust related to the xApp by ensuring that the xApp comes from a valid solution provider and is integrity protected. The solution is described in solution #1 of the present document.

6.8.3 Evaluation

Solutions #1, #3 and #4 of the present address the security requirements related to API Enablement function coming from the Near-RT RIC architecture [3] in clauses 5.1.3 and 6.2.8, specifically related to access control and considerations around the level of trust of xApps. Key issues #4, #7, #8, #10 and #11 are addressed by the solution.

6.9 Solution #8: Secure mechanisms for E2 interface

6.9.1 Introduction

Secure mechanisms for E2 interface are provided in existing WG11 specifications, in particular in O-RAN.WG11.O-RAN-Security-Requirements-Specifications [2], clause 5.2.4.

6.9.2 Solution details

Refer to O-RAN.WG11.O-RAN-Security-Requirements-Specifications [2], clause 5.2.4.

6.9.3 Evaluation

NOTE: Authorization solutions to access E2 nodes and associated services by xApps are for further study. Security mechanisms for E2 interface address key issues #3, #5 and #10.

6.10 Solution #9: Secure mechanisms for A1 interface

6.10.1 Introduction

The Near-RT RIC receives policies from the Non-RT RIC across the A1 interface. While the A1 interface is considered secure with controls that provide confidentiality, integrity, and mutual authentication, the Near-RT RIC should not assume that the policies are valid and trusted. The Near-RT RIC should provide security built-in compliant with a zero-trust architecture based upon the principle that perimeter security is insufficient to protect against internal threats.

NOTE: A1 related security mechanisms will be tackled in Non-RT RIC Security technical report (to be released)

6.10.2 Solution details

Security controls for the Near-RT-RIC that could be implemented as part of its A1 Termination include:

1. Check policies received on A1 conform to defined schema. See also clause 6.13, Solution #12: A1 Policy Authentication of the present document.
2. Validate policy values for validity and range
3. Provide rate limiting on A1 to prevent resource exhaustion and DoS
4. Implement security logging for each of the above failure events

6.10.3 Evaluation

The solution addresses key issues #9 and #6

6.11 Solution #10: Secure mechanisms for O1 interface

6.11.1 Introduction

Secure mechanisms for O1 interface are provided in existing O-RAN specifications, in particular in O-RAN.WG1.O-RAN-Security-Requirements-Specifications [2], clause 5.2.2.

6.11.2 Solution details

Refer to O-RAN.WG11.O-RAN-Security-Requirements-Specifications [2], clause 5.2.2.

6.11.3 Evaluation

NOTE: xApps are managed via Near-RT RIC, not directly by SMO. Adequacy of existing security mechanisms specified for O1 in management of xApps is for further study. Key issue #4 is addressed by O1 security mechanisms.

6.12 Solution #11: Mitigation for conflicting xApps

6.12.1 Introduction

WG3 has already defined a process to seek guidance from the Near-RT RIC Conflict Mitigation function prior to initiating a RIC function procedure. The purpose of the Conflict Mitigation function is to resolve potentially overlapping or conflicting request from multiple xApps. These are currently optional procedures [3].

This guidance can be initiated by 4 different actors at different points in the life cycle of an xApp:

- Initiated by an authorized xApp
- Initiated by Subscription Management function
- Initiated by message monitoring by the Conflict Mitigation function
- Initiated by Conflict Mitigation function

For these procedures, the assumption from WG3 is that the Conflict Mitigation function has access to sufficient information to detect a potential conflict and take a decision on an optimal mitigation solution [3].

6.12.2 Solution details

This solution proposes to make the first procedure, Conflict Mitigation guidance initiated by an authorized xApp, **mandatory** during the E2 Subscription procedure, which is performed at the end of xApp Registration. The E2 Subscription procedure is detailed in [3] in Section 9.3.2.1, with the xApp initiated E2 Guidance Request procedure described in Section 9.3.3.1 [3].

Performing this guidance request during xApp Registration would mitigate conflicts arising from newly deployed xApps. Additional consideration is needed for conflicts arising after deployment, once additional Conflict Mitigation messages and procedures are defined in WG3.

6.12.3 Evaluation

The solution addresses key issue #3.

6.13 Solution #12: A1 Policy authentication

6.13.1 Introduction

A1 policies are provided by the Non-RT RIC to the Near-RT RIC over the A1 interface, which already has security protections as defined in [2]. The security control for the A1 interface is TLS at the transport layer. In addition, the Near-RT RIC APIs include A1 related APIs for A1 Policy Setup, Update, and Delete procedures [3]. Security for these APIs is considered in key issues 10 and 11 and solutions 3, 4, and 13 of this technical report.

6.13.2 Solution details

For a malicious A1 policy to successfully affect the Near-RT RIC, the policy first must contain a valid policy type supported by the Near-RT RIC, in order to pass the policy type check at the A1 Termination in the Near-RT RIC. The policy must then be mapped to an xApp in the Near-RT RIC. Finally, to exploit an xApp for malicious intent as described in Key Issue 9, the A1 policy needs to have capabilities to execute any of the security threats outlined. The access and knowledge required to complete this sequence of events implies a skilled adversary.

Securing the endpoints of the A1 interface is another factor in preventing xApp exploitation by malicious A1 policies. This work item (WG3Sec) in WG11 addresses the Near-RT RIC side of the A1 interface. On the other end, the requirements and security controls for the SMO are FFS for the WG11 Security Requirements specification, but there is a newly started work item (SecureNonRTRIC) to secure the Non-RT RIC in the SMO. This new work item will address the security and data privacy of the Non-RT RIC, A1 interface, R1 interface, and external interfaces.

With the A1 interface itself secured, security of the A1 interface endpoints in progress, A1 related APIs security in progress for authentication and authorization, and the challenges of generating a malicious policy, the likelihood of xApp exploitation via malicious A1 policies is judged to be very low. Further examination of this issue should be performed upon completion of the relevant WG11 work items, but at the moment, no additional A1 policy authentication is deemed necessary.

6.13.3 Evaluation

This solution describes the protections against the security threats in key issue #9.

6.14 Solution #13: Network Security layer for APIs

6.14.1 Introduction

The following network layers are being proposed for the Network APIs in Near-RT RIC by O-RAN WG3:

RIC API	NATURE	NETWORK API SOLUTION (PROTOCOL/ MESSAGE ENCODING)	
		Default	Alternative (s)
A1 related APIs in A1 med towards xApp			
- Policy Enforcement API	Transactional	gRPC/Protobuf/JSON	
- Enrichment information API	Transactional	gRPC/Protobuf/JSON	
E2 related APIs in E2T/ E2M towards xApp		[SCTP/ E2related/ E2SM]	
- E2 Subscription - E2 Subscription Delete [- E2 Subscription Delete Required]	Transactional	SCTP/ Protobuf/ ASN.1	HTTP/ JSON/ ASN.1
- E2 Guidance	Transactional	SCTP/ Protobuf/ ASN.1	HTTP/ JSON/ ASN.1
- E2 Indication	Time critical Uni-Streaming	SCTP/ Protobuf/ ASN.1	
- E2 Control	Time critical Bi-Streaming	SCTP/ Protobuf/ ASN.1	
Management APIs			
- ML model related APIs	Transactional	For future study	
- FCAPS related APIs	Transactional	gRPC/Protobuf	
SDL APIs (store/ fetch/ notify/ push) - R-NIB, UE-NIB, Other data	Transactional + Uni-streaming	For future study	
Enablement APIs	Transactional	gRPC/Protobuf	HTTP/ JSON

This solution proposes to add a security layer in the current network protocol stack for the different APIs specified in Near-RT RIC architecture.

6.14.2 Solution details

TLS is proposed as a solution for transactional Near-RT RIC APIs based on HTTP or HTTP/2. REST APIs use HTTP and support Transport Layer Security (TLS). REST APIs utilizes JavaScript Object Notation (JSON) as encoding protocol. gRPC APIs use HTTP/2 and support as well TLS. In contrast with REST APIs, gRPC used Protobuf as encoding protocol. Please refer to O-RAN Security Protocols [19] for TLS 1.2, TLS 1.3 specifications.

IPSec is proposed as a preferred solution for time critical uni-streaming and bi-streaming E2 related APIs, which are based on SCTP. Please refer to O-RAN Security Protocols [19] for IPsec specifications.

DTLS is proposed as optional solution for time critical uni-streaming and bi-streaming E2 related APIs, which are based on SCTP. The solution is currently on hold due to limitation of packet size when DTLS is running over SCTP (refer to RFC 6083). Please refer to O-RAN Security Protocols [19] for DTLS specifications.

6.14.3 Evaluation

Network Security layer for APIs shall be consider as a basic requirement in Near-RT RIC architecture. The choice will depend on the selected network protocol stacks.

The solution addresses key issues #1, #2, #4, #5, #6, #7, #8, #9 and #10.

6.15 Solution #14: Security procedure for xApp registration

6.15.1 Introduction

As part of the xApp registration procedure into the Near-RT RIC platform, the platform assigns an identity (ID) to the xApp, so called xApp ID. This xApp ID is used in xApp's API request messages to the Near-RT RIC platform to facilitate the Near-RT RIC platform the identification of the xApp (API service consumer).

The solution proposes a mechanism to enhance the actual registration procedure by embedding the xApp ID into the provided certificate used for authentication (mTLS) and for authorization (OAuth – client authentication) according to the parametrization in [19].

NOTE 1: The proposed procedure is based on the security procedures specified in 3GPP TS 33.122 (Security aspects of Common API Framework (CAPIF) [31] for 3GPP northbound APIs), clause 6.1. Nevertheless, the implementation of the procedure does not require the implementation of the CAPIF framework in O-RAN Near-RT RIC platform.

In addition, the solution proposes UUID (version 4) as xApp ID format, ensuring the uniqueness and randomness of the identifier.

6.15.2 Solution details

The procedure begins with a secure session between the xApp and Near-RT RIC platform using TLS, which uses server- side certificate authentication. It is assumed that there is a pre-provisioning of necessary parameters for registration during the onboarding/deployment process of the xApp, to allow the latter to reach the Near-RT RIC platform, authenticate it, and present a registration credential (e.g., OAuth 2.0 access token, message digest, etc.) that can be verified by the Near-RT RIC platform.

NOTE 2: The configuration of the initial pre-provisioned registration credentials in the xApp as part of the deployment process are left to operator implementation. O-Cloud DMS is referred as a possible entity in charge of the provisioning task.

With the secure session established, the xApp sends a registration request message to the Near-RT RIC platform (Management Function). The message carries the initial credential obtained during the pre-provisioning of the registration information, which may be an OAuth 2.0 access token. If OAuth 2.0 mechanism is used for initial credentials, it should be profiled as specified in [19]. Other type of credentials may also be used (e.g., message digest, pre-shared key)

Figure 6.15.2-1 details the security information flow for the security procedure. The OAuth 2.0 token-based authentication initial registration credential is shown in this example.

```
@startuml
participant xapp as "xApp\n[API service consumer]"
participant nfo as "Provisioning system\n(NFO in SMO)"
participant pf as "Near-RT RIC platform \n(Security function,\nRA functionality)\n[API
service producer]"
participant pki as "Operator PKI\nCA"
```

```
xapp <- nfo : 1. Registration information\n[Near-RT RIC Platform/Security function
\n(Address,Root CA Certificate),\n OAuth 2.0 Access token]
xapp -> pf : 2. TLS (Server side certificate based authentication)
rnote over xapp
     3. Generate the key pair
     [private Key, public Key]
endnote
xapp -> pf : 4. Registration request (by xApp instance)\n[OAuth 2.0 access token, xApp
Instance CSR]
rnote over pf
     5. Verify OAuth access token
     Generate the xApp ID
     POP (Proof of Posession of Private Key)
     Embed xApp ID in the CSR (subjectAltname)
endnote
pf -> pki : 6. Enrolment protocol \n(e.g., CMPv2)
pf <- pki : 7. Issued certificate \n(embedded xApp ID)
rnote over pf
     8. Generate xApp MOI
endnote
xapp <- pf : 9. Registration Response (for xApp instance)\n[xApp ID, xApp Certificate,
(service API authentication \nand Authorization information)]
@enduml
```

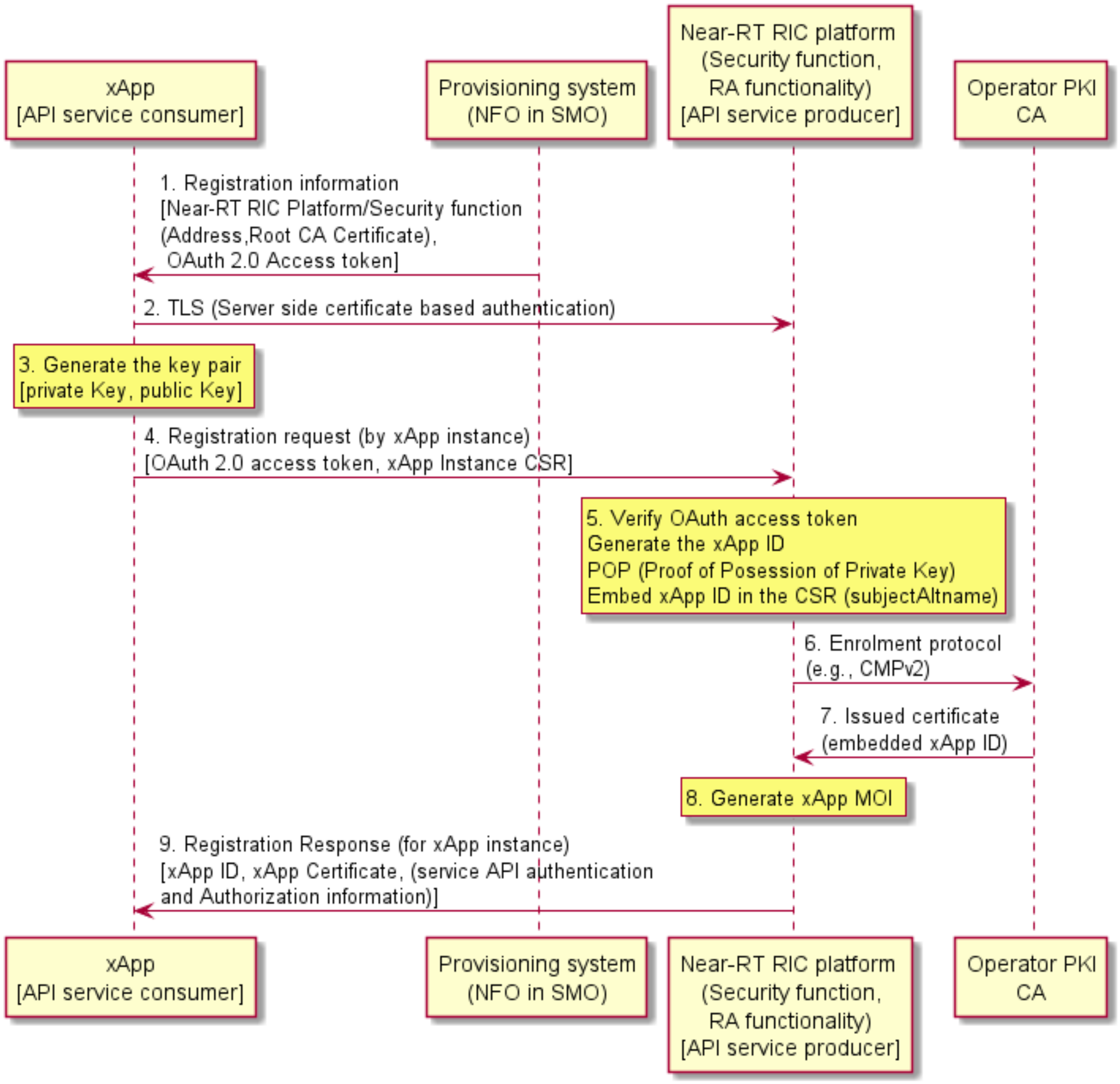

xApp
[API service consumer]
Provisioning system
(NFO in SMO)
Near-RT RIC platform
(Security function,
RA functionality)
[API service producer]
Operator PKI
CA
1. Registration information
[Near-RT RIC Platform/Security function
(Address,Root CA Certificate),
OAuth 2.0 Access token]
2. TLS (Server side certificate based authentication)
3. Generate the key pair
[private Key, public Key]
4. Registration request (by xApp instance)
[OAuth 2.0 access token, xApp Instance CSR]
5. Verify OAuth access token
Generate the xApp ID
POP (Proof of Posession of Private Key)
Embed xApp ID in the CSR (subjectAltname)
6. Enrolment protocol
(e.g., CMPv2)
7. Issued certificate
(embedded xApp ID)
8. Generate xApp MOI
9. Registration Response (for xApp instance)
[xApp ID, xApp Certificate, (service API authentication
and Authorization information)]

Figure 6.15.2-1: Security procedure for xApp registration

1. As a pre-requisite to the registration procedure, the xApp obtains information from a provisioning system (NFO in SMO) during the onboarding/deployment phase in the infrastructure. This information is used to authenticate and establish a secure TLS communication with the Near-RT RIC platform during the registration process. The information includes details of the Near-RT RIC platform (address, Root CA certificate) and includes an initial registration credential (the OAuth 2.0 Access token in the example).

2. The xApp and Near-RT RIC platform (e.g., Security function and/or Management function) establish a secure session based on TLS (server-side certificate authentication). The xApp uses the information obtained in step 1 to establish the TLS session.

Editor’s Note: if TLS communication should be mandatory in the procedure is ffs.

Editor’s Note: the termination point of the TLS communication in Near-RT RIC platform (e.g. Management Function, Security Function) is ffs.

3. The xApp generates the key pair, i.e., private key and public key

4. After successful establishment of the TLS session, the xApp instance sends a registration request message to the Near-RT RIC platform along with the pre-provisioned initial registration credential (OAuth 2.0 token in the example), and the xApp instance public key in corresponding CSR (Certificate Signing Request) message.

5. The Near-RT RIC platform (e.g., via Security function) should validate the initial registration credential and the Management Function of the platform should generate an xApp ID for that particular xApp instance. At the reception of the CSR message from the xApp, the Near-RT RIC platform (e.g., via Security function), acting as a Registration Authority (RA), should prove that the xApp instance is in possession of the private key, and if the proof of possession procedure is positive, it should embed the xApp ID in the CSR message to be forwarded to the Operator CA. The RA may use an enrolment protocol (e.g., CMPv2) to fetch the certificate from the CA

NOTE 3: Authentication mechanism between RA and CA are part of the operator PKI implementation.

6. Fetching the certificate (e.g., enrolment protocol such as CMPv2)

7. Operator CA issues a certificate, embedding the xApp ID in subjectAltName field. The issued certificate by the operator CA will be used by the xApp for subsequent authentication and authorization procedures between the xApp and the Near-RT RIC platform when services/resources are consumed by xApps via APIs.

8. The Near-RT RIC platform (Management Function) generates an xApp Managed Object Instance (MOI) as specified in [3], which may contain the mechanism for authentication (e.g., mTLS) and authorization (e.g., OAuth 2.0) between the xApp and the corresponding module of the Near-RT RIC platform.

9. The Near-RT RIC platform (Management Function) responds with a xApp registration response message. The response should include the assigned xApp ID, authentication and authorization mechanism if provided in step 5 and xApp certificate.

The solution proposes for the data type of the xApp ID a string that uniquely identifies the xApp instance. The format of this string should be a Universally Unique Identifier (UUID) version 4 (as described in IETF RFC 4122)

The solution proposes that subjectAltName in the xApp instance certificate should contain a URI-ID with the URI for the xApp ID as an URN; this URI-ID should contain the xApp ID of the xApp instance using the UUID format as described in IETF RFC 4122 (e.g., urn:uuid:f81d4fae-7dec-11d0-a765-00a0c91e6bf6)

6.15.3 Evaluation

The following diagram shows a proposal to insert the security procedure in the existing registration procedure specified in O-RAN.WG3.RICARCH, clause 9.1.4, where xApp ID is assigned:

```
@startuml
skinparam ParticipantPadding 5
skinparam BoxPadding 10
skinparam defaultFontSize 12
skinparam lifelineStrategy solid

participant smo as "SMO"
Box "O-Cloud"
participant dms as "DMS"
endbox
participant xApp as "xApp"
Box "Near-RT RIC Platform"
participant o1t as "O1 Termination"
participant ms as "Management Function + Security Function"
endbox

dms -> xApp : 0. Pre-provisioning of registration information
xApp <-> ms : TLS session
xApp -> ms  : 1. <<MS>> registration request (initial credentials)

rnote over o1t, ms
     2. registration processing, validate,
          assign xApp ID and bind it to xApp certificate,
               create xApp MOI
     endnote

ms  -> xApp  : 3. <<MS>> registration response

Opt If SMO has subscribed
o1t -> smo   : 4. <<O1>> Notify MOI Creation
End

@enduml
```

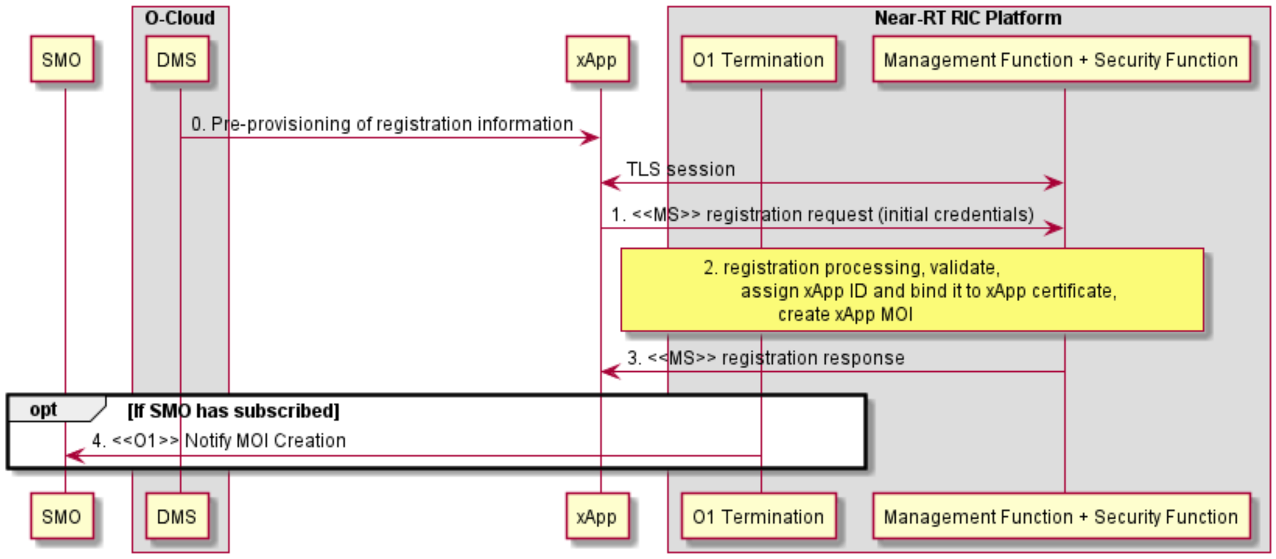

O-Cloud
Near-RT RIC Platform
SMO
DMS
xApp
O1 Termination
Management Function + Security Function
0. Pre-provisioning of registration information
TLS session
1. <<MS>> registration request (initial credentials)
2. registration processing, validate, assign xApp ID and bind it to xApp certificate, create xApp MOI
3. <<MS>> registration response
opt
[If SMO has subscribed]
4. <<O1>> Notify MOI Creation

Figure 6.15.3-1: xApp registration process with Security procedure

6.16 Solution #15: Security requirement proposals for Y1 interface

6.16.1 Introduction

The O-RAN architecture description document [32] explains that Y1 services can only be consumed by Y1 consumers after mutual authentication and authorization. That would imply that the result of the authentication for the duration of communications must be maintained, e.g. by integrity protection and/or encryption with a key that is derived from the authentication and is unknown to the adversary.

In the ongoing discussion two types of APIs are being discussed:

- RAI API: API internal to the Near-RT RIC
- Y1 API: API exported by the Y1T (Y1 Termination) to consumers outside of the Near-RT RIC (the Y1 consumers)

The RAI API is identical to other Near-RT RIC APIs regarding its requirements on security, so key issues #10 (see 5.11) and #11 (see 5.12) as well as solutions #3 (see 6.4), #4 (see 6.5), #7 (see 6.8) and #13 (see 5.14) apply.

In discussion two types of scenarios are related to the Y1 interface, involving a new entity called Y1T inside the Near RT RIC platform, providing a provider interface to be consumed by external Y1 consumers [34]:

- type 1a: Y1T consumes RAI and exposes externally to Y1 consumers
- type 2b: Near-RT RIC platform generates RAI and exposes it to ETSI MEC, as the source of Radio Network Information Service (RNIS)

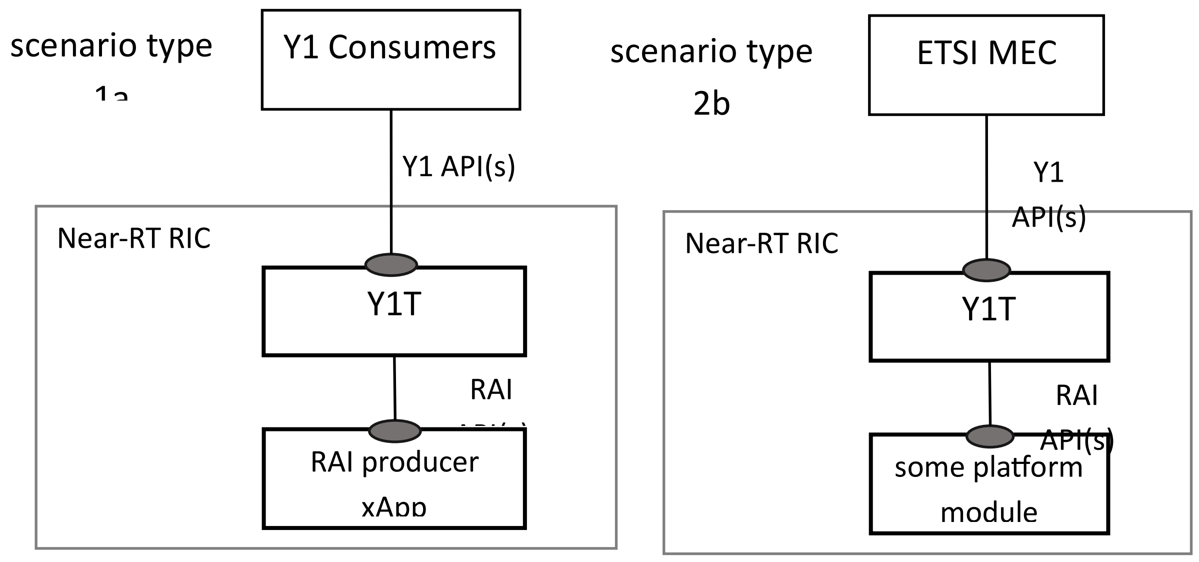


Figure 6.16-1 Scenarios relevant to the Y1 interface security

Based on the ongoing discussion it can be assumed that the topology of the PLMN trust domain should be hidden from the Y1 consumers accessing the Y1 services from outside the PLMN trust domain.

6.16.2 Solution details

The following proposals for requirements have been derived from [31], clause 4.

- The Y1 interface shall provide mechanisms to authenticate the Y1 consumer and allow for the Y1 consumer to authenticate the Y1 provider (mutual authentication)
- The Y1 interface shall authorize the Y1 consumer before allowing access to any Y1 service
- The Y1 interface shall provide mechanisms to provide confidentiality and integrity protection for all data exchanged
- The Y1 interface shall provide replay- protection for all data exchanged
- The Y1 interface shall enforce the result of the authentication for the duration of communications (e.g. by integrity protection or implicit authentication by encryption with a key that is derived from the authentication and is unknown to the adversary).
- Y1T shall provide mechanisms to hide the topology of the PLMN trust domain from the Y1 consumers accessing the Y1 services from outside the PLMN trust domain
- Further security controls for the Near-RT-RIC that could be implemented as part of its Y1T include:
  1. Validate received values for validity and range
  2. Provide rate limiting on Y1 interface to prevent resource exhaustion and DoS
  3. Implement security logging for each of the above failure events

6.16.3 Evaluation

This solution addresses key issue #13 of the present document.

6.17 Solution #16: Additional security measures for the E2 interface

6.17.1 Introduction

The Near-RT RIC receives Near real-time information from the E2 Nodes across the E2 interface. While the E2 interface is considered secure with controls that provide confidentiality, integrity, and mutual authentication, the Near-RT RIC should not assume that the data received is valid and trusted. The Near-RT RIC should provide built-in security compliant with a zero-trust architecture based upon the principle that perimeter security is insufficient to protect against internal threats.

6.17.2 Solution details

Security controls for the Near-RT-RIC that could be implemented as part of its E2 Termination include:

1. Validate received values for validity and range
2. Provide rate limiting on E2 interface to prevent resource exhaustion and DoS
3. Implement security logging for each of the above failure events

6.17.3 Evaluation

The solution addresses key issue #14.

6.18 Solution #17: Standard Identity management of xApp during xApp Registration Procedure.

6.18.1 Introduction

The Near-RT RIC platform identifies and assigns a xApp ID to the xApp that is being deployed on its platform. However, there is no universally established identity for applications in a cloud native environment that can be used by a program and other mechanisms are more suitable to identify workloads in a cloud native environment.

The solution proposes to use SPIFFE–- Secure Production Identity Framework which is a universal service identity control plane designed for cloud-native and zero-trust based architectures.

This is also in adherence to **NIST SP 800-207A** [41] which mentions that "service identity should not be subjected to spoofing and should be continuously verifiable. An example of workload identity is a SPIFFE ID which is encoded as a Uniform Resource Identifier (URI) and carried in a cryptographically verifiable document called a SPIFFE Verifiable Identity Document (SVID). "

SPIRE is a production-ready implementation of SPIFFE APIs that uses Node and workload attestation to securely issue SVIDs [X.509 certificate with SPIFFE ID] to Applications.

The main modules of the SPIFFE Specifications that are being used in the solution are:

1. SPIFFE ID: Standard URI format based on RFC 3986 with certain conditions. [EXAMPLE: spiffe://domain-name/clusterName/namespace/xAppName]
2. SVID (SPIFFE verifiable Identity document): Cryptographically verifiable document used to prove a service's identity to a peer. It uses existing formats which are X509 certificates and JWT tokens.
3. Trust bundle: The CA certificate chain (root certificate and intermediate certificates) which entities use to verify the SVID.
4. SPIRE Agent and SPIRE Server: SPIRE Agent deployed on the Node where xApp is deployed. SPIRE Server module is deployed in the Near-RT RIC Platform.
5. Optional Certificate side-car/ [SPIFFE Helper] container along with xApp Instance Pod for handling key and certificate management.

As part of the xApp registration procedure into the Near-RT RIC platform, the xApp instance will use the SPIFFE ID for registering into the Near-RT RIC platform and will allow the platform to interpret the identity of the xApp instance via the structured SPIFFE ID format.

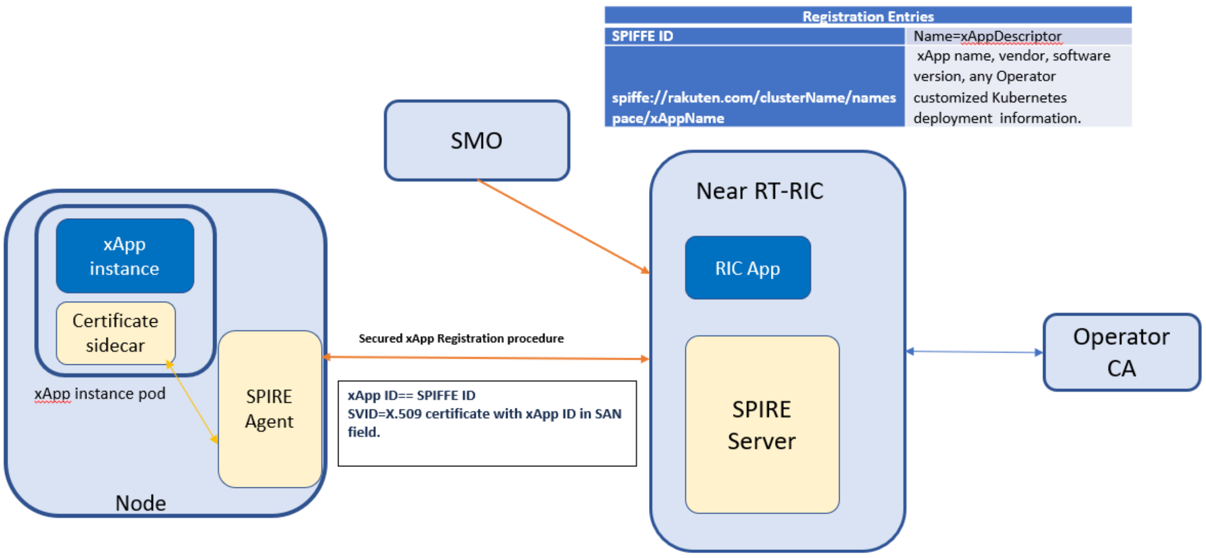

Registration Entries
SPIFFE ID
spiffe://rakuten.com/clusterName/namespace/xAppName
Name=xAppDescriptor
xApp name, vendor, software version, any Operator customized Kubernetes deployment information.
SMO
Near RT-RIC
RIC App
SPIRE Server
Operator CA
xApp instance
Certificate sidecar
xApp instance pod
SPIRE Agent
Node
Secured xApp Registration procedure
xApp ID== SPIFFE ID
SVID=X.509 certificate with xApp ID in SAN field.

Figure 6.18.1-1: Secured xApp Registration Procedure using SPIRE.

6.18.2 Solution details

The steps outlining the solution are elaborated below.

Step 1: Define the Actors and Roles in the xApp Registration Procedure and the Pre-requisite.

- xApp Instance Pod: xApp Application, (optional) Certificate Side Car originator for certificate and key management.

- SPIRE Agent: To oversee identity management and conduct workload verification within the system.
- SPIRE Server in Near-RT RIC platform: handles the registration-related message from/to xApp. A central component responsible for managing identities and issuing identity credentials.

Pre-requisites: As a pre-requisite, the xApp Instance Node should have the information details of the Near-RT RIC platform (IP address, Trust anchors) from the provisioning system.

Step 2: **Node Attestation**: SPIRE Agent and SPIRE Server Secured channel establishment.

2.i SPIRE Agent initiates the mTLS with the SPIRE Server for its attestation. SPIRE Agent uses the Node certificates in this step which can be Kubernetes issued or Operator signed certificates. Step 2.ii to 2.iv details the procedures for Nodes Certificates signed by Operator CA.
2.ii SPIRE Server responds back to SPIRE Agent with its own certificate and chain.
2.iiiThe SPIRE Agent verifies the SPIRE server certificate and responds back with its own node certificate.
2.ivThe x509pop node attestor plugin at the SPIRE Server verifies that the certificate is rooted to a trusted set of CAs and verifies that node is in possession of private key. The TLS connection gets successfully established.
2.v In case of Kubernetes issued certificates the additional steps from 2.vi to 2.ix should be followed.
2.viSPIRE Server shall generate a CSR on behalf of the SPIRE Agent and send to the Operator CA for signing.
2.vii The Operator CA signs the CSR and sends the SVID [X.509 certificate with SPIFFE ID in the SAN field] to the SPIRE Server.
2.viii The SPIRE Server sends the TRUST Bundle to the SPIRE Agent over the existing mTLS connection.
2.ixThe SPIRE Agent will re-establish the mTLS connection with the newly received SVID.

NOTE: Steps 2.i to 2.iv are for Node Attestation using Node certificates signed by Operator CA.

Step 2.v to 2.ix are for Node Attestation using Kubernetes issued Certificates. In such cases Kubernetes CA certificates are pre-configured in the SPIRE server.

Step3: **Registration entries of Workload in SPIRE Server**: As mentioned in clause 6.2.2, the SMO maps the RIC variables and instructs the O-Cloud DMS to create the xApp Deployment and allocate the resources necessary on the Near-RT RIC. As a part of this procedure, the SMO should send the xApp Registration entries [Management Descriptor] via its O1 interface to the Near-RT RIC. It would map a structured SPIFFE ID to each xApp Instance and facilitate the Near-RT RIC to identify the xApp being deployed and manage its life cycle.

NOTE: Also the SPIRE Server's Registration API, could be exposed to SMO for the SPIRE Server to obtain the workload Registration entries.

EXAMPLE: Registration Entries of two xApp Instances

Registration Entries	
SPIFFE ID	1. Name=xAppDescriptor
spiffe://domain-name/clusterName/namespace/xAppName-1	2. xApp name-1, vendor, software version, containers, pod- label(label1), and any Operator customized Kubernetes deployment information.
spiffe://domain-name/clusterName/namespace/xAppName-2	3. xApp name-2, vendor, software version, containers, pod- label(label2), and any Operator customized Kubernetes deployment information.

Step 4: **Registration Entries in Agent for SVID Issuance:** SPIRE Server sends the workload (xApp) Registration entry details to the SPIRE agents registered on it.

Step 5: **Workload Attestation**: Workload xApp Instance is instantiated. Also, an optional component of side-car container can be instantiated to handle the certificate and key management of xApp Instance which offloads the xApp Instance for security related functions.

5.i The xApp Instance Pod communicates with the SPIRE Agent over a unix domain socket-based communication.
5.ii The SPIRE Agent utilizes a combination of kernel and user space calls to gather supplementary information about the workload. The SPIRE Agent interrogates the node's kernel to identify the process ID of the caller. It then invokes any configured workload attestor plugins, providing them with the process ID of the workload. This process involves identifying the workload's pod ID through its cgroup association and subsequently obtaining information about the pod.
5.iii The agent introspects a workload to determine its properties, based on a set of selectors associated with it.
5.iv SPIRE Agent ascertains the identity of the workload by cross-referencing the discovered workload selectors with the Registration entries.

Step 6. **Generate CSR**: On successful validation in step 5, the SPIRE Agent will generate the CSR and private key for each workload (xApp). The CSRs will have the SPIFFE ID in the SAN field. The SPIFFE agent derives the SPIFFE ID from the Registration entries in Step 4 which uniquely identifies each xApp instance.

Step 7. SPIRE Agent sends xApp CSRs to the SPIRE server.

Step 8. **CSR Signing**: SPIRE Server in turns sends the xApp CSRs to the Operator CA for signing.

Step 9. **Operator PKI**: The Operator CA provides xApp SVID to the SPIRE Server. SVID are X.509 certificates with SPIFFE ID in the SAN field.

Step 10. SPIRE Server sends xApp SVID [X.509 Certificates with SPIFFE ID] and trust bundle to the SPIRE Agent which it stores in memory for a short duration of time.

Step 11. **Issuance of SVID:** SPIRE Agent then provides the xApp SVID, trust bundle to the certificate sidecar. The xApp Instance will then use the SVIDs for further communication like xApp Registration towards Near real-time RIC platform.

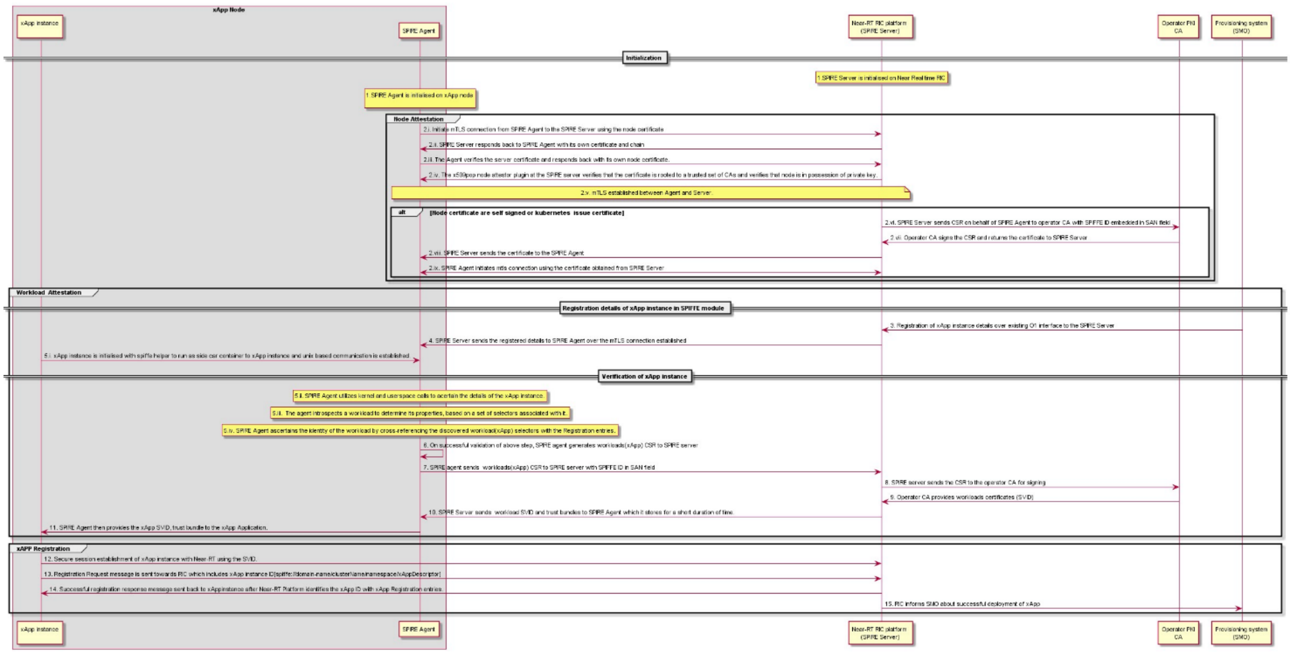
NOTE: The xApp instance would establish a secure session with the Near-RT RIC as per the Pre-requisites mentioned in Step 1.

Step 12: The xApp Instance sends Registration Request toward the Near-RT RIC platform over a secured TLS connection. The Near-RT RIC can validate the SAN field of the X.509 certificate of the xApp Instance against the Registration entries.

Step 13. On successful validation, the Near-RT RIC platform responds with xApp registration response message with the xApp ID.

Step 14. Optional: The Near-RT RIC informs SMO about the successful deployment of xApp.

The solution proposes that xApp ID use a structured ID format of SPIFFE ID which can be interpreted programmatically. The mechanism introduced does not rely on information from xApp to determine the xApp ID. Unix socket-based communication is used for process-to-process communication within the Node. Optional side-car container for key and certificate management reduces the attack surface on the Applications.


xApp Node
xApp instance
SPIRE Agent
Near-RT RIC platform (SPIRE Server)
Operator PKI CA
Provisioning system (SMO)
Initialization
Node Attestation
alt
Workload Attestation
Registration details of xApp instance in SPIFFE module
Verification of xApp instance
xAPP Registration

1.SPIRE Server is initialised on Near Real time RIC

1. SPIRE Agent is initialised on xApp node

2.i. Initiate mTLS connection from SPIRE Agent to the SPIRE Server using the node certificate

2.ii. SPIRE Server responds back to SPIRE Agent with its own certificate and chain

2.iii. The Agent verifies the server certificate and responds back with its own node certificate.

2.iv. The x509pop node attestor plugin at the SPIRE server verifies that the certificate is rooted to a trusted set of CAs and verifies that node is in possession of private key.

2.v. mTLS established between Agent and Server.

[Node certificate are self signed or kubernetes issue certificate]

2.vi. SPIRE Server sends CSR on behalf of SPIRE Agent to operator CA with SPIFFE ID embedded in SAN field

2.vii. Operator CA signs the CSR and returns the certificate to SPIRE Server

2.viii. SPIRE Server sends the certificate to the SPIRE Agent

2.ix. SPIRE Agent initiates mtls connection using the certificate obtained from SPIRE Server

3. Registration of xApp instance details over existing O1 interface to the SPIRE Server

4. SPIRE Server sends the registered details to SPIRE Agent over the mTLS connection established

5.i. xApp instance is initialised with spiffe helper to run as side car container to xApp instance and unix based communication is established

5.ii. SPIRE Agent utilizes kernel and userspace calls to acertain the details of the xApp instance.

5.iii. The agent introspects a workload to determine its properties, based on a set of selectors associated with it.

5.iv. SPIRE Agent ascertains the identity of the workload by cross-referencing the discovered workload(xApp) selectors with the Registration entries.

6. On successful validation of above step, SPIRE agent generates workloads(xApp) CSR to SPIRE server

7. SPIRE agent sends workloads(xApp) CSR to SPIRE server with SPIFFE ID in SAN field

8. SPIRE server sends the CSR to the operator CA for signing

9. Operator CA provides workloads certificates (SVID)

10. SPIRE Server sends workload SVID and trust bundles to SPIRE Agent which it stores for a short duration of time.

11. SPIRE Agent then provides the xApp SVID, trust bundle to the xApp Application.

12. Secure session establishment of xApp instance with Near-RT using the SVID.

13. Registration Request message is sent towards RIC which includes xApp instance ID[spiffe://domain-name/clusterName/namespace/xAppDescriptor]

14. Successful registration response message sent back to xAppinstance after Near-RT Platform identifies the xApp ID with xApp Registration entries.

15. RIC informs SMO about successful deployment of xApp

Figure 6.18.2-1: x-App Registration procedure using SPIFFE Framework.

6.18.3 Evaluation

The following proposal uses a cloud native solution for service identity management of applications in a cloud native environment. This is in adherence to NIST 800-207A[41]. SPIRE is a production-ready implementation of SPIFFE APIs that uses Node and workload attestation to securely issue SVID [X.509 certificate with SPIFFE ID] to Applications which can be used for service-service communication. Using SPIRE Agent and SPIRE Server in an O-RAN environment, standardized identities can be provided to Applications like xApp. The SVID issued to xApp can be used to establish a secure connection with the Near-RT RIC platform and the xApp ID can be derived from the SAN field of the X.509 certificate.

The three open item in the proposal have been listed below:

1. The RICARCH [3] does not currently describe the initial configuration of Registration entries via the O1 interface from the SMO. This procedure would need to be defined.
2. The RICARCH [3] clause 9.4.1 assumes that the Near-RT RIC platform creates the xAppID. In this proposal, the Near-RT RIC is obtaining the xApp ID from SMO via Registration entries. Another alternative would be to read the SAN field of the xApp Instance certificate.
3. Handling of private keys in SPIRE Agent and SPIRE Server as an alternative to in-memory and disk-based.

7. Conclusions

7.1 Key Issue Solution Mapping

This document currently contains 14 key issues and 17 solutions proposals to address or reduce the risk associated with the key issues. A mapping of the solutions to the key issues is shown here.

Key issue	Sol # 1	Sol #2	Sol #3	Sol #4	Sol #5	Sol #6	Sol #7	Sol #8	Sol #9	Sol #10	Sol #11	Sol #12	Sol #13	Sol #14	Sol #15	Sol #16	Sol #17
KI 1: Malicious xApps	X	X	X	X									X	X			X
KI 2: Compromised xApp	X	X	X	X									X				
KI 3: Conflicting xApps								X			X						

Key issue	Sol # 1	Sol #2	Sol #3	Sol #4	Sol #5	Sol #6	Sol #7	Sol #8	Sol #9	Sol #10	Sol #11	Sol #12	Sol #13	Sol #14	Sol #15	Sol #16	Sol #17
KI 4: Compromise of xApp isolation			X	X	X		X			X			X				
KI 5: Compromise of ML data pipelines		X	X	X	X			X					X				
KI 6: Altering ML training models		X	X	X	X				X				X				
KI 7: Database protection and privacy			X	X	X		X						X				
KI 8: Untrusted xApps	X	X	X	X			X						X				
KI 9: Malicious A1 policies			X	X					X			X	X				
KI 10: RIC API authentication security			X	X			X	X					X				
KI 11: RIC API authorization security			X	X			X										
KI 12: xAppID abuse														X			X
KI 13: Malicious Y1 Consumer															X		
KI 14:																X	

Key issue	Sol # 1	Sol #2	Sol #3	Sol #4	Sol #5	Sol #6	Sol #7	Sol #8	Sol #9	Sol #10	Sol #11	Sol #12	Sol #13	Sol #14	Sol #15	Sol #16	Sol #17
Malicious E2 Node																	

8. Recommendations

The following key issues have been agreed as contributions in O-RAN Threat Model v.03.00:

- Key issue #1: Malicious xApps deployed on the Near-RT RIC
- Key issue #10: Near-RT RIC APIs authentication
- Key issue #11: Near-RT RIC APIs authorization

The following solutions have been agreed as contributions in O-RAN Security Requirement Specification v3.0:

- Solution #1: Secure onboarding and deployment of xApps in Near-RT RIC
- Solution #3: Authentication schema for APIs
- Solution #4: Authorization framework for APIs

History

Date	Revision	Description
22.11.2023	5.00.00	Version 5 of technical report
20.07.2023	4.00.00	Version 4 of technical report
24.02.2023	3.00.00	Version 3 of technical report
08.11.2022	2.00	Version 2 of technical report
24.04.2022	2.0.1	Update of Near-RT RIC architecture and editorial corrections
23.03.2022	2.0.0	First complete version
06.12.2021	1.0.0	First draft / incomplete