Endless Subscriptions: Open RAN is Open to RIC E2 Subscription Denial of Service Attacks

Felix Klement
Chair of Computer Engineering
University of Passau
Passau, Germany
felix.klement@uni-passau.de

Alessandro Brighente
Department of Mathematics
University of Padova
Padova, Italy
alessandro.brighente@unipd.it

Anup Kiran Bhattacharjee
Networked Systems
Delft University of Technology
Delft, The Netherlands
a.k.bhattacharjee@tudelft.nl

Stefano Cecconello
Department of Mathematics
University of Padova
Padova, Italy
stefano.cecconello@unipd.it

Fernando Kuipers
Networked Systems
Delft University of Technology
Delft, The Netherlands
F.A.Kuipers@tudelft.nl

Georgios Smaragdakis
Cybersecurity
Delft University of Technology
Delft, The Netherlands
g.smaragdakis@tudelft.nl

Mauro Conti
Department of Mathematics
University of Padova
Padova, Italy
mauro.conti@unipd.it

Stefan Katzenbeisser
Chair of Computer Engineering
University of Passau
Passau, Germany
stefan.katzenbeisser@uni-passau.de

***Abstract*—Telecommunication services are essential in ensuring the operation of numerous critical infrastructures. While mobile network security increased with the advancement of generations, emerging concepts such as the Open Radio Access Network (O-RAN) are transforming the traditional operation of Radio Access Networks (RANs). Novel concepts and technologies are finding their way into RANs with a focus on softwareization and virtualization. This increases the overall attack surface and introduces new attack vectors not necessarily found in traditional RANs. This paper shows that Denial of Service (DoS) attacks leveraging subscription mechanisms can compromise O-RAN implementations. We present a novel DoS attack targeting the Near Real-Time (Near-RT) RAN Intelligent Controller (RIC). By deploying a malicious xApp, we demonstrate how an adversary can flood the Near-RT RIC with excessive subscription requests, leading to service disruption. This attack exploits the lack of rate-limiting mechanisms within the Service Model (SM), a critical component of the Near-RT RIC responsible for handling E2 subscription requests. We systematically evaluate various attack scenarios and investigate the underlying vulnerabilities exposed. Furthermore, we propose and assess countermeasures to safeguard publicly accessible O-RAN systems from such threats.**

***Index Terms*—Open RAN, O-RAN, Radio Access Networks, DoS, Network Security.**

1. Introduction

In the era of telecommunications technologies, the importance of robust and secure telecommunications systems cannot be emphasized enough. The essential elements to be protected are the integrity and availability of the individual networks and the privacy of the respective users. However, as these networks integrate a broader array of diverse technologies, they face increasing demands and a growing risk of security threats and vulnerabilities. Communication networks are an essential part of critical infrastructures, and their failure can cause far-reaching problems. This makes them particularly interesting targets and therefore highly susceptible to cyber attacks. The increasing complexity of the systems poses new challenges, for which software and virtualization is used in the respective components as a solution. However, this also makes the systems more susceptible to classic Denial of Service (DoS) attacks. Understanding and addressing these emerging threats is essential to ensure the resilience and security of next-generation wireless systems, underscoring the need for focused investigation and research in this domain.

One of the most impactful and recent advancements in Radio Access Network (RAN) technology is the introduction of the Open Radio Access Network (O-RAN) architecture [1]. O-RAN aims to bring enhanced programmability, openness, and interoperability to 5G and 6G access networks by building on the foundational principles of Software-defined Networking (SDN) and Network Function Virtualization (NFV). This shift aligns with the broader trend of migrating RAN functions to the cloud, following earlier innovations such as cloud RAN [2] and virtual RAN [3]. O-RAN has received significant attention from both academia and industry, with numerous real-world deployments and trials conducted by leading mobile operators globally [4], [5]. O-RAN has a number of use cases that currently make the concept so prominent. These include specialized enterprise networks

[6], energy efficient RAN operations [7], [8], cell-free massive Multiple Input, Multiple Output (MIMO) [9] and Artificial Intelligence (AI)-enabled RAN [10], [11]. As O-RAN becomes increasingly critical to modern network infrastructure, understanding its potential and challenges is essential, making it a compelling subject for further exploration.

The RAN Intelligent Controller (RIC) [12], a novel network component that enables intelligence through the deployment of eXtended Applications (xApps), i.e., micro-services that implement AI-enabled functions within O-RAN. Thanks to the RIC, it is possible to provide the network with high reconfiguration and adaptation capabilities, as xApps can dynamically choose RAN network functions as they see fit [13], [14]. This presents numerous opportunities for future innovation while simultaneously introducing a significant number of critical components from a security standpoint.

Motivation. The introduction of xApps necessitates a substantial number of interfaces to control and manage the desired functions within the RAN. This significantly increases the risk of traditional DoS attacks, as such interfaces represent a common attack vector for this type of component. Currently, the volume of traffic and damage caused by DoS attacks is at an all-time high, with these attacks being ranked among the top ten cyber threats by both the US Cybersecurity and Infrastructure Security Agency (CISA) [15], [16] and the European Union Agency for Cybersecurity (ENISA) [17]. Consequently, we conclude that O-RAN components are highly susceptible to such attacks and are likely to become increasingly targeted in the future. As this paper demonstrated, significant damage can often be caused through relatively simple methods. The impact on end users is also concerning. For instance, a successful attack on the Near Real-Time RIC (Near-RT RIC) could prevent users from connecting to the RAN, effectively leaving them without service.

Numerous prominent publications primarily focus on the general security implications of the ecosystem, offering discussions on potential attacks and countermeasures without conducting specific tests [18], [19]. The O-RAN alliance dedicates a working group to security evaluations. Also, in this case, the documentation provides hints on possible attacks without going into implementation details [20]. Recently, a few authors proposed some attacks on the O-RAN control plane [21]–[23], and to O-RAN xApps access control and E2 interface [24], [25].

Our objective is to study the impact of DoS to identify potential attack vectors. The subsequent section details our efforts in this domain.

Contributions. In this paper, we propose and practically demonstrate a novel DoS attack against implementations of the Near-RT RIC. Through a malicious xApp, we show that an attacker can send a large number of subscription requests to the Near-RT RIC rendering it unavailable. The attack exploits the lack of a rate-limiting solution in the Service Model (SM), a logical component in the Near-RT RIC that manages E2 subscription requests. Instead of providing a security analysis of all the relevant near-RT RIC interfaces, we focus on a specific DoS vulnerability and validate it over different near-RT RIC implementations. Compared to the existing literature on E2 security, we hence provide a practical attack and its evaluation to showcase the effect of this vulnerability and investigate possible countermeasures.

To validate our attack, we test it on two prominent implementations of the RIC, i.e., the FlexRIC [26] and the O-RAN Software Community (OSC) RIC[1].

Through our approach, we successfully identify three Common Vulnerabilities and Exposuress (CVEs) associated with both FlexRIC and OSC implementations. While the underlying attack methodology is not entirely new, our findings highlight its significant impact on contemporary software, particularly in the context of cellular network control. This underscores the pressing need for enhanced security measures in these critical systems. To mitigate the vulnerabilities exposed, we propose a straightforward rate-limiting solution and demonstrate its efficacy using the FlexRIC Software Development Kit (SDK). The results of this study emphasize the importance of addressing these threats and provide valuable insights into improving the resilience of modern telecommunication infrastructures.
We summarize our contributions as follows:

- We demonstrate that the publicly available implementations of O-RAN exhibit vulnerabilities to RIC DoS E2 subscription attacks.
- We measure and evaluate different scenarios to disrupt each implementation and investigate the root causes of the identified vulnerabilities.
- We propose and assess countermeasures for all publicly accessible O-RAN implementations.
- We disclose the vulnerabilities to the developers of the OSC and FlexRIC and got three CVEs assigned.
- We provide both the attack code and the mitigation software for FlexRIC in [27].

2. Background

In this section, we present the fundamental concepts about O-RAN. We provide a short overview of the O-RAN working principles and functional split in Section 2.1. We then provide details about the Near-RT RIC and xApps in Section 2.2, while presenting its main implementations in Section 2.3.

2.1. O-RAN Overview

Fig. 1 depicts an overview of O-RAN's main components and their interconnections within the *data plane* and the *control plane*.

In the data plane, the O-RAN architecture splits the RAN components into separate entities. The cell site performs basic operations, such as radio frequency and low-level physical-layer processing [1]. The edge O-Cloud [28], directly connected to the cell site, processes higher-level information through its three main components, i.e., the O-RAN Distributed Unit (O-DU), the O-RAN Central Unit - User Plane (O-CU-UP), and the O-RAN Central Unit - Control Plane (O-CU-CP). The edge O-Cloud then communicates with the control plane via the *E2 interface*, which enables communications with the regional O-Cloud [28]. In particular, thanks to the *E2 termination*, the Near-RT RIC receives data-plane information

1. https://wiki.o-ran-sc.org/display/RICP/Introduction+and+guides

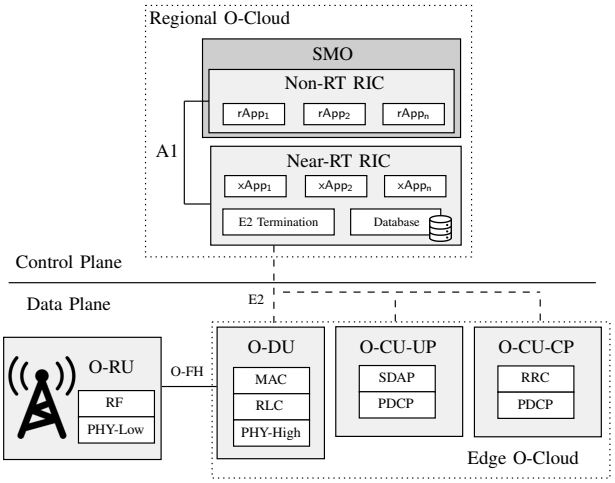


Figure 1: O-RAN architecture. We emphasize the functional split in the access network proposed by the O-RAN alliance and the separation between elements of the control plane and the data plane.

and processes it to provide near-real-time control actions for tasks, such as scheduling and resource allocation. Via the A1 interface, the Near-RT RIC exchanges information with the Service Management and Orchestration (SMO) framework, and in particular with the Non Real-Time RIC (Non-RT RIC), which provides control over a longer period (non-real-time). Notice that a detailed explanation of the exact role and working principle of all components is beyond the scope of this paper. For that, we refer the reader to the official O-RAN documentation [1].

2.2. The Near-Real Time RIC

The Near Real-Time RIC (Near-RT RIC) [29] assumes responsibility for critical radio functions, including connection management, mobility management, and the execution of trained AI/ML models. Implemented above the cloud computing platform known as the O-Cloud [28], the Near-RT RIC operates with control loops ranging from 10 milliseconds to less than one second. Third parties can deploy applications, known as eXtended Applications (xApps), which operate on the Near-RT RIC platform. xApps communicate with RAN functions using the E2 interface [30] based on various SMs. For instance, they enable efficient control of RAN nodes [31], Cell Configuration Control on a cell-level basis [32], and Key Performance Measurement for enabling measurements/statistics of RAN nodes [33]. Throughout the development of an xApp, two primary parties are involved: the solution provider, who creates the applications, and the service provider, who integrates and utilizes these applications within their network.

The end-to-end lifecycle [34] of an xApp can be categorized into three phases: App Development, App Onboarding, and App Operations. *App Development* consists of the creation of the xApp. This phase involves operations such as gathering customer feedback on use-case requirements, incorporating feature requests, and addressing defect notifications. These inputs are integrated into the app development cycle for implementation, resulting in the creation of container images using SDKs tailored for the intended deployment environment. Subsequently, the built containers are conveyed to the service provider (RIC owner) through the onboarding process. *App Onboarding* operations include establishing proper configuration, policies, measurements, and required analytics. *App Operations* determines when the xApp needs to be deployed or decommissioned. These operations could include decisions regarding production or non-production execution of the xApp. Additionally, this phase involves continuous monitoring of the deployed xApp, including alert management, event handling, and incident management. Analysis of the monitoring data informs further actions such as termination, healing, and scaling to ensure optimal performance and reliability.

In the context of a hypothetical xApp, consider the scenario of localization data for positioning of User Equipment (UE). An xApp can be a machine-learning algorithm that receives as input relevant data from E2 nodes and provides as output UE location information. During training, the xApp must collect required UE-related data, such as signal strength from E2 nodes, to properly train the prediction model. The solution provider receives a set of requirements from the service provider, along with sample data for example, to facilitate the xApp development process. Subsequently, the developed application undergoes onboarding and validation by the service provider, including real-life testing for UE localization, with feedback and optimization requests provided to the solution provider. Upon successful testing and validation, the service provider deploys the xApp for real-world usage by customers, who utilize the localization data.

2.3. RIC Implementations

The O-RAN specification defines the components and protocols defining the Near-RT RIC via openly accessible documents, allowing practitioners to develop their own solutions. An increasing number of publicly available implementations are currently in various stages of development. Notably, their focus and intended use often differ considerably. There is a significant difference in the platform approaches and direct component implementations, such as for the Near-RT RIC. Some have developed their own components, while others utilize already established open-source components and integrate them. To our current knowledge, there are eight main open-source O-RAN platform implementation and development efforts. These can be seen in Table 1 together with the respective compatible Near-RT RIC implementations.

Within the scope of our research, we examine two independent standard-compliant open-source implementations of the Near-RT RIC: FlexRIC and the O-RAN OSC RIC. These have become well-established within the field of O-RAN research and are supported by a substantial user base. These are used directly or indirectly in 7 of the 8 platform approaches. This allows us to demonstrate a certain general impact among existing open-source approaches with our research.

The FlexRIC [43] has been developed under the Mosaic5G project [38], and is nowadays a widely tested platform that proved significant improvements to the 5G RAN. Indeed, under the wider umbrella of the OpenAir Interface [37], FlexRIC has provided extended capabilities

TABLE 1: O-RAN platform implementation and development efforts with the respective Near-RT RIC implementations.

● ≡ Directly covered with our Approach

● ≡ Not covered with our Approach

O-RAN Platforms	Near-RT RIC		
	FlexRIC	O-RAN OSC	SD-RAN
● OAIC [35]		✓	
● SD-RAN [36]			✓
● Open Air Interface [37]	✓		
● Mosaic5G [38]	✓		
● O-RAN OSC [39]		✓	
● srsRAN [40]	✓	✓	
● X5G [41]		✓	
● POWDER [42]		✓	✓

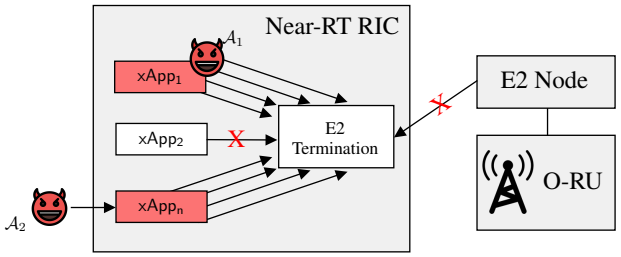


Figure 2: Depiction of the threat model. We consider both insider attacker $\mathcal{A}_1$ and outsider attacker $\mathcal{A}_2$. The malicious/compromised xApp sends a large number of subscription requests to the E2 termination, preventing legitimate services (xApps or E2 node) from properly communicating (denoted via the red X in the figure).

to manage the heterogeneous 5G use cases and being compliant with the O-RAN specification [44]. FlexRIC comprises O-RAN compliant E2 Node Agent emulators, a Near-RT RIC, and xApps developed in C/C++ and Python. It is compatible with major open-source RAN projects like OpenAirInterface [37] and srsRAN [40]. During the development phase of FlexRIC, their development team encountered limitations within the O-RAN E2 interface specification of that time. Consequently, the decision was made to introduce new types of messages to the E2 interface and rename it as E42 in the context of the FlexRIC project. OSC is instead the official software from the O-RAN community, the reference point for practitioners and developers interested in this technology [45]. The software developed by the O-RAN alliance has been proven effective in real-life demonstrators of O-RAN's performance [4], [5]. Hence, the security of these two RIC implementations has a significant impact on the whole community of O-RAN practitioners and developers.

In general, however, the implementation of FlexRIC has a number of advantages, especially for emerging 5G and 6G use cases and for O-RAN developers. The SDK has already been evaluated on two specialized state-of-the-art 5G use cases and showed 50% reduced round-trip time, about 10 times less CPU consumption, and only one-third of the memory consumption compared to the OSC reference implementation [26]. Furthermore, the FlexRIC Near-RT RIC was developed as a SDK with a modular architecture that enables the construction of specialized service-oriented controllers. Throughout this paper, we therefore use these two standard compliant Near-RT RIC implementations with a strong focus on the FlexRIC implementation.

3. Threat model

In this section, we discuss the attacker's aim and capabilities. We introduce our proposed threat model in Section 3.1. In Section 3.2, we provide an overview of the closest related threats discussed in O-RAN specifications.

3.1. Our Proposed Threat Model

We assume the victim is an O-RAN compliant service provider. In particular, the attacker targets the Near-RT RIC and its capability to establish connections between xApps and E2 nodes. The attack aims to disrupt the victim's O-RAN architecture service. This includes slowing down the services provided to all/some clients or completely blocking the services provided. A successful attack is assumed when the Near-RT RIC crashes, thereby making it impossible for new users to access the RAN.

In Fig. 2, we plot our considered threat model. The attacker's objective is to send an elevated number of subscription requests to the E2 termination such that the Near-RT RIC can not keep up with a large number of requests and thus fails in providing services to legitimate components.

To achieve this objective, we assume that the attacker can control one or more xApps deployed in the target Near-RT RIC. This can be achieved by two different types of attackers, i.e., the insider attacker $\mathcal{A}_1$ and the outsider attacker $\mathcal{A}_2$. In the first, the attacker $\mathcal{A}_1$ is an authorized user of the Near-RT RIC. This includes two possible actors: (i) a third-party company that develops xApps to be deployed by the network operator; and (ii) a network operator sharing the RAN hardware infrastructure with other operators. Notice that the second actor is feasible only thanks to the RAN sharing capabilities enabled by O-RAN [46]. Currently, no security measure exists to check whether xApps have malicious intentions. Therefore, the Near-RT RIC owner has no means to assess whether the xApp in the onboarding process will perform malicious activities. Despite considering the operator (RIC owner) as competent and trusted, onboarding a malicious xApp still represents a severe threat to the O-RAN architecture. The motivations for $\mathcal{A}_1$ include disrupting the network service for operators and preventing them from leveraging the advantages offered by xApps. For instance, considering the shared RAN scenario, a competitor operator could onboard a malicious xApp to degrade the service quality of other operators and induce users into changing their service provider.

In the second scenario, the attacker $\mathcal{A}_2$ is an external user that has no legitimate access to the Near-RT RIC, but manages to compromise an already deployed xApp. Noticing that xApps are containerized microservices (e.g., Docker instances in a Kubernetes cluster in OSC), the attacker can leverage known vulnerabilities to achieve privilege escalation [47]. Possible motivations for $\mathcal{A}_2$ include disrupting the access network on behalf of competitor

operators, military operations, terrorism, or hacktivism.

We assume that the attacker has no privileged position compared to other regular users. This implies that once the attack is successful, the service will no longer be available even to the attacker. Therefore, we assume that, for the attacker, the benefit of blocking the service for all users is the main goal.

3.2. O-RAN Reference Threat Model

We base our threat model primarily on the O-RAN Security WorkGroup specifications [20], [48]–[50]. The O-RAN system faces internal and external threat entry points, encompassing risks from within and outside its infrastructure. The O-RAN alliance identifies both threat surfaces and the associated security threats.

Considering our scenario, relevant threat surfaces include but are not limited to, the decoupling of components and their retrieval from a Trust Chain, containerization and virtualization that further separate software and hardware, and the reliance on open-source code which elevates exposure to public exploits. These surfaces do not find effective security measures in the current state-of-the-art in the O-RAN domain.

Based on the identified threat surfaces, entry points, and vulnerabilities, the O-RAN alliance identifies specific threats that pose risks to its functionality. Regarding our considered scenario, the O-RAN Security WorkGroup 11 provides an extensive categorization of such threats [20], [48]–[50]. They fall into three main categories: threats concerning O-RAN system components, threats related to lifecycle management (LCM), and threats targeting the O-Cloud infrastructure. O-RAN system component threats encompass risks directed at the system itself, including general threats (T-ORAN-01, T-ORAN-02, T-ORAN-03, T-ORAN-06, T-ORAN-09), threats against the Near-RT-RIC (T-NEAR-RT-01), and threats against the xApps (T-xApp-01, T-xApp-02, T-xApp-03). Life Cycle Management (LCM) threats pertain to the threats related to the xApp life cycle management as explained in Section 2.2, such as T-App-LCM-01, T-App-LCM-02, T-App-LCM-03, T-App-LCM-04. Additionally, they identified threats such as T-GEN-01 targeting the cloud infrastructures hosting the O-RAN system components. Interested readers are encouraged to consult [20], [48]–[50] for further details on these threats.

4. Methodology

In our study, we thoroughly analyze the attack vector of subscription DoS and assess its impact on the components in three different deployment configurations of a Near-RT RIC. In the following sections, we evaluate the resilience of the Near-RT RIC in different scenarios.

Given that the Subscription Manager (SubM) is a fundamental platform service within the Near-RT RIC, it assumes responsibility for orchestrating E2 subscriptions from xApp to the E2 node (e.g., eNodeB or gNodeB). Fig. 3 illustrates the interrelationships between the relevant components. A HyperText Transfer Protocol (HTTP)-based Representational State Transfer (REST) interface facilitates the connection between the xAPP and the SubM, as well as between the Routing Manager and the SubM.

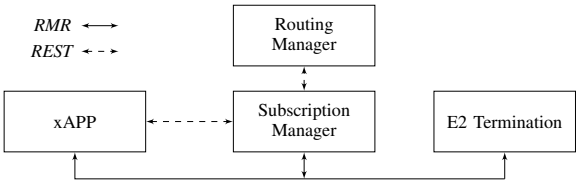


Figure 3: Interface overview of the SubM and relevant neighboring nodes.

Communication between the E2 Termination and the SubM relies on RIC Message Router (RMR) messages. An xApp subscription request message may encompass multiple E2 subscriptions, each requiring the xApp to generate a distinct instance ID. A notable limitation lies in the restriction to only one concurrent E2 subscription or subscription deletion procedure towards the E2 node per time unit across the E2 interface. This constraint arises from the SubM's ability to consolidate new E2 subscriptions contingent upon having already received a successful response from the E2 node. Consequently, E2 subscriptions and subscription deletions may experience queuing temporarily, turning the SubM into a possible bottleneck, particularly under a high volume of requests [51].

4.1. Attack Procedure

In all scenarios under consideration, we conduct a DoS attack leveraging a manipulated xApp deployed within the system. This involves the adaption of a prototype xApp to continuously send Subscription Requests (Sub_{Reqs}) to the Near-RT RIC. Fig. 4 shows a schematic representation of the expected request interactions within the FlexRIC implementation. The sequence diagram for the attack within the OSC Near-RT RIC implementation exhibits an identical structure, with the only distinction being the replacement of the E42 abstraction layer introduced by FlexRIC with conventional E2 requests.

The underlying premise behind the attack vector is the flooding of the Near-RT RIC with a substantial volume of requests originating from the xApp. This influx may eventually surpass the system's processing capacity, leading to potential consequences such as reduced service performance, characterized by increased latency, or, in severe cases, system failure attributed to internal software anomalies.

Should such a scenario arise, all components situated southbound of the architectural hierarchy would become disconnected from the system. In most scenarios, this disconnection would affect the entire data plane. The formal description of the attack procedure can be as follows. A malicious xApp is characterized as

$$xApp_n = \mathrm{LoopSub_{Req}}(), \tag{1}$$

which transmits a certain total number of Subscription Request (Sub_{Req}) to the connected Near-RT RIC per time unit T_u, defined as

$$\delta = \sum_{T_u} Sub_{Req}. \tag{2}$$

A maximum number ξ of processable Sub_{Reqs} per T_u is allocated to the Near-RT RIC before it is either overloaded

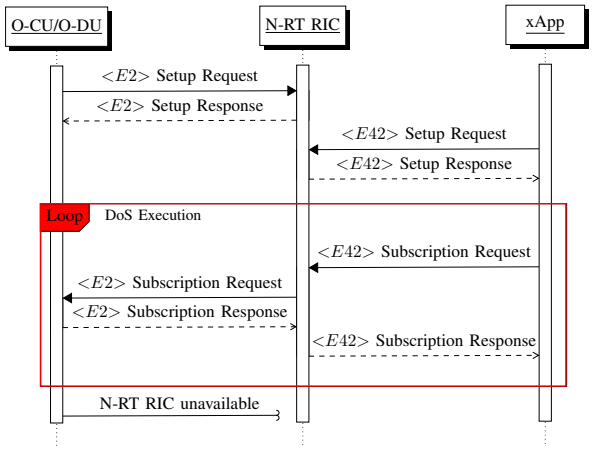


Figure 4: Sequence diagram of the Subscription DoS attack on the FlexRIC Near-RT RIC.

and can no longer keep up with the processing of the individual requests or before an internal software problem arises. Hence, if

$$\delta > \xi, \tag{3}$$

regular operation of the Near-RT RIC is no longer possible.

This issue presents a significant problem for end users: new users are unable to establish connections, although existing connections are likely to remain active. The situation becomes increasingly problematic if, as described in Section 6.1, all components (Central Unit (CU), Distributed Unit (DU), and Near-RT RIC) must be restarted to restore normal operation following an attack. Consequently, existing connections would be terminated, resulting in a complete service failure for the end user.

4.2. Baseline Scenarios

To conduct the attack successfully, a compliant Next Generation Node Base (gNB) setup comprising O-RAN Central Unit (O-CU) and O-RAN Distributed Unit (O-DU) is essential. For simplicity, these components are referred to collectively as E2 nodes. Additionally, seamless connectivity between the E2 nodes and the Near-RT RIC must be established via an E2 setup request. Subsequently, registration of the malicious xApp (or multiple instances, if required to amplify the payload) is necessary, accomplished through a setup request to the Near-RT RIC. These preparations form the fundamental prerequisites. Throughout our integration testing, we encountered significant challenges in assembling a functional test environment for the individual software components. This difficulty primarily stems from the necessity for numerous independently developed software components to operate cohesively. Consequently, the ongoing development of individual components often results in rapidly emerging dependencies, causing operational complications. To facilitate the reproducibility of our experiments, we meticulously documented the specific git commit versions utilized, as presented in Table 2. Furthermore, we have developed a Docker environment [27] specific to SC_1 and SC_2, that enables a quick replication of our attacks with minimal effort by using our DoS setup script.

In the following, we provide a concise overview of four fundamental scenarios implemented in Sections 4.2.1 through 4.2.4, outlining the interaction among components and highlighting the intended outcomes following a successful attack.

4.2.1. SC_1 - FlexRIC Standalone. This scenario leverages the most recent version of the FlexRIC [43] project, involving the deployment of the Near-RT RIC alongside two E2 node agent emulators, emulating the functionalities of the O-CU and O-DU. Subsequently, the designated xApps can be executed via command-line calls. It is expected that upon activation of the malicious xApp, there will be a notable escalation in CPU utilization by the Near-RT RIC, culminating in system instability leading to eventual failure.

4.2.2. SC_2 - FlexRIC, srsRAN, Open5GS. To add a higher degree of realism to our experiments, beyond the limits of emulation as observed in SC_1, we introduce a more complex scenario. Herein, we employ the srsRAN[2] gNB in conjunction with srsUE, which gives us the flexibility to opt for either physical evolved Node Bases (eNBs) and UEs utilizing Software-defined Radios (SDRs) for over-the-air transmissions, or virtual radios facilitated through ZeroMQ (ZMQ) for the transmission of radio patterns between applications. Anticipated outcomes in this scenario mirror those observed in SC_1, characterized by not being able to establish connections with new UEs and disruptions to the connectivity of existing gNBs and UEs with the core network.

4.2.3. SC_3 - OSC O-RAN, ORAN-SIM H-Release. Given that the first two scenarios leverage the adaptable SDK FlexRIC, and we aim to also evaluate our methodology utilizing the Near-RT RIC provided by the OSC platform, we opt for the H-Release, accompanied by its integrated E2 Simulator, constructed on top of the E2 Application Protocol (E2AP) version 1. We hypothesize a scenario outcome where the subscription management pod within the Kubernetes cluster, responsible for overseeing subscriptions, encounters a failure event, subsequently initiating an automated regeneration and restart process.

4.2.4. SC_4 - OSC O-RAN, ORAN-SIM I-Release. For SC_4, in contrast to SC_3, the changes are limited to the software components and are incorporated into the latest software version, which corresponds to the I-Release enabled by OSC. Comprehensive information regarding the individual commit versions is accessible in Table 2.

5. Implementation

In the following sections, we discuss the implementation details explaining the methodology behind executing the attacks. Our introduction of distinct scenarios featuring diverse software components enables comprehensive coverage of prevalent O-RAN software. Nevertheless, minor discrepancies arise in the execution of the attack due to the varied software configurations. Assessing structural security vulnerabilities across all implementations is a

2. https://www.srsran.com

TABLE 2: Listing of O-RAN software component implementations evaluated in our study with associated Git commit.

Software	Git Commit
	$\mathbf{SC_1}$
FlexRIC	1f04cc558ebc8da9de6a620762bc02f5db4ecb4a
	$\mathbf{SC_2}$
srsRAN_Project	2f90c8b60e9396a7aed59645c98dbcbccda2bf7c
srsRAN_4G	ec29b0c1ff79cebcbe66caa6d6b90778261c42b8
Open5GS	be7d08bffc4919475e5c87355eda22e051ccc5b2
FlexRIC	ddb0a6add3c866d5ba50f8c60336b3d403c92b58
	$\mathbf{SC_3}$
ric-dep	b0753dc1d115fd4e996e07846610f6e469b562b3
E2-interface	215d350dbe06a4c7ee370815f26128a7f7e160cb
appmgr	139295eb66cb5cedbae5e3746d9ace128104bb69
hw-go	3a0d348e429ea0f3f3d2a1d5eb54ec8758d1a262
	$\mathbf{SC_4}$
ric-dep	0a6f18efda7b20b9d3d9d6bf80a9e0feedddd8e7
E2-interface	95005a3d8b62f04e46cca615bdbffa2842827bdd
appmgr	361faacb46d1f379406b205034c818fef37a9d76
hw-go	3a0d348e429ea0f3f3d2a1d5eb54ec8758d1a262

challenging task, as these systems vary significantly and are often still in the early stages of development. However, the underlying methodology remains consistent with the aforementioned description. Below, we draw a distinction between FlexRIC and the Near-RT RIC realized by OSC.

Table 2 lists the commit of each component we leveraged for our analysis. All commits refer to the most recent versions of their respective releases available at the time of publication. Specifically, for OSC software, the H and I releases are utilized. For FlexRIC, version 2.0.0 released in December 2023 is employed, while a newer version from February 2024 is used in the experiments involving the srsRAN components.

5.1. FlexRIC

The FlexRIC [43] deployment adheres fully to the O-RAN standards. It incorporates various service models that support the standard Key Performance Measurement (KPM) versions 2.01, 2.03, 3.00, and RC version 1.03. It integrates different encoding schemes based on the utilized service model (ASN.1, flatbuffer, and plain). For xApps, a *sqlite3* database connection is enabled to persistently store received indication data for subsequent applications. This versatility renders FlexRIC highly adaptable and compatible with nearly all O-RAN use cases.

To realize our methodology, we have made slight modifications to the KPM monitoring xApp example sourced from FlexRIC. Specifically, we have encapsulated an infinite loop around the query process for KPM metrics. This loop continuously dispatches queries alongside preceding subscription requests to the Near-RT RIC until both the Near-RT RIC and the xApp experience a crash. The respective code excerpt and the associated alteration are detailed in Algorithm 1.

Algorithm 1 Original FlexRIC xApp source code, wrapped in an endless loop to achieve a DoS.

```
1: while True do
2:     kpm_sub_data_t kpm_sub = {0};
3:     uint64_t period_ms = 100;
4:     kpm_sub.ev_trg_def = gen_ev_trig(period_ms);
5:     const int KPM_ran_function = 2;
6:     for i = 0; i < nodes.len; ++i do
7:         kpm_sub.sz_ad = 1;
8:         kpm_sub.ad = CALLOC(1, sizeof(kpm_act_def_t));
9:         assert(kpm_sub.ad ≠ NULL and "MEM Exh.");
10:        ngran_node_t const t = nodes.n[i].id.type;
11:        bool du_or_gnb = t == ngran_gNB
             or t == ngran_gNB_DU;
12:        const char* act = du_or_gnb?
             "DRB.RlcSduDelayDl" :
    "DRB.PdcpSduVolumeDL";
13:        *kpm_sub.ad = GEN_ACT_DEF(act);
14:        h[i] = REPORT_SM_XAPP_API(
             &nodes.n[i].id, KPM_ran_function,
             &kpm_sub, sm_cb_kpm);
15:        assert(h[i].success == true);
16:        FREE_KPM_SUB_DATA(&kpm_sub);
17:    end for
18: end while
```

Executing FlexRIC necessitates the initial steps of building and installing the FlexRIC project, including the modified xApp into the build pipeline. Subsequently, the Near-RT RIC can be initiated, alongside the E2 node agent emulators for CU and DU with default configuration settings. Given that the operational dynamics of FlexRIC xApps diverge slightly from those in an OSC O-RAN, the process only involves launching the compiled C file. Subsequent observation enables the assessment of the effects of the DoS attack.

5.2. OSC O-RAN

Deploying an xApp within the Near-RT RIC framework of the OSC implementation entails a more intricate and error-prone sequence of steps from the developer's perspective compared to the FlexRIC deployment. Primarily, this process necessitates the flawless operation of the complete Kubernetes cluster, alongside the containerized E2 interface simulator responsible for emulating the gNB. Subsequently, it is imperative to establish successful connectivity between the gNB and the Near-RT RIC, followed by the creation of an entry in the routing table facilitated by the routing manager. Only thereafter can docker containers, be on-boarded to the chartmuseum and subsequently installed within the cluster utilizing the *dms_cli* tool. For implementation purposes, we have employed the hw-go sample application, within which an infinite loop has been integrated within the xAppStart callback. Algorithm 2 illustrates this straightforward adaptation within the authentic code segment.

Algorithm 2 OSC xApp source code for the endless sending of subscription requests.

```
1: nbList ← e.GETNBLIST;
2: while True do
3:     for each nb in nbList do
4:         e.SENDSUBSCRIPTION(nb.InventoryName);
5:     end for
6: end while
```

6. Evaluation

In this section, we present our findings derived from the execution of distinct attacks across the respective scenarios, explaining the effects of each. The experiments and measurements were conducted within Virtual Machines (VMs) hosted on a Proxmox environment. Each VM is configured identically, equipped with 6 CPU cores of Intel(R) Core(TM) i5-9500 CPU @ 3.00GHz and 64 GB of RAM, operating under Ubuntu 20.04-2.0.

Experiments SC_1 and SC_2 underwent comprehensive measurement and evaluation encompassing a diverse array of metrics. Detailed documentation on the replication of the attacks is accessible via our dedicated GitHub repository [27]. Furthermore, we explain the identified consequences on the OSC implementations for the attack scenarios SC_3 and SC_4.

In our evaluation, we concentrate our experiments on analyzing CPU load during the respective attacks. Although we recorded memory consumption for all measurements, it did not exhibit any significant characteristics. This can be explained by the fact that flooding the endpoint with packets necessitates processing each incoming message, which is highly CPU-intensive. While certain DoS attacks can significantly impact memory consumption, our preliminary evaluations indicate that this is not the case for our attacks. Therefore, we have chosen to focus exclusively on a detailed analysis of the effects on CPU consumption in this section.

6.1. SC_1 - FlexRIC Standalone

To establish a comparative reference for our attacks, we initiate a ground truth measurement as a baseline. As depicted in Fig. 5, this baseline illustrates the simultaneous start of all required components, maintaining a consistent level of CPU utilization throughout the 120-second runtime. Subsequently, we conduct two distinct Attack Case (AC) test runs:

- $AC_\alpha SC_1 \rightarrow$ Single execution of the DoS attack with no subsequent component interaction.
- $AC_\beta SC_1 \rightarrow$ Single execution of the DoS attack with two subsequent restarts of the Near-RT RIC.

In the setting of $AC_\alpha SC_1$, our implementation exhibits the expected effect described above. Following a brief startup phase, the Near-RT RIC significantly increases its CPU power consumption under the constant load of subscription requests. Shortly after executing the xApp, the Near-RT RIC collapses. Fig. 6 clearly illustrates this rise and the abrupt stop shortly before the 40-second mark. This indicates that from approximately second 40, there is no longer a connection between the CU, as well as the DU, and the control plane. Hence, it is reasonable to assume that the condition outlined in (3) manifested.

The total peak CPU load during an attack increases, typically leading to the collapse of the Near-RT RIC within a few seconds of the attack. In general, the duration of a successful attack falls within the range of 10 seconds or more. Additionally, there is roughly double the CPU performance of CU and DU compared to the baseline measurement, attributed to their efforts to send metrics to a large number of subscribed xApps. In the case of $AC_\beta SC_1$, the subsequent restart of the Near-RT RIC exhibits interesting behavior characterized by a sudden increase in CPU payload. These spikes amount to a peak of 133% (a number greater than 100% is possible since the virtual execution environment has 6 CPU cores and thus theoretically 6 × 100% is achievable). The cause is that the Near-RT RIC immediately attempts to resolve all RIC indication messages previously accumulated by the DoS attack after restarting. This behavior can also be seen after a second restart, as in Fig. 7. Our tests show that the problem persists even after multiple attempts. Only a complete restart of all components, including CU and DU, allows normal operation to be restored after a subscription request DoS attack.

6.2. SC_2 - FlexRIC, srsRAN, Open5GS

This scenario is analogous to SC_1, affirming the assumption described in Section 4.2. Only the emulated CU and DU are replaced by srsRAN components. In essence, the baseline ground truth measurement in Fig. 8 depicts an increased CPU consumption of the srsRAN gNB and UE. This augmentation arises because, unlike in SC_1, a purely emulated data plane is not employed here; instead, a 5G-compliant network is utilized, comprising the components outlined in Section 4.2.2. Furthermore, we also split in this scenario the remaining experiments into two distinct groups:

- $AC_\alpha SC_2 \rightarrow$ Single execution of the DoS attack with no subsequent component interaction.
- $AC_\beta SC_2 \rightarrow$ Single execution of the DoS attack with two subsequent restarts of the Near-RT RIC.

The same equivalent effects of the Sub_{Req} DoS attack, as previously noted in SC_1, are observable in Fig. 9 and 10. This demonstrates that not only does the emulated scenario entail far-reaching effects in the event of such an attack, but also components such as the srsRAN 4G and 5G software radio suites are impacted by the attack's effects if FlexRIC is employed as the Near-RT RIC within the RANs. This further underscores the threat scenario for real, as opposed to merely simulated, RANs.

6.3. SC_3 - OSC O-RAN, ORAN-SIM H-Release

In the current implementation of SC_3, the underlying technology has transitioned towards the adoption of the OSC O-RAN. This scenario investigates the impact of Sub_{Req} DoS attacks on the pods operating within the Kubernetes cluster and assesses their effects on the simulated gNB. A prerequisite for this assessment is the successful installation and error-free operation of all pods, alongside the proper functioning of the gNB simulation facilitated by the E2 simulator, which must be running and connected to the Near-RT RIC. Upon successful onboarding of the malicious xApp into the operational cluster through the *dms_cli*, as outlined in Section 5.2, our attack protocol is initiated. The attack simulation, denoted as $AC_\alpha SC_3$, is configured to trigger an immediate restart in the event of component failure, which is attempted up to three times before further steps are discontinued. Fig. 11 depicts the 120-second measurements of Kubernetes pod states for both the E2 manager (ricplt-e2mgr) and the malicious

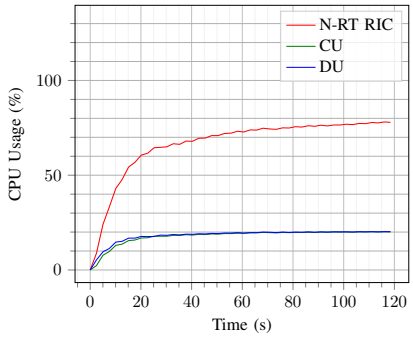


Figure 5: Baseline CPU Usage for SC_1.

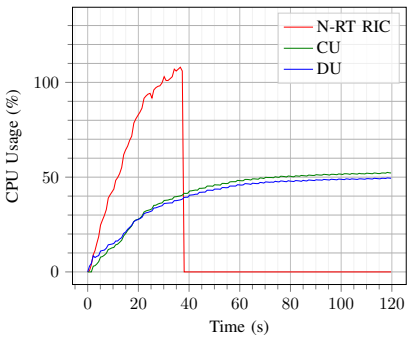


Figure 6: DoS for $AC_{\alpha}SC_1$.

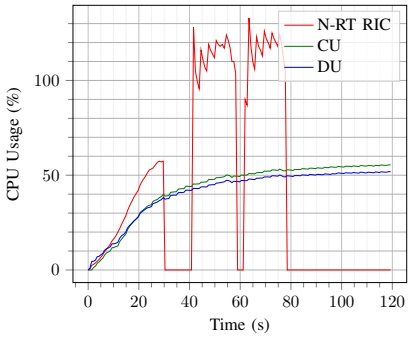


Figure 7: DoS with restarts for $AC_{\beta}SC_1$.

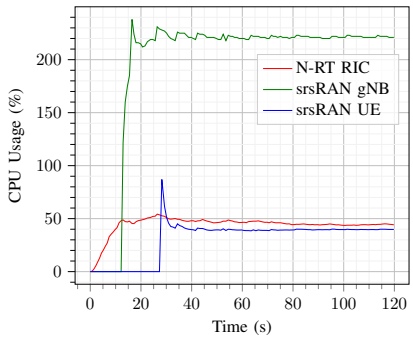


Figure 8: Baseline CPU Usage for SC_2.

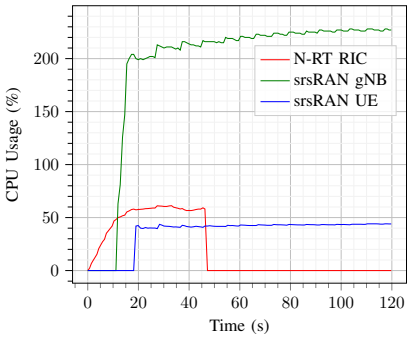


Figure 9: DoS for $AC_{\alpha}SC_2$.

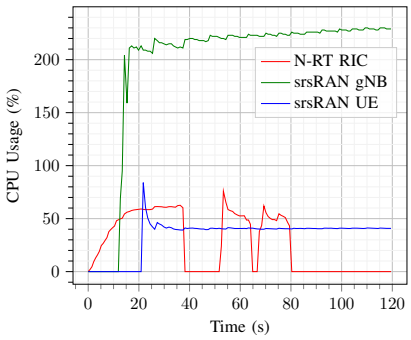


Figure 10: DoS with restarts for $AC_{\beta}SC_2$.

xApp (ricxapp-hw-dos), along with the gNB connection status.

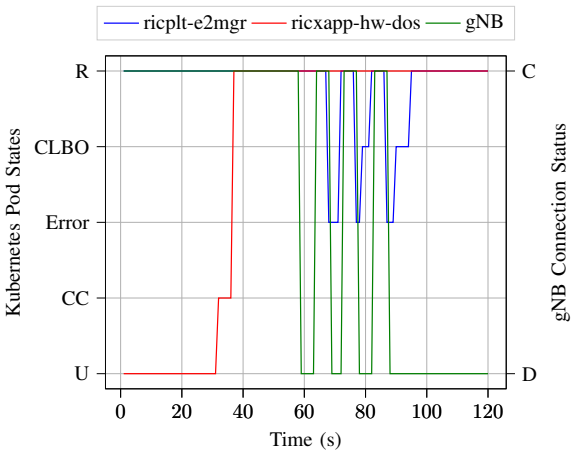
The analysis of these measurements yields the following observations:

1) Following the initiation of the malicious xApp, it takes approximately 22 seconds for the gNB to crash.

2) Approximately 4 seconds subsequent to the gNB restart, the E2 Manager crashes, encountering a runtime error attributed to segmentation violation, consequently transitioning into an error state.

3) Following this, a similar pattern to (2) emerges during the subsequent two restarts of the gNB, with a slight variation: the pod transitions to the CrashLoopBackOff state before resuming normal operation. This is a result of the internal handling of pod restarts within Kubernetes, which imposes a back-off delay with each restart. This cumulative delay increases with iterative failures, leading to an increased delay in subsequent restart attempts.

In our experimental trials, we find that exclusively executing a full redeployment of the entire set of Near-RT RICs without deploying the malicious xApp proves effective in resolving this behavior. This reaffirms the conclusions drawn in SC_1 and SC_2, indicating that following a Sub_{Req} DoS incident, only a comprehensive reset of the RIC, accompanied by complete downtime for all users and services, can extinguish the impact of the attack.



Abbreviations: C ≡ Connected, D ≡ Disconnected, U ≡ Undeployed, CC ≡ ContainerCreating, CLBO ≡ CrashLoopBackOff, R ≡ Running

Figure 11: Kubernetes pod states with gNB connection status for SC_3 during Sub_{Req} DoS attack with gNB restarts.

6.4. SC_4 - OSC O-RAN, ORAN-SIM I-Release

Scenario SC_4 mirrors the structure of SC_3, with the only modification being the utilization of the latest version of the I release. In this iteration, we notice a disparity in the impact of the OSC O-RAN implementation compared to our previous observation in SC_3, where the Sub_{Req} DoS attack was conducted.

In our experimental framework, executing an attack now results in the effective prevention of new E2 gNB

simulations from establishing connections. Should our modified xApp start before the actual gNB connection, leading to a significant influx of Sub_{Reqs} directed towards the Near-RT RIC, we can handicap the initiation of new gNB connections. To gain a better understanding of these effects, we conduct multiple simulations with varying degrees of malicious loads. To modulate the rate of Sub_{Reqs} transmission, we introduce a simple delay before each iterative processing of the requests. We categorize these simulations into five distinct groups, each characterized by a different average rate of Sub_{Reqs} transmission per unit of time:

- $SM_4M_\alpha \rightarrow$ This measurement serves as a ground truth reference, wherein the RIC is utilized, and no xApp exhibits malicious behavior.
- $SM_4M_\beta \rightarrow$ This measurement is conducted without introducing any delay in the transmission of Sub_{Reqs}.
- $SM_4M_\gamma \rightarrow$ In this measurement, a delay of 2 ms is incorporated into the request loop.
- $SM_4M_\delta \rightarrow$ This measurement involves the inclusion of a 5 ms delay in the request loop.
- $SM_4M_\epsilon \rightarrow$ Here, a delay of 10 ms is introduced into the request loop.

During the execution of the individual test categories, we record timestamps in a log file corresponding to the moment when each Sub_{Req} was dispatched. Additionally, for measurements SM_4M_γ to SM_4M_ϵ, we introduce a delay in milliseconds prior to sending each request. This intentional delay manipulation enables a certain degree of control over the total number of Sub_{Reqs} per unit of time.

The outcomes of this test series are illustrated in Fig. 12. The figure presents the average total count of Sub_{Reqs} per 10-second interval against the probability percentage of a successful gNB connection. Evidently, as the volume of Sub_{Reqs} increases, the likelihood of a successful setup response diminishes. This underscores the potential of a determined attacker to prevent new gNB connections with a considerable degree of success, provided that sufficient effort is made in terms of Sub_{Req} volume.

Such a scenario poses significant challenges, especially in instances where a gNB experiences a failure and undergoes a restart. Routine reboots, such as those occurring after updates or similar events, are also conceivable. If an attack is ongoing during such critical moments, the gNB faces obstacles in resuming normal operation within the RIC post-restart.

6.5. Sources of Vulnerabilities

Due to the implementation differences, we also identified independent potential causes of errors. Within the FlexRIC SDKs, an assertion error triggers application crashes, with more detailed information available in Section 7.1, where we analyze the underlying problem.

Regarding the issues within SC_3, we suspect an algorithm complexity problem, expressed in the terminology of the Common Weakness Enumeration (CWE) Root Cause Mapping of Vulnerabilities (RCMV). Initially, the gNB aborts with a core dump, subsequently leading to a runtime panic error in the *e2mgr*, with indications of an invalid memory address or a nil pointer dereference

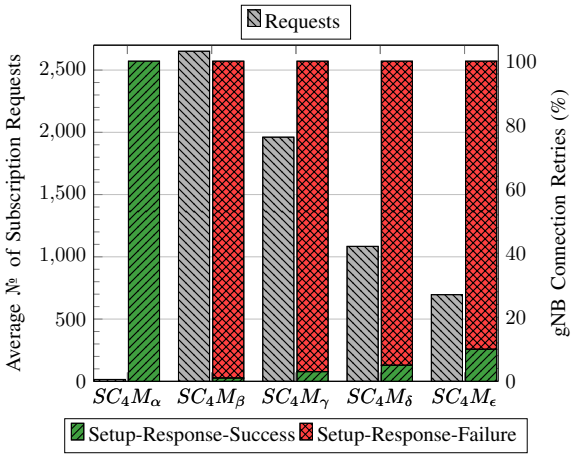


Figure 12: Average number of Sub_{Reqs} per 10-second time window in comparison to the percentage of successfully started gNBs in SC_4.

upon restart. Consequently, the *e2mgr* crashes with the *SIGSEGV* segmentation violation, *code=0x1 addr=0x0 pc=0x91d016*.

In SC_4, we characterize the described issue within the CWE RCMV as resource exhaustion. The key aspect here lies in the overwhelming inflow of Sub_{Reqs}, leading to an overload of the Near-RT RIC and consequently rendering other legitimate components, such as new connecting gNBs, incapable of establishing new connections via legitimate requests. Consequently, this results in a *SETUP-RESPONSE-FAILURE* within the timing of the gNB.

6.6. Ethics and Responsible Disclosure

We took measures for responsible disclosure to prevent the discovered vulnerabilities from affecting real-life deployments and the diffusion of our work. We followed different approaches based on their original release model for the affected components.

In particular, we contacted the project's original developers for the FlexRIC. They acknowledged our findings and proposed a first solution. However, after testing, we found that the problem persisted, and we shared our proposed solution with the FlexRIC team.

Regarding the OSC implementation, we opened two issues on the Jira page of the OSC Near-RT RIC project[3]. The issues have been assigned and are currently under investigation.

We have been assigned three CVEs (CVE-2024-34034, CVE-2024-34035, CVE-2024-34036) for our findings within the FlexRIC and OSC implementations from the MITRE CVE Numbering Authority. All three were reviewed and accepted by the CVE Numbering Authority (CNA).
Next, we present the claims in each filed CVE:

- CVE-2024-34034:
  Sub_{Reqs} vulnerability in FlexRIC 2.0.0 - The component crashes during a Subscription Request DoS attack, triggered by an assertion error. An attacker

3. https://jira.o-ran-sc.org/projects/RIC/issues/

Algorithm 3 Basic pseudocode for handling subscription request DoS prevention in Near-RT RIC.

Require: $maxSimultaneousXappsSubs \leftarrow \alpha$
Ensure: $cSubs \leq maxSimultaneousXappsSubs$
if new.SUBREQ($xAppID$) **then**
  $cSubs \leftarrow get$.COUNTCURRENTSUBREQ($xAppID$);
  if $cSubs \leq maxSimultaneousXappsSubs$ **then**
    $handle$.SUBSCRIPTIONREQUEST($xAppID$);
    add.SUBREQARRAY($xAppID$)
  else
    break
  end if
else if del.SUBREQ($xAppID$) **then**
  $test$
end if

must send a high number of E42 Subscription Requests to the Near-RT RIC component.

- CVE-2024-34035:
  Sub_{Reqs} vulnerability in O-RAN OSC Near-RT RIC H-Release - To trigger the crashing of the e2mgr, an adversary must flood the system with a significant quantity of E2 Subscription Requests originating from an xApp.
- CVE-2024-34036:
  Sub_{Reqs} vulnerability in O-RAN OSC Near-RT RIC I-Release - To exploit this vulnerability, an attacker can disrupt the initial connection between a gNB and the Near-RT RIC by inundating the system with a high volume of E2 Subscription Requests via an xApp.

7. Countermeasures

In this section, we show how a simple countermeasure can mitigate our proposed attack without significantly impacting the performance under regular network usage. This is intended to demonstrate how such attacks can potentially be avoided by means of simple measures. In particular, we focus on a countermeasure that efficiently prevents the attacks for FlexRIC shown in SC_1 and SC_2.

Algorithm 3 shows a schematic pseudocode of how our DoS prevention works.

The main idea is to create a global dynamic array containing the number of active subscription requests of an xApp. Upon receiving a new subscription request, the algorithm checks the number of active subscriptions of the requesting xApp and allows for a new one only if the total number of active subscriptions is below a pre-defined threshold value α.

7.1. Problem Analysis

In the context of investigating the faulty Near-RT RIC functionality under DoS Sub_{Req} attacks, several critical observations can be made that lead to the clarification of potential problems within the operational framework of FlexRIC. First, it can be observed that the Near-RT RIC module is prone to lose connection with the associated E2 nodes as a result of the high volume of Sub_{Reqs}. This loss of connection, which is assumed to be possibly related to the timeout of the *SUSBCRIPTION_REQUEST_RESPONSE*, interrupts the seamless interaction between the Near-RT RIC and the E2 nodes, triggering a change of the RIC state to *PENDING_EVENT* within the *e2_event_loop*.

Subsequently, under such circumstances, the Near-RT RIC tries to handle the pending event by attempting to consume a file descriptor. This action entails invoking the *consume_fd* function within the codebase, designed to read from the file descriptor to facilitate its removal. However, it is observed that during this process, an assertion check for the file descriptor's validity, specifically $fd > 0$, fails to hold true. In the particular scenario under investigation, the file descriptor falls below the expected threshold, indicating an anomaly within the system. It is possible that the reduced value of the file descriptor could be due to various factors, including possible socket errors or other forms of undefined behavior.

Following discussions with FlexRIC developers, this assumption has been validated. The prevailing method in FlexRIC is to promptly confirm and subsequently fail upon a timeout occurrence. We believe that this approach is suboptimal, as timeouts can be induced by malicious activities, as demonstrated in our case. A first attempt[4] to solve this problem tried to overcome the assertion problem of $fd > 0$, which occurs after a certain time under DoS. However, this proved to be ineffective against our attack, from which we conclude that an all-encompassing DoS protection scheme based on our initial preventive measures is necessary. The fail-fast approach used in the FlexRIC project offers advantages in troubleshooting and simplifies implementation, but is suboptimal from a security standpoint. Such an approach is susceptible to exploitation by attackers who can induce repeated failures, potentially resulting in a denial of service (DoS) by rendering the system inaccessible to legitimate users, as observed in our case. Additionally, this approach can leave the system in an inconsistent or insecure state, where certain processes are terminated while others remain operational, thereby creating potential vulnerabilities.It is also concerning that the software is already being deployed in this manner in some customer projects.

Of particular concern is the absence of robust error-handling or recovery mechanisms within the codebase to address and mitigate such abnormal states. This deficiency underscores a critical gap in the Near-RT RICs fault tolerance and resilience strategies, necessitating further investigation and mitigation efforts to fortify its operational stability and reliability. The capacity of the RIC to process Sub_{Reqs} is inherently dependent on the hardware utilized, the allocated resources (e.g., threads), and the frequency of indication messages requested by xApps. However, scaling up resources should not be the sole solution to managing high request volumes. In order to mitigate resource-based attacks, approaches such as our proposed mechanism need to be integrated and further developed with existing security solutions.

7.2. Implemented Measure Details

We have created a premise within the build script that allows the subscription request DoS prevention

4. https://gitlab.eurecom.fr/mosaic5g/flexric/-/commit/2af360ca7727198f6bd624ef77746b027d42ec17

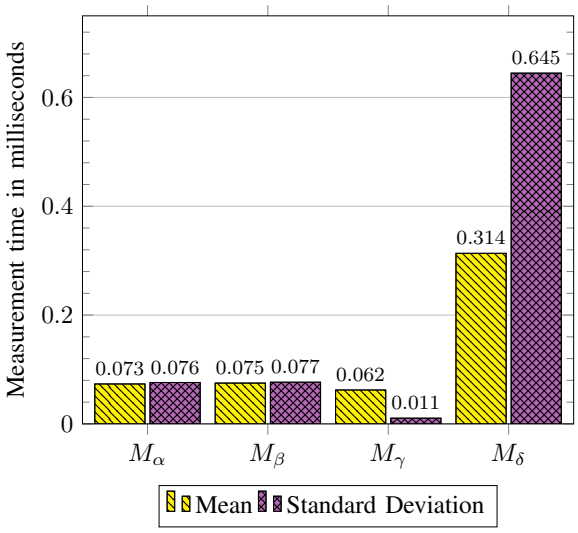


Figure 13: Overhead measurements for FlexRIC Sub_{Req} DoS prevention in milliseconds (ms).

to be activated or deactivated. If it is to be activated, *msg_handler_iapp_dosprev.h* is imported instead of *msg_handler_iapp.h*. Within this, a subscription tracker is created at the beginning in the initial message iapp handle. In the case of an *E42_RIC_SUBSCRIPTION_REQUEST* handle, the DoS prevention trigger is tested before the actual subscription request is executed. This checks whether the xApp from which the subscription request comes exceeds the number of permitted simultaneous xApp subscriptions with this request. If this is not the case, the subscription request is executed normally and the xApp ID is then added to the list within DoS prevention. In all other cases, the subscription request is ignored. A global parameter can be used to define how many simultaneous subscription requests an xApp can create and maintain. The list into which new Sub_{Reqs} are inserted functions as a buffer for a potential surge of requests. This mechanism can lead to a bottleneck during an attack, resulting in delays for legitimate requests. Nonetheless, this is an acceptable trade-off to mitigate attacks. Our initial load tests already show that FlexRIC can maintain stability and does not crash.

7.3. Prevention Overhead Evaluation

The reason for carrying out overhead measurements is the need to strike a balance between security and performance. While implementing robust security measures to protect against DoS attacks is essential, these measures often lead to additional computational and communication overhead. This overhead can affect the overall performance of the system and the user experience. Therefore, a thorough assessment of the overhead created by the countermeasures deployed is essential to make informed decisions about how to deploy them.

By quantifying the overhead caused by our DoS prevention mechanism, we aim to provide insights into the trade-offs between security and performance within the Near-RT RIC. Our empirical evaluation of the mean and standard deviation in Fig. 13 allows us to show that with a minimal average overhead of around 0.015 ms, the Sub_{Req} DoS can be effectively prevented. The results presented are divided into 4 measurement categories:

- $M_\alpha \rightarrow$ Shows the measurements with DoS prevention activated and related to the value for maximum simultaneous Sub_{Reqs} per xApp of 10.
- $M_\beta \rightarrow$ Indicates measurements with DoS prevention enabled and with reference to the value for maximum simultaneous Sub_{Reqs} per xApp of 100.
- $M_\gamma \rightarrow$ Measured values for implementation without DoS prevention in standard operation.
- $M_\delta \rightarrow$ Runtime values measured for execution without DoS prevention under load of the Sub_{Req} DoS attack shortly before collapse of the Near-RT RIC.

All measurements are taken for the same total runtime and under the number of twenty simultaneously deployed xApps. The measured time is the duration it takes the handle callback function, triggered for new incoming E42 Sub_{Reqs}, to process and confirm them. It is very unlikely that multiple malicious xApps can be deployed simultaneously. As a hypothetical, unrealistic upper limit, we have set twenty as the maximum number of parallel malicious xApps. The number of xApps would have to be enormously high to see a major impact. Since it is extremely unlikely that an attacker could secretly deploy so many xApps, this does not fall within our threat model.

The measurements of M_α and M_β indicate a relative homogeneity of the transit times, as the standard deviation is almost equal to the mean value and thus indicates a symmetrical distribution. This shows a relatively stable and consistent overhead of a maximum of 0.015 ms in the case of 100 possible simultaneous Sub_{Reqs}. Which we consider to be a minimal and therefore acceptable amount of computational time overhead for an active prevention of the described attack. Considering that it is improbable for a typically functioning xApp to transmit over 100 Sub_{Reqs}, our preventive measure does not yield any additional drawbacks.

In the case of M_δ, the widely differing mean value and standard deviation reflect the large significant scattering of the runtimes around the mean value during the attack. This can be explained by the fact that the measurements are taken shortly before the Near-RT RIC collapses and that the data set still contains fast response times and, subsequently, increasingly higher response times.

Changing the request pattern, e.g. by varying the request frequency, has no impact on our approach as a producer-consumer buffer absorbs such peaks. Apart from that, in our tests, we have the worst case (for the Near-RT RIC component in terms of incoming requests to be processed) with the highest possible load by sending all requests immediately.

7.4. Potential Additional Steps

The measures we have taken so far to prevent the attack are only a first step. Several additional security measures need to be implemented to address this attack strategy before the system is fully secured against this problem. Furthermore, they should be properly extended to cover the different possible DoS exploit points. Some of these measures include:

- xApp authentication. Authentication plays a fundamental role in understanding the source of messages. Indeed, with a proper authentication mechanism, it would be possible for the Near-RT RIC to properly queue the received requests.
- Timing Threshold. Given the authentication framework, the Near-RT RIC could impose a service-based timing threshold on xApps. In particular, based on the specific functionality of the xApp, the Near-RT RIC could impose a maximum rate of acceptable requests without impairing the xApp-provided service.
- Cloud-based DoS prevention. Similarly to commercially available solutions, the SMO could include a service dedicated to packet filtering, such that subscription requests are not directly handled by the Near-RT RIC if not properly validated by the external DoS prevention service.

Furthermore, more focus needs to be put on concrete vulnerability research. One possible approach to varying request patterns is protocol fuzzing, which is orthogonal to our proposal. In our approach, we generate requests with valid methods (system functions, interfaces, communication data - are not changed) that lead to a problematic system state. Hence, we do not modify the request payload, as this would not result in a denial of service. However, changing the payload to unforeseen content is important to check all possible system states. Therefore, it is necessary to pursue such approaches to prevent potential invalid system states triggered by non-valid input data.

7.5. Applicability to Other Implementations

In theory, this simple mitigation measure can also be applied in the OSC implementation. Currently, this approach assumes that there is no malicious entity in the system and therefore all xApps behave exactly as specified. Due to the implementation differences of the individual approaches, the integration of the countermeasures must be customized. For the OSC implementation, we tend towards a more far-reaching and all-encompassing overall solution. This could be implemented in the form of a generic security monitor. We envision a central security authority that can effectively enforce request rate limiting for the specific endpoints, active DoS detection, general anomaly detection, and regular checks of security guidelines.

As already described, the mitigation measure we have shown is merely a first demonstration of how easy it is to effectively prevent such attacks. It is not a final solution. Thanks to the simple use and the proven low overhead, significantly worse scenarios and effects can be effectively and quickly eliminated. If this measure is now combined with the measures described in Section 7.4 and possibly combined within a security monitor as just described, we believe that the general attack surface for the Near-RT RIC will be drastically reduced.

8. Related Work

The security of O-RAN is currently understudied. The majority of works focus on developing novel and efficient solutions for resource management leveraging the novel features of O-RAN. There currently exist survey papers exploring the possible threat surface and attack implications on O-RAN [18], [19], [52]. As their scope is to provide a comprehensive overview of the technology and its possible security implications, the authors did not perform evaluations to assess whether the proposed threat vectors are exploitable and if they lead to severe consequences for an O-RAN deployment.

8.1. O-RAN Security Research

Soltani et al. [21] leverage the similarities between Software Defined Networks (SDN) and O-RAN to showcase how an SDN security flaw (i.e., poisoning bearer context migration) may affect an O-RAN deployment and significantly affect its performance. Tseng et al. [53] exploit the lack of a mutual authentication mechanism between routing services in the Near-RT RIC to poison the routing table and hence disrupt the victim's services.

The paper closest to our work is from Hung et al. [24], who focus on the E2 interface and the access control of xApps towards its services. The authors show that the RIC routing manager is not able to correctly handle duplicate xApps, only routing packets to the latest deployed one, thus creating possibilities for an attacker to stop the information flow to the legitimate xApp via a duplicate malicious one. Furthermore, the authors show that an xApp sending a RIC subscription response to the E2 termination causes a crash, making the E2 termination unavailable. Our work differs from that of Hung et al. [24] in two key points. First, our threat model envisions a malicious xApp sending subscription requests to the E2 termination of the Near-RT RIC. Different from malicious responses that could be identified by checking for the presence of their request counterparts, malicious subscription requests are harder to detect. Indeed, they are fundamental components of the RAN function lifecycle and cannot be discarded. The second key difference is that we show that the vulnerability we disclose affects all Near-RT RIC implementations currently available, rather than only the OSC implementation.

In recent work, Chiejina et al. [25] conducted one of the first system-level analyses of adversarial attacks and defensive mechanisms targeting the intelligent components integrated as xApps within the near-RT of an O-RAN system. Using a lab environment, they identified the vulnerability of ML models to adversarial attacks and the subsequent adverse effects on both the models and network performance. They also demonstrated that these effects persisted despite the system's rapid RTT capabilities to meet stringent latency constraints. Their work unveiled shortcomings of the current O-RAN architectures and proposed a future O-RAN system design and security direction.

A novel methodology has been developed in [54], utilizing the MITRE ATT&CK framework to objectively assess specific threats in O-RAN. This work analyzes the O-Cloud component within the O-RAN ecosystem as a representative example, demonstrating that no single threat class possesses complete security. The analysis reveals that the entire component responsible for the virtualization of the RANs reaches a high average exploitability score

of 8.7 out of 10. This underscores the susceptibility to existing vulnerabilities and highlights the necessity for addressing security risks within O-RAN.

A recent work by Thimmaraju et al. [23] highlighted the architectural weakness of O-RAN centralized controller, similar to vulnerabilities in Software-Defined Networking (SDN) controllers. The authors conducted a security risk assessment study of two state-of-the-art open-source Near-RT RICs for known security vulnerabilities via dependency and configuration analysis by developing an O-RAN A1 interface security testing tool called OAITT. They present novel timing and storage covert channels between Non-RT RICs and the Near-RT RIC that exploit the A1 protocol. Their research led to the reporting of Threat "T-A1-04" in the O-RAN Study on Security for Non-RT RIC. They also identified and reported a denial of service vulnerability in the µONOS (version 1.4) A1 interface implementation. Complementing to our work, the above studies show that the architecture of O-RAN RIC is vulnerable to covert channel and DoS attacks.

Few other research works proposed security solutions for O-RAN. Wen et al. [55] proposed 5GSPECTOR, a methodology to detect layer 3 attacks (e.g., overshadowing), leveraging the programmability offered by O-RAN, its xApps, and their metering capabilities. Atalay et al. [56] proposed an authorization framework for xApps, where xApp onboarding on the Near-RT RIC is subject to the presence of valid JSON web tokens. Neither of these two solutions can mitigate our proposed attack. Indeed, 5GSPECTOR focuses on attacks where the malicious user exploits the physical layer. Therefore, it cannot detect repeated malicious subscription requests (our attack). While the solution from Atalay et al. may seem like a possible countermeasure for this attack, their authorization framework only prevents a malicious user from onboarding non-authorized xApps (external attackers). However, we also focus on insider attacks, i.e., attackers that can deploy or exploit seemingly legitimate xApps to generate a volumetric DoS attack.

8.2. DoS Research in O-RAN

Currently, there are not many papers that deal with DoS specifically for O-RAN. Some papers mention DoS as a possible attack vector and also partially outline possible DoS attack paths. However, these publications do not conduct a feasibility study or other more detailed necessary investigations. To the best of our knowledge, there are only three publications that describe and investigate a DoS attack in detail. They all focus on the Fronthaul Interface (FHI) and the A1 interface. Table 3 presents related work on the topic DoS in O-RAN as well as the attacked interface and the O-RAN component.

Liao et al. [22] developed a control-plane DoS attack against the fronthaul interface of O-RAN. With a software tool, they simulate attacks and evaluate their impact using a DU-Radio Unit (RU) testbed based on OSC source code. Their experiments demonstrate that C-Plane DoS attacks can cause significant disruptions, particularly in user-plane message processing. However, their current testbed lacks full compliance with real-world O-RAN deployments, necessitating further evaluation.

Work	Interface	Component
Liao et al. [22]	FHI	C-Plane
Felina et al. [57]	FHI	C/U-Plane
Thimmaraju et al. [23]	A1	Near-RT-RIC
Our work	E2	Near-RT-RIC

TABLE 3: Comparison of existing work for DoS in O-RAN and their affected components, as well as the used interface.

Felina et al. [57] also perform a DoS attack targeting the control- and user-plane of the O-RAN fronthaul interface. They introduce a custom-built tool that is compliant with O-RAN end-to-end test specifications and generates high-volume packets targeting the O-DU and O-RAN Radio Unit (O-RU). They vary attack parameters such as traffic type, data rate, and spoofed source Medium Access Control (MAC) addresses to evaluate the impact on system throughput, block error rate, and overall stability. Their experiments demonstrate varying levels of system vulnerability, highlighting that certain configurations are susceptible to severe service degradation.

The work by Thimmaraju et al. [23] mentioned above also found that µONOS (version 1.4) is susceptible to a DoS attack. The attack exploited the A1 interface by sending a high rate of legitimate policy PUT requests, causing excessive processing delays and ultimately crashing the Kubernetes A1 pod. This resulted in a complete service disruption, requiring manual efforts to restart the pod and reinstall policies.

9. Conclusion

In this paper, we demonstrate that a relatively simple subscription-based DoS attack can disrupt popular O-RAN implementations. The proposed novel DoS attack against the Near-RT RIC implementations proves highly effective. Indeed, employing a malicious xApp, we illustrate how an attacker can flood the Near-RT RIC with numerous subscription requests, rendering it inaccessible. The resulting issues lead to the initial unavailability of the service for new user connections, followed by a complete service outage for the end user when all affected components must be restarted to restore normal operation. Our measurements are conducted under realistic conditions mirroring those encountered in real RAN deployments. The exploit takes advantage of the lack of a rate-limiting mechanism in the SM, a logical component within the Near-RT RIC responsible for managing E2 subscription requests in current implementations. It is noteworthy that the requisite number of requests for a successful attack varies across different implementations, underscoring the adaptability and potential widespread impact of such attacks.

Additionally, we discuss simple countermeasures that prove highly effective with minimal operational overhead. Given the increasing significance of O-RAN and 5G architectures in future networks, it is imperative to scrutinize their security. To facilitate further exploration in this domain, we are making the source code for the attack and the mitigation for FlexRIC publicly available. As part of our future research agenda, we intend to investigate the impact of subscription-based DoS attacks on other components of O-RAN architecture.

Acknowledgments

We would like to thank Kashyap Thimmaraju and Jean-Pierre Seifert for their feedback on an earlier version of this paper. The authors acknowledge the financial support by the German Federal Ministry of Education and Research – Bundesministerium für Bildung und Forschung (BMBF), as part of the Project "6G-RIC: The 6G Research and Innovation Cluster" (project number 825026). This research was supported by the Netherlands' National Growth Fund through the Dutch 6G flagship project "Future Network Services" and the European Commission under the Horizon Europe Programme as part of the projects SafeHorizon (Grant Agreement #101168562) and RECITALS (Grant Agreement #101168490). The content of this article does not reflect the official opinion of the Dutch government or the European Union. The responsibility for the information and views expressed therein lies entirely with the authors.

References

[1] *O-RAN Architecture Description 11.0*, O-RAN Alliance, Oct. 2023.

[2] A. Checko, H. L. Christiansen, Y. Yan, L. Scolari, G. Kardaras, M. S. Berger, and L. Dittmann, "Cloud RAN for mobile networks—A technology overview," *IEEE Communications surveys & tutorials*, vol. 17, no. 1, pp. 405–426, 2014.

[3] X. Foukas, G. Patounas, A. Elmokashfi, and M. K. Marina, "Network slicing in 5G: Survey and challenges," *IEEE Communications Magazine*, vol. 55, no. 5, pp. 94–100, 2017.

[4] Vodafone. (2019) Vodafone pioneers innovative network tech to increase suppliers and extend rural internet access. [Online]. Available: https://www.vodafone.com/news/technology-news/vodafone-pioneers-innovative-network-tech-to-increase-suppliers-and-extend-rural-internet-access

[5] Orange. (2021) MWC 2021: With Pikeo, 5G Networks Switch to the Cloud and Become Autonomous. [Online]. Available: https://hellofuture.orange.com/en/mwc-2021-with-pikeo-5g-networks-switch-to-the-cloud-and-become-autonomous/

[6] T. Zhang, J. O. Boateng, T. Ui Islam, A. Ahmad, H. Zhang, and D. Qiao, "ARA-O-RAN: End-to-End Programmable O-RAN Living Lab for Agriculture and Rural Communities," in *IEEE INFOCOM 2024 - IEEE Conference on Computer Communications Workshops*, 2024, pp. 1–6.

[7] M. Hoffmann and M. Dryjański, "Energy Efficiency in Open RAN: RF Channel Reconfiguration Use Case," *IEEE Access*, vol. 12, pp. 118 493–118 501, 2024.

[8] A. I. Abubakar, O. Onireti, Y. Sambo, L. Zhang, G. K. Ragesh, and M. Ali Imran, "Energy Efficiency of Open Radio Access Network: A Survey," in *2023 IEEE 97th Vehicular Technology Conference*, 2023, pp. 1–7.

[9] O. T. Demir, M. Masoudi, E. Björnson, and C. Cavdar, "Cell-Free Massive MIMO in O-RAN: Energy-Aware Joint Orchestration of Cloud, Fronthaul, and Radio Resources," *IEEE J.Sel. A. Commun.*, vol. 42, no. 2, p. 356–372, Feb. 2024. [Online]. Available: https://doi.org/10.1109/JSAC.2023.3336187

[10] H. Lee, J. Cha, D. Kwon, M. Jeong, and I. Park, "Hosting AI/ML Workflows on O-RAN RIC Platform," in *2020 IEEE Globecom Workshops (GC Wkshps*, 2020, pp. 1–6.

[11] B. Balasubramanian, E. S. Daniels, M. Hiltunen, R. Jana, K. Joshi, R. Sivaraj, T. X. Tran, and C. Wang, "RIC: A RAN Intelligent Controller Platform for AI-Enabled Cellular Networks," *IEEE Internet Computing*, vol. 25, no. 2, pp. 7–17, 2021.

[12] ——, "RIC: A RAN intelligent controller platform for AI-enabled cellular networks," *IEEE Internet Computing*, vol. 25, no. 2, pp. 7–17, 2021.

[13] R. Ntassah, G. M. Dell'Aera, and F. Granelli, "xApp for Traffic Steering and Load Balancing in the O-RAN Architecture," in *ICC 2023-IEEE International Conference on Communications*. IEEE, 2023, pp. 5259–5264.

[14] C.-C. Chen, M. Irazabal, C.-Y. Chang, A. Mohammadi, and N. Nikaein, "FlexApp: Flexible and low-latency xApp framework for RAN intelligent controller," in *ICC 2023-IEEE International Conference on Communications*. IEEE, 2023, pp. 5450–5456.

[15] US Cybersecurity and Infrastructure Security Agency (CISA), "Cost of a Cyber Incident: Systematic Review and Cross-Validation," 2020.

[16] U.S. Cybersecurity and Infrastructure Security Agency (CISA), "Cyber Incident Reporting for Critical Infrastructure Act of 2022 (CIRCIA)," https://www.cisa.gov/circia, 2023.

[17] European Union Agency for Cybersecurity (ENISA), "Threat Landscape Report 2023," https://www.enisa.europa.eu/publications/enisa-threat-landscape-2023, 2023.

[18] M. Liyanage, A. Braeken, S. Shahabuddin, and P. Ranaweera, "Open RAN Security: Challenges and Opportunities," *Journal of Network and Computer Applications*, vol. 214, p. 103621, 2023.

[19] 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.

[20] *O-RAN Study on Security for Near Real Time RIC and xApps 5.0*, O-RAN Alliance, Feb. 2024.

[21] S. Soltani, M. Shojafar, A. Brighente, M. Conti, and R. Tafazolli, "Poisoning Bearer Context Migration in O-RAN 5G Network," *IEEE wireless communications letters*, vol. 12, no. 3, pp. 401–405, 2022.

[22] S.-H. Liao, C.-W. Lin, F. A. Bimo, and R.-G. Cheng, "Development of C-plane DoS attacker for O-RAN FHI," in *Proceedings of the 28th Annual International Conference on Mobile Computing And Networking*, 2022, pp. 850–852.

[23] K. Thimmaraju, A. Shaik, S. Flück, P. J. F. Mora, C. Werling, and J. Seifert, "Security Testing The O-RAN Near-Real Time RIC & A1 Interface," in *Proceedings of the 17th ACM Conference on Security and Privacy in Wireless and Mobile Networks, WiSec 2024, Seoul, Republic of Korea, May 27-29, 2024*. ACM, 2024, pp. 277–287.

[24] 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*, 2024.

[25] A. J. Chiejina, B. Kim, K. Chowhdury, and V. K. Shah, "System-level Analysis of Adversarial Attacks and Defenses on Intelligence in O-RAN based Cellular Networks," in *Proceedings of the 17th ACM Conference on Security and Privacy in Wireless and Mobile Networks, WiSec 2024, Seoul, Republic of Korea, May 27-29, 2024*. ACM, 2024, pp. 237–247.

[26] R. Schmidt, M. Irazabal, and N. Nikaein, "FlexRIC: an SDK for next-generation SD-RANs," in *Proceedings of the 17th International Conference on Emerging Networking EXperiments and Technologies*, ser. CoNEXT '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 411–425. [Online]. Available: https://doi.org/10.1145/3485983.3494870

[27] F. Klement, "FlexRIC Subscription DoS Attack," https://github.com/fklement/flexric_e42_sub_dos_oran.git, 2025.

[28] *O-RAN Cloud Architecture and Deployment Scenarios for O-RAN Virtualized RAN 6.0*, O-RAN Alliance, Feb. 2024.

[29] *O-RAN Near-RT RIC Architecture 5.0*, O-RAN Alliance, Oct. 2023.

[30] *O-RAN E2 General Aspects and Principles (E2GAP) 5.0*, O-RAN Alliance, Feb. 2024.

[31] *O-RAN E2 Service Model (E2SM), RAN Control 5.0*, O-RAN Alliance, Feb. 2024.

[32] *O-RAN E2 Service Model (E2SM) Cell Configuration and Control 3.0*, O-RAN Alliance, Feb. 2024.

[33] *O-RAN E2 Service Model (E2SM) KPM 4.0*, O-RAN Alliance, Oct. 2023.

[34] *O-RAN Operations and Maintenance Architecture 11.0*, O-RAN Alliance, Oct. 2023.

[35] NSF CISE, "Open AI Cellular (OAIC) NSF CISE community research infrastructure project," Jan 2025. [Online]. Available: https://www.openaicellular.org

[36] Aether, "SD-RAN - Open Software-based Split-RAN," Jan 2025. [Online]. Available: https://aetherproject.org/sd-ran/

[37] OpenAirInterface, "OpenAirInterface5G," https://gitlab.eurecom.fr/oai/openairinterface5g, 2024.

[38] Communication System Department of Eurecom, "Mosaic5G: A Consortium For Building Agile 5G Service Platforms and Opening Fast Wireless Innovation," Jan 2025. [Online]. Available: https://mosaic5g.io

[39] O-RAN Software Community The Linux Foundation, "O-RAN Software Community (SC)," Jan 2025. [Online]. Available: https://o-ran-sc.org

[40] srsRAN, "srsRAN Project," https://docs.srsran.com/projects/project/en/latest/index.html, 2024.

[41] Institute for the Wireless Internet of Things at Northeastern University, "An Open, Programmable, Multi-vendor 5G O-RAN Testbed with NVIDIA ARC and OpenAirInterface," Jan 2025. [Online]. Available: https://x5g.org

[42] University of Utah, "Powder (the Platform for Open Wireless Data-driven Experimental Research)," Jan 2025. [Online]. Available: https://www.powderwireless.net

[43] EURECOM, "Flexible RAN Intelligent Controller (FlexRIC) and E2 Agent," https://gitlab.eurecom.fr/mosaic5g/flexric, 2024.

[44] Eurecom. (2021) FlexRIC: A Flexible RAN Intelligent Controller. [Online]. Available: https://www.virtualexhibition.o-ran.org/classic/generation/2021/category/intelligent-ran-control-demonstrations/sub/intelligent-control/140

[45] O-RAN Alliance. (2024) Open Software for the RAN. [Online]. Available: https://www.o-ran.org/software

[46] *O-RAN Use Cases Detailed Specification*, O-RAN Alliance, Oct. 2023.

[47] D. P. V. S, S. C. Sethuraman, and M. K. Khan, "Container security: Precaution levels, mitigation strategies, and research perspectives," *Computers & Security*, p. 103490, 2023.

[48] *O-RAN Security Threat Modeling and Risk Assessment 2.0*, O-RAN Alliance, Feb. 2024.

[49] *O-RAN Study on Security for Application Lifecycle Management 3.0*, O-RAN Alliance, Feb. 2024.

[50] *O-RAN Study on Security for O-Cloud 5.0*, O-RAN Alliance, Feb. 2024.

[51] O-RAN Project, "O-RAN Subscription Manager Documentation," https://docs.o-ran-sc.org/projects/o-ran-sc-ric-plt-submgr/en/latest/user-guide.html, 2024.

[52] F. Klement, S. Katzenbeisser, V. Ulitzsch, J. Krämer, S. Stanczak, Z. Utkovski, I. Bjelakovic, and G. Wunder, "Open or not Open: Are conventional radio access networks more secure and trustworthy than Open-RAN?" *arXiv preprint arXiv:2204.12227*, 2022.

[53] C.-h. Tseng, C.-F. Hung, B.-K. Hong, and S.-M. Cheng, "On Manipulating Routing Table to Realize Redirect Attacks in O-RAN by Malicious xApp," in *2023 26th International Symposium on Wireless Personal Multimedia Communications (WPMC)*. IEEE, 2023, pp. 288–292.

[54] 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, 2024.

[55] H. Wen, P. Porras, V. Yegneswaran, A. Gehani, and Z. Lin, "5G-SPECTOR: An O-RAN Compliant Layer-3 Cellular Attack Detection Service," in *NDSS 2024*. Internet Society, 2024.

[56] T. O. Atalay, S. Maitra, D. Stojadinovic, A. Stavrou, and H. Wang, "Securing 5G Open RAN with a Scalable Authorization Framework for xApps," in *IEEE INFOCOM 2023-IEEE Conference on Computer Communications*. IEEE, 2023, pp. 1–10.

[57] F. Feliana, T. Hung, B. Chen, and R. Cheng, "Evaluation of Control/User-Plane Denial-of-Service (DoS) Attack on O-RAN Fronthaul Interface," in *IEEE INFOCOM 2024 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, 2024, pp. 1–6.

Appendix A.
Open Science and Data Availability

The code necessary to carry out the attack from SC_1 is available in an GitHub repository [27]. This repository enables the execution of the E2 Subscription DoS attack on the FlexRIC Near-RT RIC implementation. The attack is fully executable within a Docker container and can be initiated via a Makefile, which enables a quick and uncomplicated execution. However, for SC_2, the deployment of srsRAN and Open5GS is required, while for SC_3 and SC_4, a fully operational OSC O-RAN Kubernetes setup is necessary. As a result, we do not provide a fully automated test environment.