Covert Communication Channels in the O-RAN SC Near-Real Time RIC

Daniel Balint
Humboldt University of Berlin
Berlin, Germany

Kashyap Thimmaraju
Technical University of Berlin
Berlin, Germany

Stefan Schmid
Technical University of Berlin
Fraunhofer SIT
Weizenbaum Institute
Berlin, Germany

***Abstract*—The O-RAN Near-Real-Time RAN Intelligent Controller (RIC) relies on strict isolation between xApps to enforce security boundaries as recommended by O-RAN WG11. In this paper, we demonstrate that the current O-RAN Software Community (O-RAN SC) reference implementation violates these assumptions. We identify two previously undocumented vulnerabilities in the Subscription Manager that enable covert communication between network-isolated xApps. The first channel redirects subscription responses to unintended recipients, enabling low-rate data leakage without special privileges. The second abuses a shared debugging endpoint to establish a high-throughput, bidirectional covert channel. We experimentally evaluate both channels on a deployed O-RAN SC Near-RT RIC and discuss their security impact and mitigations.**

***Index Terms*—O-RAN, Near-Real Time RIC, xApps, covert channels, network security, Subscription Manager**

I. Introduction

A. Background: O-RAN and Services

The O-RAN Architecture [1] is a paradigm shift in how mobile networks are designed and deployed. It was created by the O-RAN Alliance e.V. [2], a global community of over 300 mobile network operators, manufacturers and vendors, government agencies, research and academic institutions and numerous other entities. It aims to increase openness and innovation by eliminating reliance on vendor-specific hardware, instead decomposing traditional base stations into modular, logically separated network functions connected through standardised, open interfaces, enabling vendors open, unified, and extensible access to radio access networks.

Figure 1 shows an overview of the logical architecture of O-RAN. The Open Cloud (O-Cloud) is a cloud infrastructure made up of O-RAN compliant physical nodes designed to host the O-RAN network functions, the required supporting software stack (including OS, VM, and container runtime), and the orchestration and management capabilities that align with O-RAN specifications. It provides fault, configuration, accounting, performance, and security (FCAPS) services to other network functions and exposes the O2 interface, which is used by the Service Management and Orchestration Framework (SMO Framework) to manage e.g. configurations, workload, faults or performance of the O-Cloud. It also contains the Non-Real Time RIC (RAN Intelligent Controller) which collects data and information about the RAN from the SMO Framework and the O2 interfaces. It exposes these to vendor specific rApps installed on the Non-Real Time RIC. The rApps use data analytics tools and/or AI/ML to optimise the RAN, triggering actions (e.g. reconfiguration) and policies (e.g. to prioritise emergency calls) via the O1, O2 and A1 interfaces in a control loop in timescales >1s. The actual radio signaling takes place in the Open Central Unit (O-CU), Open Distributed Unit (O-DU) and Open Radio Unit (O-RU), containing different parts of the overall protocol stack.

ISBN 978-3-903176-82-9

B. Our Focus: Near-Real Time RIC and xApps

The Near-Real Time RIC [3] collects near-real time (<1s) data from RAN nodes and policies from the Non-Real Time RIC and then enforces those policies and performs time critical RAN optimisations, such as radio resource management, cell handover management, load balancing and interference detection and mitigation. It performs its function by allowing vendors to install xApps consisting of one or more microservices. xApps and other RIC components may communicate with each other via a shared messaging infrastructure. They can also save data like time series or information about user equipment in the Shared Data Layer (SDL), which can then be accessed by other xApps. The RIC connects to the rest of the RAN via E2 interfaces, exchanging E2 Application Protocol (E2AP) [4], [5] messages via SCTP. The E2AP specifies procedures for xApps to be able to subscribe to E2 nodes, coordinated by the RICs subscription manager. The E2 nodes expose so called RAN functions which can either be queried (e.g. for KPI) and/or controlled (e.g. to save energy) by an xApp through a subscription to provide its services of RAN optimisation.

C. Our Motivation: Security Considerations and Isolation

Because base stations are usually critical infrastructure, the O-RAN Alliance e.V. has its own security workgroup WG11, conducting and releasing security analyses and risk assessments of various architectural components, accompanied by guidance on appropriate safeguards and mitigation measures. Their "*Study on Security for Near Real Time RIC and xApp*" [6] outlines 14 potential security risks associated with xApps and the Near-Real Time RIC. Key issue #4 identified in the study is concerned with the isolation of xApps from each other. The main issue they outline is the possibility of an xApp disrupting or degrading the service of other network functions

using the same platform by depleting available resources, such as CPU, I/O bandwith or memory. Of particular relevance to this paper is their observation that xApps can have different threat profiles and/or varying access privileges to data. The establishment of a covert communication channel would facilitate the exfiltration of data to unintended recipients. The security work group recommends appropriate isolation of xApps into different security levels.

D. Our Contributions

We revisit and analyze the security of the O-RAN critical infrastructure. We find that there is risk of violation of the WG11 security guidelines, by performing side-channel attacks on the O-RAN Software Community's reference implementation of the Near-Real Time RIC, enabling covert communication between isolated xApps via the Subscription Manager [7].

Isolation can be circumvented in two ways. Firstly, a subscribing (attacker) xApp can choose a 16-bit identifying integer in its subscription request which is returned by the Subscription Manager in its second response. This response can be redirected to any xApp by simply specifying them as the return endpoint in the request.

Secondly, the Subscription Manager has a shared REST endpoint intended for debugging, which lists the *ClientEndpoint*s of active subscriptions. Because entries are not validated, arbitrary attacker-controlled data can be deposited into the debugging endpoint via the *ClientEndpoint* field, which can then be read by other xApps.

We implemented both exploits on the J-Version of the O-RAN SC Near-Real Time RIC and evaluated their performance on consumer hardware.

E. Novelty

Existing literature tends to fall in one of two main categories. The first category comprises papers which discuss the O-RAN architecture at a high level and identify general security concerns and threat vectors [8]–[11]. The second category of papers are concerned with disruptive denial of service attacks on the O-RAN infrastructure [12]–[15]. Our paper differs from the published papers because it discusses a specific vulnerability, focusing specifically on the O-RAN SC implementation of the Near-Real Time RIC and how to exploit it, rather than focusing on possible attack vectors in general.

II. Preliminaries and Methodology

This section first introduces additional preliminaries and the implementation of the Near-Real Time RIC. Then, we present our methodology.

A. The O-RAN SC Near-Real Time RIC

The O-RAN Software Community (SC) [16] is a collaborative effort between the O-RAN Alliance and the Linux Foundation. They create open source software for radio access networks based on the specifications of the O-RAN Alliance with the goal of building solutions that can be deployed in commercial settings, meeting the requirements for real-time systems. One of their developments is an implementation of the Near-Real Time RIC. It can be set up as a Kubernetes cluster with each Pod running a different part of the RIC, such as the Subscription Manager or the E2 termination serving the E2 interface. Internal messaging is handled by the RIC Message Router (RMR) [17]. The shared data layer consists of a Pod running Redis.

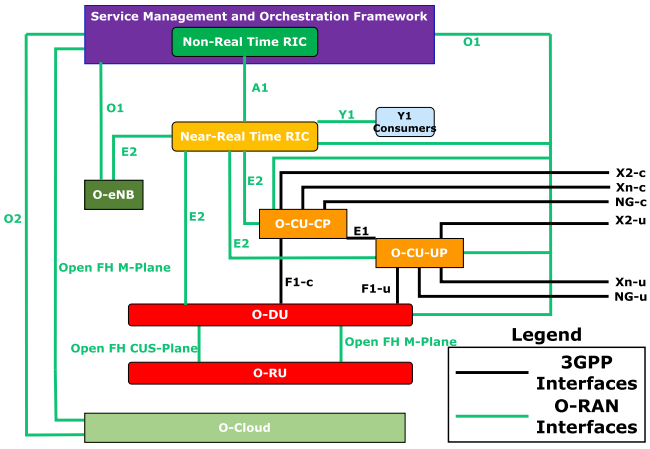


Fig. 1: O-RAN Architecture [1]

B. Methodology

In order to analyze the security properties, we set up the J-Release of the O-RAN SC Near-Real Time RIC in a VM on a consumer laptop. We consider two xApps deployed on the same Near-RT RIC but assigned to different security domains. Direct communication via RMR, REST, or shared storage is blocked by network policies. Both xApps are otherwise benign and operate within their assigned privileges.

We consider two attackers with access to xApp onboarding, e.g. a malicious mobile provider, one with a higher privilege than the other. The aim is to exfiltrate data from the higher-privileged xApp to the lower-privileged one without violating isolation rules. For the proof of concept, the RIC was set up for a scenario where a single shared RIC is serving every component, as opposed to splitting a physical RIC into multiple logical ones, or using multiple RICs (e.g. one for 4G and one for 5G).

As a starting point, we revisited CVE-2023-41627 [18], allowing an attacker to send forged RMR routing tables to E2 nodes. We concluded that the approach of manipulating components' routing tables into relaying messages is infeasible without disruption of RAN services because there is no way of obtaining a valid routing table of another component to manipulate in the first place.

Further experimentation with the RIC led to the investigation of the subscription mechanism (Fig. 2). Although internal RIC communication relies on the RMR, subscription establishment itself is initiated by an xApp through an unsecured HTTP REST API exposed by the Subscription Manager [19].

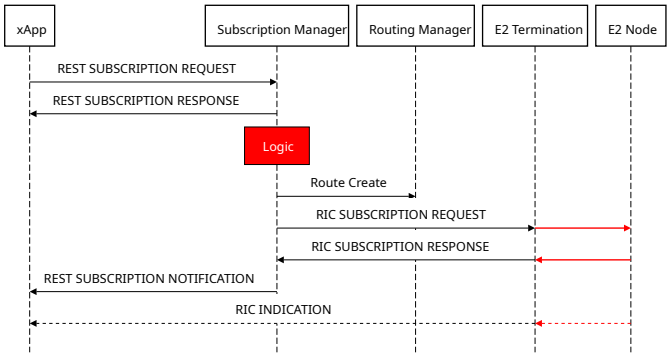


Fig. 2: Communication flow of a successful subscription request [7]

The Subscription Manager immediately sends a subscription response back to the xApp, confirming that the request has been received, then coordinates the subscription with the E2 Node via the RMR. Once the E2 node processed the request, or the request timed out, the Subscription Manager sends a subscription notification over HTTP back to the xApp, indicating whether coordinating the subscription was successful or not. Subsequent data flow from a subscription (e.g. metrics, events) is again handled by the RMR. The Subscription Manager and its exposed HTTP REST API enable the construction of two covert channels between isolated xApps.

III. Attack 1: Subscription Notification Redirection

Figure 3 shows the relevant parts of an HTTP REST API subscription request sent to the Subscription Manager by an xApp. The *SubscriptionId* is needed for modifying existing subscriptions and will be returned in the first subscription response. *ClientEndpoint* specifies the xApp endpoint which will use the subscription. The subscription notification and all further subscription traffic (Fig. 2) will be sent to this endpoint. This is necessary, because xApps may consist of multiple independent containers with different tasks. *Meid* identifies the E2 node to subscribe to, and *RANFunctionID* the function. The *SubscriptionDetails* specify what triggers the subscription and what actions should be taken upon being triggered. Most importantly, these details include the *XappEventInstanceId*, a 16-bit Integer chosen by the subscriber that is returned in the subscription notification to identify which specific subscription failed or succeeded (one subscription may include multiple requests).

```
{
"SubscriptionId": "",
"ClientEndpoint": {
    "Host":
     ↪  "service-ricxapp-listener1-http.ricxapp",
    "HTTPPort": 8080,
    "RMRPort": 4560
},
"Meid": "changing",
"RANFunctionID": 40,
"E2SubscriptionDirectives": {
...
},
"SubscriptionDetails": [
{
    "XappEventInstanceId": 8452,
    "EventTriggers": [8, 3, 231],
    "ActionToBeSetupList": [
    {
    ...
```

Fig. 3: Part of a Subscription request used in the exploit

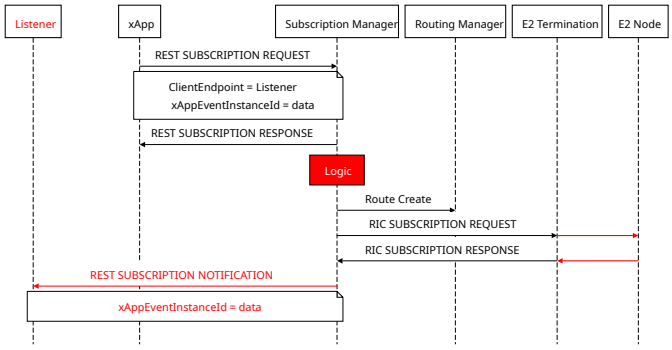


Fig. 4: Redirecting the subscription notification

Because the *ClientEndpoint* is never validated to belong to the subscribing xApp, it is possible to redirect the subscription notification to a different listener with information embedded in the *XappEventInstanceId*.
In our proof of concept we used failing subscriptions to avoid the overhead of managing active subscriptions. That means that an xApp can keep subscribing to an E2 node that does not support a specific requested function or action, while sending 16 bits of information to an arbitrary endpoint each time. The only limitation is that the subscription has to be sent to and fail on actual E2 node. Subscribing to a non-existent *Meid* will return an empty *SubscriptionInstances* array, missing the *XappEventInstanceId*.

```
{
        "SubscriptionId": "string",
        "SubscriptionInstances": [
        {
                "XappEventInstanceId":
                ↪  8452,
                "E2EventInstanceId": 0,
                "ErrorCause": "E2Node
                ↪  Cause: (Cause:5, Value
                ↪  0)",
                "ErrorSource": "E2Node"
        }
        ]
}
```

Fig. 5: REST subscription notification of a failed subscription

IV. Attack 2: Debug Endpoint Injection

Although we used failing, invalid subscriptions in our proof of concept, it is still necessary to explicitly unsubscribe from an invalid subscription using the *SubscriptionId* from the subscription notification. This was discovered when we queried the Subscription Managers' unsecured HTTP debugging endpoint [7] which lists the endpoints of every active and failed subscription unless explicitly unsubscribed:

```
vboxuser@OpenRAN2:~$ curl -X GET
 ↪  "http://10.244.0.232:8080/ric/v1/get_all
["service-ricxapp-listener1-http.ricxapp"]
```

In this case, the endpoint still lists the previous exploits' endpoint. The Subscription Managers' IP-Address can be obtained using Kubernetes' DNS.

Previously, the *ClientEndpoint* field was used to redirect the subscription notification. In this exploit, the information is directly embedded into the *ClientEndpoint* string. It does not matter if the endpoint is invalid, its string is written into the debugging endpoint and the sender receives the subscription ID from the first subscription response. We used the format „magicvalue-seqNr-base64data“, which includes a magic value for the listener to find, a sequence number to avoid duplicate information, and arbitrary base64 information. We did not encounter illegal characters or a size limit, embedding up to 32KiB of data in that field. A listener can continuously query the debugging endpoint, while the sender writes data into it, waits, then unsubscribes so data is not retained. As long as the listener queries the endpoint faster than the sender deposits information, this enables a continuous flow of information.

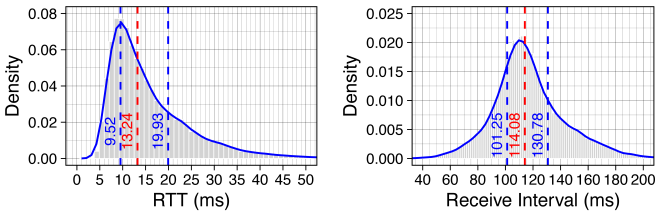
V. Evaluation

We evaluate the performance and reliability of the two identified covert channels by measuring sender-side round-trip times (RTT), receiver-side inter-arrival times, message loss, and effective throughput. RTTs were measured between sending a subscription request and receiving the first subscription response from the Subscription Manager. All experiments were conducted on the O-RAN SC Near-RT RIC J-release using a compliant E2 node implementation. and performed in a virtualised environment on consumer-grade hardware.

A. Attack 1: Subscription Notification Redirection

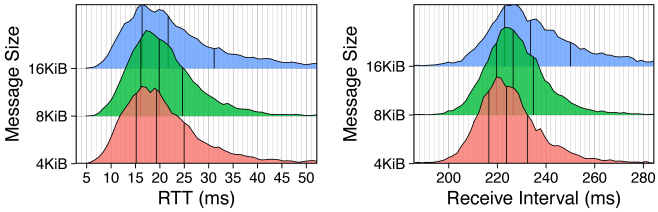
For the first covert channel, we conducted 15 runs transmitting 4 KiB of data per run using 2048 messages carrying 2 B of payload each. Messages were sent at a fixed interval of 100 ms, which was the lowest interval that did not cause noticeable degradation of the Subscription Manager or xApp performance. The RIC cluster was kept running continuously between runs to capture long-term effects.

Figure 6 shows aggregated sender-side RTT and receiver-side inter-arrival time densities. The sender-side median RTT is 13 ms, with approximately two thirds of responses received



(a) Sender RTT Density (b) Receive Interval Density

Fig. 6: RTT and Receive Interval Densities



(a) Sender RTT Density over Message Size (b) Receive Interval Density over Message Size

Fig. 7: RTT and Receive Interval Densities over Message Size

within 20 ms. However, receiver-side inter-arrival times exhibit substantial jitter: 25% of messages arrive in less than 100 ms, indicating reordering or buffering effects within the channel. These effects are consistent with concurrent handling of REST-based subscription requests and internal buffering or retry mechanisms in the Subscription Manager and RMR.

Across consecutive runs, total transmission time increases progressively, indicating accumulating load or resource contention. Redeploying the cluster restores performance to baseline levels. Using a 100 ms sending interval, the channel achieves a median inter-arrival time of approximately 114 ms and a 75th percentile of 130 ms, corresponding to effective data rates of 17.5 B/s and 15.4 B/s, respectively. While these rates impose measurable strain on the RIC and are unsuitable for sustained high-volume transfer, they are sufficient for low-bandwidth data exfiltration.

B. Attack 2: Debug Endpoint Injection

For the second covert channel, we conducted four runs of 5000 messages each using message sizes of 4, 8, 16, and 32 KiB. For each message, the sender writes data into the Subscription Manager's debugging endpoint, waits 200 ms for retrieval by the listener, and unsubscribes to clear the entry. The receiver continuously polls the endpoint. To eliminate long-term effects, the RIC cluster was fully redeployed between runs.

Figure 7 shows sender-side RTT and receiver-side inter-arrival time densities across message sizes. For 4 KiB and 8 KiB messages, median sender RTT remains near 20 ms with minimal variation, indicating underutilized resources. At 16 KiB, higher percentiles shift upward, suggesting operation near system capacity. Receiver-side inter-arrival times increase

correspondingly, with median values of approximately 224 ms (4 KiB), 226 ms (8 KiB), and 234 ms (16 KiB).
The 32 KiB run fails to complete successfully, resulting in severe performance degradation and the loss of 117 messages. Latency increases sharply early in the run, ultimately rendering the cluster unrecoverable. For message sizes up to 16 KiB, message loss remains minimal, with no losses observed at 8 KiB.
At 16 KiB per message, the channel achieves a median inter-arrival time of 234 ms and a 75th percentile of 250 ms, corresponding to effective data rates of approximately 68 KiB/s and 64 KiB/s, respectively. For improved stability, 8 KiB messages achieve approximately 35 KiB/s. These rates are substantially higher than those of the first channel and sufficient for transferring sensitive data, assuming shared access to the debugging endpoint.

VI. Discussion and Conclusion

A. Risks and Feasibility

The risks that an attacker is detected depend entirely on how the system is monitored and the nature and volume of other traffic. xApps can subscribe to the functions they need at the start of their operational lifespan and exclusively receive indications from that point onwards. There are also RAN functions such as RAN Control [20], which rely on a stream of subscription requests to continuously modify them.
If the environment is otherwise conservative with the amount of request traffic, the continuous transmission of subscriptions by an xApp might contrast sharply to the system's typical traffic profile. Examining the E2 termination's logs, the amount of failed subscriptions might be reason for suspicion, although this could also be attributed to faulty code initiating erroneous subscription requests in an endless loop. In principle, it is feasible for these exploits to be carried out with valid subscription requests which create valid subscriptions. However, those would either have to be kept track of within the xApp to be unsubscribed from at a later point, or unsubscribed from shortly after a successful subscription which also looks like unusual activity on the system.
The first exploit can not be detected using the Subscription Manager's debugging endpoints [7]. Subscriptions simply show up as if they were requested by the listening xApp, not the attacker. It also does not necessitate any special configuration, only an E2 node which has properly been set up with a connection to the Subscription Manager. Endpoint addresses are known to developers and can easily be shared out of band. However, it is impossible to differentiate subscription responses coming from the intended sender or from another source. This should not occur though, as subscription responses are usually intended for the requestor. It is also trivial to expose a dedicated endpoint within the xApp specifically for the covert channel.
The second exploit is less subtle. It requires the Subscription Manager's debugging endpoint to be accessible by both parties. Given that the scenario under consideration involves an environment where xApps are isolated from each other, an open debugging endpoint like this appears to be unlikely. If it is accessible, encoded data is directly visible in the *ClientEndpoint* string, which would reveal the exploit immediately if the debugging endpoint was to be examined. There is a risk that, in the event that the unsubscribe process is unsuccessful for some reason, the Subscription Manager may become overloaded with all the subscriptions it is required to maintain, even those that have not successfully been cancelled. In summary though, if the environment for it is given, both exploits are very easily implemented.

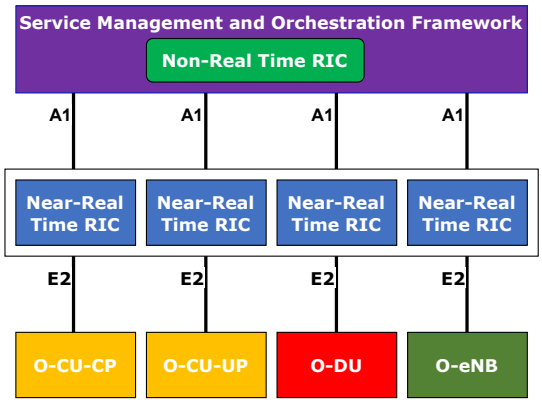


Fig. 8: One physical, multiple logical RIC [1]

B. Mitigation

The O-RAN Security Work Group issues recommendations for resolving the issue of isolating xApps in their "Study on Security for Near-Real Time RIC and xApps" [6]. As per the premise for this paper, the recommendation is to categorise xApps into security zones based on their threat profile, with communication between boundaries having to be authorised and authenticated. It is these recommendations that can be circumvented using the exploits shown. However, the paper also puts forward a number of solutions that address several of the other issues it identifies [6]. One solution is an authentication schema for APIs like the subscription interface using mTLS, IPSec and client authentication. This would eliminate the need for the *ClientEndpoint* field of the subscription request, eliminating the possibility of redirecting the subscription notification from the Subscription Manager. The other solution proposes the implementation of an authorisation framework for APIs using OAuth, which would ensure that an xApp is granted permission to utilise a particular function or API. Simply not allowing deployed xApps to use the debugging API unless requested would mitigate the exploit.
It is also possible to implement proper xApp isolation without modifying the software. The O-RAN specification allows for the use of multiple Near-Real Time RIC. Figure 8 shows an implementation where one physical Near-Real Time RIC is divided into multiple logical ones. In Figure 9 there is multiple physical Near-Real Time RIC. In those cases, each xApp can run on their own respective RIC with their own Subscription Manager, isolated from the other RIC. In summary, there are

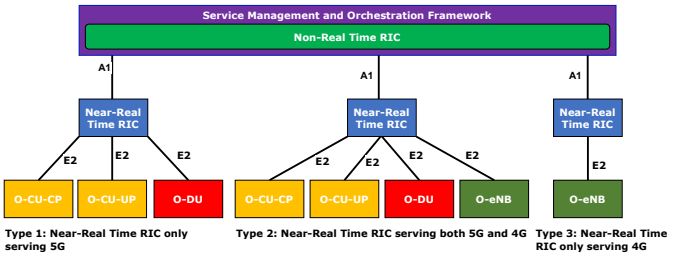


Fig. 9: Multiple physical RIC [1]

numerous solutions for the vulnerabilities. The RIC can be set up with isolation already in mind, a *ClientEndpoint* sanity check is quick and easy to implement as a fix, and different forms of authentication can then be added to further harden security.

Acknowledgments

Research supported by the Federal Ministry for Research, Technology and Space (BMFTR) in Germany, project xG-NOVA, grant: 16KISS003, project xG-RIC, grant: 16KIS2434, project Q-Fiber, grant: 16KISQ124, project 6G Research, Federal Ministry for Digital and Transport (BMDV), project O-RAN Lab, grant: 19OL22004C.

References

[1] O-RAN Work Group 1 (Use Cases and Overall Architecture), "O-RAN Architecture Description (O-RAN.WG1.OAD-R003-v12.00)," Technical Specification, O-RAN ALLIANCE e.V., 2024.

[2] "O-RAN ALLIANCE e.V — o-ran.org." https://www.o-ran.org/. [Accessed 28.03.2025].

[3] O-RAN Working Group 3 (Near-Real-time RAN Intelligent Controller and E2 Interface Workgroup), "Near-RT RIC Architecture (O-RAN.WG3.RICARCH-R003-v04.00)," technical specification, O-RAN ALLIANCE e.V., Mar. 2023.

[4] O-RAN Working Group 3 (Near-Real-time RAN Intelligent Controller and E2 Interface Workgroup), "E2 General Aspects and Principles (E2GAP) (O-RAN.WG3.E2GAP-R003-v03.00)," technical specification, O-RAN ALLIANCE e.V., Mar. 2023.

[5] O-RAN Working Group 3 (Near-Real-time RAN Intelligent Controller and E2 Interface Workgroup), "E2 Application Protocol (E2AP) (O-RAN.WG3.E2AP-v02.03)," technical specification, O-RAN ALLIANCE e.V., Mar. 2022.

[6] O-RAN Work Group 11 (Security Work Group), "Study on Security for Near Real Time RIC and xApps (O-RAN.WG11.Security-Near-RT-RIC-xApps-TR.0-R003-v05.00)," tech. rep., O-RAN ALLIANCE e.V., 2024.

[7] "User-Guide (new) 2014; ric-plt/submgr master documentation — docs.o-ran-sc.org." https://docs.o-ran-sc.org/projects/o-ran-sc-ric-plt-submgr/en/latest/user-guide.html. [Accessed 31-03-2025].

[8] 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 and Tutorials*, vol. 25, no. 2, pp. 1376–1411, 2023.

[9] Y.-Z. Chen, T. Y.-H. Chen, P.-J. Su, and C.-T. Liu, "A brief survey of open radio access network (o-ran) security," 2023.

[10] J. Groen, S. D'Oro, U. Demir, L. Bonati, M. Polese, T. Melodia, and K. Chowdhury, "Implementing and evaluating security in o-ran: Interfaces, intelligence, and platforms," *IEEE Network*, vol. 39, p. 227–234, Jan. 2025.

[11] A. S. Abdalla and V. Marojevic, "End-to-end o-ran security architecture, threat surface, coverage, and the case of the open fronthaul," *IEEE Communications Standards Magazine*, vol. 8, p. 36–43, Mar. 2024.

[12] 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)*, p. 288–292, IEEE, Nov. 2023.

[13] W. Lin, Z. Li, B. Chen, J. Liu, R.-G. Cheng, and F. Zhang, "5g-muffler: Covert dos attacks over open fronthaul interface of o-ran 5g network," in *IEEE INFOCOM 2025 - IEEE Conference on Computer Communications*, pp. 1–10, 2025.

[14] P. Baguer, G. M. Yilma, E. Municio, G. Garcia-Aviles, A. Garcia-Saavedra, M. Liebsch, and X. Costa-Pérez, "Attacking o-ran interfaces: Threat modeling, analysis and practical experimentation," *IEEE Open Journal of the Communications Society*, vol. 5, pp. 4559–4577, 2024.

[15] 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, p. 1197–1203, 2024.

[16] "O-RAN Software Community — o-ran-sc.org." https://o-ran-sc.org/. [Accessed 31.03.2025].

[17] "RIC Message Router 2013; RMR 2014; ric-plt-lib-rmr master documentation — docs.o-ran-sc.org." https://docs.o-ran-sc.org/projects/o-ran-sc-ric-plt-lib-rmr/en/latest/. [Accessed 31-03-2025].

[18] Salim S.I., Richard Y. Lin, "Opening Critical Infrastructure: The Current State of Open RAN Security." https://www.trendmicro.com/en_us/research/23/l/the-current-state-of-open-ran-security.html, Jan. 2023. Accessed: 31.03.2025.

[19] "User-Guide (new) 2014; ric-plt/submgr master documentation — docs.o-ran-sc.org." https://docs.o-ran-sc.org/projects/o-ran-sc-ric-plt-submgr/en/latest/user-guide.html. [Accessed 07-09-2025].

[20] O-RAN Working Group 3 (Near-Real-time RAN Intelligent Controller and E2 Interface Workgroup), "E2 Service Model (E2SM) RAN Control (O-RAN.WG3.E2SM-RC-R003-v03.00)," technical specification, O-RAN ALLIANCE e.V., Mar. 2023.