Assessing Supply Chain Risks in 5G O-RAN Components Using Static Analysis

Himashveta Kumar[1] Tianchang Yang[1] Arupjyoti Bhuyan[2] Syed Rafiul Hussain[1]
[1]*The Pennsylvania State University* [2]*Idaho National Laboratory*
hkk5340@psu.edu, tzy5088@psu.edu, arupjyoti.bhuyan@inl.gov, hussain1@psu.edu

***Abstract*—The emergence of the 5G Open Radio Access Network (O-RAN) architecture introduces increased flexibility and modularity to cellular networks, but its sudden shift toward software-centric and multi-vendor deployments also expands the software supply chain (SSC) attack surface, which is particularly concerning given the critical role of 5G infrastructure. SSC vulnerabilities can lead to severe consequences, including service disruption, unauthorized backdoors, and code injection. In this work, we systematically identify and analyze SSC vulnerabilities in O-RAN RAN Intelligent Controller, which performs latency-sensitive edge control and optimization in 5G networks. Using static analysis tools, we evaluate production-grade O-RAN components primarily implemented in Go and find 57 security-relevant issues after manual validation. We highlight key limitations of off-the-shelf analyzers, quantify false-positive results, and contextualize identified risks within O-RAN deployments. Our findings emphasize the need for improved SSC security practices tailored to O-RAN systems.**

I. Introduction

To meet 5G's demands of high bandwidth, low latency, and massive device connectivity, the Open Radio Access Network (O-RAN) architecture has been proposed [1]. O-RAN introduces software-defined control and open interfaces to increase flexibility and interoperability among radio components, enabling multi-vendor support, reduced costs, and rapid development and deployment cycles. However, this openness and software-centric design also expand the attack surface, exposing O-RAN systems to critical software security threats such as memory safety violations, remote code executions, logical vulnerabilities such as authentication bypass, and service degradation [2]–[4]. In this work, we focus on security risks introduced through the software supply chain (SSC) [5]–[8], which remain under-examined in O-RAN due to complex inter-component interactions, deep dependency chains, and fragmented build and deployment pipelines. SSC vulnerabilities could arise from poor coding practices, developer oversights, or malicious backdoors [5]–[7] and lead to severe consequences [6], [7], including system crashes and remote code execution. As 5G and O-RAN are increasingly adopted to support critical communication systems [9], [10], the need for rigorous supply chain risk management to ensure the security and resilience of 5G systems is further emphasized [11].

Static program analysis enables early identification of vulnerabilities without requiring code execution or reliance on observed runtime behavior. While dynamic analysis is also effective in detecting vulnerabilities in mobile RAN [4], [12]–[14], its reliance on deployment-dependent testbeds, realistic message traces, and complex orchestration makes it less suitable as a lightweight check that can be integrated into daily development pipelines. Conversely, static analysis scales well to large and deeply nested dependencies and is more readily applicable across evolving, multi-vendor O-RAN ecosystems.

In this work, we apply static analysis tools to 12 O-RAN RAN Intelligent Controller (RIC) components, which are central to intelligent edge control of RAN nodes and radio resources, covering both platforming components and xApps. Security issues in the RIC can propagate to other RAN elements, including Distributed Units (O-DUs) and Centralized Units (O-CUs) via the E2 interface, amplifying their potential impact. Most open-source O-RAN implementations are written in Go, which is widely adopted for its memory-safety properties and cloud-native support. However, prior work has shown that Go ecosystems remain susceptible to software supply chain vulnerabilities, particularly through third-party dependencies [15]. Rather than aiming to discover new vulnerabilities through fuzzing [4], [14] or other testing techniques [12], [13] that require extensive harnessing and deployment-specific setup, we focus on identifying supply chain risks arising from known vulnerability patterns and vulnerable dependencies using readily available static checkers. We apply Go-specific static analysis tools [16]–[18] to production-grade O-RAN codebases [19], [20], uncovering 57 valid security-relevant issues alongside 65 false positives that introduce significant noise for operators. Our goal is not to survey static analysis tools, but to provide a manually validated empirical characterization of their precision and failure modes on O-RAN RIC codebases. Based on these findings, we highlight the need for O-RAN-specific context and CI/CD integration to improve the precision and operational usefulness of static supply chain security analysis.

To summarize, we make the following contributions.

- We present a manually validated empirical evaluation of four widely used static analyzers on production-grade O-

Workshop on Security and Privacy of Next-Generation Networks (FutureG) 2026
23 February 2026, San Diego, CA, USA
ISBN 979-8-9919276-9-7
https://dx.doi.org/10.14722/futureg.2026.23070
www.ndss-symposium.org

RAN Near-RT RIC repositories.

- We quantify precision gaps across tools and identify key sources of false positives in real RIC codebases, highlighting practical limitations of generic static analysis.
- Based on these insights, we derive actionable implications for O-RAN CI/CD security testing, arguing that reachability- and repository-structure-aware scanning are key requirements for effective O-RAN hardening.

II. Background and Related Works

O-RAN disaggregates the traditionally monolithic RAN into modular components connected by standardized interfaces (Figure 1), enabling interoperability and rapid feature delivery [1]. RAN Intelligent Controllers (RICs) provide software-defined control and optimization at the network edge through modular applications (xApps/rApps) [21]. This openness expands the software supply chain (SSC), as each component depends on direct and transitive third-party packages and multi-vendor applications, making dependency compromise and vulnerable updates critical security threats [6], [7].

Prior works. Prior work explains why SSC threats are prevalent and hard to detect at scale in various domains [5]–[8], [22]–[28]. Empirical evaluations show static analyzers often have uneven coverage and high false-positive rates, requiring extensive manual validation [29]. In the cellular domain, most prior systems [4], [12]–[14] focus on discovering new vulnerabilities through domain-informed dynamic or specification-aware analysis. In contrast, our work targets software supply chain risk by identifying known vulnerability patterns and dependency-related issues in real RIC codebases. In O-RAN components, Thimmaraju et al. identify security issues from deployment and interface configurations [30]. This work is orthogonal, as we focus on SSC risk introduced through third-party dependencies and multi-vendor components.

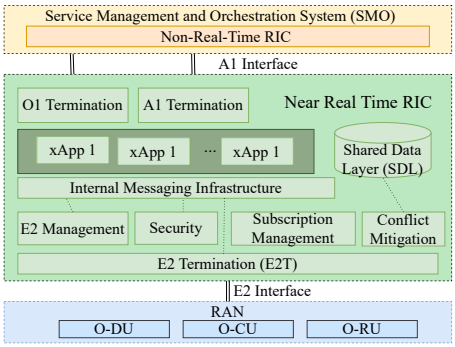

Fig. 1: O-RAN architecture

III. Analysis Overview

A. SSC Analysis Challenges in O-RAN

O-RAN's multi-vendor, modular architecture significantly amplifies software supply chain (SSC) risk. The complex interdependencies and control-plane interactions among components further enable faults or malicious behaviors to propagate to RAN nodes and end users [6], [31]. Detecting SSC issues in O-RAN systems, however, is challenging due to strong context dependence and ecosystem heterogeneity. Malicious or vulnerable behaviors are only exploitable under specific calling paths, configurations, or deployment modes, while static analysis tools that ignore calling context and function semantics often generate excessive false positives (§V-C2). Furthermore, the multi-repository, dependency-heavy RIC ecosystems require manual, context-aware validation to interpret findings in realistic RIC workflows [6], [32].

TABLE I: Static analysis tools evaluated

Tool	Focus	Mechanism	Key Limitation
Staticcheck [18]	correctness/quality	SSA/control-flow	not security-specific
Govulncheck [16]	dependency CVEs	import/reachability	only known CVEs
Gosec [17]	security patterns	AST/rule-based	context-limited; FP-prone
CodeQL [33]	semantic code scanning	queries + taint/-dataflow	build/extraction + context still required

B. Evaluated Static Analysis Tools

We use 4 complementary Go tools to balance SSC coverage and precision (Table I): (i) `govulncheck` [16] for known-vulnerability reporting with call-graph reachability; (ii) `gosec` [17] for security anti-pattern detection via AST/rule-based analysis; (iii) `staticcheck` [18] for correctness and quality issues that can become security-relevant; and (iv) `CodeQL` [33] to identify program anti-patterns, dataflow vulnerabilities, and non-Go issues (e.g., C/C++ calls in Go).

IV. Methodology

A. Target Components and Scope

We choose Go-based RIC services, SDKs, and xApps from two commercially-adopted [34], [35] open-source O-RAN systems: the O-RAN Software Community (OSC) [19] and ONOS SD-RAN [20] (Table IV in Appendix).

B. Tool Execution

For each repository, we (i) resolve Go module dependencies (when possible); (ii) execute the static analyzers using recommended/default configurations; and (iii) record each analyzer's per-component output. Example outputs for each tool are shown in Listings 1, 2, 3 and 4. Repositories that fail dependency resolution, build steps, or result in analysis errors for a given analyzer are discarded. Details on tool execution failures are summarized in Appendix A.

C. Manual Classification

Previous studies show vulnerability scanning produces significant false alerts [29], [30] (e.g., due to the tool's over-approximation). We manually label each alert as:

- **True Positive (TP):** a valid issue with a plausible exploitation path or SSC risk in realistic O-RAN execution.
- **False Positive (FP):** non-exploitable, test-only, or unreachable issues that do not lead to true exploitation.

Additional details on our manual labeling methodology are provided in Appendix B.

V. Results

A. Findings and Results Breakdown

We present a sample output for each evaluated tool in Appendix G. Across the 4 evaluated tools, we collected 136 total findings, as summarized below:

Govulncheck: 39 total (22 TP and 17 FP). Findings primarily correspond to known dependency vulnerabilities with partial reachability evidence, including denial-of-service patterns (e.g., panics) and HTTP/2 resource-exhaustion risks [36], [37]. In several cases, the same advisory appeared in both a reachable (TP) and a non-reachable/test-only (FP) context. Overall, govulncheck had the highest precision among the tools in our dataset.

Staticcheck: 15 total (4 TP and 11 FP). In general, staticcheck flagged mostly those items related to maintainability or correctness issues (e.g., dead/test-only code, ineffective assignments, or minor misuse of the API). A TP label is assigned only to the subset of findings deemed to have operational or SSC impact (i.e., deprecated/insecure APIs or configuration mistakes), whereas the FPs were style/refactoring-oriented issues that were not exploitable.

Gosec: 68 total (31 TP and 37 FP). The dominant FP drivers were benign/test-only paths and missing deployment reachability/context, while TPs mostly reflect TLS/config issues, unsafe file permissions, and input-handling anti-patterns [17], [38].

CodeQL: 14 total (1 TP and 13 FP). Most FPs were test-only, debug-only, or not reachable in production call paths. Several of these FPs match `gosec` findings (notably TLS `InsecureSkipVerify` discussed in Appendix C and integer conversion patterns), indicating that the same recurring anti-patterns drive false alarms across tools. The remaining alert is a request-forgery finding (CWE-918) flagged by CodeQL's `go/request-forgery` check [39].

Most of the TP counts concentrate in control-plane components (e.g., `onos-rsm`), while also appear in policy/control (e.g., `onos-a1t`, `onos-topo`, `onos-ric-sdk-go`) around TLS configuration and file handling [40]–[43]. Results are summarized in Figure 2 (Appendix).

B. Overlap Analysis

Overlap reports between the evaluated tools were limited, as each tool aims at detecting distinct classes of issues. In practice, `govulncheck` provided the highest precision for known, dependency-reachable vulnerabilities, `gosec` provided the broadest coverage but with higher noise (FP alerts), and `staticcheck` helped flag correctness and dead/test-only code that reduced false alarms during manual triage. `CodeQL` corroborated many of the results from `gosec` with some code patterns beyond Go-specific linters (e.g., C/C++ issues), but in general, it produced very low precision and required more manual validation to filter false positives.

C. Case Studies

Below, we present case studies of representative true and false alerts reported by evaluated tools. Additional false positive cases are included in Appendix C.

1) Representative True Positives:

a) Index Panic (DoS) - `onos-pci` [44]: Using govulncheck, we identified an index-out-of-range panic in `asn1.BitString.GetLen()`, which reads `bs.Bytes[0]` without checking length. When malformed ASN.1 inputs induce an empty slice, the handler can panic and crash the `onos-pci` process, enabling denial-of-service. `onos-pci` depends on `onos-lib-go@v0.10.24`, which predates a guard fix; upgrading to a version that includes commit `55579ff` mitigates this issue.

b) gRPC HTTP/2 CONTINUATION Flood (DoS) - `ric-sdk` [43]: `govulncheck` reports that `ric-sdk` depends on `golang.org/x/net@v0.8.0`, vulnerable to HTTP/2 CONTINUATION-flood denial-of-service (`GO-2024-2687` [45], [46]). The risk is relevant for gRPC-facing modules and is mitigated by upgrading to `v0.23.0` or later, which enforces frame limits.

2) Representative False Positives:

a) Weak Crypto Primitive (MD5) Used Non-Cryptographically - `onos-a1t` [41]: `gosec` reports a G401 (CWE-328) warning for MD5 usage in `pkg/rnib/rnib.go`. Manual inspection showed MD5 is used for deterministic identifier generation rather than cryptographic integrity and therefore is not impacted by the weak cryptographic guarantee of MD5.

b) Dependency Present but Not Runtime-Reachable - SD-RAN `sdran-in-a-box` [47]: In SD-RAN's `sdran-in-a-box`, `govulncheck` reports a `containerd` vulnerability because the module appears in the dependency graph. Manual inspection indicates the references occur only in deployment/test helpers (e.g., Helm/Kubernetes orchestration) without reaching the vulnerable `containerd` code.

c) Error String Capitalization (ST1005) - `onos-topo` [42]: In `onos-topo`, `staticcheck` reports an ST1005 warning in `pkg/northbound/service.go:221`, indicating that an error string begins with a capital letter, which violates Go's error formatting convention. This is just a style-only issue that does not affect control flow, security properties, or SSC exploitability. While it is not a security issue, it introduces unnecessary noise and could flood the security alerts.

VI. Discussion

A. Key Observations

We summarize the precision evaluation across tools in Table II. `govulncheck` is the most precise tool in our setting because it ties known vulnerabilities to dependency usage with (partial) reachability evidence [36], [37]. The most critical recurring issues were DoS risks, e.g., ASN.1 parsing panics in `onos-lib-go`, and HTTP/2 resource-exhaustion vulnerabilities affecting gRPC-heavy modules (e.g., `ric-sdk`, `onos-uenib`) [36], [43], [48]. However, deployment/test scaffolding still triggered reports, requiring manual context checks to separate operational risk from non-reachable code [47], [49], [50]. We summarize dominant drivers of

false positives in Table III in Appendix. staticcheck primarily surfaces correctness and maintainability issues (e.g., unused code, style) that we consider FP, as it is not adversarially exploitable. However, staticcheck identifies unreachable/test-only code regions, reducing noise compared to other tools. Only a smaller subset of reported issues was actionable (e.g., deprecated io/ioutil usage, missing configuration fields) [18], [38]. gosec provides the broadest coverage but also the highest noise [17], [29]. Many integer-conversion and weak-crypto reports are FP after context inspection, while the most relevant TPs involve insecure TLS settings, path/permission handling, unchecked unmarshalling/error handling, and injection-style risks that can enable tampering or MiTM-style threats under externally influenced inputs [17], [38], [51].

Across the repositories where CodeQL executed successfully, we found 14 alerts (12 labeled by CodeQL as *High*, 2 *Critical*). The only TP represents a request forgery vulnerability (CWE-918) and is labeled *Critical*, and the other *Critical* an FP, arising from debug-only/local-file logic (CWE-134). Most CodeQL FPs overlapped with recurring gosec anti-pattern classes (e.g., TLS InsecureSkipVerify and integer-conversion patterns), demonstrating that severity alone does not reflect SSC exploitability. We discuss additional precision trade-off insights in Appendix D.

TABLE II: Precision comparison of evaluated tools

Tool	Total Findings	TP	FP	Precision (%)
gosec	68	31	37	45.59%
govulncheck	39	22	17	56.41%
staticcheck	15	4	11	26.67%
codeql	14	1	13	7.14%

B. Threats to Validity

Manual labeling may introduce subjectivity despite the use of consistent, well-defined criteria, particularly for context-dependent cases [29], [38]. We mitigate this risk by having two domain experts independently label the data and cross-validate results to ensure consistency. Tool coverage is further constrained by build-process and environment assumptions, such as repositories that fail go mod resolution or require deployment-specific scaffolding [38]. Finally, our study is limited to a representative set of O-RAN repositories and static analysis tools, and broader survey across additional components, ecosystems, programming languages, and dynamic validation techniques could strengthen generalizability [29].

C. Recommendations & Future Work

No single static analyzer is sufficient for securing O-RAN RIC codebases. Instead, the evaluated tools exhibit complementary coverage, and in practice, combining them with lightweight manual triage provides the best overall results. However, this approach also produces substantial noise from false positive reports (Table II). In addition, existing tools rely primarily on template-based detection and therefore focus on known vulnerability patterns, which limits their ability to identify trigger-based behaviors such as logic bombs or backdoors that activate only under specific runtime conditions in cross-service workflows [32], [52].

a) Reachability- and O-RAN-Aware Contextual Triage: Our analysis indicates that false positives can be significantly reduced by incorporating deployment reachability together with O-RAN–specific context. Future work can explore automating this process by mapping reported findings to realistic Near-RT RIC entry points, including A1/E2 handlers and service endpoints, while explicitly filtering code that is only exercised during testing, benchmarking, or CI pipeline stages. Such domain-informed triage can leverage repository structure, build artifacts, and pipeline metadata to distinguish test-only scaffolding from production-relevant components. We further observe that multiple tools often report the same recurring anti-patterns, such as TLS InsecureSkipVerify or integer conversion warnings, many of which are unreachable in deployed services. Leveraging tool overlap to de-duplicate such findings and prioritizing only production-reachable alerts can substantially improve the operational usefulness of static analysis in O-RAN CI/CD pipelines.

b) O-RAN-Specific Trigger Modeling: Beyond CVE matching and syntactic anti-patterns, future analyses should account for O-RAN-specific triggers that influence control-plane behavior. One promising direction is to treat decoded E2AP/E2SM message fields as structured sources and security-relevant control actions as sinks. This enables targeted checks for missing validation on protocol fields, unsafe use of externally influenced values in resource-amplifying loops, and configuration changes that affect security-sensitive settings, improving both coverage and precision [52].

D. Responsible Disclosure

We followed standard responsible disclosure practices and reported all 50 code-level issues to the corresponding project maintainers. Two reported issues have already been confirmed by maintainers, and one has been patched, mitigating downstream impact in components including onos-config and onos-topo. The remaining reports are pending responses.

VII. Conclusion

Our empirical analysis of O-RAN Near-RT RIC components using off-the-shelf static analyzers shows that, while existing tools exhibit complementary strengths, notable precision gaps remain. These findings highlight the need for more context-aware and domain-informed static analysis. Improving such tooling enables developers to better identify and mitigate software supply chain risks at build time, reducing exposure in O-RAN deployments.

Acknowledgements

This work is supported by the Public Wireless Supply Chain Innovation Fund (PWSCIF) under Federal Award ID Number 51-60-IF007.

REFERENCES

[1] O-RAN Alliance, "O-ran: Towards open and intelligent radio access networks." [Online]. Available: https://www.o-ran.org/

[2] M. Liyanage, A. Braeken, S. Shahabuddin, and P. Ranaweera, "Open ran security: Challenges and opportunities," 2022. [Online]. Available: https://arxiv.org/abs/2212.01510

[3] W. Azariah, F. A. Bimo, C.-W. Lin, R.-G. Cheng, N. Nikaein, and R. Jana, "A survey on open radio access networks: Challenges, research directions, and open source approaches," *Sensors*, vol. 24, no. 3, p. 1038, Feb. 2024. [Online]. Available: http://dx.doi.org/10.3390/s24031038

[4] T. Yang, S. M. M. Rashid, A. Ranjbar, G. Tan, and S. R. Hussain, "ORANalyst: Systematic testing framework for open RAN implementations," in *33rd USENIX Security Symposium (USENIX Security 24)*. Philadelphia, PA: USENIX Association, 2024, pp. 1921–1938. [Online]. Available: https://www.usenix.org/conference/usenixsecurity24/presentation/yang-tianchang

[5] S. Ladisa and K. Rieck, "Sok: Taxonomy of attacks on open-source software supply chains," in *Proceedings of the 44th IEEE Symposium on Security and Privacy (S&P)*. IEEE, 2023.

[6] R. Duan, O. Alrawi, R. P. Kasturi, R. Elder, B. Saltaformaggio, and W. Lee, "Towards measuring supply chain attacks on package managers for interpreted languages," in *Network and Distributed System Security (NDSS) Symposium*, 2021. [Online]. Available: https://www.ndss-symposium.org/wp-content/uploads/ndss2021_1B-1_23055_paper.pdf

[7] M. Ohm, H. Plate, A. Sykosch, and M. Meier, "On the feasibility of detecting software supply chain attacks," in *IEEE MILCOM*. IEEE, 2020, pp. 476–481.

[8] M. Fourné, Y. Acar, and S. Fahl, "It's like flossing your teeth: On the importance and challenges of reproducible builds for software supply chain security," in *Proceedings of the 44th IEEE Symposium on Security and Privacy (S&P)*. IEEE, 2023.

[9] U.S. Department of Defense, "Dod awards project to develop open radio access network prototype at fort bliss," May 2024. [Online]. Available: https://tinyurl.com/oran-proto

[10] "DeepSig and SRS Chosen by DOD Future G Office to." [Online]. Available: https://www.deepsig.ai/deepsig-and-srs-chosen-by-dod-futureg-office-to-lead-ocudu-the-open-source-5g-6g-ran-initiative/

[11] U.S. Department of Defense, "Private 5g deployment strategy," https://tinyurl.com/dod-5g-strategy, 2024, accessed: 2025-05-29.

[12] E. Kim, M. W. Baek, C. Park, D. Kim, Y. Kim, and I. Yun, "Basecomp: A comparative analysis for integrity protection in cellular baseband software," in *32nd USENIX Security Symposium (USENIX Security 23)*, 2023, pp. 3547–3562, august 9–11, 2023, Anaheim, CA, USA. [Online]. Available: https://www.usenix.org/system/files/usenixsecurity23-kim-eunsoo.pdf

[13] E. Kim, D. Kim, C. Park, I. Yun, and Y. Kim, "Basespec: Comparative analysis of baseband software and cellular specifications for l3 protocols," in *Network and Distributed System Security (NDSS) Symposium 2021*, 2021, 21–25 February 2021, Virtual. [Online]. Available: https://www.ndss-symposium.org/wp-content/uploads/ndss2021_6B-4_24365_paper.pdf

[14] N. Bennett, W. Zhu, B. Simon, R. Kennedy, W. Enck, P. Traynor, and K. R. B. Butler, "Ransacked: A domain-informed approach for fuzzing lte and 5g ran-core interfaces," in *Proceedings of the 2024 ACM SIGSAC Conference on Computer and Communications Security (CCS '24)*, 2024, october 14–18, 2024, Salt Lake City, UT, USA. [Online]. Available: https://nathanielbennett.com/publications/ransacked.pdf

[15] J. Hu, L. Zhang, C. Liu, S. Yang, S. Huang, and Y. Liu, "Empirical analysis of vulnerabilities life cycle in golang ecosystem," in *Proceedings of the 46th International Conference on Software Engineering (ICSE '24)*. New York, NY, USA: Association for Computing Machinery, 2024. [Online]. Available: https://doi.org/10.1145/3597503.3639230

[16] G. S. Team, "govulncheck: Go vulnerability checker," Go tool documentation, 2023, https://pkg.go.dev/golang.org/x/vuln/cmd/govulncheck.

[17] S. Team, "gosec: Golang security checker," GitHub repository, 2023, https://github.com/securego/gosec.

[18] D. Honnef, "staticcheck: Advanced linter for go," Tool documentation, 2023, https://staticcheck.io/.

[19] "O-RAN Software Community," https://o-ran-sc.org/.

[20] "SD-RAN," https://opennetworking.org/open-ran/.

[21] O-RAN Alliance, "Near-real-time ran intelligent controller (ric) architecture," 2021. [Online]. Available: https://www.o-ran.org/specifications

[22] A. Sejfia and M. Schäfer, "Practical automated detection of malicious npm packages," in *Proceedings of the 44th International Conference on Software Engineering (ICSE '22)*. ACM, 2022, pp. 1681–1692. [Online]. Available: https://dl.acm.org/doi/10.1145/3510003.3510104

[23] J.-U. Holtgrave, K. Friedrich, F. Fischer, N. Huaman, N. Busch, J. H. Klemmer, M. Fourné, O. Wiese, D. Wermke, and S. Fahl, "Attributing open-source contributions is critical but difficult: A systematic analysis of github practices and their impact on software supply chain security," in *Network and Distributed System Security Symposium (NDSS)*. Internet Society, Feb. 2025. [Online]. Available: https://www.ndss-symposium.org/wp-content/uploads/2025-613-paper.pdf

[24] A. S. A. Yelgundhalli, P. Zielinski, R. Curtmola, and J. Cappos, "Rethinking trust in forge-based git security," in *Network and Distributed System Security Symposium (NDSS)*. Internet Society, Feb. 2025. [Online]. Available: https://www.ndss-symposium.org/wp-content/uploads/2025-1008-paper.pdf

[25] M. Moore, A. S. A. Yelgundhalli, and J. Cappos, "Securing automotive software supply chains," in *Symposium on Vehicle Security and Privacy (VehicleSec)*. Internet Society, Feb. 2024. [Online]. Available: https://www.ndss-symposium.org/wp-content/uploads/vehiclesec2024_23015_paper.pdf

[26] F. Reyes, F. Bono, A. Sharma, B. Baudry, and M. Monperrus, "Maven-hijack: Software supply chain attack exploiting packaging order," in *Proceedings of the ACM Workshop on Software Supply Chain Offensive Research and Ecosystem Defenses (SCORED)*, 2025. [Online]. Available: https://arxiv.org/abs/2407.18760

[27] X. Gao, X. Sun, S. Cao, K. Huang, D. Wu, X. Liu, X. Lin, and Y. Xiang, "Malguard: Towards real-time, accurate, and actionable detection of malicious packages in pypi ecosystem," in *34th USENIX Security Symposium (USENIX Security 25)*. USENIX Association, Aug. 2025. [Online]. Available: https://www.usenix.org/system/files/usenixsecurity25-gao-xingan.pdf

[28] A. Aijaz, S. Gufran, T. Farnham, S. Chintalapati, A. Sánchez-Mompó, and P. Li, "Open ran for 5g supply chain diversification: The beacon-5g approach and key achievements," in *2023 IEEE Conference on Standards for Communications and Networking (CSCN)*. IEEE, 2023, pp. 1–7. [Online]. Available: https://arxiv.org/abs/2310.03580

[29] A. Arusoaie, G. Ciobanu, and G. Rosu, "An empirical study on the effectiveness of static c/c++ analyzers for vulnerability detection," in *Proceedings of the 31st ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA)*. ACM, 2022, pp. 318–329.

[30] K. Thimmaraju, R. Kashyap, A. Shaik, S. Flück, P. J. Fullana Mora, C. Werling, and J.-P. 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, pp. 277–287.

[31] M. Zimmermann, C.-A. Staicu, C. Tenny, and M. Pradel, "Small world with high risks: A study of security threats in the npm ecosystem," in *28th USENIX Security Symposium (USENIX Security 19)*. Santa Clara, CA: USENIX Association, 2019, pp. 995–1010. [Online]. Available: https://www.usenix.org/conference/usenixsecurity19/presentation/zimmerman

[32] S. Torres-Arias, H. Afzali, T. K. Kuppusamy, R. Curtmola, and J. Cappos, "in-toto: Providing farm-to-table guarantees for bits and bytes," in *28th USENIX Security Symposium (USENIX Security 19)*. Santa Clara, CA: USENIX Association, 2019, pp. 1393–1410. [Online]. Available: https://www.usenix.org/conference/usenixsecurity19/presentation/torres-arias

[33] "Codeql query suites," GitHub Docs, accessed 2026-01-04.

[34] "Intelligent Private 5G Solution Based on Near-RT RIC," https://stage-o-ran-v2.azurewebsites.net/classic/generation/2023/category/intelligent-ran-control-demonstrations/sub/intelligent-control/251.

[35] "ONF and Deutsche Telekom demonstrate fully disaggregated Open RAN," https://opennetworking.org/news-and-events/press-releases/onf-and-deutsche-telekom-demonstrate-fully-disaggregated-open-ran-with-open-ric-platform/.

[36] Go Authors, "golang.org/x/vuln/cmd/govulncheck (package documentation)," pkg.go.dev, 2025, accessed: 2025-12-30. [Online]. Available: https://pkg.go.dev/golang.org/x/vuln/cmd/govulncheck

[37] The Go Team, "Govulncheck v1.0.0 is released!" Go Blog, 2023, accessed: 2025-12-30. [Online]. Available: https://go.dev/blog/govulncheck

[38] B. Johnson, Y. Song, E. Murphy-Hill, and R. Bowdidge, "Why don't software developers use static analysis tools to find bugs?" in *Pro-*

ceedings of the 35th International Conference on Software Engineering (ICSE). IEEE, 2013, pp. 672–681.
[39] "Uncontrolled data used in network request (go request forgery)," GitHub CodeQL Query Help, accessed 2026-01-04.
[40] O. Project, "Ran slice management xapp of onos," https://github.com/onosproject/onos-rsm, accessed: 2025-05-17.
[41] ——, "A1 termination of onos," https://github.com/onosproject/onos-a1t, accessed: 2025-05-17.
[42] ——, "Ric topology service of onos," https://github.com/onosproject/onos-topo, accessed: 2025-05-17.
[43] ——, "Onos ric sdk," https://github.com/onosproject/onos-ric-sdk-go, accessed: 2025-05-17.
[44] ——, "Pci xapp of onos," https://github.com/onosproject/onos-pci, accessed: 2025-05-17.
[45] "Vulnerability report: Go-2024-2687," https://pkg.go.dev/vuln/GO-2024-2687, Apr. 2024, go Vulnerability Database. Accessed: 2026-01-06.
[46] "net/http, x/net/http2: close connections when receiving too many headers (cve-2023-45288) (issue #65051)," https://github.com/golang/go/issues/65051, Jan. 2024, golang/go issue tracker. Accessed: 2026-01-06.
[47] ONOS Project, "sdran-helm-charts: Helm charts for sd-ran," GitHub repository, 2025, accessed: 2025-12-30. [Online]. Available: https://github.com/onosproject/sdran-helm-charts
[48] O. Project, "Ue information base of onos," https://github.com/onosproject/onos-uenib, accessed: 2025-05-17.
[49] The Go Team, "Tutorial: Find and fix vulnerable dependencies with govulncheck," Go Documentation, 2025, accessed: 2025-12-30. [Online]. Available: https://go.dev/doc/tutorial/govulncheck
[50] ONOS Project, "sdran-in-a-box (riab): Sd-ran development/test environment over kubernetes," GitHub repository, 2025, accessed: 2025-12-30. [Online]. Available: https://github.com/onosproject/sdran-in-a-box
[51] J. Lauinger, L. Baumgärtner, A.-K. Wickert, and M. Mezini, "Uncovering the hidden dangers: Finding unsafe go code in the wild," 2020. [Online]. Available: https://arxiv.org/abs/2010.11242
[52] Y. Fratantonio, A. Bianchi, W. Robertson, E. Kirda, C. Kruegel, and G. Vigna, "Triggerscope: Towards detecting logic bombs in android applications," in *2016 IEEE Symposium on Security and Privacy (S&P)*, 2016, pp. 377–396. [Online]. Available: https://sites.cs.ucsb.edu/~vigna/publications/2016_SP_Triggerscope.pdf
[53] SPDX Workgroup (Linux Foundation), "SPDX: The system package data exchange," https://spdx.dev/, accessed: 2025-12-30.
[54] CycloneDX Project (OWASP), "Cyclonedx specification overview," https://cyclonedx.org/specification/overview/, accessed: 2025-12-30.

APPENDIX

A. Tool Execution Failures

Some repositories did not produce analyzable output for specific tools due to build-time constraints rather than tool crashes. Common causes include (i) Go module resolution failures involving internal O-RAN dependencies (e.g., Gerrit-hosted modules), (ii) repositories that require deployment-only scaffolding or generated code to type-check, and (iii) CGO dependencies on external system headers/libraries unavailable in our environment. For example, `ric-app-kpimon-go` requires the F1AP wrapper headers via CGO and fails to compile without them, preventing tools that rely on successful compilation/type-checking from running. We therefore report each tool's results over the subset $S_{\texttt{tool}}$ where that tool executed successfully.

B. Labelling methodology and insights

Methodology of identifying the findings and what we learned: We took a repeatable triage approach that assigned a higher priority to the important factors of whether a finding can be exploited and the relevance to SSC. The initial step for every alert was to confirm precisely where in the code

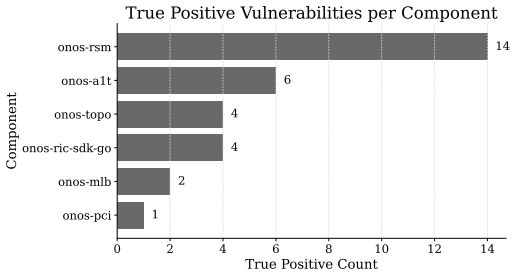

Fig. 2: True positive vulnerability detections per component

(i.e., within a file/line) the report referred to as it relates to the semantics of the rule, and then to follow the call path through to the calling function to determine if the code could actually be called from a production environment (e.g., via server initialization, using gRPC handler, etc.) as opposed to being called from test related tools or examples, CI scripts or deployment manifests. In the case of configuration-driven issues (e.g., TLS settings), we then inspected what the actual values were set to (i.e., default, flags, environment variables or Helm charts). When labelling the results of our effort, we labelled TP (True Positive) when there was a reasonable likelihood of exploiting the finding or the potential that it could have an adverse effect within normal O-RAN operation; in contrast, we labelled a finding as FP (False Positive) if we could definitively demonstrate that the finding was unreachable, only associated with test cases, relied on dead code or was stylistic in nature. When the results of our work contained ambiguity, we conservatively recorded sufficient descriptive information based on what we had used to assess against our criteria for reachability and impact and therefore we can revisit these items and provide them a different label as the context surrounding deployments evolves.

C. Additional False-Positive Examples

a) TLS Verification Disabled (`InsecureSkipVerify`) in `onos-topo`: `gosec` reports a G402 (CWE-295) warning for `TLS InsecureSkipVerify` being set to `true` in `onos-topo/cmd/topo-scale/topo-scale.go:85`. While disabling certificate verification can enable server impersonation and traffic interception if used in production, manual inspection indicates this setting is confined to the `topo-scale` benchmarking/scale harness rather than the deployed `onos-topo` service path; we therefore classify it as FP. [17], [42]

D. Other Result Insights

Our results show a coverage–precision tradeoff across static tools. `gosec` produces the most TPs (31) but also substantial noise (37 FP), `staticcheck` yields fewer security-relevant TPs (4) with more FPs (11), and `govulncheck` achieves the best balance (22 TP vs. 17 FP) via known-vulnerability reporting with reachability evidence [17], [18], [36]–[38].

E. False positive drivers.

As summarized in Table III, across the evaluated tools, most false positives stem from structural mismatches between what analyzers scan and what is actually shipped/exercised in Near-RT RIC deployments. First, a frequent occurrence of false alerts appears exclusively in *test, benchmark, or local harness code* (e.g., CodeQL/`gosec` reports of TLS `InsecureSkipVerify` that were confined to benchmarking clients or unit-test scaffolding), which inflates severity despite being non-production. Second, many findings are *non-deployment-reachable*: dependency advisories or pattern matches occur in optional modules, build helpers, or orchestration logic that is not reachable from realistic service entry points (e.g., `govulncheck` flagging a `containerd` advisory in SD-RAN `sdran-in-a-box`, where the vulnerable code path is referenced exclusively in deployment/test helpers and not exercised at runtime). Third, *build-tag and configuration-specific paths* can cause tools to analyze code that is excluded under release build constraints, yielding alerts that do not apply to the deployed artifact. Fourth, tools often over-flag *benign uses of flagged primitives* when threat context is missing (e.g., `gosec` G401 for MD5 used for deterministic identifiers rather than cryptographic integrity). Finally, a smaller but important source of noise is *missing O-RAN semantics*: generic analyzers lack protocol context to map sources/sinks to E2/A1-triggered behaviors, so they may both over-approximate exploitability in generic input-handling patterns and under-capture protocol-specific constraints. These drivers collectively explain why reachability evidence and repository-/deployment-context signals are necessary to make static scanning actionable in O-RAN CI/CD.

F. CI/CD Integration to Reduce Manual Effort

Our current evaluation required substantial manual validation because generic tool outputs do not encode deployment reachability or artifact provenance [38]. A practical next step is to integrate security check pipelines into CI/CD so that each repository build produces (i) scanner reports (`govulncheck`, `gosec`, `staticcheck`, and CodeQL where applicable), (ii) SBOM/provenance artifacts, and (iii) an O-RAN-specific filter that suppresses test-only findings and flags regressions when reachable issues appear [32], [53], [54].

G. Sample Outputs of Evaluated Tools

The listings below are examples of the raw tool output that was utilised during the manual triage process. The raw output remains unchanged, but due to its size has been truncated. The original wordings and fields are preserved with the exception of any repetitive areas (i.e. long trace lists or repeated style warnings). These examples illustrate the reporting level differences for each tool—`govulncheck` provides fixed-version reference information, `gosec/staticcheck` highlights patterns at the code-level, while `CodeQL`reports on dataflow-taints of information. CodeQL did not report any issues for `ric-plt-a1`, so another example for `ric-app-kpimon-go` is given.

```
:~/ric-plt-a1$ govulncheck ./...
=== Symbol Results ===

Vulnerability #1: GO-2025-3447
    Timing sidechannel for P-256 on ppc64le in
    ↪ crypto/internal/nistec
  More info: https://pkg.go.dev/vuln/GO-2025-3447
  Standard library
    Found in: crypto/internal/nistec@go1.22
    Fixed in: crypto/internal/nistec@go1.22.12
    Platforms: ppc64le

...


Vulnerability #13: GO-2022-0322
    Uncontrolled resource consumption in
    ↪ github.com/prometheus/client_golang
  More info: https://pkg.go.dev/vuln/GO-2022-0322
  Module: github.com/prometheus/client_golang
    Found in: github.com/prometheus/client_golang@v0.9.3
    Fixed in: github.com/prometheus/client_golang@v1.11.1

Your code is affected by 13 vulnerabilities from 1 module
    ↪ and the Go standard library.
This scan also found 5 vulnerabilities in packages you
    ↪ import and 14
vulnerabilities in modules you require, but your code
    ↪ doesn't appear to call
these vulnerabilities.
```

Listing 1: `govulncheck` output for `ric-plt-a1` (A1 mediator).

```
Results:

[ric-plt-a1/pkg/rmr/rmr.go:156] - G104 (CWE-703): Errors
    ↪ unhandled (Confidence: HIGH, Severity: LOW)
    json.Unmarshal([]byte(payload), &result)

...

[ric-plt-a1/pkg/resthooks/resthooks.go:742] - G104
    ↪ (CWE-703): Errors unhandled (Confidence: HIGH,
    ↪ Severity: LOW)
    rh.deleteMetadata(policyTypeId, policyInstanceID)

Summary:
  Gosec  : dev
  Files  : 65
  Lines  : 9281
  Nosec  : 0
  Issues : 6
```

Listing 2: `Gosec` output for `ric-plt-a1` (A1 mediator).

```
staticcheck ./...

-: # gerrit.o-ran-sc.org/r/ric-plt/a1/config
    ↪ [gerrit.o-ran-sc.org/r/ric-plt/a1/config.test]
config/configuration_test.go:31:34: config.Logging
    ↪ undefined (type *Configuration has no field or
    ↪ method Logging)
config/configuration_test.go:33:32: config.Rmr undefined
    ↪ (type *Configuration has no field or method Rmr)
    ↪ (compile)

pkg/policy/policyManager.go:38:5: error var
    ↪ policyTypeNotFoundError should have name of the
    ↪ form errFoo (ST1012)
pkg/policy/policyManager.go:38:31: error strings should not
    ↪ be capitalized (ST1005)
pkg/policy/policyManager.go:39:5: error var
    ↪ policyInstanceNotFoundError should have name of the
    ↪ form errFoo (ST1012)
pkg/policy/policyManager.go:39:35: error strings should not
    ↪ be capitalized (ST1005)

pkg/restful/restful.go:59:6: should omit comparison to bool
    ↪ constant, can be simplified to !resp (S1002)
```

```
pkg/resthooks/resthooks.go:54:5: error var typeAlreadyError
     ↪ should have name of the form errFoo (ST1012)
pkg/resthooks/resthooks.go:54:24: error strings should not
     ↪ be capitalized (ST1005)
pkg/resthooks/resthooks.go:55:5: error var
     ↪ InstanceAlreadyError should have name of the form
     ↪ ErrFoo (ST1012)
pkg/resthooks/resthooks.go:55:28: error strings should not
     ↪ be capitalized (ST1005)
pkg/resthooks/resthooks.go:56:5: error var
     ↪ typeMismatchError should have name of the form
     ↪ errFoo (ST1012)

...

pkg/resthooks/resthooks.go:59:31: error strings should not
     ↪ be capitalized (ST1005)
pkg/resthooks/resthooks.go:395:2: this value of err is
     ↪ never used (SA4006)

pkg/resthooks/types.go:41:6: type iRMRClient is unused
     ↪ (U1000)
```

Listing 3: `Staticcheck` output for `ric-plt-a1` (A1 mediator).

```
CodeQL results:

[e2ap/wrapper.c:662] - cpp/tainted-format-string (CWE-134):
     ↪ Uncontrolled format string
    char* text = (char*)calloc(numbytes, sizeof(char));
    fread(text, sizeof(char), numbytes, fp3);
    fclose(fp3);
    printf(text);

Description:
  The value of this argument may come from string read by
     ↪ fread and is being used as a formatting argument to
     ↪ printf(__format).
  The program uses input from the user as a format string
     ↪ for printf style functions; this can crash the
     ↪ program, disclose information, or enable code
     ↪ execution.

Recommendation:
  Use a constant format string, e.g., printf("%s", text).
```

Listing 4: CodeQL output for `ric-app-kpimon-go`: uncontrolled format string (CWE-134).

TABLE III: Dominant drivers of false positives, ranked by frequency.

*Each FP is only assigned a single dominant driver.

FP Driver	**#FP**	**Representative examples**
Missing semantic or threat-model context (benign primitive use)	38	`gosec` flags MD5 usage that does not affect cryptographic integrity (e.g., deterministic identifiers), bounded integer conversions, or patterns that can only lead to benign startup failures.
Test vs. production code separation	25	TLS `InsecureSkipVerify` and similar anti-patterns confined to unit tests, benchmarks, or scale/debug utilities (e.g., `test/` directories or `*-scale` tooling).
Tool scope mismatch (style or correctness warnings)	12	`staticcheck` style or correctness findings (e.g., ST1005 error-string capitalization) that do not lead to exploitability.
Lack of deployment reachability	4	Advisories or patterns reported through transitive dependencies that are not reachable from deployed service entry points (e.g., unused HTTP/2 or container tooling paths).

TABLE IV: O-RAN components and corresponding dependencies

Project	Summary	Dependencies
ric-plt-a1	A1 interface for policy control between Non-RT and Near-RT RIC.	onos-api, onos-lib-go, viper, Go OpenAPI
ric-plt-xapp-frame	SDK for rapid xApp development on Near-RT RIC.	onos-lib-go, golog, REST, Prometheus, stdlib
ric-app-hw-go	Basic xApp with A1 interface, config, and health checks.	ric-plt-xapp-frame, internal O-RAN SC libs
ric-app-kpimon-go	xApp for collecting RAN KPIs via E2.	ric-plt-xapp-frame, InfluxDB, internal libs
onos-e2t	Manages E2 interface to connect RIC and RAN nodes.	onos-api, onos-lib-go, Atomix, gRPC, protobuf
onos-a1t	A1 interface for injecting policies into the RIC.	onos-api, onos-lib-go, ONOS microservices
onos-topo	Shared topology service for RAN and xApps.	onos-api, onos-lib-go, Atomix
onos-uenib	Tracks UE state for mobility and performance use.	onos-api, onos-lib-go, Atomix
onos-ric-sdk-go	SDK for xApp APIs; abstracts RIC's gRPC internals.	onos-api, onos-lib-go, gRPC, protobuf
onos-pci	xApp for PCI management and conflict resolution.	onos-ric-sdk-go, onos-api, onos-lib-go, OpenAPI
onos-mlb	xApp for cell load balancing via handover tuning.	onos-ric-sdk-go, onos-api, onos-lib-go, Prometheus
onos-mho	xApp for handover decisions using UE and RAN data.	onos-ric-sdk-go, onos-api, onos-lib-go
onos-rsm	xApp for managing RAN slicing and UE assignments.	onos-ric-sdk-go, onos-api, onos-lib-go, REST

TABLE V: CodeQL alerts and manual FP labeling (cross-check against Go tools).

Component	Description	File/Line	Sev.	Comparison	FP?
onos-a1t	Incorrect conversion between integer	pkg/manager/manager.go:124	High	GoSec Match	YES
onos-topo	Disabled TLS certificate check (test)	test/utils/topo.go:25	High	NO	YES
onos-topo	Disabled TLS certificate check	cmd/topo-scale/topo-scale.go:85	High	GoSec Match	YES
onos-ric-sdk-go	Disabled TLS certificate check	pkg/utils/creds/creds.go:22	High	GoSec Match	YES
onos-rsm	Incorrect conversion between integer types (test)	pkg/slicing/manager.go:205	High	GoSec Match	YES
onos-rsm	Incorrect conversion between integer types (test)	pkg/slicing/manager.go:106	High	GoSec Match	YES
onos-rsm	Incorrect conversion between integer types (test)	pkg/nib/uenib/uenib.go:321	High	GoSec Match	YES
onos-rsm	Disabled TLS certificate check (test)	test/utils/cli_client.go:29	High	GoSec Match	YES
ric-plt-rtmgr	Incorrect conversion between integer types	pkg/sbi/sbi.go:162	High	NO	YES
ric-plt-rtmgr	Incorrect conversion between integer types	pkg/nbi/httprestful.go:702	High	NO	YES
ric-plt-submgr	Incorrect conversion between integer types	pkg/control/debug_rest_if.go:42	High	NO	YES
ric-plt-xapp-frame	Zip Slip (not reachable in production code)	pkg/xapp/Utils.go:166	High	NO	YES
ric-app-kpimon-go	Uncontrolled format string (CWE-134; debug-only local file input)	e2ap/wrapper.c:662	Critical	NO	YES
ric-app-kpimon-go	Uncontrolled data used in network request (CWE-918; constrained SSRF/internal request forgery)	f1apHelper/f1apServer.go:76	Critical	NO	NO