Towards LLM-Assisted Vulnerability Detection and Repair for Open-Source 5G UE Implementations

Rupam Patir[†*], Qiqing Huang[†*], Keyan Guo[†], Wanda Guo[‡], Guofei Gu[‡], Haipeng Cai[†], Hongxin Hu[†]
[†]University at Buffalo,
[‡]Texas A&M University,
Email: [†]{rupampat, qiqinghu, keyanguo, haipengc, hongxinh}@buffalo.edu
[‡]{wdguo, guofei}@tamu.edu

***Abstract*—The rapid evolution of software systems in 5G networks has heightened the need for robust security measures. Traditional code analysis methods often fail to detect vulnerabilities specific to 5G, particularly vulnerabilities stemming from complex protocol interactions. In this work, we explore the potential of LLM-assisted techniques in vulnerability detection and repair in open-source 5G implementations. We introduce a novel framework leveraging Chain-of-Thought (CoT) prompting in two phases: *first*, vulnerability detection based on 5G Vulnerability Properties (VPs); *second*, vulnerability repair guided by 5G Secure Coding Practices (SCPs). We conducted a case study on an open-source 5G User Equipment (UE) implementation that illustrates how our framework leverages vulnerability properties and SCPs to identify and remediate vulnerabilities. Our testing results indicate successful detection and repair, demonstrating the practicality and effectiveness of our approach. While challenges persist, including the identification of 5G-specific security properties and SCPs and the complexity of their integration, this study provides a foundation for advancing automated LLM-assisted solutions to strengthen the security of open-source 5G systems.**

I. Introduciton

Fifth Generation (5G) networks have emerged as a cornerstone of modern communication, promising ultra-high bandwidth, low latency, and massive connectivity for diverse application scenarios [1]. However, the increased complexity of 5G systems—spanning different layers of protocols and incorporating real-time, heterogeneous components—introduces new security challenges that traditional software security methodologies [2]–[4] fail to address.

Ensuring the security and reliability of 5G software components, particularly in User Equipment (UE), demands advanced, context-aware analysis techniques. In general, 5G vulnerabilities can be divided into design flaws and implementation flaws [5]. On the design side, formal analyses have revealed a wide range of LTE/5G design weaknesses [6]–[9], yet these findings do not necessarily translate into secure implementations. For implementation flaws, most existing work relies on black-box approaches—such as fuzzing, over-the-air probing, and negative testing—to identify 5G implementation vulnerabilities [5], [10]–[22]. Although these techniques can effectively uncover functional anomalies, they rarely leverage open-source 5G UE projects (e.g., Open5GS [23], srsRAN [24], and OpenAirInterface (OAI) [25]) for comprehensive, code-level secuirty. Meanwhile, pattern-based code analysis tools (e.g., CodeQL [4]) can detect common security risks—such as buffer overflows and SQL injections—but often struggle to pinpoint 5G-specific or semantic-level flaws, leading to false negatives [26], [27]. Furthermore, the complex control flows inherent in open-source 5G UE software can inflate the number of false positives [28], and manual patching of these issues remains labor-intensive and prone to human error [26], [27], [29]. As a result, delivering secure, large-scale open-source 5G UE implementations requires more robust, context-aware analysis methods.

Recent advancements in Large Language Models (LLMs) have demonstrated strong reasoning capabilities [30] and a deep understanding of 3GPP standards and 5G protocols [31]–[33]. Additionally, LLMs have shown remarkable performance in vulnerability detection and automated patch generation for traditional software [34]–[39]. However, their application in identifying and repairing vulnerabilities in open-source 5G implementations remains largely unexplored. To bridge this gap, we investigate the following key research questions: *(1) How can LLMs be effectively leveraged to detect complex vulnerabilities in open-source 5G UE implementations?* and *(2) How can LLMs facilitate or automate the repair of those vulnerabilities for open-source 5G UE implementations?*

In this work, we present the *first* exploration of integrating LLMs into 5G software security workflows. Specifically, we introduce an LLM-assisted vulnerability detection approach that leverages 5G vulnerability properties and Chain-of-Thought (CoT) prompting [40] to identify 5G-specific vulnerabilities, such as missing validation of security-critical fields and protocol compliance issues. Building on this, we also propose a vulnerability repair process that combines 5G Secure Coding Practices (SCPs) with generic SCPs, enabling LLMs to generate effective fixes for the detected vulnerabilities. To evaluate our approach, we conduct a case study[1] on an open-source 5G UE implementation, demonstrating its effectiveness

[*] Equal contribution.

Workshop on Security and Privacy of Next-Generation Networks (FutureG) 2025
24 February 2025, San Diego, CA, USA
ISBN 979-8-9919276-7-3
https://dx.doi.org/10.14722/futureg.2025.23021
www.ndss-symposium.org

[1]Our demo video is available at https://github.com/qiqingh/LLM_assisted_vulnerability_detection_repair.

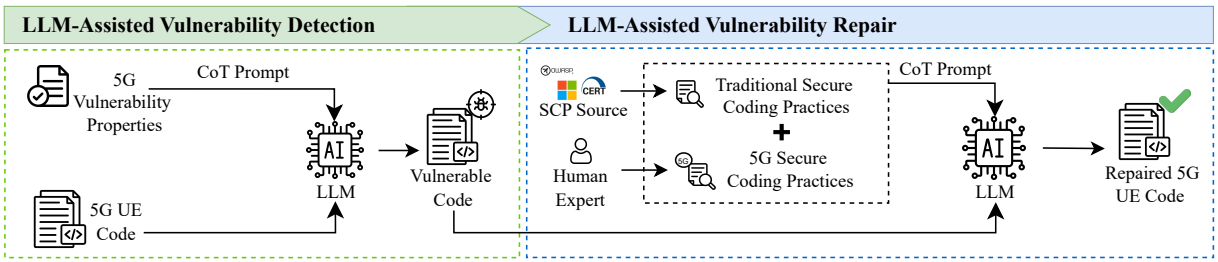

Fig. 1. Overview of Approach.

through a comparative analysis of vulnerability identification and repair before and after applying our methodology.

II. Approach

This section outlines our framework for enhancing 5G UE security. As depicted in Figure 1, the framework includes two phases: it first identifies 5G-specific vulnerabilities using curated 5G Vulnerability Properties, then applies 5G-specific and generic SCPs for repair in the next phase.

A. LLM-Assisted Vulnerability Detection

The goal of this stage is to identify vulnerabilities specific to 5G UE implementations. The complexity of 5G systems, including protocol behaviors and multi-layer interactions, often exceeds the capabilities of general-purpose code analysis. To address this, we leverage 5G vulnerability properties (VPs), derived from security testing results, which include findings from prior LTE/5G security research [11] [5] [21] [20] and empirical evaluations of UE implementations. These properties encapsulate common security weaknesses found in open-source 5G software and are defined through a structured verification process. Each property is analyzed across three key dimensions: (1) Exploitable Input, determining whether a vulnerability can be triggered via a 5G remote message (e.g., NAS/RRC/PHY); (2) Security Impact, assessing potential threats such as DoS, authentication bypass, or data leakage; and (3) Practical Exploitability, evaluating the real-world feasibility of an attack. Table I shows several examples of VPs.

To fully leverage these defined properties, we employ CoT prompting [40], which enhances the LLM's ability to systematically reason through complex tasks like vulnerability detection [41]–[43]. CoT prompting improves accuracy by guiding the model through structured reasoning steps, ensuring a nuanced understanding of code behaviors in 5G systems. To integrate the properties into CoT reasoning, we structure the CoT prompt (detailed in Appendix A) to systematically guide the LLM in analyzing a code snippet and identifying whether it exhibits a VP.

The process begins with a clear definition of the VP under consideration. The LLM is first instructed to review the property's characteristics and security implications. It then performs a semantic analysis of the code, examining execution pathways, data dependencies, and how different components interact. This ensures the LLM develops a comprehensive understanding of the code structure and behavior before proceeding to the detection step. Finally, the LLM identifies specific lines of code that exhibit the vulnerability and provides structured explanations detailing why they are problematic. These findings are reported in a standardized format, highlighting each vulnerable line and explaining its security impact.

By embedding VPs into CoT reasoning, we ensure the LLM systematically evaluates them through explicit, structured steps. This integration allows for a precise and context-aware vulnerability detection process, improving the reliability of identifying security weaknesses in 5G UE implementations.

B. LLM-Assisted Vulnerability Repair

The repair stage builds upon the vulnerabilities identified during the detection phase, aiming to address these issues through actionable SCPs. SCPs are a set of proven guidelines that developers use to write secure and robust code, addressing critical areas such as input validation, error handling, and memory safety. LLMs are trained on vast amounts of code data, including numerous examples of these practices. This enables LLMs to apply these practices when repairing code, and prior methods have effectively leveraged SCPs to repair vulnerabilities in traditional software [44], [45]. The SCPs are sourced from trusted sources such as OWASP [46], Microsoft [47], and CERT [48], which provide comprehensive and reliable secure coding standards. SCPs are highly relevant for 5G implementations, as the underlying code is written in C.

However, since the logic of these implementations is 5G-specific, there is a need for SCPs that are tailored to the unique environment of 5G. To address this gap, we propose a combined approach that utilizes both general SCPs and 5G-specific SCPs in the repair process.

5G Secure Coding Practices are guidelines designed to address vulnerabilities unique to 5G UE implementations. These practices extend general SCPs by targeting the specific challenges of 5G UE environments, such as ensuring proper input validation, maintaining protocol integrity, and validating resource allocation parameters. We curate this list of SCPs via human expert knowledge corresponding to the vulnerabilities properties identified in Section II-A. A sample list of 5G-specific and general SCPs is provided in Table II.

Prior work has demonstrated that LLMs can be leveraged to repair traditional software vulnerabilities [49], [50]. CoT prompts can enhance this process by guiding LLMs through

TABLE I
5G VULNERABILITY PROPERTY EXAMPLES

Category	ID	Vulnerability Property (VP)	Exploitable Input	Security Impact	Practical Exploitability
Input Validation Issues	VP1	Lack of ASN.1 input validation allowing malformed messages.	NAS Attach RRC Setup	DoS UE crash	Fake base station sends malformed RRC/NAS messages
	VP2	Missing validation of security-critical protocol fields allowing security bypass.	NAS Security Mode RRC Reconfig	Bypassing authentication Misconfigurations	Fake base station sends manipulated NAS Security Mode Command or RRC Reconfiguration
Memory & Resource Issues	VP3	Insufficient validation in resource allocation leading to resource exhaustion.	RRC Connection NAS Service Request	DoS Excessive resource usage	Compromised UE or botnet UE triggers excessive RRC/NAS signaling
	VP4	Lack of boundary checks in buffer management causing memory corruption.	PDCP PDU NAS Downlink	UE crash Memory corruption	Malformed NAS/RRC packets trigger memory corruption in UE baseband
Protocol Compliance Issues	VP5	Failure to enforce integrity protection allowing message forgery or replay attacks.	NAS Registration RRC Security Mode	Identity spoofing, Downgrade attacks Replay attacks	MITM attack injects unprotected NAS messages
	VP6	Accepting undefined or reserved protocol parameters leading to unexpected behavior.	RRC Capability NAS Security Mode	Unexpected protocol behavior Security bypass	Unexpected behavior triggered by incorrect RRC/NAS parameter handling

TABLE II
5G AND GENERAL SECURE CODING PRACTICES

Category	Type	Secure Coding Practices
Input Validation	5G-specific	Perform validation on all ASN.1 messages to detect malformed packets before processing.
	5G-specific	Enforce rigorous validation of security-critical protocol fields to prevent security bypass or message forgery.
Memory and Resource Management	General	Implement strict resource allocation validation to prevent resource exhaustion attacks.
	General	Apply boundary checks on buffers and memory operations to prevent corruption.
Protocol Compliance	5G-specific	Require integrity checks and cryptographic validation for messages to prevent forgery and replay attacks.
	5G-specific	Reject undefined or reserved protocol parameters to avoid unexpected behavior.

reasoning. To leverage this, we constructed a CoT prompt, detailed in Appendix A, that directs the LLM in applying SCPs. This prompt takes a vulnerability, the original code snippet, and the list of SCPs as inputs, and systematically leads the LLM through the repair process. It starts by analyzing the vulnerability and understanding its root cause. Then, the LLM examines the data and control flows related to the vulnerability and pinpoints where the issue arises. The LLM then applies the appropriate SCPs to mitigate the vulnerability, incorporating both 5G-specific and general SCPs as needed. The output of the repair process is a corrected code snippet that resolves the detected vulnerability.

III. CASE STUDY

This case study demonstrates how Vulnerability Property 1 (VP1) listed in Table I arises in OpenAirInterface (OAI) [25]. Specifically, we focus on the `nr_generate_pucch0` function in OAI. Using ChatGPT [51], we identified a potential flaw, verified its exploitability, and proposed a fix. We further validated both the vulnerability and our patch using 5Ghoul [52] by injecting malicious RRC messages, confirming that the vulnerability existed and was successfully mitigated after our repair. Details are provided in Appendix B.

A. Vulnerability Detection

Our approach systematically evaluates each Vulnerability Property (VP) to ensure a structured and thorough analysis of security flaws in 5G UE implementations. While the framework is designed to assess all defined VPs, this case study focuses on VP1: Lack of ASN.1 Validation. Exhibiting this property can lead to unintended execution behaviors, including denial-of-service (DoS) conditions, memory corruption, and out-of-bounds memory access.

Following the CoT-based vulnerability detection methodology outlined in Section II-A, the LLM first reviews VP1, recognizing its characteristics and security risks, including buffer overflows and unauthorized memory access. Once the vulnerability property is understood, the LLM proceeds with a semantic analysis of the code, where it examines execution pathways, data dependencies, and interactions between components. This analysis helps identify scenarios where incorrect computations could lead to security risks. Finally, the LLM analyzes the `nr_generate_pucch0` function to assess whether it exhibits characteristics associated with VP1. During this analysis, the expression `txdataF[0][(l2*frame_parms->ofdm_symbol_size)+re_offset]` is flagged as exhibiting the VP. The LLM reasons that `l2` and `re_offset` originate from fields within ASN.1 packets, highlighting their dependence on protocol message processing. It also recognizes the indexing operation within `txdataF`, determining that it relies on these ASN.1 influenced values, thereby establishing a direct link between ASN.1 packet handling and memory access. The LLM finds that these values are not validated before use, meaning that if they are set incorrectly, they could lead to out-of-bounds memory access, buffer overflows, or unintended memory writes. To validate these findings, we manually analyzed the code to confirm the LLM's reasoning. Additionally, we modified the ASN.1 packet that influences these values and constructed an exploit. As illustrated at the top of Figure 2, the exploit successfully triggered a system crash, confirming the identified vulnerability. The full details of this verification process and the corresponding exploit are provided in Appendix B.

B. Vulnerability Repair

In the repair phase, the LLM was tasked with repairing the code via the methodology described in Section II-B. By leveraging the list of SCPs tailored for 5G systems, the LLM generated a comprehensive patch to resolve the issue. The patch introduced guard checks to ensure that key parameters remained within valid ranges, preventing potential memory

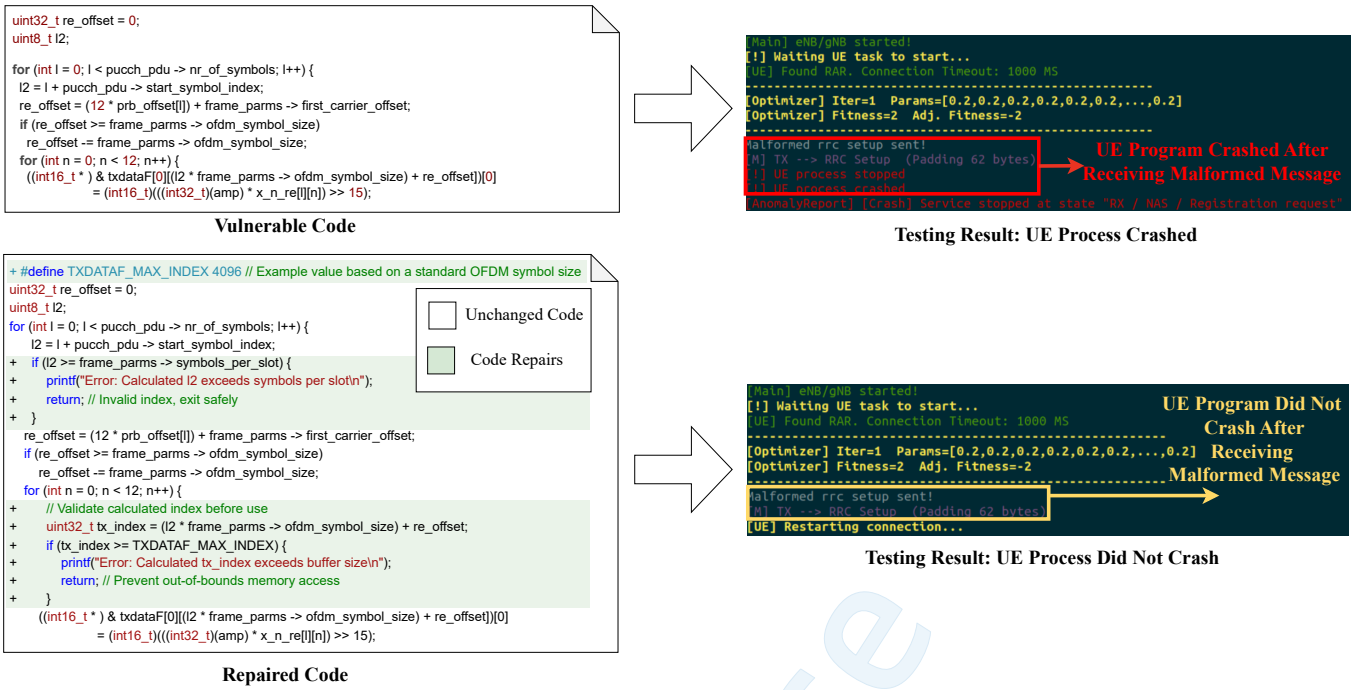

Fig. 2. Case Study Testing Results Illustrating the Successful Prevention of UE Program Crashes Through the LLM-Assisted Detection and Repair Process.

corruption or unintended behavior. For instance, bounds were established for variables like `l2` and `tx_index`, and validations were implemented to prevent out-of-range calculations. The LLM further demonstrated contextual understanding by deducing appropriate ASN.1 schema related boundaries for these variables and incorporating global constants where needed. The effectiveness of the repair was evaluated by retesting the patched code against the same exploit. Unlike the vulnerable version, the patched code successfully handled the malformed input without crashing, confirming that the repair was effective. This outcome highlights the robustness of the LLM-assisted repair process in mitigating real-world vulnerabilities. Figure 2 showcases the original vulnerable code, the code repair generated, and screenshots from the testing environment, illustrating how the patch addresses the identified issue.

IV. Limitations and Future Directions

In this work, we present the first study of an LLM-assisted approach for vulnerability detection and remediation in open-source 5G UE implementations, highlighting the potential of leveraging LLMs to enhance 5G security. While challenges such as hallucinations in LLMs [53] and scalability issues in code analysis [54] exist and should be addressed, we also recognize the following key limitations that warrant further investigation in future work:

Identification of 5G-Specific Vulnerability Properties and SCPs: Our current approach requires manual identification of 5G Vulnerability Properties and SCPs, relying on expert knowledge. This labor-intensive process risks missing emerging vulnerabilities. In the future, we will develop an automated system to derive these security requirements by analyzing data from 3GPP standards, large-scale 5G code repositories, and historical vulnerability disclosures. Combining this with SCPs, we plan to build an evolving library of 5G-specific checks and mitigations, minimizing manual effort and improving LLM-driven vulnerability detection and repair.

Utilization of 5G Vulnerability Properties and SCPs: While our current framework utilizes CoT prompting to apply 5G Vulnerability Properties and SCPs in vulnerability detection and repair, the complexity of managing and efficiently utilizing these properties and SCPs increases as the system evolves. To address this, future work will focus on developing advanced methods for utilizing properties and SCPs. One approach involves leveraging the interrelationships among properties and SCPs to dynamically prioritize the most contextually relevant ones, thereby reducing computational overhead and enhancing efficiency. Additionally, we will investigate structuring properties and SCPs hierarchically, applying critical or high-level concepts first, followed by progressively detailed elements to streamline their application and enhance scalability.

Evaluation Across the 5G Ecosystem: Our methodology was demonstrated primarily through a single case study. While this case study highlights the feasibility and effectiveness of our framework, it lacks a comprehensive evaluation across other components, such as 5G base stations or 5G core networks. Consequently, the generalizability of the proposed framework has yet to be established. To address this, future work will expand the evaluation of LLM-assisted vulnerability detection across the broader 5G ecosystem. By testing the methodology in different segments of the 5G architecture, our objective is to enhance our understanding of its applicability and identify opportunities to improve the detection and mitigation of vulnerabilities across the entire 5G infrastructure.

REFERENCES

[1] T. Trainer, "Applications of 5g network," 2023. [Online]. Available: https://www.telecomtrainer.com/explain-technically-in-detail-applications-of-5g-network/

[2] Y. Yu, Y. Xu, K. Huang, and J. Liu, "Tapfixer: Automatic detection and repair of home automation vulnerabilities based on negated-property reasoning," in *33rd USENIX Security Symposium (USENIX Security 24)*. USENIX Association, 2024, pp. 4945–4962.

[3] S. Groß, S. Koch, L. Bernhard, T. Holz, and M. Johns, "Fuzzilli: Fuzzing for javascript jit compiler vulnerabilities." in *NDSS*, 2023.

[4] G. S. Lab, "Codeql: Code analysis and security tool," https://codeql.github.com/, 2025, accessed: 2025-01-06.

[5] C. Park, S. Bae, B. Oh, J. Lee, E. Lee, I. Yun, and Y. Kim, "{DoLTEst}: In-depth downlink negative testing framework for {LTE} devices," in *31st USENIX Security Symposium (USENIX Security 22)*, 2022, pp. 1325–1342.

[6] S. Hussain, O. Chowdhury, S. Mehnaz, and E. Bertino, "Lteinspector: A systematic approach for adversarial testing of 4g lte," in *Network and Distributed Systems Security (NDSS) Symposium 2018*, 2018.

[7] S. R. Hussain, M. Echeverria, I. Karim, O. Chowdhury, and E. Bertino, "5greasoner: A property-directed security and privacy analysis framework for 5g cellular network protocol," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, 2019, pp. 669–684.

[8] I. Karim, S. R. Hussain, and E. Bertino, "Prochecker: An automated security and privacy analysis framework for 4g lte protocol implementations," in *2021 IEEE 41st International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2021, pp. 773–785.

[9] M. Akon, T. Yang, Y. Dong, and S. R. Hussain, "Formal analysis of access control mechanism of 5g core network," in *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*, 2023, pp. 666–680.

[10] D. Rupprecht, K. Jansen, and C. Pöpper, "Putting {LTE} security functions to the test: A framework to evaluate implementation correctness," in *10th USENIX Workshop on Offensive Technologies (WOOT 16)*, 2016.

[11] H. Kim, J. Lee, E. Lee, and Y. Kim, "Touching the untouchables: Dynamic security analysis of the lte control plane," in *2019 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2019, pp. 1153–1168.

[12] M. Chlosta, D. Rupprecht, T. Holz, and C. Pöpper, "Lte security disabled: misconfiguration in commercial networks," in *Proceedings of the 12th conference on security and privacy in wireless and mobile networks*, 2019, pp. 261–266.

[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 *NDSS*, 2021.

[14] H. Wang, B. Cui, W. Yang, J. Cui, L. Su, and L. Sun, "An automated vulnerability detection method for the 5g rrc protocol based on fuzzing," in *2022 4th International Conference on Advances in Computer Technology, Information Science and Communications (CTISC)*, 2022, pp. 1–7.

[15] M. E. Garbelini, Z. Shang, S. Chattopadhyay, S. Sun, and E. Kurniawan, "Towards automated fuzzing of 4g/5g protocol implementations over the air," in *GLOBECOM 2022-2022 IEEE Global Communications Conference*. IEEE, 2022, pp. 86–92.

[16] G. Hernandez, M. Muench, D. Maier, A. Milburn, S. Park, T. Scharnowski, T. Tucker, P. Traynor, and K. Butler, "Firmwire: Transparent dynamic analysis for cellular baseband firmware," in *Network and Distributed Systems Security Symposium (NDSS) 2022*, 2022.

[17] S. Nie, Y. Zhang, T. Wan, H. Duan, and S. Li, "Measuring the deployment of 5g security enhancement," in *Proceedings of the 15th ACM Conference on Security and Privacy in Wireless and Mobile Networks*, 2022, pp. 169–174.

[18] D. Klischies, M. Schloegel, T. Scharnowski, M. Bogodukhov, D. Rupprecht, and V. Moonsamy, "Instructions unclear: Undefined behaviour in cellular network specifications," in *32nd USENIX Security Symposium (USENIX Security 23)*, 2023, pp. 3475–3492.

[19] G. K. Gegenhuber, W. Mayer, E. Weippl, and A. Dabrowski, "{MobileAtlas}: Geographically decoupled measurements in cellular networks for security and privacy research," in *32nd USENIX Security Symposium (USENIX Security 23)*, 2023, pp. 3493–3510.

[20] E. Bitsikas, S. Khandker, A. Salous, A. Ranganathan, R. Piqueras Jover, and C. Pöpper, "Ue security reloaded: Developing a 5g standalone user-side security testing framework," in *Proceedings of the 16th ACM Conference on Security and Privacy in Wireless and Mobile Networks*, 2023, pp. 121–132.

[21] S. Khandker, M. Guerra, E. Bitsikas, R. P. Jover, A. Ranganathan, and C. Pöpper, "Astra-5g: Automated over-the-air security testing and research architecture for 5g sa devices," in *Proceedings of the 17th ACM Conference on Security and Privacy in Wireless and Mobile Networks*, 2024, pp. 89–100.

[22] F. Dolente, R. G. Garroppo, and M. Pagano, "A vulnerability assessment of open-source implementations of fifth-generation core network functions," *Future Internet*, vol. 16, no. 1, p. 1, 2023.

[23] O. Project, "Open5gs: Open source implementation of 5g core and epc," 2025, accessed: 2025-01-10. [Online]. Available: https://open5gs.org/

[24] S. Project, "srsran: Open source 4g and 5g software radio suite," 2025, accessed: 2025-01-10. [Online]. Available: https://www.srsran.com/

[25] O. G. Wireless. (2023) Openairinterface 5g wireless implementation. [Online]. Available: https://gitlab.eurecom.fr/oai/openairinterface5g

[26] J. Wang, M. Huang, Y. Nie, and J. Li, "Static analysis of source code vulnerability using machine learning techniques: A survey," in *2021 4th International Conference on Artificial Intelligence and Big Data (ICAIBD)*. IEEE, 2021, pp. 76–86.

[27] Z. Xu, "Source code and binary level vulnerability detection and hot patching," in *Proceedings of the 35th IEEE/ACM International Conference on Automated Software Engineering*, 2020, pp. 1397–1399.

[28] X. Xu, Q. Zheng, Z. Yan, M. Fan, A. Jia, Z. Zhou, H. Wang, and T. Liu, "Patchdiscovery: Patch presence test for identifying binary vulnerabilities based on key basic blocks," *IEEE Transactions on Software Engineering*, vol. 49, no. 12, pp. 5279–5294, 2023.

[29] Y. Hu, Y. Zhang, and D. Gu, "Automatically patching vulnerabilities of binary programs via code transfer from correct versions," *IEEE Access*, vol. 7, pp. 28 170–28 184, 2019.

[30] J. Wei, X. Wang, D. Schuurmans, M. Bosma, F. Xia, E. Chi, Q. V. Le, D. Zhou *et al.*, "Chain-of-thought prompting elicits reasoning in large language models," *Advances in neural information processing systems*, vol. 35, pp. 24 824–24 837, 2022.

[31] H. Zou, Q. Zhao, Y. Tian, L. Bariah, F. Bader, T. Lestable, and M. Debbah, "Telecomgpt: A framework to build telecom-specfic large language models," *arXiv preprint arXiv:2407.09424*, 2024.

[32] R. Nikbakht, M. Benzaghta, and G. Geraci, "Tspec-llm: An open-source dataset for llm understanding of 3gpp specifications," *arXiv preprint arXiv:2406.01768*, 2024.

[33] Y. Cheng, L. K. Shar, T. Zhang, S. Yang, C. Dong, D. Lo, S. Lv, Z. Shi, and L. Sun, "Llm-enhanced static analysis for precise identification of vulnerable oss versions," *arXiv preprint arXiv:2408.07321*, 2024.

[34] R. Meng, M. Mirchev, M. Böhme, and A. Roychoudhury, "Large language model guided protocol fuzzing," in *Proceedings of the 31st Annual Network and Distributed System Security Symposium (NDSS)*, 2024.

[35] M. D. Purba, A. Ghosh, B. J. Radford, and B. Chu, "Software vulnerability detection using large language models," in *2023 IEEE 34th International Symposium on Software Reliability Engineering Workshops (ISSREW)*. IEEE, 2023, pp. 112–119.

[36] S. Yan, S. Wang, Y. Duan, H. Hong, K. Lee, D. Kim, and Y. Hong, "An llm-assisted easy-to-trigger backdoor attack on code completion models: Injecting disguised vulnerabilities against strong detection," *arXiv preprint arXiv:2406.06822*, 2024.

[37] S. Wang, X. Wang, K. Sun, S. Jajodia, H. Wang, and Q. Li, "Graphspd: Graph-based security patch detection with enriched code semantics," in *2023 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2023, pp. 2409–2426.

[38] J. Wang, L. Yu, and X. Luo, "Llmif: Augmented large language model for fuzzing iot devices," in *2024 IEEE Symposium on Security and Privacy (SP)*. IEEE Computer Society, 2024, pp. 196–196.

[39] K. Tamberg and H. Bahsi, "Harnessing large language models for software vulnerability detection: A comprehensive benchmarking study," *arXiv preprint arXiv:2405.15614*, 2024.

[40] J. Wei, X. Wang, D. Schuurmans, M. Bosma, B. Ichter, F. Xia, E. Chi, Q. Le, and D. Zhou, "Chain-of-thought prompting elicits reasoning in large language models," 2023. [Online]. Available: https://arxiv.org/abs/2201.11903

[41] Y. Nong, M. Aldeen, L. Cheng, H. Hu, F. Chen, and H. Cai, "Chain-of-thought prompting of large language models for discovering and fixing software vulnerabilities," 2024. [Online]. Available: https://arxiv.org/abs/2402.17230

[42] D. Noever, "Can large language models find and fix vulnerable software?" 2023. [Online]. Available: https://arxiv.org/abs/2308.10345
[43] Anonymous, "To err is machine: Vulnerability detection challenges LLM reasoning," in *Submitted to The Thirteenth International Conference on Learning Representations*, 2024, under review. [Online]. Available: https://openreview.net/forum?id=Q0mp2yBvb4
[44] J. Res, I. Homoliak, M. Perešíni, A. Smrčka, K. Malinka, and P. Hanacek, "Enhancing security of ai-based code synthesis with github copilot via cheap and efficient prompt-engineering," 2024. [Online]. Available: https://arxiv.org/abs/2403.12671
[45] Y. Fu, E. Baker, Y. Ding, and Y. Chen, "Constrained decoding for secure code generation," 2024. [Online]. Available: https://arxiv.org/abs/2405.00218
[46] OWASP Foundation, "Owasp secure coding practices," n.d., accessed: 2024-08-04. [Online]. Available: https://owasp.org/www-project-secure-coding-practices-quick-reference-guide/stable-en/02-checklist/05-checklist
[47] Microsoft, "Secure coding guidelines," 2022, accessed: 2024-08-04. [Online]. Available: https://learn.microsoft.com/en-us/dotnet/standard/security/secure-coding-guidelines
[48] CERT, "Cert secure coding standards," n.d., accessed: 2024-08-04. [Online]. Available: https://wiki.sei.cmu.edu/confluence/display/seccode/SEI+CERT+Coding+Standards
[49] H. Pearce, B. Tan, B. Ahmad, R. Karri, and B. Dolan-Gavitt, "Examining zero-shot vulnerability repair with large language models," 2022. [Online]. Available: https://arxiv.org/abs/2112.02125
[50] U. Kulsum, H. Zhu, B. Xu, and M. d'Amorim, "A case study of llm for automated vulnerability repair: Assessing impact of reasoning and patch validation feedback," 2024. [Online]. Available: https://arxiv.org/abs/2405.15690
[51] OpenAI, "Chatgpt: Optimizing language models for dialogue," 2023, accessed: 2025-01-10. [Online]. Available: https://openai.com/chatgpt
[52] A. Group, "5ghoul: Exploring 5g security vulnerabilities," https://asset-group.github.io/disclosures/5ghoul/, 2025, accessed: 2025-01-08.
[53] Z. Ji, N. Lee, R. Frieske, T. Yu, D. Su, Y. Xu, E. Ishii, Y. J. Bang, A. Madotto, and P. Fung, "Survey of hallucination in natural language generation," *ACM Computing Surveys*, vol. 55, no. 12, pp. 1–38, 2023.
[54] M. Allamanis, M. Brockschmidt, and M. Khademi, "Learning to represent programs with graphs," *arXiv preprint arXiv:1711.00740*, 2017.

Appendix A
Chain-of-Thought Prompt Templates for Vulnerability Detection and Repair

```
{Code Snippet}

You are tasked with analyzing the code snippet
above for vulnerabilities related to 5G systems.
The specific 5G Vulnerability Property to focus on
is:

{5G Security Vulnerability}

Follow these instructions to ensure a thorough and
systematic analysis:

1. Review the definition of the 5G Vulnerability
Property, understanding its characteristics and
security implications.

2. Conduct a semantic analysis of the code,
examining execution pathways, data dependencies,
and interactions between components. Assess whether
 security mechanisms regulate data propagation and
execution logic.

3. Identify specific lines of code that exhibit the
 vulnerability. For each line, explain why it is
vulnerable by describing what the code is doing (or
 failing to do) and how it relates to the
vulnerability property.

4. Provide your findings in the following format:
   Line of Code: [The Code that is found to be
   vulnerable]
   Explanation: [Explanation in one or two lines]

Ensure your analysis is comprehensive and that you
review the entire code snippet. Do not attempt to
fix the code. Focus solely on identifying and
explaining the vulnerabilities.
```

Listing 1. Chain-of-Thought Prompt for Vulnerability Detection.

```
You are tasked with repairing the provided code
above against the following vulnerability in 5G
protocol message processing.

{Detected Vulnerable Lines of Code}
{Explanation of Vulnerability}

Follow these steps:

1. Analyze the Vulnerability: Examine the provided
code snippet and identify the root cause of the
issue.

2. Analyze Data and Control Flows: Trace the data
and control flows from the identified lines of code
 to understand how input propagates, any checks or
transformations applied, and where the
vulnerability occurs.

3. Apply Secure Coding Practices: {SCP}

4. Plan the Patch: Outline the approach to patching
 the vulnerability, detailing specific fixes and
their justification.

5. Implement the Fix: Write the repaired code,
including comments to explain each change and
whether it aligns with 5G-specific requirements or
traditional practices.

6. Justify Constants: If constants are added,
provide comments explaining their purpose, chosen
values, and possible alternatives.

Format of Output:

Plan to Patch: [Write the plan for fixing the
vulnerability]
Repaired Code: [Include the repaired code snippet,
with detailed inline comments]
Constants: [List all constants, with comments in
code explaining their configuration and
alternatives]
```

Listing 2. Chain-of-Thought Prompt for Vulnerability Repair.

Appendix B
Additional Details for Case Study

Testing Environment The testing environment was designed to validate the identified vulnerability in a controlled yet realistic setting. It consisted of a base station (gNB) and a UE, both implemented using the open-source software provided by the OpenAirInterface (OAI) project [25]. To facilitate the testing process, we utilized the exploit interface provided by the 5Ghoul platform [52]. This interface enabled us to intercept downlink packets transmitted from the gNB to the UE, manually locate the offset position of the `k2` field within the packet payload, and apply targeted mutations to the `k2`

field before forwarding the modified packets to the UE. This approach allowed for precise manipulation of packet content while maintaining the natural flow of 5G network operations. By leveraging the flexibility of the 5Ghoul exploit interface, we ensured that the environment supported reproducible and targeted testing of the OAI UE implementation without interference from external factors.

Vulnerability Analysis: We manually analyzed the correctness of their reasoning. The `l2` value is directly affected by the `start_symbol_index`, a key parameter influenced by the ASN input. The `start_symbol_index` is calculated based on the `k2` field in the `pusch-TimeDomainAllocation` structure, detailed in Listing 3. Specifically, `k2` determines the offset used to calculate `start_symbol_index`. Therefore, a targeted mutation of `k2` was carried out. This mutation triggered a cascading effect: `start_symbol_index` changed, which led to `l2` changing, and ultimately caused the array indexing in `txdataF` to become misaligned. As a result, unintended memory locations were accessed, leading to out-of-bounds access.

Verification Process To verify the presence of the identified vulnerability and assess the effectiveness of the repair, a structured testing and validation process was carried out. First, the `k2` field in the downlink packet, transmitted from the gNB to the UE, was mutated to generate targeted test cases. The mutation was crafted to manipulate the memory alignment in the UE software, leading to a program crash when this vulnerability was exploited as shown in Figure 2. After confirming the vulnerability, the affected code segment was repaired, and the entire OAI UE program was recompiled accordingly. To validate the success of the repair, the same mutated test case was retransmitted to the UE. No crashes occurred, confirming the patch's effectiveness and highlighting LLM-assisted vulnerability detection and repair.Through LLM analysis, we identified the vulnerability in the UE's code and used this insight to guide the repairing process. This demonstrated the practical value of LLM-assisted approaches.

```
PUSCH-ConfigCommon ::= SEQUENCE {
    timeDomainAllocationList PUSCH-
    TimeDomainResourceAllocationList,
    hoppingId INTEGER (0..1023) OPTIONAL,
    ...
}

PUSCH-TimeDomainResourceAllocationList ::= SEQUENCE
 (SIZE (1..maxNrofTimeDomainResourceAllocations))
OF PUSCH-TimeDomainResourceAllocation

PUSCH-TimeDomainResourceAllocation ::= SEQUENCE {
    k2 INTEGER (0..31) OPTIONAL,
    mappingType ENUMERATED {typeA, typeB},
    startSymbolAndLength BIT STRING (SIZE (14))
}
```

Listing 3. pusch-ConfigCommon ASN.1 Schema

5G Vulnerability Property:
Fails to validate ASN.1 encoded inputs to prevent malformed packets and protocol violations in 5G.

Findings:

Line of Code:
```
for (int l=0; l<pucch_pdu->nr_of_symbols; l++) {
```
Explanation:
`pucch_pdu->nr_of_symbols` is directly used without validating whether its value falls within an acceptable range. Malformed or tampered ASN.1 encoded inputs could set `nr_of_symbols` to invalid values, leading to out-of-bounds access during the loop.

Line of Code:
```
l2=l+pucch_pdu->start_symbol_index;
```
Explanation:
`pucch_pdu->start_symbol_index` is used without validation. If this field is manipulated in the ASN.1 encoded input, it could result in `l2` pointing to an invalid symbol index, potentially leading to incorrect resource element (RE) mapping or buffer overflows.

Line of Code:
```
re_offset = (12*prb_offset[l]) + frame_parms->
first_carrier_offset;
```
Explanation:
The variable `prb_offset[l]` is used without bounds checking. If the ASN.1 encoded input indirectly influences `prb_offset`, an invalid value could cause incorrect or out-of-bounds offsets, leading to memory corruption or unintended behavior.

Line of Code:
```
if (re_offset>= frame_parms->ofdm_symbol_size)
re_offset-=frame_parms->ofdm_symbol_size;
```
Explanation:
This code attempts to handle out-of-bounds `re_offset` values but does not prevent them from being set initially. This could allow malformed ASN.1 inputs to bypass proper validation and result in undefined behavior in subsequent operations.

Line of Code:
```
((int16_t *)&txdataF[0][(l2*frame_parms->
ofdm_symbol_size) + re_offset])[0] = (int16_t)(((
int32_t)(amp) * x_n_re[l][n])>>15);
((int16_t *)&txdataF[0][(l2*frame_parms->
ofdm_symbol_size) + re_offset])[1] = (int16_t)(((
int32_t)(amp) * x_n_im[l][n])>>15);
```
Explanation:
The array indexing `txdataF[0][(l2*frame_parms->ofdm_symbol_size) + re_offset]` depends on `l2` and `re_offset`, both of which are indirectly influenced by the ASN.1 encoded inputs. Malformed inputs could cause buffer overflows, memory corruption, or writing to unintended memory locations.

Summary of Vulnerabilities:
- **Lack of Input Validation:**
  Fields such as `nr_of_symbols`, `start_symbol_index`, and `prb_offset` are used without verifying their validity or range.
- **Out-of-Bounds Access:**
  Variables like `l2`, `re_offset`, and array indices derived from them are prone to out-of-bounds access due to inadequate input validation.
- **Missing Error Handling:**
  The code lacks mechanisms to detect and gracefully handle malformed ASN.1 encoded inputs.

Listing 4. Complete LLM Output for Vulnerability Detection

The vulnerability stems from improper validation of fields in `pucch_pdu` and related calculations, which are influenced by ASN.1-encoded messages. Specifically:
1. **Missing Boundary Checks**:
   Fields like `nr_of_symbols`, `start_symbol_index`, and `prb_offset` are used without verifying their validity, potentially leading to out-of-bounds access.
2. **No Handling of Malformed Messages**:
   Invalid or tampered ASN.1 packets are not detected or safely discarded, risking system instability.
3. **Improper Memory Operations**:
   Indexing operations involving `re_offset` and `txdataF` lack safeguards against invalid calculations.

To fix the vulnerability:
- Add strict range checks for all `pucch_pdu` fields before their use.
- Validate calculated offsets (`re_offset`) against defined boundaries.
- Use recovery mechanisms to handle invalid protocol messages gracefully, ensuring the system remains stable.
- Incorporate predefined global constants representing valid ranges to enforce compliance with 5G standards.

Repaired Code:

```
void nr_generate_pucch0(PHY_VARS_NR_UE *ue,
                        int32_t **txdataF,
                        NR_DL_FRAME_PARMS *
                        frame_parms,
                        int16_t amp,
                        int nr_slot_tx,
                        fapi_nr_ul_config_pucch_pdu
                         *pucch_pdu) {
  // Define valid ranges (see Constants section for
   justification)
  if (pucch_pdu->nr_of_symbols <= 0 || pucch_pdu->
  nr_of_symbols > MAX_NR_OF_SYMBOLS) {
    printf("Error: Invalid number of symbols in
    pucch_pdu\n");
    return; // Discard invalid message
  }
  if (pucch_pdu->start_symbol_index < 0 ||
  pucch_pdu->start_symbol_index >= frame_parms->
  symbols_per_slot) {
    printf("Error: Invalid start_symbol_index in
    pucch_pdu\n");
    return; // Discard invalid message
  }
  uint32_t re_offset = 0;
  uint8_t l2;
  for (int l = 0; l < pucch_pdu->nr_of_symbols; l
  ++) {
    l2 = l + pucch_pdu->start_symbol_index;
    if (l2 >= frame_parms->symbols_per_slot) {
      printf("Error: Calculated l2 exceeds symbols
      per slot\n");
      return; // Invalid index, exit safely
    }
    // Validate prb_offset value
    if (prb_offset[l] < 0 || prb_offset[l] >=
    MAX_PRB_OFFSET) {
      printf("Error: Invalid PRB offset for symbol
      %d\n", l);
      return; // Discard invalid message
    }
    re_offset = (12 * prb_offset[l]) + frame_parms
    ->first_carrier_offset;
    if (re_offset >= frame_parms->ofdm_symbol_size)
      re_offset -= frame_parms->ofdm_symbol_size;
    for (int n = 0; n < 12; n++) {
      // Validate calculated index before use
      uint32_t tx_index = (l2 * frame_parms->
      ofdm_symbol_size) + re_offset;
      if (tx_index >= TXDATAF_MAX_INDEX) {
        printf("Error: Calculated tx_index exceeds
        buffer size\n");
        return; // Prevent out-of-bounds memory
        access
      }
      ((int16_t *)&txdataF[0][tx_index])[0] = (
      int16_t)(((int32_t)(amp) * x_n_re[l][n]) >>
      15);
      ((int16_t *)&txdataF[0][tx_index])[1] = (
      int16_t)(((int32_t)(amp) * x_n_im[l][n]) >>
      15);
      re_offset++;
      if (re_offset >= frame_parms->
      ofdm_symbol_size)
        re_offset -= frame_parms->ofdm_symbol_size;
    }
  }
}
```

Constants:

```
// Maximum number of symbols allowed in pucch_pdu (
based on 5G standard constraints)
#define MAX_NR_OF_SYMBOLS 14 // Typical maximum for
 PUCCH in a single slot

// Maximum PRB offset value (assuming a system with
 100 MHz bandwidth and 273 PRBs)
#define MAX_PRB_OFFSET 273

// Maximum index for txdataF buffer (determined by
OFDM symbol size and implementation limits)
#define TXDATAF_MAX_INDEX 4096 // Example value
based on a standard OFDM symbol size

// Explanation of constants:
// - `MAX_NR_OF_SYMBOLS`: The value 14 ensures
compliance with TS 38.211, which defines the
maximum symbols in a slot.
// - `MAX_PRB_OFFSET`: This is based on the maximum
 number of Physical Resource Blocks (PRBs) in the
system bandwidth.
// - `TXDATAF_MAX_INDEX`: Calculated based on
implementation-defined limits for the `txdataF`
buffer. Ensure this matches the hardware
configuration.
```

Justification of Fix:
- **Strict Boundary Checks**:
  All critical fields (e.g., `nr_of_symbols`, `start_symbol_index`, `prb_offset`) are validated against predefined limits.
- **Compliance with Standards**:
  Constants are derived from 5G specifications (e.g., TS 38.211).
- **Recovery Mechanisms**:
  The function exits gracefully when invalid input is detected, ensuring no undefined behavior occurs.
- **Memory Safety**:
  Validation prevents out-of-bounds memory operations, adhering to secure coding principles.

Listing 5. Complete LLM Output for Vulnerability Repair