Feedback-Guided API Fuzzing of 5G Network

Tianchang Yang, Sathiyajith K S, Ashwin Senthil Arumugam, Syed Rafiul Hussain
The Pennsylvania State University
{*tzy5088, sxk6394, ajs9797, hussain1*}*@psu.edu*

***Abstract*—We present our work-in-progress on designing and implementing a black-box evolutionary fuzzer for REST APIs, specifically targeting 5G core networks that utilize a service-based architecture (SBA). Unlike existing tools that rely on static generation-based approaches, our approach progressively refines test inputs to explore deeper code regions in the target system. We incorporate a thorough analysis of the limited response message feedback available in black-box settings and employ a carefully crafted mutation method to generate effective state-aware test inputs. Evaluation of our current implementation has uncovered two previously unknown vulnerabilities in open-source 5G core network implementations, resulting in the assignment of two CVEs. Additionally, our approach already demonstrates superior performance compared to existing black-box fuzzing methods.**

I. Introduction

The 5G core network serves as the backbone of modern telecommunications, supporting features such as ultra-low latency, massive device connectivity, and enhanced mobile broadband. To achieve these functionalities, it adopts a Service-Based Architecture (SBA), which relies on REST APIs for communication between its functional modules, Network Functions (NFs) [1]. These APIs facilitate the exchange of critical control information throughout the operation of the core network, making them vital for the proper functioning of the overall cellular network. While the SBA design offers significant flexibility and scalability, it also expands the attack surface, exposing each NF to potential malicious API requests that could lead to severe consequences, such as service disruption, unauthorized access, or exposure of sensitive data, potentially affecting millions of users and devices. This underscores an urgent need for robust testing methodologies to ensure the reliability and security of 5G core network implementations.

Testing REST APIs presents unique challenges, particularly in the context of 5G core networks. Black-box testing approaches are often the only feasible option due to restricted access to source code and the underlying infrastructure. Existing black-box tools [2], [3], [4], [5] primarily rely on static generation-based methods to produce test cases. However, these methods struggle to explore deep code regions and are ineffective at handling stateful interactions [3]. Furthermore, they often depend on static dictionaries for parameter values and fail to fully utilize the feedback provided by response messages [2], [5], [4], missing critical opportunities to refine test inputs and uncover hidden vulnerabilities.

We present our work-in-progress of addressing these limitations by designing and implementing a black-box evolutionary fuzzer specifically tailored for REST APIs in 5G core networks. Our approach introduces the following key improvements over existing work: (1) **Dynamic refinement of test inputs:** unlike static generation-based methods, our evolutionary fuzzer progressively refines test inputs by analyzing feedback from response messages, enabling deeper exploration of the target system. (2) **State-aware testing:** we carefully design mutation strategies that respect dependencies between requests and generate state-aware test cases to uncover vulnerabilities in stateful interactions. (3) **Feedback-driven analysis:** our method incorporates a fine-grained analysis of response messages, utilizing not only response codes but also the content of messages to guide test case generation more effectively.

Scope. This work focuses on identifying implementation issues in the 5G core network, in contrast to formal methods that analyze mobile network specifications to detect specification errors [6], [7]. Our current testing targets easily observable 5XX errors, which indicate unexpected server-side failures caused by improperly handled requests or server crashes.

Evaluation & contribution. To demonstrate the efficacy of our approach, we evaluate our implementation on real-world 5G core network implementations. Our fuzzer identifies two previously unknown vulnerabilities, resulting in the assignment of two CVEs. The contributions of this work are:

- We propose a novel black-box evolutionary fuzzing approach for REST APIs, specifically designed for the service-based architecture of 5G core networks.
- We introduce a state-aware mutation and a feedback-driven analysis framework to dynamically refine test cases and effectively guide exploration.
- We evaluated our approach on open-source 5G core network implementations, demonstrating its effectiveness in discovering vulnerabilities and achieving superior code coverage compared to existing state-of-the-art tools.

II. Background

A. 5G Core Network and REST API

Cellular networks consist of the core network, base stations, and User Equipment (UE) such as smartphones. The core network handles user authentication, request processing, and connectivity management. 5G core networks revolutionize the monolithic, closed design of previous generations by adopting a service-based architecture, dividing operations into modular

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.2025.23071
www.ndss-symposium.org

components called Network Functions (NFs). Key NFs include the **AMF** (Access and Mobility Management Function), which handles user registration, mobility management, and connection establishment, and the **UPF** (User Plane Function), which manages PDU sessions, packet forwarding, and routing.

NFs within the core network communicate with each other via REST APIs. REST APIs facilitate communication between clients and servers using standard HTTP methods such as GET, POST, PUT, and DELETE. Requests typically include an HTTP method, a URL (representing a resource), headers (e.g., authentication tokens or content type), and optionally a body. Server responses include an HTTP status code (e.g., 2XX for success, 4XX for client errors, and 5XX for unexpected internal server errors), headers, and often a body with requested data or response messages. Swagger [8] is a specification language used to describe and define REST APIs, offering a structured format to detail API endpoints, request formats, request methods, parameters, and response structures.

III. Related Work

A. Testing of 5G and Beyond Mobile Networks

Existing efforts on core network security primarily employ formal verification methods [9], [6], [7], which rely on protocol specifications to identify specification errors but are ineffective at detecting implementation flaws. Prior implementation testings of cellular networks [10], [11], [12], [13], [14], [15], [16], [17], [18] focus on specific interfaces in mobile networks. These approaches typically require significant manual effort to construct message grammars and often rely on code coverage information (i.e., white- or gray-box access to instrument the program) to achieve optimal results. However, they overlook the potential of directly utilizing well-defined REST API specifications [19] for core network testing, which could substantially reduce harnessing and manual effort.

B. REST API Fuzzing

REST API fuzzing [2], [3], [4], [5], [20], [21], [22], [23], [24], [25], [26], [27] uncovers hidden errors within the reachable execution states of a cloud service. REST API fuzzers automate request sequence generations to test a target server and use the responses to identify errors. A typical REST API fuzzer [2], [3] generally comprises the following components.
Compiler is responsible for parsing REST API specifications (e.g., Swagger [8]) to produce internal grammars and request templates for the fuzzer. During this process, the fuzzer: (1) constructs an internal grammar and structural representation for each request type, (2) builds a dictionary to generate values for each basic-type parameter during fuzzing, (3) and infers dependencies between different request methods.
Generation Module creates test requests to explore new code regions and uncover vulnerabilities in the target system. It has two primary tasks: generating meaningful request sequences and populating parameter values for each message in a sequence. REST API requests are inherently stateful. For instance, testing a GET or DELETE method often requires the client to first POST an object to create it. To address these dependencies, the compiler initially extracts relationships statically from the API specification (e.g., ensuring a PUT command is executed before GET for the same URL object). During fuzzing, the fuzzer dynamically refines request sequences by exploring different combinations of messages. If a constructed sequence consistently results in 2XX status code, it is considered valid and stored as a reusable template. Parameter values for each request in a sequence template are instantiated using the dictionary generated during the compilation phase. These values can be user-provided or predefined.
Checker evaluates the outcomes of testing sequences to identify vulnerabilities. The checker first detects whether the server returns a 5XX status code, which signals an internal server error. Additional sophisticated validations can also be implemented. For instance, a use-after-free check ensures that after a DELETE method is invoked, subsequent GET requests on the same object do not incorrectly return success.
Limitations. However, these existing approaches are insufficient for uncovering vulnerabilities due to several key limitations: (1) Their generation-based techniques lack the capability to systematically explore deeply-rooted vulnerabilities, as they do not iteratively refine inputs based on prior findings. (2) They rely heavily on pre-defined dictionaries to generate test inputs, making them unable to detect vulnerabilities triggered by edge cases or values not present in the dictionary. (3) They fail to fully utilize available feedback. While black-box testing is inherently constrained, existing methods typically rely only on response codes to evaluate test inputs, neglecting detailed response messages, which often contain critical information like error causes and other contexts. As a result of these limitations, existing tools struggle to achieve high code coverage, typically falling below 50%, and fail to identify deeply-rooted vulnerabilities [28], [29].

IV. Approach

The 5G core network is maintained by network operators, restricting direct testing access. Consequently, testing is typically limited to black-box approaches, where access to the source code or the machines hosting each NF is not permitted. Instead, only observable black-box metrics, such as response messages, can be utilized. Due to similar challenges, REST API testing tools [2], [3], [4], [5] typically all employ this design. In this work, we adopt a black-box testing methodology.

To overcome the limitations of existing REST API testing tools' generation-based methods, we adopt a mutation-based approach inspired by mutation fuzzers [30], [31]. As illustrated in Figure 1, our method leverages the limited feedback available in black-box testing to iteratively refine and mutate test inputs that trigger interesting behaviors, enabling progressive exploration of the target system. Our fine-grained response analysis identifies and preserves inputs that lead to novel system states. While we retain the compiler and checker components from previous methods, we replace the static generation module with a dynamic *mutator* to generate effective test cases by mutating inputs from the test corpus. Through

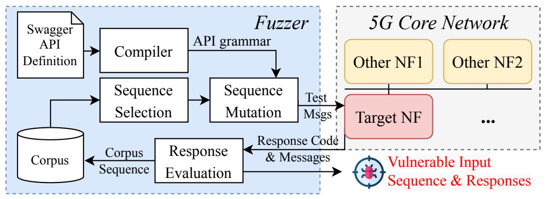

Fig. 1: Architecture of Our Approach

this iterative refinement, our approach uncovers vulnerabilities more efficiently and targets deeper system states.

A. Response Evaluation

For each sequence of inputs, we analyze the corresponding responses to determine whether the sequence has invoked previously unseen behaviors (i.e., discovered new system states). Specifically, we evaluate the type, code, and message of the response. If any part reveals a previously unobserved behavior, the corresponding input sequence is retained as an interesting test case. To avoid saving redundant or meaningless test cases, we implement a filtering mechanism for 4XX responses. A 4XX response typically indicates an invalid request, which often renders the subsequent messages in the sequence uninteresting. Therefore, if a 4XX response is encountered, we disregard the remaining messages in the sequence. Without this mechanism, the approach risks accumulating long, arbitrary concatenations of irrelevant test cases in the corpus, which dilutes the effectiveness of further mutations.

B. Sequence-Aware Mutation

Effective mutation requires careful selection of both the sequence and the specific message to mutate. We calculate a priority score for each sequence in the corpus based on its length and mutation depth. The intuition is to prioritize longer sequences, as they are more likely to explore deeper implementation states, and sequences that have undergone more mutations, to prioritize the exploration of newly-discovered program regions. Conversely, sequences that fail to reveal interesting behaviors even after multiple mutations are deprioritized. Score $\mathcal{S}$ for a sequence s_i is given by:

$$\mathcal{S}(s_i) = \alpha \cdot \text{length}(s_i) + \beta \cdot \text{depth}(s_i),$$

where $\text{length}(s_i)$ is the number of messages in the sequence s_i, $\text{depth}(s_i)$ is the number of mutations the sequence has undergone, and α and β are weights that balance the importance of length and mutation depth. The possibility of each sequence s_i being selected to mutate is based on normalized scores:

$$P(S_i) = \frac{\mathcal{S}(s_i)}{\sum_j \mathcal{S}(s_j)},$$

This ensures that sequences with higher scores are more likely to be selected, while still maintaining diversity in selection.

Once a sequence is selected, we apply only one mutation at a time to minimize deviations from the original sequence and ensure a high probability that the mutated test sequence is still accepted by the target. Within the selected sequence, we calculate a priority score for each message m_j in s_i based on its position in the sequence and its mutation history. The score $\mathcal{S}$ for a message m_j is given by:

$$\mathcal{S}(m_j) = \gamma \cdot \frac{j}{\text{len}(s_i)} + \delta \cdot \frac{1}{1 + \text{mutations}(m_j)},$$

where $\text{len}(s_i)$ represents the total length of the sequence, so $\frac{j}{\text{len}(s_i)}$ assigns higher weights to later messages in the sequence to reduce early rejection of the mutated sequence. $\text{Mutations}(m_j)$ is the number of times the message has been mutated, which prioritizes under-tested messages. γ and δ are weights that balance the influence of position and mutation history. The message is also selected based on the normalized probability similar to sequence selection. Each selected message undergoes one of the following mutation strategies: (1) **In-place mutation**, which modifies the values within the message, such as altering parameter values. (2) **Replacement** that swaps the selected message with another valid message, either randomly or selected from a different sequence in the corpus. (3) **Deletion**, which removes the message from the sequence. (4) **Addition**, that inserts a new message or multiple messages into the sequence, selected from another sequence in the corpus. (5) **Crossover**, by combining parts of the selected message with another message to create a hybrid test case.

V. Evaluation

To evaluate the effectiveness of our approach, we aim to address the following research questions:

- **RQ1.** How effective is our approach in identifying real vulnerabilities in 5G core networks?
- **RQ2.** How does our approach compare to existing REST API testing tools?

A. Identified Vulnerabilities

To address RQ1, we evaluated our approach on the major NFs of a popular open-source 5G core network implementation, free5GC [32] over a 24-hour testing period. During this evaluation, we uncovered two previously unknown vulnerabilities that caused system crashes. Both issues were assigned CVEs and acknowledged by the developers. Patches to fix both issues have been merged. The identified vulnerabilities highlight critical flaws in input validation and error handling in the target implementation. Both issues occur in AMF's Event Exposure API, which provides subscription management for AMF events. The first vulnerability is triggered by the absence of a mandatory EventList parameter. Instead of returning a client error (e.g., a 400 Bad Request), AMF fails to handle the missing parameter gracefully, leading to a null pointer dereference and triggering a 500 Internal Server Error. Similarly, the second vulnerability arises when AMF attempts to access a malformed or incomplete payload, leading to a similar Internal Server error. Both issues stem from insufficient checks for null values before dereferencing critical fields.

The identified vulnerabilities pose significant risks to the reliability and security of 5G core networks. Exploiting these weaknesses, an attacker could craft malicious requests with missing or malformed parameters to induce a denial of service attack on the AMF. This could disrupt critical functionalities

TABLE I: Comparison of REST API Testing Tools

Tool	Approach	Test Generation	Feedback/Oracle	Stateful	Notes
Ours	Black-box	Evolutionary-based	Response codes & messages	Yes	Dynamically refines test cases using mutation; prioritizes exploration based on response diversity.
RESTler [2]	Black-box	Generation-based	Response codes	Yes	Extracts dependencies from API specifications and uses static dictionaries for parameter values; limited by lack of mutation.
Miner [3]	Black-box	Generation-based	Response codes	Yes	Applies neural network to generate valid long request sequences. However, it lacks dynamic refinement of test inputs.
EvoMaster [4]	Black-/White-box	Random/ mutation	Response codes	Yes	Black-box mode applies random test generations; white-box applies mutation algorithms, but only supports Java targets.
Forest [5]	Black-box	Generation-based	Response codes	Yes	Applies tree-based approach to infer API-dependency graphs from specifications. Employs static dictionary-based test generation.
RestTestGen [26]	Black-box	Nominal/error tests	Response codes & format	Yes	Performs nominal tests using generation values and error tests by mutating nominal tests to invalid values/fields.
RESTest [20]	Black-box	Model-based	Response codes	Yes	Analyzes inter-parameter dependencies in specifications to produce nominal/faulty values for tests.
WuppieFuzz [27]	Black-/White-box	Coverage-guided	Response codes, code coverage	Yes	Supports code coverage from languages supported by LibAFL, and utilizes API specifications for request generation and mutation.

such as event subscription and notification handling, affecting network stability. However, these vulnerabilities are challenging to detect using existing tools because: (1) Static test cases existing generation-based tools employ may not include edge cases such as missing or malformed fields. (2) Detecting these issues requires understanding stateful dependencies between API requests. For example, the Event Exposure API must validate and maintain the subscription state, but existing tools cannot model such dependencies effectively.

B. Comparison Against Existing Tools

To answer RQ2, we conduct a comparative analysis of our approach with other popular REST API testing tools, presented in Table I. Our approach is the only one that employs an evolutionary algorithm in black-box settings, which enables dynamic refinement of test cases, compared to the static random generation-based techniques used by other black-box tools. Our approach leverages state-aware sequence mutation to explore complex interactions effectively. Compared to WuppieFuzz and EvoMaster, whose efficiency depends on their white-box coverage-guided testing, our black-box approach is more flexible, operating effectively without requiring instrumentation. This makes our tool more applicable to test 5G core networks where source access is unavailable.

To further evaluate our approach, we conducted a quantitative comparison with popular existing tools, including RESTler [2], foREST [5], and EvoMaster (black-box mode) [4]. Each tool was executed with its default configurations for a duration of 24 hours on the Event Exposure API of the AMF in free5GC [32]. The coverage growth over time is shown in Figure 2. As illustrated, our approach achieves the highest code coverage and demonstrates continuous growth throughout the testing period. In contrast, other compared tools stagnate after approximately six hours, where their generation-based methods struggle to uncover additional coverage.

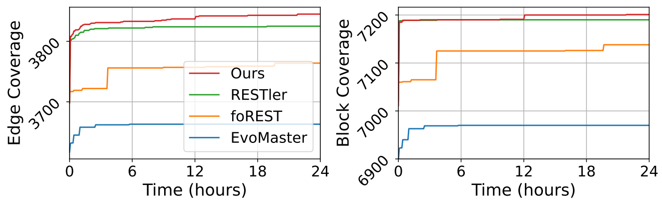

Fig. 2: Edge/Basic Block Coverage of Different Tools Over Time

VI. Conclusion & Future Direction

We presented our work-in-progress of designing and implementing a black-box evolutionary fuzzer for REST APIs in 5G networks, addressing the limitations of existing static generation-based methods through dynamic feedback-driven refinement and state-aware sequence mutations. Our evaluation uncovered two critical vulnerabilities in a widely used 5G core implementation, demonstrating the effectiveness of our approach in finding flaws that could disrupt 5G operations. We plan to enhance our approach in the following key directions.

Domain-informed test generation. Incorporating domain-specific knowledge into sequence generation can significantly improve the effectiveness of testing. Understanding the specific roles and interactions of NFs in 5G core networks and their associated procedures can help craft sequences that target inter-function dependencies and stateful interactions. For instance, the registration procedure in 5G involves intricate communication between various NFs with complex parameter dependencies. Existing approaches fail to infer such dependencies, making it challenging to generate a complete and successful registration sequence. Leveraging domain-informed heuristics could guide the mutation process to produce test cases that reflect realistic traffic patterns and uncover edge cases specific to telecommunication protocols.

LLM-enhanced parameter & sequence construction. Large language models (LLMs) offer significant potential for enhancing parameter generation and sequence construction through semantic understanding. LLMs trained on API specifications, technical documentation, and runtime logs could infer parameter constraints, relationships, and valid values, enabling the creation of highly realistic and contextually relevant test cases. Additionally, LLMs could assist in synthesizing complex API sequences by reasoning about dependencies and plausible interactions. By integrating LLM-powered insights with feedback-driven refinement, our approach could achieve greater depth and breadth of exploration, uncovering complex, multi-step vulnerabilities that traditional methods cannot find.

Acknowledgements

This work has been supported by the NSF under grants 2145631, 2215017, 2226447, and 2326898, and by DARPA under contract number D22AP00148.

References

[1] 3rd Generation Partnership Project (3GPP), "3GPP TS 23.501 V19.2.0: System Architecture for the 5G System (5GS)," 3GPP, Tech. Rep., September 2025, accessed: 2025-01-07. [Online]. Available: https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=3144

[2] V. Atlidakis, P. Godefroid, and M. Polishchuk, "Restler: Stateful rest api fuzzing," in *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*, 2019, pp. 748–758.

[3] C. Lyu, J. Xu, S. Ji, X. Zhang, Q. Wang, B. Zhao, G. Pan, W. Cao, P. Chen, and R. Beyah, "Miner: a hybrid data-driven approach for rest api fuzzing," in *Proceedings of the 32nd USENIX Conference on Security Symposium*, ser. SEC '23. USA: USENIX Association, 2023.

[4] A. Arcuri, "Evomaster: Evolutionary multi-context automated system test generation," in *2018 IEEE 11th International Conference on Software Testing, Verification and Validation (ICST)*, 2018, pp. 394–397.

[5] J. Lin, T. Li, Y. Chen, G. Wei, J. Lin, S. Zhang, and H. Xu, "forest: A tree-based black-box fuzzing approach for restful apis," in *2023 IEEE 34th International Symposium on Software Reliability Engineering (ISSRE)*, 2023, pp. 695–705.

[6] 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.

[7] 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.

[8] "Swagger: API Documentation and Design Tools." [Online]. Available: https://swagger.io/

[9] D. Basin, J. Dreier, L. Hirschi, S. Radomirovic, R. Sasse, and V. Stettler, "A formal analysis of 5g authentication," in *Proceedings of the 2018 ACM SIGSAC conference on computer and communications security*, 2018, pp. 1383–1396.

[10] S. R. Hussain, I. Karim, A. A. Ishtiaq, O. Chowdhury, and E. Bertino, "Noncompliance as deviant behavior: An automated black-box noncompliance checker for 4g lte cellular devices," in *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, 2021, pp. 1082–1099.

[11] D. Rupprecht, K. Kohls, T. Holz, and C. Pöpper, "Call me maybe: Eavesdropping encrypted lte calls with revolte." in *USENIX Security Symposium*, 2020, pp. 73–88.

[12] 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.

[13] D. Maier, L. Seidel, and S. Park, "Basesafe: Baseband sanitized fuzzing through emulation," in *Proceedings of the 13th ACM conference on security and privacy in wireless and mobile networks*, 2020, pp. 122–132.

[14] 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, Aug. 2024, pp. 1921–1938. [Online]. Available: https://www.usenix.org/conference/usenixsecurity24/presentation/yang-tianchang

[15] K. Tu, A. A. Ishtiaq, S. M. M. Rashid, Y. Dong, W. Wang, T. Wu, and S. R. Hussain, "Logic gone astray: A security analysis framework for the control plane protocols of 5g basebands," in *33rd USENIX Security Symposium (USENIX Security 24)*. Philadelphia, PA: USENIX Association, Aug. 2024, pp. 3063–3080. [Online]. Available: https://www.usenix.org/conference/usenixsecurity24/presentation/tu

[16] G. Nakas, P. Radoglou-Grammatikis, G. Amponis, T. Lagkas, V. Argyriou, S. Goudos, and P. Sarigiannidis, "5g-fuzz: An attack generator for fuzzing 5gc, using generative adversarial networks," in *2023 IEEE Globecom Workshops (GC Wkshps)*, 2023, pp. 347–352.

[17] I. Siroš, D. Singelée, and B. Preneel, "Covfuzz: Coverage-based fuzzer for 4g5g protocols," 2024. [Online]. Available: https://arxiv.org/abs/2410.20958

[18] S. L. S. C. S. S. E. K. Matheus E. Garbelini, Zewen Shang, "5Ghoul : Unleashing Chaos on 5G Edge Devices ," https://asset-group.github.io/disclosures/5ghoul/, 2023.

[19] 3rd Generation Partnership Project (3GPP), "3GPP 5G Specification series," 2024. [Online]. Available: https://www.3gpp.org/dynareport?code=29-series.htm

[20] A. Martin-Lopez, S. Segura, and A. Ruiz-Cortés, "RESTest: Automated Black-Box Testing of RESTful Web APIs," in *Proceedings of the 30th ACM SIGSOFT International Symposium on Software Testing and Analysis*, ser. ISSTA '21. Association for Computing Machinery, 2021.

[21] "Schemathesis: Property-Based Testing for Your APIs." [Online]. Available: https://github.com/schemathesis/schemathesis

[22] "Tcases: Model-Based API Testing Framework." [Online]. Available: https://github.com/Cornutum/tcases

[23] "Dredd: Language-Agnostic API Validation Tool." [Online]. Available: https://github.com/apiaryio/dredd

[24] "bBOXRT: Black-Box RESTful Testing Tool." [Online]. Available: https://git.dei.uc.pt/cnl/bBOXRT

[25] "APIFuzzer: REST API Security Fuzzer." [Online]. Available: https://github.com/KissPeter/APIFuzzer

[26] D. Corradini, A. Zampieri, M. Pasqua, and M. Ceccato, "Resttestgen: An extensible framework for automated black-box testing of restful apis," in *2022 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 2022, pp. 504–508.

[27] "WuppieFuzz: coverage-guided REST API fuzzer." [Online]. Available: https://github.com/TNO-S3/WuppieFuzz

[28] M. Zhang and A. Arcuri, "Open problems in fuzzing restful apis: A comparison of tools," *ACM Trans. Softw. Eng. Methodol.*, vol. 32, no. 6, Sep. 2023. [Online]. Available: https://doi.org/10.1145/3597205

[29] M. Kim, Q. Xin, S. Sinha, and A. Orso, "Automated test generation for rest apis: no time to rest yet," in *Proceedings of the 31st ACM SIGSOFT International Symposium on Software Testing and Analysis*, ser. ISSTA '22. ACM, Jul. 2022. [Online]. Available: http://dx.doi.org/10.1145/3533767.3534401

[30] "American Fuzzy Lop (AFL)." [Online]. Available: https://lcamtuf.coredump.cx/afl/

[31] "libFuzzer: A Library for Coverage-Guided Fuzzing." [Online]. Available: https://llvm.org/docs/LibFuzzer.html

[32] "free5GC: An Open-Source 5G Core Network Implementation." [Online]. Available: https://free5gc.org/