5GC-Fuzz: Finding Deep Stateful Vulnerabilities in 5G Core Network with Black-box Fuzzing

Yu Sun[1,2], Xinyu Liu[1,2], Qian Sun[3,4], Jiaming Wang[1,2], Lin Tian[3,4], Jianwei Liu[1,2]
[1]School of Cyber Science and Technology, Beihang University, China
[2]Key Laboratory of Aerospace Network Security Ministry of Industry and Information Technology, China
[3]Institute of Computing Technology, Chinese Academy of Sciences, China
[4]Nanjing Institute of InforSuperBahn, China
Email: {sunyv, LXYbhu}@buaa.edu.cn, sunqian@ict.ac.cn, WangJM@buaa.edu.cn,
tianlindd@gmail.com, liujianwei@buaa.edu.cn

***Abstract*—Given the large-scale deployment of 5G, rigorous testing of its core network (5GC) is essential to ensure security and robustness. Fuzzing is currently one of the most popular vulnerability discovery techniques. However, existing fuzzers suffer from low coverage of 3GPP-specified 5GC states, invalid long signaling sequence generation when exploring deep 5GC states, and coarse-grained feedback of closed-source 5G systems. This paper presents 5GC-Fuzz, a black-box fuzzing framework to detect deep stateful vulnerabilities in 5GC implementations. 5GC-Fuzz integrates three innovative techniques: (1) a systematic construction of a 5GC state machine derived from 3GPP specifications to guide the fuzzing process; (2) a 5G grammar-aware signaling sequence mutation method based on protocol stack interception to generate test cases while maximally guaranteeing the syntactic, semantic, and cryptographic correctness; and (3) a fine-grained state-transition-path feedback mechanism based on 5GC logs to optimize test states and sequences selection. The 5GC-Fuzz was evaluated on three popular 5GC implementations and achieves 152.6% more states and 206.7% more state transition paths than the state-of-the-art fuzzers. Moreover, 5GC-Fuzz exposed 22 security-critical vulnerabilities, with 6 CVEs assigned. In general, 5GC-Fuzz could explore deeper states and uncover more vulnerabilities in 5GC, significantly enhancing the security of mobile communication infrastructures.**

***Index Terms*—5G Core Network, fuzzing, vulnerabilities.**

I. Introduction

The 5G Core Network (5GC) is essential for the infrastructure of 5G technology, which requires extensive testing to verify its robustness and security. The Non-Access Stratum (NAS) and Next Generation Application Protocol (NGAP), which operate through the N1 and N2 interfaces, are critical components. Flaws in these protocols could jeopardize the entire network's security and functionality. Given the predominantly closed-source configuration of the 5GC, internal testing poses significant challenges. However, the N1 and N2 interfaces, which are designed for interactions with external devices, provide viable points for external testing. Concentrating on these interfaces for evaluating the NAS and NGAP protocols offers a focused approach to assess the most vulnerable and essential aspects of the 5GC.

Yu Sun and Xinyu Liu contribute equally.
Corresponding author: Qian Sun.

Fuzzing has emerged as one of the most effective techniques for detecting vulnerabilities in network protocol implementations. Given a target protocol program, a fuzzer works by continuously generating packets and sending them to the program while observing for potential anomalies. State-of-the-art tools for general-purpose network protocol fuzzing, such as AFLNET [1], SGFUZZ [2], and NSFUZZ [3], instrument the code of the target program and utilize code coverage for real-time feedback. However, the close-sourced nature of 5G systems renders these tools ineffective. Some recent approaches [4]–[10] have focused on fuzzing the 5G system, but they still suffer from the following problems:(1) These methods overlook the internal states and state transitions of the 5GC. However, the 5GC system is stateful and state transitions significantly affect the execution of the core network protocol stack. Vulnerabilities often reside within deeper states. He *et al.* [4] tackles this problem by transitioning the core network state during testing but limits its focus on only four states, which is insufficient for thorough testing. (2) Current strategies fail to generate test cases that can effectively penetrate deeply into the 5GC protocol logic. Due to the complexity of the 5G signaling message format and contextual dependencies, the use of random mutation often results in syntactic or semantic corruption, leading to early rejection by the protocol stack and rendering further tests meaningless. Secchecker [9] generates counter examples against security properties based on the predefined rules, but this requires tedious manual efforts. Beyond that, the 5G protocol includes numerous cryptographic operations, such as responding to AMF challenges during authentication or securing NAS messages, which are not sufficiently addressed in existing research. (3) Existing methods do not incorporate feedback mechanisms during the fuzzing process, which leads to inefficient and blind testing. To optimize fuzzing, Garbelini *et al.* [11] intercepts messages to label them by type and direction and leverages the labels as state feedback, yet this approach remains too coarse-grained.

This paper proposes a novel 5GC black-box fuzzing framework named 5GC-Fuzz to identify deep stateful vulnerabilities in 5GC implementations. To realize this framework, several challenges must be overcome: (C_1) ***States***: Achieving thorough 5GC state coverage requires a state machine to

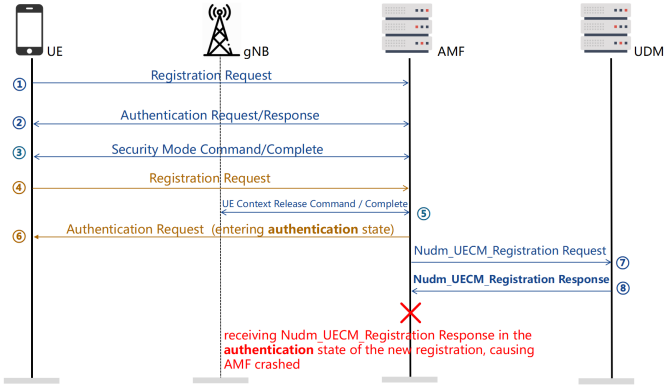

Fig. 1: Discovering AMF Unexpected Message Crash.

guide the fuzzing process, but this is challenging due to the extreme complexity of 5GC protocols and the lack of a formal specification. (C_2) ***Mutation***: To generate effective long signaling sequences as test cases, guaranteeing the syntactic, semantic, and cryptographic correctness is essential but non-trivial. In-depth fuzzing requires a mutation strategy that considers the packet format, the contextual dependency, critical timing constraints and 5G security contexts all together with diverse mutation operators. (C_3) ***Feedback***: To optimize fuzzing, developing a fine grained feedback mechanism that only leverages limited information from the black-box 5GC implementation is inherently demanding.

Figure 1 illustrates the steps of triggering the ***Unexpected Message Crash*** in Open5GS [12] (*CVE-2023-50019*), which first discovered by 5GC-Fuzz. Finding this vulnerability is exceedingly challenging, as it necessitates the simultaneous satisfaction of three specific conditions: (1) during the initial registration process, completing the 5G authentication and security mode selection, followed by initiating the registration with the UDM (blue steps); (2) re-transmitting the registration message at an appropriate moment to start a second registration process, thereby inducing the core network into the authentication state (brown steps); and (3) the AMF receiving a particular signaling message in an inappropriate state (Step ⑧), all of which require the long signaling sequence depicted in Steps ①-⑧. The issue occurs at Step ⑧, where the AMF processes the response message from the UDM for the previous registration process. This signaling triggers an assertion failure in the 5GC implementation, leading to a crash of the AMF. Notably, this vulnerability was not detected by state-of-the-art fuzzing tools during comparative experiments for several reasons: (1) these tools are stateless or employ incomplete state machine learning, and thus are unable to trigger complex state transitions; (2) their blind testing approach uses coarse-grained feedback, preventing efficient selection of specific states (such as authentication and security mode state) and sequences (such as registration requests); and (3) entering deep protocol state requires valid long signaling sequences, but these tools fail to account for the syntactic, semantic, or cryptographic issues, leading to premature rejection of their generated test cases by the 5G protocol stack.

To address the aforementioned challenges, 5GC-Fuzz utilizes three key techniques. First, to focus on exploring 5GC state space, a 5GC state machine is developed based on the 3GPP Specifications. This state machine guides the fuzzing process and assesses behavior correctness of the core network. Second, to generate effective test cases, 5GC-Fuzz implements a 5G grammar-aware signaling sequence mutation method based on protocol stack interception. It takes control of the access network, intercepts traffic and injects mutated signaling sequences. This method enables flexible sequence-level mutation operators like message delay, re-transmission and disordering while maintaining the syntactic, semantic and cryptographic correctness. Finally, to efficiently select test states and signaling sequences, 5GC-Fuzz employs a state-transition-path feedback mechanism, utilizing a novel coverage metric, state transition path, as fuzzing feedback. This metric, derived from 5GC implementation logs, replaces traditional code coverage and consists of hash values that represent multiple state labels, detailing state transitions of the core network during test iterations.

5GC-Fuzz has been implemented with UERANSIM [13]. It not only provides a comprehensive evaluation of network security and performance but also accommodates the proprietary constraints of 5GC systems. In summary, this paper yields three main contributions:

- To improve fuzzing in 5GC implementations, three innovative techniques are proposed: (1) a systematic construction of a 5GC state machine derived from 3GPP specifications to guide the fuzzing process; (2) a 5G grammar-aware signaling sequence mutation method based on protocol stack interception to generate test cases while maximally guaranteeing the syntactic, semantic, and cryptographic correctness; and (3) a fine-grained state-transition-path feedback mechanism based on 5GC logs to optimize test states and sequences selection.
- Based on the three techniques, a novel fuzzing framework named 5GC-Fuzz is designed to effectively test 5GC implementations. To our best knowledge, 5GC-Fuzz is the first comprehensive approach to systematically and automatically fuzz arbitrary 5GC implementations.
- 5GC-Fuzz is evaluated on three popular 5GC implementations (Open5GS, Free5GC and OAI-5GC), and find 22 real vulnerabilities (including 19 crashes and 3 semantic problems). All of these vulnerabilities have been confirmed and fixed by related developers. 6 CVEs are assigned due to their severe security impact. Moreover, when compared with the state-of-the-art fuzzing approaches, 5GC-Fuzz finds more real vulnerabilities missed by these approaches with higher testing coverage.

The rest of the paper is organized as follows. Section II introduces the background and related work. Section III introduces the framework and key techniques of 5GC-Fuzz. Section IV shows the evaluation and compares 5GC-Fuzz to existing fuzzing tools and Section V concludes this paper. **Responsible disclosure:** We have followed the responsible

disclosure policy and all the vulnerabilities have been confirmed and fixed by related developers.

II. Background and Related Work

In this section, we introduce the 5G core network and fuzzing methods in mobile communication networks.

A. 5G Core Network

Key Network functions (NFs) within the 5GC include the Access and Mobility Management Function (AMF), which manages UE registration and mobility; the Session Management Function (SMF), responsible for session establishment and management; the Authentication Server Function (AUSF) and the Unified Data Management (UDM) play crucial roles in the 5G authentication process, ensuring secure user verification and data management. All the network functions work together seamlessly, enabling the efficient operation and management of the 5G network, as shown in Figure 2.

Central to the operation of the 5GC are the NAS and NGAP protocols. The NAS protocol operates over the N1 interface between the UE and the AMF, managing mobility and session states, including authentication, security control, and bearer management. The NGAP protocol facilitates signaling over the N2 interface between the gNodeB(5G base station) and the AMF, handling procedures such as UE context management, registration, paging, and handover. Given their critical roles, ensuring the robustness and security of the NAS and NGAP protocols is paramount.

This paper focuses on fuzzing these protocols by intercepting and mutating NAS and NGAP packets at the N1 and N2 interfaces, respectively, to identify and mitigate potential vulnerabilities within the 5GC. The testing entry point is the AMF, the boundary network function of the 5GC control plane, making this approach suitable for closed-source 5G systems due to the AMF's exposure to external N1 and N2 interfaces.

B. Fuzzing for Mobile Communication Networks

Fuzzing is a crucial technique for ensuring the security and reliability of communication protocols. It works by providing malformed, unexpected, or random data as input to the protocol program being tested. The goal is to observe how the program handles these inputs and to identify flaws that could be exploited by malicious actors. Previous works such as SPIKE [14], PEACH [15] and AFLSMART [16] have explored semantic-aware fuzzing and syntax tree mutations for network protocols [17], but these approaches struggle to handle the complex state transitions and cryptographic requirements inherent in 5G signaling procedures.

In recent years, several works [4]–[11], [18]–[21] have utilized fuzzing to uncover potential attacks and threats in mobile communication systems like 4G and 5G. Some works focus on individual packet generation but lose sight of state or context information. AMFuzz [7] and 5GReplay [8] mutate existing traffic to generate test cases, but the mutated packets can be easily dropped by the protocol stack due to lack of the format specification. T-FUZZ [21] and Berserker [20] utilize

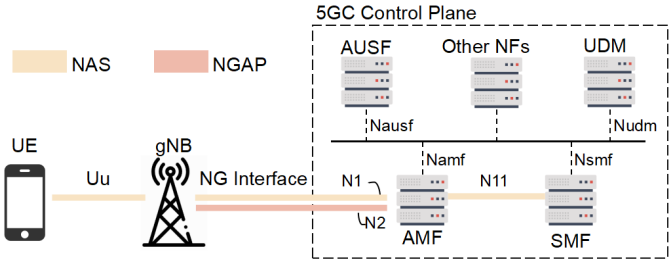

Fig. 2: 5G Architecture with boundary NFs.

structured mutation methods to generate syntactically valid test cases, but they do not consider the contextual dependency or cryptographic procedures. LTEFuzz [18] and SecChecker [9] manually generate test cases that violate security properties to reveal vulnerabilities in the 4G and 5G systems. However, they require tedious manual efforts. Moreover, without state information, these approaches may miss vulnerabilities that only manifest after a specific sequence of messages or state transitions. Additionally, the absence of the feedback mechanism further limits their effectiveness. To deal with feedback, Hu *et al.* [6], which targets the 5GC, employs a grey-box approach using code coverage as feedback to optimize its proposed field-weight table for packet field mutations. However, this method is not applicable to closed-source 5G systems. Some recent works also introduces state awareness for 4G and 5G protocols. DoLTEst [19] and He *et al.* [4] select specific states for testing during the fuzzing process. However, they lack adequate feedback mechanisms to avoid the blind state selection. Garbelini *et al.* [11] uses predefined mapping rules to parse the packet types in the responses from the target, forming a state machine to guide the testing process. Nevertheless, this approach remains coarse-grained.

In comparison, 5GC-Fuzz is a stateful approach, utilizing its 5GC state machine to thoroughly explore the state space. With the novel 5G grammar-aware mutation method based on protocol stack interception, it can generate syntactically, semantically, and cryptographically valid signaling sequences. Without instrumentation, 5GC-Fuzz provides a state-transition-path feedback mechanism by leveraging the information of the 5G logs to navigate protocol-logic exploration.

III. Framework and Key Components

In this section, we first present an overview of 5GC-Fuzz with its main components and workflow. Then we illustrate the important components in detail.

A. Architecture and Workflow

System Architecture. Figure 3 illustrates the architecture of 5GC-Fuzz, which is composed of four main modules: (i) the Fuzzing and Feedback components, which guides the fuzzing process and provides results and feedback; (ii) the State and Sequence Selector, which selects test states and the corresponding signaling sequences that can transition the core network to these states; (iii) the Mutator, which mutates the

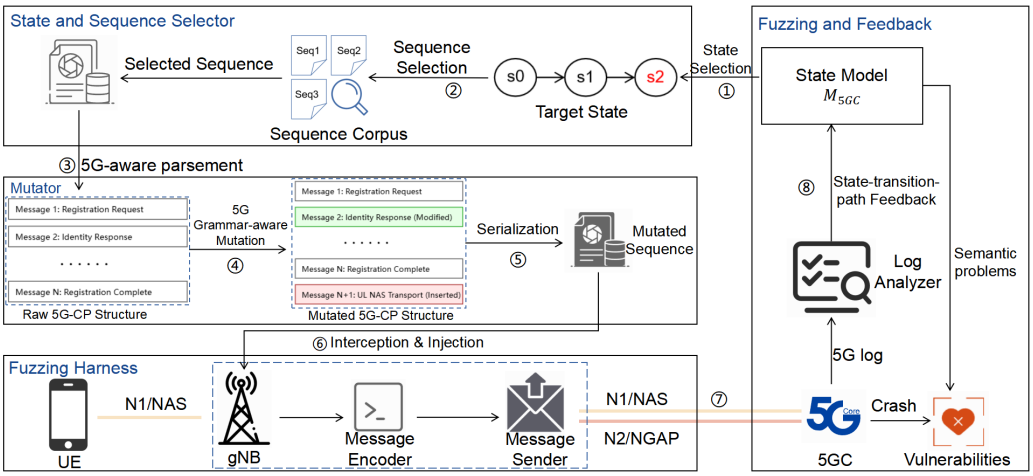

Fig. 3: Architecture of 5GC-Fuzz.

signaling sequences; and (iv) the Fuzzing Harness, which allows the fuzzer to fully control the live communication process between the access network and the core network, sending the mutated signaling sequences to the core network. The components of 5GC-Fuzz interact and collaborate with each other to attain the aim of discovering potential vulnerabilities in the implementation of 5GC.

Initialization. The 5GC state machine and the sequence corpus are constructed first. The fuzzer is under the guidance of the 5GC state machine model M_{5GC} to select a target state. The sequence corpus, collected from individual requests in the correct order from captured network traffic sent to the core network by triggering functionalities using CLI commands with UERANSIM, is used to select sequences corresponding to different states.

Fuzzing Iteration.: 5GC-Fuzz leverages information from the M_{5GC} to select the next state to focus on (Step ①). A state is given higher priority if: (1) it has previously contributed successfully to state coverage, or (2) it is rarely selected, to identify blind spots that might reveal new state transitions and vulnerabilities. Once a target state s is chosen, the fuzzer selects a signaling sequence M from the sequence corpus that can reach state s (Step ②). To ensure that the mutated sequence M' can exercise the chosen state s, our fuzzer splits the original sequence M into three parts: (1) the prefix M_1 is required to reach the selected state s, (2) the candidate subsequence M_2 contains all messages that can be executed after M, and (3) the suffix M_3 is simply the leftover subsequence such that $\langle M_1, M_2, M_3 \rangle = M'$ [1]. During the mutation stage, a 5G grammar-aware sequence mutation method generates the mutated message sequence. Raw data is first parsed into a 5G Control Plane (5G-CP) message structure that preserves both NGAP and NAS protocol hierarchies according to 3GPP specifications [22], [23]. After applying mutation operations on this structure, the modified messages are then serialized back into protocol-compliant binary format (Steps ③-⑤). The mutated seed input is then prepared for fuzzing. Normal communication traffic is intercepted, the correct encryption

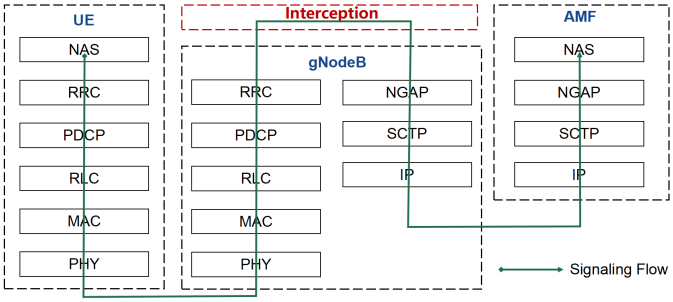

Fig. 4: Simplified Signaling Process in the 5G Protocol Stack.

and integrity protection calculations are performed, and the original packets are replaced with these mutated sequences before being sent to the core network (Steps ⑥-⑦). 5GC-Fuzz focus on the boundary network functions in the 5GC. The signaling flow from the UE to the core network is illustrated in Figure 4. 5GC-Fuzz is fundamentally an interceptive approach [7]. The insertion point is established at the NGAP layer, shown in Figure 4, to modify the corresponding messages (NAS messages are encapsulated within NGAP messages as protocol data units). Utilizing UERANSIM, 5GC-Fuzz has gained complete control over the security context in real-time communications. This ensures the correctness of security algorithms, preventing messages from being discarded by the protocol stack due to encryption or integrity protection issues.

Subsequently, the state of the core network and its logs are observed to determine if a vulnerability has been detected. State coverage feedback is derived from the logs for each testing iteration, which optimizes subsequent state and sequence selections (Step ⑧). This heuristic approach increases the probability of selecting states or sequences that have led to higher coverage in previous tests. The fuzzing iteration ends if: (1) all messages of the test sequence have been sent to the AMF, (2) a vulnerability is detected, or (3) the fuzzer times out based on a predefined time limit.

Vulnerability detection. For crashes, 5GC-Fuzz checks if network functions are still alive by ping testing. As for semantic vulnerabilities, theoretically, the state transition paths

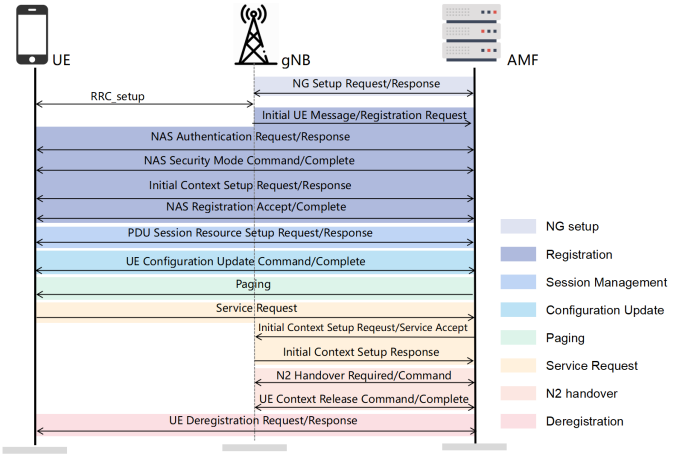

Fig. 5: The 8 important core network procedures.

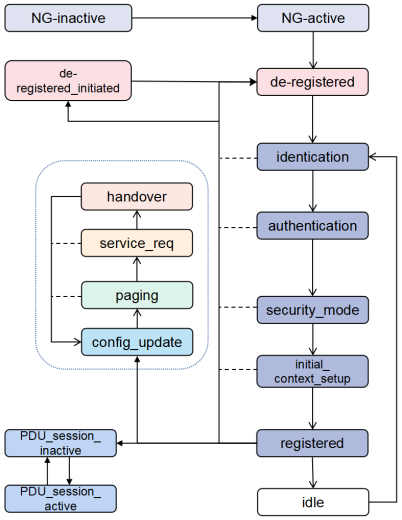

Fig. 6: Simplified representation of the state machine.

derived from the logs can be fully mapped to paths in the 5GC state machine. Therefore, if a state transition path causes corruption in the state machine, a semantic vulnerability may have occurred, which will be analyzed further.

B. Threat Model

The threat model considers an active attacker who possesses all UE cryptographic keys and can perform legitimate cryptographic operations. The attacker can modify, inject, replay and drop packets on the channel between the access network and the core network, with the goal of discovering implementation flaws in the core network.

C. The 5GC State Machine Model

5GC-Fuzz aims to detect flaws in 5GC implementations. Particularly, 8 important core network procedures including NG setup, registration, session management, deregistration, configuration update, service request, paging and N2 handover are studied as illustrated in Figure 5. 5GC-Fuzz works during this process and it is guided by a 5GC state machine.

Based on 3GPP specifications [22]–[24], the protocol states and behaviors are modeled on the boundary network functions in the network side. The state machine is essentially a quintuple, comprising a finite set of input variables, a finite set of output variables, a finite set of state variables, an initial state set, and a finite set of assignments for state variables. The state machine updates the state variables based on the assignment set, thereby achieving system state migration [25]. The state machine serves the two purposes: (1) guiding the fuzzing process, and (2) providing subsequent feedback and verification of core network behaviors (to assess the reasonableness of behaviors that violate the state machine). Specifically, the state machine defines 16 states and 79 state transitions. A simplified representation of the state machine is presented in Figure 6.

Initially, the core network starts in the ***NG-inactive*** state. After successful NG Setup between gNB and AMF, it transitions to the ***de-registered*** state. When gNB forwards the UE's Registration Request via Initial UE Message, the AMF moves to the ***identification*** state if the identity is unknown, requesting UE's SUCI. Upon receiving the identity response, AMF enters the ***authentication*** state and initiates the authentication procedure with the UE by sending the key selector, RAND and AUTN to the UE. The UE then responds to the authentication challenge. If the authentication is successful, UE and AMF will establish a security context in NAS messages. Then the network transitions to the ***security_mode*** state to establish NAS security algorithms, then proceeds to the ***initial_context_setup*** state and initiates a session setup with the gNB by sending the Initial Context Setup Request message. The Initial Context Setup Response message notifies AMF to establish the UE context. Finally, the core network sends the Registration Accept message to UE informing that the core network accepts initial registration. The core network proceeds to the ***registered*** state. When UE requests PDU session establishment, the network moves to the ***PDU_session_inactive*** state, followed by the ***PDU_session_active*** state upon acceptance. During periods of UE inactivity, AMF transitions to the ***idle*** state.

After successful attachment, UE can start other important procedures. The network enters the ***configuration_update*** state when updating UE context with new 5G-GUTI, TAL, service area list or LADN information. For paging, when the network needs to signal a UE in the CM-IDLE state [22], it enters the ***paging*** state and awaits a Service Request response. During N2 handover, the network transitions to the ***handover*** state as it manages the transfer between source and target gNBs. For deregistration, the network transitions to the ***de-registered-initiated*** state upon receiving a UE Deregistration request, eventually returning to the ***de-registered*** state.

D. Mutation with Protocol Stack Interception

1) 5G grammar-aware mutation: Due to the highly complex logic of 5G protocols, particularly in NGAP and NAS layers, traditional fuzzing approaches that rely on blind message mutations often corrupt protocol logic, preventing the core network from reaching deeper states and resulting in low testing efficiency. To address this challenge, a 5G grammar-

aware mutation method is proposed based on protocol stack interception to preserve 5G protocol semantics.

a) Structure of the 5G Signaling Sequence: The 5G signaling sequence follows a hierarchical structure in which NGAP messages are encoded according to the ASN.1 notation, while NAS messages adopt the TLV (Type-Length-Value) encoding. During the parsing process, the raw protocol messages are converted into a hierarchical 5G-CP structure that preserves these encoding characteristics. The structural representation of the 5G signaling sequence is illustrated in Figure 7. Let $\mathcal{S}$ denote a valid 5G signaling sequence, where each sequence consists of multiple NGAP messages $M_k \in \mathcal{S}$ with their corresponding delay parameters δt_k that controls its transmission timing. Each NGAP message comprises a structure where the message header contains Procedure Code and Length, followed by Information Elements (IEs). Among these IEs, the NAS-PDU is a special IE that encapsulates the NAS layer messages. The NAS-PDU follows a bit-level format where each field has specific attributes:

- Type attribute t_i: defines the parameter type according to NAS specifications
- Mutability flag $\mu_i \in \{0, 1\}$: determines if the parameter can be mutated

Taking the Initial Registration Request as an example, the NAS-PDU contains mandatory fields such as Extended Protocol Discriminator (EPD), Security Header Type, Message Type, and 5GS Registration Type, each with their corresponding type attributes and mutability flags. Some fields, like EPD and 5GS Mobile Identity, can be marked as immutable ($\mu = 0$) to preserve protocol semantics, while others, such as Security Header and UE Security Capability, are mutable ($\mu = 1$) to enable fuzzing exploration.

b) 5G Grammar-Aware Mutation Strategy: Based on the hierarchical structure defined above, two types of mutation operators are proposed:

Packet-Level Mutation Operator. This operator performs field-wise mutations within a single message. For mutable NAS parameters ($\mu_i = 1$), it enables bit-level modifications such as bit flips and boundary value testing, while respecting parameter dependencies such as sequence numbers and UE identity information within the same registration process. This strategy ensures thorough testing of the parameter space while maintaining protocol compliance.

Sequence-Level Mutation Operator. This operator modifies message sequences through three atomic operations:

$$\begin{aligned} D(S, i) &= \{M_1, ..., M_{i-1}, M_{i+1}, ..., M_n\} \\ I(S_1, S_2, i) &= \{M_1, ..., M_i, M', M_{i+1}, ..., M_n\} \\ R(S_1, S_2, i) &= \{M_1, ..., M_{i-1}, M', M_{i+1}, ..., M_n\} \end{aligned} \quad (1)$$

where D, I and R represent deletion, insertion, and replacement operations respectively. Based on these mutation operators, 5GC-Fuzz can achieve sequence-level mutations such as packet delay, duplication, disordering and flooding. Before these mutated signaling packets are sent to the core network for testing, they require additional processing, such as encryption and integrity protection. This is accomplished through the subsequent interception approach.

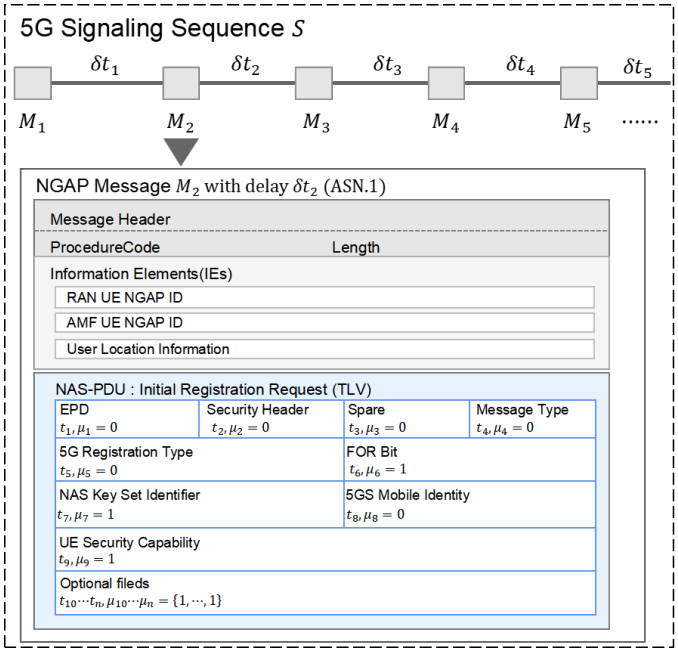

Fig. 7: Hierarchical structure of the 5G signaling sequence.

2) Interception approach: The essence of 5GC-Fuzz is an interceptive approach which operates during real-time communication between the access network and the core network by intercepting traffic and injecting mutated signaling sequences. The entire 5G communication process encompasses several procedures that necessitate meticulous attention. For instance, during the Authentication and Key Agreement (AKA) phase, the AMF dispatches a fresh challenge (i.e., never sent before), and the UE must respond with a result strictly tied to the challenge [7]. Furthermore, all NAS messages are encrypted and integrity-protected following the completion of the NAS security procedure. To ensure that the core network processes the mutated signaling sequences correctly without prematurely discarding them, full control of the 5G access network is achieved by registering hooks in UERANSIM. UERANSIM is a state-of-the-art open-source 5G UE and RAN simulator Specifically, the *sendNgapUeAssociated* and *sendNgapNonUe* functions in the UERANSIM gNB are employed to send NAS and NGAP packets to the core network respectively. The two functions are modified to send the desired signaling sequences in the correct order. For NAS messages, all necessary data (e.g., session keys, RAND value, and integrity algorithm) are acquired from the startup configuration and real communication traffic. This ensures correct responses to the core network's challenges during the authentication phase and enables the performance of encryption and integrity protection calculations. Subsequently, it is ensured that all messages maintain their integrity and correctness throughout the transmission process, including accurate MAC values for SCTP messages.

In summary, the mutation methodology maximizes syntac-

tic, semantic, and cryptographic correctness. Specially, the 5G grammar-aware mutation approach ensures that generated test cases maintain protocol validity. The delay attribute accounts for the critical timing of packet transmission, while the mutable attribute ensures the correctness of related semantics. Additionally, by utilizing the interception approach, it ensures that the cryptographic processes are correctly handled by taking full control of the control network. It can be extended to other 5G protocols following ASN.1 or TLV encoding.

E. State-Transition-Path Feedback Mechanism

Due to the closed nature of 5G protocol stacks, code coverage cannot serve as feedback to guide the fuzzing process. For a given input signaling sequence, the 5GC protocol's state is always changed when handling each message in this sequence. Furthermore, if protocol behaviors are different, the transition between two states should not be treated as the same. Through in-depth investigation of the log systems of 5GC implementations, it is found that these log systems are very similar and provide detailed demonstrations of the program's comprehensive state and its handling of transmitted and received messages. This makes them sufficient for use as feedback for fuzzing. Inspired by this observation, a new coverage metric named state transition path(STP) is proposed to describe both states and state transitions based on the 5G log analyzer. For an iteration of testing, the state transition path is a hash value formed by concatenating multiple state segments extracted from the logs.

Figure 8 illustrates the method for obtaining state transition paths. Three types of predicates including validity predicates, presence predicates and grouping predicates are employed in order to accurately describe the 5G protocol stack behavior [25]. Validity predicates encompass elements such as MAC and the validity of RES parameters (e.g., xres_matched_sres). Presence predicates cover components like subscription permanent identifier (SUPI) and the presence of NAS security contexts (e.g., security context completed or not exists). Grouping predicates include categorization of core network rejection reasons for user connectivity requests. All these predicates and incoming and outgoing messages can be considered as environmental variables.

First, the logs are summarized (Step ①), removing unnecessary information such as timestamps and line numbers, and then use regular expressions to filter out key information already defined in the state model, such as program states, messages being processed, messages sent, and key environmental variables. Subsequently, the summarized logs are segmented into individual fragments. The program output, i.e., packets, can be adopted as indicators of its internal state. For example, if the AMF sends an Identity Request message, it can be considered to be in the identification state; if the AMF sends an Authentication Request message, it can be considered to have completed the identity acquisition of the UE and started the AKA authentication process. The logs between two sent messages are referred to as a state fragment. As shown in Figure 8, in both iteration 1 and iteration 2, the logs are

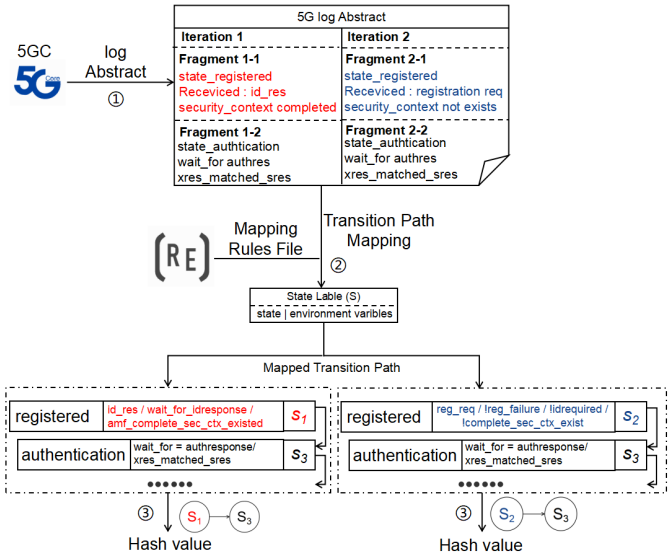

Fig. 8: State-Transition-Path feedback mechanism.

segmented into fragments representing the state registered and the state authentication. Predefined mapping rules are later used to map state fragments to independent log labels (Step ②). The mapping rules are straightforward: based on the order of appearance in the logs, a state label is characterized as a concatenation of specific 5G states and environmental variables, consistent with the 5GC state machine described in Section III-C. As shown in Figure 8, both transitions are from the state registered to the state authentication, but they are treated as two different transitions because the state label for registered is different. Thus, the state transition corresponds to the transition between two state labels, which is also an edge in the core network state machine. After an iteration of fuzz testing, a complete log is parsed, and multiple state labels are obtained. Finally, the hash value of the entire state transitions is calculated (Step ③), representing the state transition path for a fuzzing iteration. Each hash value represents an independent state transition path in the state machine. The testing goal is to obtain as many unique state transition paths as possible, which implies exploring more paths and increasing the likelihood of discovering vulnerabilities. In subsequent tests, paths that have been rarely tested before or are more likely to lead to new paths will be selected as test objects.

IV. Evaluation

To evaluate the efficiency of 5GC-Fuzz, we answer the following research questions (RQs):

- RQ1, **State space exploration ability**: How effective is 5GC-Fuzz in terms of covering state space in closed-source 5GC implementations compared to existing fuzzing tools?
- RQ2, **Ablation study**: How do the state-transition-path feedback mechanism and the 5G grammar-aware mutation method contribute to the effectiveness of 5GC-Fuzz?

TABLE I: Basic information about the benchmark.

Implementation	Version	Language
Open5GS	v2.7.0	c
OAI-5GC	v2.0.1	c
Free5GC	v3.4.1	Go

TABLE II: Testing coverage of 5GC-Fuzz.

5GC	States	Transitions / Model coverage	STP
Open5GS	16	40 / 50.6%	2047
Free5GC	16	44 / 55.7%	2372
OAI-5GC	16	35 / 44.3%	1813

TABLE III: Comparison results of 5GC-Fuzz and the baselines.

5GC	5GC-Fuzz		5GReply				He *et al.*				AFLNET			
	state	STP	state	Improve	STP	Improve	state	Improve	STP	Improve	state	Improve	STP	Improve
Open5GS	16	2047	4	300%	125	1537.6%	6	166.7%	508	303.0%	9	77.8%	1402	46.0%
Free5GC	16	2372	4	300%	143	1558.7%	6	166.7%	512	363.3%	-	-	-	-
OAI-5GC	16	1813	4	300%	98	1750.0%	6	166.7%	427	324.6%	9	77.8%	1453	24.8%
AVG	-	-	-	300%	-	1615.4%	-	166.7%	-	330.3%	-	77.8%	-	35.4%

- RQ3, **Real vulnerabilities finding ability**: How effective is 5GC-Fuzz in terms of finding new vulnerabilities in closed-source 5GC implementations?

A. Experimental Setup

Benchmark: Three most widely-used 5GC implementations, including Open5GS [12], Free5GC [26], and OAI-5GC [27], are used as the benchmark. Table I shows the basic information about the implementations.

Baselines: To make the comparison, AFLNET [1], 5GReplay[2] and He *et al.* [4], which most closely align with our research, are selected as the baselines. To enable 5GC testing, two protocol interface functions in AFLNET, *extract_requests* and *extract_response_codes*, are modified to recognize the basic message formats of NAS and NGAP with correct cryptographic procedures for a more fair comparison. Each target is fuzzed with different fuzzers for 48 hours for reporting the testing coverage (including states, state transitions and state transition paths) and the discovered vulnerabilities.

All experiments were performed on a virtual machine running Ubuntu 20.04 LTS with 8GB RAM and 4 CPU cores.

B. State Space Exploration (RQ1)

To evaluate the state space exploration ability of 5GC-Fuzz, the state coverage of different fuzzers is checked. Table II shows the overall results of testing coverage of 5GC-Fuzz. Specifically, the 5GC state machine model employs a total of 16 states and 79 valid transitions. 5GC-Fuzz covers all the states defined in the model. The third column of Table II holds the number of different transitions traversed between states. Each implementation traverses the model differently and does not trigger all possible valid transitions. This is because these implementations do not complete all possible 5GC functions. For example, Open5GS does not implement emergency UE registration.

State and State Transition Path. Table III and Figure 9 show the comparison of state coverage and state transition paths (STP) between 5GC-Fuzz and the baselines. As shown in the results, 5GC-Fuzz covered a greater number of states across all tested core network implementations. On average, 5GC-Fuzz achieves 152.6% more states and 206.7% more state transition paths than the state-of-the-art fuzzers. Overall, 5GC-Fuzz demonstrates superior performance over the baselines in both state coverage and state transition path exploration.

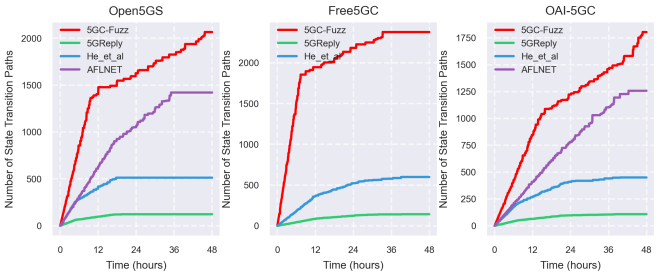

Fig. 9: State-Transition-Path growth of various fuzzers.

Branch coverage. Although our target is closed-source 5G core networks, these open-source core network implementations provided us with the opportunity to measure code branch coverage. Only the AMF is instrumented, as it is the main network function handling NAS and NGAP, thereby enhancing efficiency. Since AFLNET does not support instrumentation for Go programs, branch coverage is not collected for free5GC. Table IV presents the branch coverage (Br.) results of the fuzzers. Although branch coverage was not used as feedback in 5GC-Fuzz, its branch coverage outperformed AFLNET by 19.0%, 5GReply by 71.1% and He *et al.* by 46.2% across the two implementations. Higher branch coverage tends to correlate with a higher probability of vulnerability discovery. This demonstrates the robust performance of 5GC-Fuzz in testing closed-source black-box 5GC implementations.

TABLE IV: Code branch coverage compared with baselines.

5GC	5GC-Fuzz	AFLNET		5GReplay		He *et al.*	
	Br.	Br.	Imp.	Br.	Imp.	Br.	Imp.
Open5GS	52686	40666	29.6%	28759	83.2%	34105	54.5%
OAI-5GC	39647	36584	8.4%	24943	58.9%	28746	37.9%
AVG	-	-	19.0%	-	71.1%	-	46.2%

C. Ablation Study (RQ2)

Based on the 5GC state machine model, 5GC-Fuzz comprises two key components: (1) an automated 5G grammar-aware signaling mutation method, and (2) a state transition path feedback mechanism based on 5G logs. To evaluate the contribution of each component to the increase in state coverage, an ablation study is conducted. Specifically, based on the 5GC state machine and the interception approach, two tools are developed: (1) 5GC-trivial performs random

TABLE V: Improvements in terms of state-transition-paths compared with the baseline.

5GC	5GC-trivial (Baseline)	5GC-rand		5GC-Fuzz	
		Value	Improve	Value	Improve
Open5GS	1358	1735	27.8%	2047	50.7%
Free5GC	1743	1974	13.2%	2372	36.1%
OAI-5GC	1278	1523	19.2%	1813	41.9%
AVG	1460	1744	19.5%	2077	42.3%

TABLE VI: Vulnerabilities found.

5GC	5GC-Fuzz	AFLNET	5GReply	He *et al.*
Open5GS	13	5	1	2
Free5GC	6	1	1	0
OAI-5GC	3	0	0	0
Total	22	6	2	2

TABLE VII: Statistics of 10 important vulnerabilities discovered by 5GC-Fuzz.

ID	5GC	Model state	Impact	Root cause	Status
1	Open5GS	ng-active	Crash/DoS	Broken pipe	CVE-2023-50020
2	Open5GS	de-registered	Crash/DoS	Buffer-Overflow when handling pfcp message pool	CVE-2024-40129
3	Open5GS	de-registered	Crash/DoS	Memory leak	CVE-2024-33382
4	Open5GS	authentication	Crash/DoS	Assertion failure when handling Nudm response	CVE-2023-50019
5	Open5GS	registered	Crash/DoS	Assertion failure when handling UE Context transfer	CVE Requested
6	Open5GS	all states	Crash/DoS	Buffer-Overflow when handling message encoding	CVE Requested
7	Free5GC	de-registered	Crash/DoS	Null pointer de-reference	CVE Requested
8	Free5GC	pdu_sess_inactive	Crash/DoS	Overflow when handling PDU session setting up	CVE Requested
9	Free5GC	registered	DoS	Improper Authentication	CVE Requested
10	OAI-5GC	pdu_sess_active	Crash/DoS	Assertion faliure when handling PDU session releasing	CVE Requested

mutations on signaling sequences and randomly selects test states and sequences without feedback, and (2) 5GC-rand performs grammar-aware mutations but without any feedback mechanism. Table V presents the results of the ablation study. Overall, both key components significantly increased the number of discovered state transition paths during testing. Specifically, compared to the baseline, the number of state transition paths increased by an average of 19.5%. When combined with the feedback mechanism, the increase reached 42.3%. These results demonstrate that the integration of both strategies makes our approach the most effective.

D. Vulnerabilities Finding (RQ3)

In this experiment, the utility of 5GC-Fuzz is evaluated by checking whether it is able to discover zero-day vulnerabilities in the benchmark. The results are shown in Table VI. 5GC-Fuzz found a total of 22 unique and previously unknown vulnerabilities in the benchmark. CVE IDs have been requested for these vulnerabilities. In comparison, 5Greply and He *et al.* each identified only 2 vulnerabilities, while AFLNET found 6 vulnerabilities, resulting in a total of 7 unique vulnerabilities discovered by these tools. Notably, 5GC-Fuzz successfully discovered all these 7 vulnerabilities, along with 15 additional vulnerabilities that were missed by the other tools. Indeed, 5Greply only performs message reply, neglecting the state of the core network, the mutation strategy of AFLNET leads to corruption in syntax, semantics, and cryptographic issues and He *et al.* only focus on a few states, which preventing them from finding more deep stateful vulnerabilities.

Influences of the found vulnerabilities. From the initial ng-active state to the deeper pdu_sess_inactive state, vulnerabilities are dispersed across various states of the core network, demonstrating the effectiveness of 5GC-Fuzz in thoroughly exploring core network states. All these vulnerabilities stem from implementation issues. Among these, 19 vulnerabilities result in immediate crash of the core network, causing a denial of service. Additionally, three other semantic vulnerabilities include two that force the UE to go offline and one that causes the implementation to deviate from 3GPP specifications. 10 significant vulnerabilities have been detailed in Table VII. Overall, 5GC-Fuzz has the ability to uncover new vulnerabilities across all tested implementations.

V. Conclusion and Future Work

This paper proposes a novel black-box fuzzing framework named 5GC-Fuzz, to effectively detect deep stateful vulnerabilities in 5GC implementations. Guided by a meticulously constructed 5GC state machine, 5GC-Fuzz utilizes a 5G grammar-aware signaling mutation method based on the protocol interception for test case generation. Additionally, 5GC-Fuzz employs a state-transition-path feedback mechanism to efficiently select test states and signaling sequences. 5GC-Fuzz was evaluated on three well-known 5GC implementations and uncovered 22 previously unknown vulnerabilities. Compared to existing fuzzing tools, it is able to identify more vulnerabilities with higher state coverage, which significantly enhances the security of mobile communication infrastructures, offering a robust safeguard against potential vulnerabilities.

Limitations and future work. One key limitation is that constructing the 5GC state machine requires substantial domain expertise. By solely relying on 3GPP specifications, it may miss vulnerabilities that exist outside the standard protocol paths. The state-transition-path feedback mechanism may not be generalizable across all implementations, particularly for closed-source commercial systems. In the future, we plan to apply 5GC-Fuzz to commercial 5G core networks deployed in the field.

Acknowledgments

We thank the anonymous reviewers and the related developers for their constructive feedback and suggestions. This work was supported by the Super SIM Project.

REFERENCES

[1] V.-T. Pham, M. Böhme, and A. Roychoudhury, "Aflnet: A greybox fuzzer for network protocols," in *2020 IEEE 13th International Conference on Software Testing, Validation and Verification (ICST)*. IEEE, 2020, pp. 460–465.

[2] J. Ba, M. Böhme, Z. Mirzamomen, and A. Roychoudhury, "Stateful greybox fuzzing," in *31st USENIX Security Symposium (USENIX Security 22)*, 2022, pp. 3255–3272.

[3] S. Qin, F. Hu, Z. Ma, B. Zhao, T. Yin, and C. Zhang, "Nsfuzz: Towards efficient and state-aware network service fuzzing," *ACM Transactions on Software Engineering and Methodology*, vol. 32, no. 6, pp. 1–26, 2023.

[4] F. He, W. Yang, B. Cui, and J. Cui, "Intelligent fuzzing algorithm for 5G nas protocol based on predefined rules," in *2022 International Conference on Computer Communications and Networks (ICCCN)*. IEEE, 2022, pp. 1–7.

[5] Y. Cao, Y. Chen, and W. Zhou, "Detection of pfcp protocol based on fuzz method," in *2022 2nd International Conference on Computer, Control and Robotics (ICCCR)*. IEEE, 2022, pp. 207–211.

[6] Y. Hu, W. Yang, B. Cui, X. Zhou, Z. Mao, and Y. Wang, "Fuzzing method based on selection mutation of partition weight table for 5G core network ngap protocol," in *Innovative Mobile and Internet Services in Ubiquitous Computing: Proceedings of the 15th International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS-2021)*. Springer, 2022, pp. 144–155.

[7] F. Mancini, S. Da Canal, and G. Bianchi, "Amfuzz: Black-box fuzzing of 5G core networks," in *2024 19th Wireless On-Demand Network Systems and Services Conference (WONS)*. IEEE, 2024, pp. 17–24.

[8] Z. Salazar, H. N. Nguyen, W. Mallouli, A. R. Cavalli, and E. Montes de Oca, "5greplay: A 5G network traffic fuzzer-application to attack injection," in *Proceedings of the 16th International Conference on Availability, Reliability and Security*, 2021, pp. 1–8.

[9] C. Yu, S. Chen, Z. Wei, and F. Wang, "Secchecker: Inspecting the security implementation of 5G commercial off-the-shelf (cots) mobile devices," *Computers & Security*, vol. 132, p. 103361, 2023.

[10] 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.

[11] 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.

[12] S. Lee, "Open5gs," https://github.com/open5gs/open5gs, 2020.

[13] UERANSIM, "Ueransim," https://github.com/aligungr/UERANSIM, 2020.

[14] D. Aitel, "The advantages of block-based protocol analysis for security testing," *Immunity Inc., February*, vol. 105, p. 106, 2002.

[15] MozillaSecurity, "Peach," https://github.com/MozillaSecurity/peach, 2021.

[16] V.-T. Pham, M. Böhme, A. E. Santosa, A. R. Căciulescu, and A. Roychoudhury, "Smart greybox fuzzing," *IEEE Transactions on Software Engineering*, vol. 47, no. 9, pp. 1980–1997, 2019.

[17] F. Hu, J. Ji, H. Shu, Z. Li, T. Liu, and C. Zhang, "Formatted stateful greybox fuzzing of tls server," in *2024 IEEE Conference on Software Testing, Verification and Validation (ICST)*. IEEE, 2024, pp. 151–160.

[18] 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.

[19] 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.

[20] S. Potnuru and P. K. Nakarmi, "Berserker: Asn. 1-based fuzzing of radio resource control protocol for 4G and 5G," in *2021 17th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)*. IEEE, 2021, pp. 295–300.

[21] W. Johansson, M. Svensson, U. E. Larson, M. Almgren, and V. Gulisano, "T-fuzz: Model-based fuzzing for robustness testing of telecommunication protocols," in *2014 IEEE Seventh International Conference on Software Testing, Verification and Validation*. IEEE, 2014, pp. 323–332.

[22] 3GPP, "Non-access-stratum (nas) protocol for 5G system (5gs)," TS 24.501 V16.6.0, 3rd Generation Partnership Project (3GPP), Jul. 2020.

[23] ——, "Ng-ran; ng application protocol (ngap)," TS 38.413, 3rd Generation Partnership Project (3GPP), 2019.

[24] ——, "Security architecture and procedures for 5G system (5gs)," TS 33.501 V16.4.0, 3rd Generation Partnership Project (3GPP), Jul. 2020.

[25] 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.

[26] Free5GC, "free5gc," https://github.com/free5gc/free5gc, 2020.

[27] OAI, "Oai-5g," https://github.com/OPENAIRINTERFACE/openair-cn, 2020.