Toward a Robust Ingress for Open-Sourced 5G Core Network

Jin-Wei Hsu, Xin-Ye Jiang, I-Wei Chen, Kai-Jung Chen, Chuan Ou-Yang, and Chun-Ying Huang *Senior Member, IEEE*
Department of Computer Science, College of Computer Science
National Yang Ming Chiao Tung University

***Abstract*—The security of fifth-generation (5G) networks hinges on the robustness and reliability of their software implementations from the core network infrastructure to end-user devices. Fortifying these networks against emerging threats and vulnerabilities requires rigorous testing. This paper proposes a systematic approach for identifying flaws in 5G core network implementations. Focusing on the attack surfaces at 5G core network entry points, we identified flaws in handling the next-generation application protocol (NGAP) and nonaccess stratum (NAS) protocol with a Fuzzer-in-the-Middle (FitM) architecture that systematically evaluates multistage 5G core network protocol implementation; this architecture was applied to well-known open-source 5G core network implementations for evaluation. Specifically, the FitM architecture fuzzes valid user and base-station NAS and NGAP packets, which are then transmitted to the core network. In addition to recognizing 20 known implementation flaws, the FitM approach successfully recognized eight unknown flaws in various network functions in the core network implementations. The findings were reported to the developer community, and the issues have been fixed in most implementations. The proposed approach can be seamlessly integrated into the software development life cycle. The approach is both practical and extensible and can help communities develop more reliable core networks.**

***Index Terms*—5G, core network, fuzzing, implementation flaw, software vulnerability**

I. Introduction

THE 3rd Generation Partnership Project (3GPP) began standardizing fifth-generation (5G) mobile communication systems in 2017, publishing both standalone (SA) and non-standalone (NSA) specifications. The advent of 5G communication heralds a new telecommunications era characterized by an open and software-based design. Rooted in global standards set by the 3GPP, 5G networks leverage open architectures that foster interoperability among diverse vendors, enabling a dynamic ecosystem. The software-centric nature of 5G accelerates deployment and facilitates agile updates and system enhancements; both are crucial for adapting to rapidly evolving technological landscapes. Because of this confluence of openness and software-driven innovation, 5G technologies are resilient and adaptable and will shape a future in which connectivity is fast and seamlessly responsive to the evolving needs of a connected world.

* Corresponding Author: Chun-Ying Huang (chuang@cs.nycu.edu.tw). The work was partially supported by the NSTC of Taiwan (#113-2221-E-A49-186-MY3, #114-2634-F-A49-002-MBK, and #114-2218-E-A49-017) and the Taiwan Academic Cybersecurity Center (TACC) project at National Yang Ming Chiao Tung University.

To achieve the aforementioned vision, 5G technologies must be secure. A vulnerability is defined as an error, a flaw, or a mistake in computer software that permits or causes an unintended behavior to occur [1]. The specifics of a vulnerability differ between organizations; for example, the National Institute of Standards and Technology states that vulnerabilities include both software flaws and configuration problems, whereas "vulnerability" strictly refers to software flaws[1] in the security content automation protocol. Although human-written software is invariably flawed, software errors must be minimized to ensure robust, secure, and reliable 5G networks. Therefore, a steadfast commitment to identifying, addressing, and preventing vulnerabilities in software is essential. Researchers have proposed several methods for rigorously scrutinizing and fortifying the code underpinning 5G infrastructure. However, most studies [2], [3], [4], [5], [6], [7] have examined the behavior of specific 5G core network protocols in isolation. An integrated understanding of the broader network's state is crucial because the behavior of the 5G core network is intrinsically linked to the system's operational status. Systematic vulnerabilities within the 5G core network are difficult to identify; however, a fuzzing test system capable of controlling a 5G network's operational state can facilitate the discovery of such vulnerabilities.

This paper proposes a systematic approach for detecting implementation flaws in 5G core networks. Specifically, we target three open-source and well-known 5G core network implementations: free5GC, Open5GS, and OpenAirInterface-CN-5G. The proposed approach focuses on the implementation of the entry points of the core network, which handles packets sent from a mobile client. The proposed approach differs from existing solutions by addressing the following challenges typically encountered in the evaluation of complex network protocol implementations:

1) **Network protocol implementation complexity:** Testing 5G protocols is challenging because their implementations are inherently complex on account of their requirements for high-speed data transfer and support for diverse applications, communication scenarios, and device types in a dynamic ecosystem. Therefore, conventional testing methods may overlook corner cases arising from complex interactions between network components. In the present

[1]Based on the definitions from NIST, we use the terms "vulnerability" and "flaws" interchangeably throughout this paper.

study, valid simulated 5G traffic was used as an initial seed for fuzzing to effectively capture the complexity of actual systems.

2) **Network operational state maintenance:** 5G protocols are adaptive to network conditions and user demands; thus, replicating a network state for testing is challenging. Moreover, transitioning between states can result in failures whose simulation and detection require a sophisticated testing environment. An innovative testing approach that can navigate the intricacies of protocol state maintenance is essential.
3) **Payload encryption:** Encrypted 5G payloads present unique challenges during system validation because they cannot be inspected directly, limiting the scope of vulnerability assessments. However, encryption is a cornerstone of 5G security. Therefore, effective testing strategies for encrypted data are critical for ensuring the resilience of encrypted 5G payloads against emerging threats.

The proposed approach includes several novel elements for handling complexity, maintaining states, and validating encrypted traffic. First, we leveraged a user equipment (UE)/radio access network (RAN) emulator [8] to generate valid protocol message payloads (Section IV-C). Second, we propose a Fuzzer-in-the-Middle (FitM) architecture to handle multistage protocol states and encrypted protocol payloads (Section IV-D). The FitM can be configured to mutate protocol messages during a selected handshake stage. Finally, we integrated protocol parsers to recognize protocol fields and perform field-aware black-box mutation operations to enhance fuzzing performance (Section IV-E). The proposed FitM architecture operates between the RAN and the access and mobility management function (AMF) and can recognize possible attack surfaces for the entry point of the core network. Our evaluation results indicate that the proposed approach successfully recognized more than 20 implementation flaws, including eight previously unknown flaws, in the selected open-source 5G implementations. We reported the newly recognized flaws to the developer community, and the flaws have been fixed.

The remainder of this paper is organized as follows. Section II discusses the literature on long-term evolution (LTE) and 5G protocol testing and details the features and limitations of relevant studies. Section III presents the necessary background regarding the 5G core network architecture, NG application protocol (NGAP), and nonaccess stratum (NAS) protocol. Section IV describes the threat model, overall structure, and operation of the proposed FitM architecture. Section V provides a detailed introduction to the experimental environment and discusses the experimental results. Finally, Section VI presents the conclusions.

II. Related Work

Numerous studies have evaluated the security and reliability of mobile networks; however, many challenges remain. This section briefly discusses relevant studies and unsolved challenges in this field. Specifically, it focuses on the evaluation of 5G interface security, which is crucial in the 5G architecture [9].

A. Test Case Design

Srinath et al. [3] designed Berserker, a fuzzer that can be used with any version of the 4G and 5G Radio Resource Control (RRC) technical specifications. Their approach extends the ASN.1-based RRC message format to fuzz NAS messages enclosed in RRC packets. In contrast to typical fuzzing frameworks, Berserker omits the implementation of RRC procedure handling and state management. Instead, it relies on a concrete implementation of UE for uplink fuzzing and a RAN for downlink fuzzing; this approach enables communication networks to be set up such that RRC messages can be delivered to the target system. Hussain et al. [4] proposed a framework called 5GReasoner, which analyzes a protocol's technical specifications to generate an abstract cellular protocol model and extract desired security properties. Chen et al. [5] proposed creating test cases by leveraging natural language processing and machine learning techniques to scan large volumes of LTE specification documents. Kim et al. [10] proposed LTEFuzz for 4G/LTE networks. LTEFuzz generates test cases based on security properties retrieved from LTE control plane specifications. It enhances the efficiency of test case generation and considerably improves the depth of security analyses. Apart from specification-based test case generation, Yang et al. [11] proposed generating test cases based on formal verification of 5G network specifications. They focused primarily on authentication flows and generated fuzzing test cases specifically for flows that were classified as uncertain or unsafe.

B. Protocol Message Fuzzing

One typical approach for fuzzing a network protocol is black-box fuzzing. Boofuzz [12] is a well-known black-box fuzzer that has been studied in the literature. This fuzzer is the successor of the Sulley [13] fuzzer, which can be used to fuzz network protocols. In Boofuzz, the protocol architecture model must be defined manually before it can begin fuzzing. Boofuzz can generate various types of data to send to a target and monitor whether the target is active. It also provides crash analysis and reporting features to assist users in finding problems.

In addition to black-box fuzzing, gray-box tools have been used to fuzz network protocols. One popular gray-box network fuzzer is American Fuzzy Lop for Network Fuzzing (AFLNET) [14]. AFLNET is seeded with recorded messages exchanged between two communication parties. It then acts as one party and replays variations of the original sequence of messages sent to the other party. Variations that were effective at increasing the code coverage or state space are retained. AFLNET can automatically connect with a target system to fuzz the network protocol. Although this method enables the discovery of network vulnerabilities or anomalous behavior, each fuzzing iteration requires a network connection to be re-established. Although this requirement incurs substantial time costs, the aforementioned method is considered an efficient network fuzzer. Some gray-box network fuzzers with improved efficiency, such as Nyx-Net [15] and SNPSFuzzer [16], have been developed. Both fuzzers adopt

snapshot technology to preserve the network state before testing different inputs; this approach enables them to recover an initial state, ensuring that testing is repeatable. Test results from ProFuzzBench [17]indicate that both fuzzers considerably outperform AFLNET in terms of coverage and fuzzing speed. Furthermore, Nyx-Net can identify some vulnerabilities that AFLNET cannot. Snapshotting considerably reduces time costs when behaviors are repeated and improves fuzzing performance. However, black-box and gray-box tools, which act as clients that send mutated network payloads, often generate payloads ineffectively, and their exploration of the runtime states is limited.

C. Mutation Technique

Mutation methods for fuzzing can be broadly categorized as those that send malformed packets and those that assign specific field weights. Yebei et al. [2] used a malformed packet method to analyze the security of the packet forwarding control protocol. Specifically, they tested whether a target system accepted data packets with incorrectly sized protocol fields and recorded any unexpected errors. Zujany et al. [18] designed an open-sourced 5G network traffic fuzzer called 5Greplay, which generated malformed Stream Control Transmission Protocol (SCTP) packets to evaluate the robustness of the AMF against unexpected entries during runtime. Weighted packet field methods include that of Fengjiao et al. [6], who reviewed the NAS specifications and designed a new rule extractor. Their program identifies several key fields and applies various mutation strategies to these fields on the basis of the field attributes. This method increases the intelligence of the message mutation process. Hu et al. analyzed the NGAP [19] to assemble mutant fields and normal fields as NGAP-compatible packets, which were sent to the core network. They monitored the current operating status of the core network to allocate field weights.

Field-based mutation has been demonstrated to improve the fuzzing process. Angora [20] and RedQueen [21]are two effective field-aware mutation fuzzers that improve testing efficiency. These fuzzers can discover vulnerabilities by conducting a deep analysis of program logic followed by intelligent mutations of specific fields. Angora performs dynamic tracing of branch conditions during program execution and employs a gradient-based search strategy to precisely adjust input data to explore more execution paths, especially hidden paths requiring specific conditions. RedQueen employs "input-to-state correspondence" techniques to identify key conditions during program execution and automatically adjusts its mutation strategies to bypass or meet these conditions. The commonality between the aforementioned fuzzers lies in performing targeted mutations on specific input fields; such targeting is not performed in conventional fuzzing methods. Although input field recognition requires in-depth program instrumentation and analysis, which can be infeasible for complicated 5G core network systems, incorporating field-based mutation in black-box fuzzers still benefits their overall performance.

In summary, generating appropriate test cases for fuzzing is vital for network fuzzers. Studies have attempted to create test cases by implementing customized traffic generators or replaying prerecorded protocol messages. Customized traffic generators require an in-depth understanding of the protocol specification or message grammar to enhance fuzzing capability; achieving this goal is challenging in the modern era of rapidly evolving protocol specifications. However, replay methods suffer from protocol state management problems. The proposed FitM architecture employs a novel design to address these problems by leveraging protocol emulators and focusing primarily on core fuzzing tasks. It seamlessly bridges real-world spec-compliant network flows with black-box fuzzing algorithms and can systematically perform in-depth fuzzing for selected multistage bidirectional 5G protocols.

III. Background

This section briefly introduces the required background for the design and implementation of the proposed approach. The architecture of the FitM network is described first, followed by the encryption algorithms and involved protocol messages, including the NGAP and NAS protocols.

A. Architecture of the Adopted 5G Core Network

Figure 1 depicts the architecture of the 5G network [22], [23] adopted in the proposed approach, including the access and 5G core networks. The access network comprises the UE and a base station, which is called a gNodeB, and connects to the 5G core network. An essential element of the 5G core network is its service-based architecture, which has a modular framework in which different network functions interact with each other through a service-based interface (SBI) [22]. SBIs are usually based on HTTP/2 or other modern protocols, and interactions occur through RESTful application programmable interfaces [24]. Authorized network functions can access services available on the data bus through control plane protocols. The N1 interface between the UE and the AMF exchanges signaling and control information, such as mobility management and authentication information. The N2 interface connects a gNodeB and the AMF and primarily processes signaling and control messages, including those related to RRC message forwarding and session setup. The N3 interface between a gNodeB and the user plane function (UPF) is dedicated to user data transmission; it manages the flow of data plane information. Because our approach focuses on recognizing possible attack surfaces from the access network, we focused on the N1 and N2 interfaces. Details regarding the relevant components in the architecture are provided as follows.

User Equipment (UE). The UE [25] is a device used to establish connections with the 5G core network. This device usually comprises two parts: mobile equipment (ME) and a universal subscriber identity module (USIM) card. The ME is a hardware device that supports user communication. This device stores an international ME identity number that uniquely identifies it. A USIM card is issued by the operator and stores information such as the subscriber's subscription permanent identity (SUPI), the root key, and the operator's public key. This information can uniquely identify a legitimate

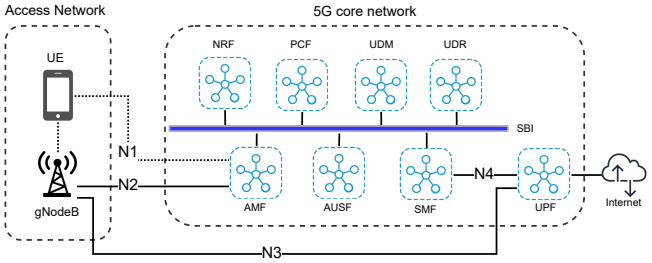

Fig. 1: Architecture of the 5G network adopted in the proposed approach.

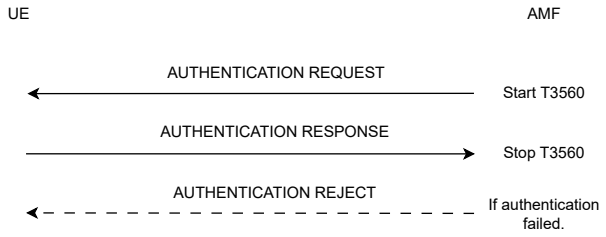

Fig. 2: 5G-AKA-based primary authentication procedure.

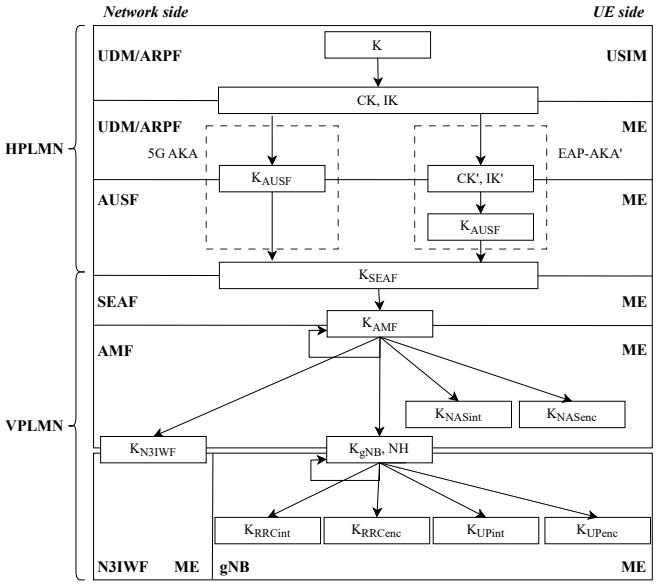

Fig. 3: Key derivation and distribution scheme in a 5G network.

subscriber and enables the completion of mutual authentication whenever the UE attempts to access the 5G network. The UE accesses the 5G core network through a gNodeB, and the N1 interface between the UE and the AMF processes NAS messages.

gNodeB. A gNodeB [26] is a 3GPP-compliant implementation of a 5G New Radio (NR) base station. It comprises independent network functions that implement 3GPP-compliant NR RAN protocols. The N2 interface between a gNodeB and the AMF processes NGAP packets, including NAS messages from the UE. This interface supports control plane signaling between the RAN and the 5G core for UE context management and packet data unit (PDU) session resource management procedures. The N3 interface between a gNodeB and the UPF conveys user data from the RAN to the UPF.

Access and Mobility Management Function (AMF). The AMF involves the termination of NAS signaling on the network side and is responsible for registration management, connection management, reachability management, and mobility management in the 5G core network.

Authentication Server Function (AUSF). The AUSF is responsible for UE authentication through a backend service that computes the authentication data and keying materials when 5G authentication and key agreement (AKA) or EAP-AKA'[2] is used.

Unified Data Management (UDM). The UDM unit performs data management functions, such as computing authentication data and keying materials for the AUSF in accordance with the selected authentication method.

B. 5G Security Architecture

Understanding the 5G security architecture is essential for handling encrypted protocol payloads. Figure 2 presents the 5G-AKA-based procedure for primary AKA; this procedure involves mutual authentication between the UE and the network and helps the two parties agree on the keys KAUSF, KSEAF, and KAMF [27]. The procedure is initiated by sending an AUTHENTICATION REQUEST to the UE. The network then starts the timer T3560[3] [28]. The authentication procedure fails if an AUTHENTICATION REJECT message is received or the timer expires.

[2]EAP-AKA' is an improved version over EAP-AKA.
[3]The timer is typically set to 6s.

Figure 3 shows the key derivation and distribution scheme in 5G networks [27]. In the present study, we focused on the keys related to authentication, starting from keys K and CK/IK. In the case of EAP-AKA', the keys CK' and IK' are derived from CK and IK, respectively. The key hierarchy includes the following keys: KAUSF, KSEAF, KAMF, KNASint, KNASenc, KN3IWF, KgNB, KRRCint, KRRCenc, KUPint, and KUPenc. The most relevant keys for our approach are KNASint and KNASenc, which are used for the integration verification operation and encryption/decryption operation, respectively. Therefore, creating testing NAS packet payloads requires using these keys to ensure the core network components can appropriately handle encrypted mutated messages. Given that K is known in the evaluation environment, our approach must follow the key derivation and distribution scheme to generate the required subkeys appropriately for each involved component, including the ARPF, AUSF, SEAF, and AMF. Readers may refer to Appendix A for more detailed introductions to the terms relevant to the key derivation process.

C. Next-Generation Application Protocol (NGAP)

The NGAP [29] supports the N2 interface between the AMF and a gNodeB. An NGAP message includes an elementary

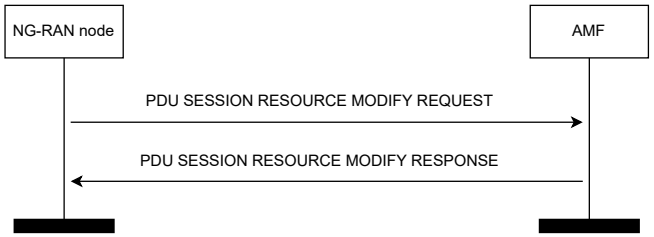

Fig. 4: Example of a successful operation.

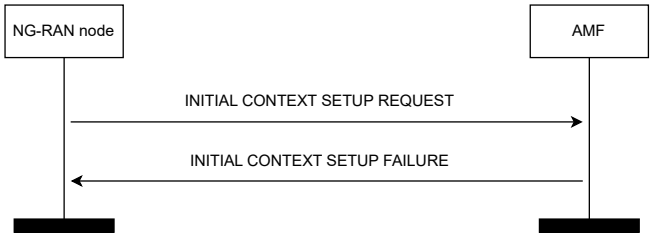

Fig. 5: Example of an unsuccessful operation.

procedure (EP) that represents the primary intention of the message. EPs can be classified as those requiring responses (Class 1) and those not requiring responses (Class 2). Class 2 EPs are always considered successful. Invoking a Class 1 EP leads to a successful or unsuccessful response. Figure 4 [29] shows a successful signaling message (PDU session resource modification request) that has been completed with the receipt of a response. By contrast, Figure 5 [29] shows an unsuccessful signaling message (initial context setup request). A Class 1 EP may be unsuccessful if an expected response is not received because of message expiration.

D. Non-Access-Stratum (NAS) Protocol

NAS [28] is a protocol of the N1 interface, which is the control plane interface between the UE and the AMF for 3GPP and non-3GPP access. The main functions of the NAS protocol are supporting UE mobility, including typical procedures such as identification, authentication, UE configuration updates, and security mode command procedures. Moreover, it supports session management procedures to establish and maintain data connectivity between the UE and the data network.

Figure 6 shows the NAS message sequence for the attach procedure of the 5G core network. The first message of the attach procedure is the initial NAS message sent after the UE has established a radio link with a gNodeB. The UE sends cleartext (unencrypted) information elements (IEs) containing the UE's identity information for verification by the AMF. The authentication procedure is then initiated when the UE's identity is confirmed. In the NAS protocol, integrity and confidentiality protection are enabled by sending the "secure mode command." Subsequently, the derived keys, namely KNASint and KNASenc, are used to secure the NAS protocol payloads. To implement this process appropriately, an in-depth

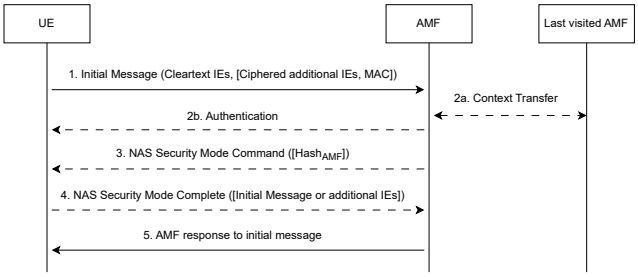

Fig. 6: The attach procedure of the 5G core network.

protocol implementation evaluation system must be capable of inspecting and mutating secured protocol messages.

E. Effects of Flaws in 5G Core Network Implementations

Many 5G core network implementations are software-based, making them susceptible to typical software vulnerabilities. This article evaluates several open-source 5G core network solutions in later sections. Vulnerabilities in key components, such as the AMF, AUSF, and UDM, can be exploited through fuzzing techniques, which may trigger crashes and expose underlying flaws. Such crashes can disrupt the operation of the entire 5G core network, rendering it unable to deliver services. More seriously, memory-related bugs, such as buffer overflows, use-after-free, and heap corruptions, can lead to severe consequences beyond service disruption. These vulnerabilities may be exploited to achieve remote code execution (RCE), allowing an attacker to gain unauthorized control over critical 5G core components. In such cases, an attacker could manipulate network behavior, exfiltrate sensitive data, or persist within the system, posing significant threats to the security and integrity of the entire 5G infrastructure.

IV. Approach

A. Threat Model and Scope

Figure 7 illustrates the network scenario considered in this study and two possible threat sources. We assume that a gNodeB or the UE can be malicious. A malicious gNodeB can connect to the AMF directly and send NGAP messages. It can then send crafted NGAP and NAS protocol messages. By contrast, the UE can only send (malicious) NAS protocol messages to the core network through a gNodeB. Because a gNodeB might not inspect NAS protocol messages, it might forward a malicious NAS protocol message to the core network, which is then processed by flawed components in this network.

The proposed approach results in the systematic generation of real-world test cases that trigger core network implementation flaws; it does not involve anomaly detection. None of the flaws recognized by the proposed approach are false positives; the detected flaws should be addressed appropriately to enhance network reliability.

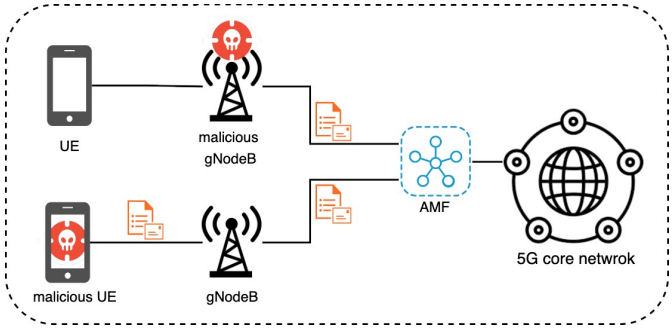

Fig. 7: Threat models considered in this study.

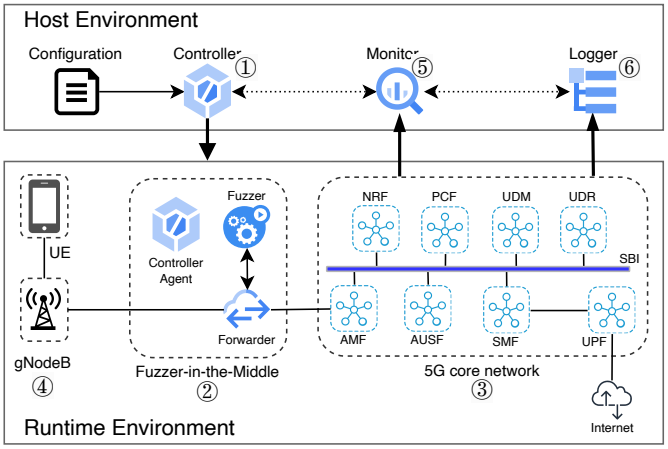

Fig. 8: System architecture for the proposed approach.

B. System Architecture for the Proposed Approach

Figure 8 displays the system architecture for the proposed approach. The architecture comprises the host environment and runtime environment.

The host environment executes and monitors the entire testing process. The process is initiated by the controller ①, which loads the testing configuration and boots all components in order, including the FitM ②, UE/gNodeB ④, and 5G core network ③. The component boot order is critical for avoiding service dependency problems. When a component sends a request, a corresponding service for handling the request must be available.

In the runtime environment, each involved component is containerized; Docker containers are used in the designed system. The controller requests each UE and gNodeB component to repeatedly perform operations until it detects an abnormal event or the testing process is terminated by a user. The testing network traffic flows from the UE/gNodeB ④, passes the FitM architecture ②, and reaches the target [i.e., the 5G core network ③]. When packets between the gNodeB and the AMF pass the FitM, the FitM inspects the packet content and performs mutation operations to generate test cases for each system configuration. All links involved in the network topology are emulated using wired virtual local area network links. Specifically, the wireless link between the emulated UE and the gNodeB is emulated using the ZeroMQ messaging library [30].

Two essential components in the host environment are the monitor ⑤ and logger ⑥. The monitor checks whether the core network containers remain active during the test process. These checks can be performed (1) by ensuring that the core network containers are still active by using process inspection utilities and (2) by ensuring that the network is still connected by periodically sending network packets from the UE/RAN emulator. If abnormal behavior is detected, the logger collects information from the FitM and core networks; the collected information includes the generated test cases, console logs, log files, and crash dumps. The controller then restarts the testing process by recovering all containers to their initial state.

Our system leverages containerization to efficiently manage the complexities of 5G core network implementations; this approach ensures that tests are repeatable and reproducible. The system architecture enables dynamic testing for various network configurations and protocol interactions, thus facilitating the comprehensive evaluation of the network's performance and reliability. Although our framework is optimized for containerized networks, it also accommodates noncontainerized implementations. Manual resets may slow down testing for these networks; however, our system minimizes this inefficiency and conducts a comprehensive evaluation regardless of the network's deployment method.

In summary, our streamlined approach addresses the challenges of testing complex 5G networks, offering a robust and flexible testing environment that captures the intricacies of core network implementations. The details of the runtime components are provided in the rest of this section.

C. UE/RAN Emulator

The UE/RAN emulator is an essential component in the runtime environment. We employed UERANSIM [8] to generate network flows for testing. UERANSIM is a cutting-edge tool that has emerged as a valuable asset in 5G network development and testing. Developed as an open-source project, UERANSIM is designed to emulate the UE and RAN (gNodeB) in a 5G network to test various scenarios and functionalities in simulated environments. This tool is pivotal in validating the performance, security, and interoperability of 5G network elements.

UERANSIM has many of the required functionalities for attaching to 5G core networks and establishing communication channels. It generates spec-compliant NGAP and NAS protocol messages. UERANSIM implements the NGAP features of critical operations, session resource management, context setup and modification, messaging, and error indication; it also supports many essential NAS features, such as primary AKA, security mode control, identification, and various registration and session management processes. In this study, the network flows generated by UERANSIM were forwarded and mutated by the proposed FitM to generate test cases to trigger unexpected behavior in the evaluated core network.

Using UERANSIM reduces the complexity of network protocol implementation in a fuzzing scenario. The success of a fuzzing process heavily relies on the quality of its initial seeds (test cases). Thus, applying mutation to valid messages

exchanged between the emulator and the core network considerably shrinks the exploration space for finding appropriate test inputs.

D. Fuzzer-in-the-Middle (FitM) Architecture

The FitM module is the most crucial component of the proposed approach. The proposed FitM comprises three submodules: a controller agent, a forwarder, and a fuzzer. The controller agent is the interface between the FitM and the host controller. It configures the FitM module and reports the testing status to the host controller. A user can also configure the FitM module to evaluate a target protocol, such as the NGAP or NAS protocol, or target states, such as the procedure code in the NGAP or the message type in the NAS protocol. The FitM module addresses two of the primary challenges discussed in Section I: network operational state maintenance and payload encryption.

Complex communication protocols often involve state transitions, and the appropriate management of these states is crucial for maintaining the stable operation of the fuzzing process. The proposed FitM module maintains correct protocol states by forwarding spec-compliant network flows with selective protocol message mutations. Algorithm 1 describes the workflows of the forwarder and fuzzer in the FitM module. The forwarder processes the packets received by the FitM module. On the basis of the configured target testing protocol, procedure code, and message type of the FitM module, the forwarder decides whether the fuzzer should mutate a packet before forwarding it. If the packet fields match a target protocol or state in the fuzzer configuration, the packet is first passed to the fuzzer for mutation; otherwise, the packet is forwarded to the next hop directly.

The proposed fuzzer focuses on the N1 and N2 interfaces, for which payloads conform to the NAS protocol and NGAP. The FitM module first identifies fields consistent with the configured testing protocol, procedure code, and message type and randomly selects some of these fields for mutation. For each mutation request, the fuzzer mutates the same packet multiple times; each mutated packet is then returned for forwarding. The fuzzing process continues until a user terminates the FitM module from the controller or the number of fuzzed test cases reaches a user-defined threshold.

After user authentication is completed, all messages between the UE and the core network are encrypted; this process is challenging in the mutation method. Therefore, the FitM employs a cryptographic module to process encrypted protocol messages. First, the forwarder detects whether a packet is encrypted; encrypted packets are decrypted before they are passed to the fuzzer. Subsequently, following mutation, the mutated packets are re-encrypted, digitally signed if required, and forwarded to the next hop.

To perform decryption, encryption, and digital signing, the cryptographic module must have access to the K, IK, and CK keys of the core network as well as the nonces[4] exchanged between the two endpoints. In our testing scenario, these keys were easily obtained because the UE, gNodeB, and core network were completely controlled in the simulations. The forwarder tracks the nonces exchanged between two endpoints and provides them to the cryptographic module.

[4]Numbers used only once.

Algorithm 1 Workflow of the proposed FitM

Input: *proto*: Target protocol
Input: *proc*: Target procedure code
Input: *msgtype*: Target message type
Input: *keys*: cryptographic keys: K, IK, and CK

```
1:  nonce = ∅
2:  while not stopped do
3:      pkt = receive_packet()          ▷ blocking mode with a timeout period
4:      if pkt == None then
5:          log_report_and_restart()
6:      end if
7:      update_nonce(nonce, pkt)                ▷ Track nonces
8:      if state_match(pkt, proto, proc, msgtype) then
9:          p = is_security_protected(pkt)
10:         if p == True then
11:             pkt = decrypt(pkt, keys, nonce)
12:         end if
13:         mutated_pkts = fuzzer(pkt)
14:         for s ∈ mutated_pkts do
15:             if p == True then
16:                 epkt = encrypt_and_mac(s, keys, nonce)
17:             end if
18:             forward_packet(epkt)
19:             if check_service_state() == Dead then
20:                 log_report_and_restart()
21:             end if
22:             if stopped then
23:                 terminate()
24:             end if
25:         end for
26:         go to 13
27:     else
28:         forward_packet(pkt)
29:         if check_service_state() == Dead then
30:             log_report_and_restart()
31:         end if
32:     end if
33: end while
```

During testing, the forwarder also monitors whether the AMF is reachable by identifying NGAP connections that fail to establish or become disconnected. If such connections are detected, the forwarder reports an abnormal status to the host controller, which logs the event and restarts the testing process.

It is important to note that our proposed algorithm is designed for a black-box fuzzer, rather than a grey-box one. While a detailed exploration of grey-box fuzzing is beyond the scope of this study, our preliminary experiments indicate that integrating lightweight grey-box instrumentation for coverage collection, maintenance, and synchronization results in performance degradation of over 10× compared to the black-box approach. Additionally, incorporating grey-box mutation techniques may conflict with our state-aware packet fuzzing

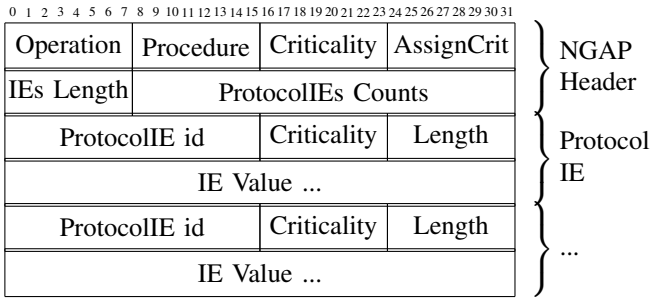

Fig. 9: Format of an NGAP packet.

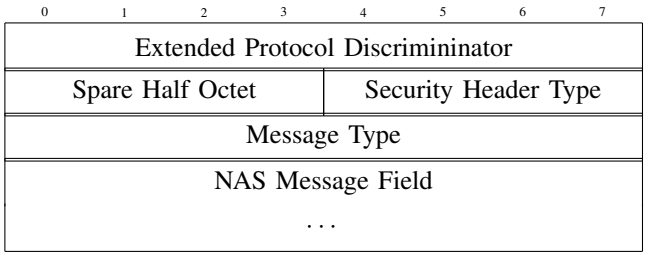

Fig. 10: Format of a plain NAS packet.

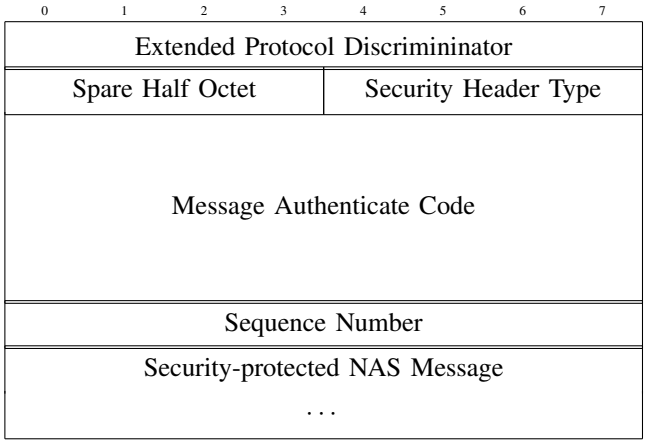

Fig. 11: Format of a security-protected NAS packet.

strategy. In particular, mapping code coverage back to specific protocol states poses significant challenges because: (1) the same code paths may be exercised across multiple states, (2) some state transitions do not correspond directly to distinct code branches, and (3) coverage signals can be noisy or ambiguous due to shared or reused execution paths.

E. Fuzzer: The Protocol Mutator

The goal of the proposed approach is to explore the attack surfaces at the entry point of the core network; therefore, the FitM is located between a gNodeB and the AMF, and this fuzzer evaluates the implementation of the NGAP and NAS protocol. This section provides details regarding how the FitM handles protocol payloads.

Figure 9 displays the format of an NGAP packet. The NGAP can be used to signal the UE/RAN emulator and the core network. The "procedure" code in the NGAP packet specifies the signal to be delivered. Typical signals include NGSetup, InitialContextSetup, InitialUEMessage, and NAS-related procedures such as UplinkNASTransport and DownlinkNASTransport. For each procedure, several IEs may also be included in the packet. The mutator is applied to both the protocol fields and IEs.

NAS payloads are enclosed in the protocol IE with id 38 in NGAP packets. Although NAS protocol messages are also sent over the NGAP, a dedicated mutation algorithm is implemented for handling NAS procedure codes because the UE can initiate a NAS protocol message and send it to the core network without the message being carefully inspected at the RAN (gNodeB). An attacker can leverage this lack of inspection to use NAS protocol messages to launch attacks directly against the core network.

Figures 10 and 11 depict the formats of plaintext and encrypted NAS protocol messages. NAS messages are initially transmitted in plaintext format; after security procedures are invoked and completed, subsequent NAS messages are sent in the secure format. Both secure and plaintext NAS messages can be mutated using the cryptographic module in the FitM.

1) Field-Aware Mutation: Fuzzing research works [20], [21] have indicated that fuzzing performance improves if mutation is performed with knowledge of field boundaries. Therefore, the proposed FitM employs a field-aware mutation strategy to ensure that precise field mutations can be performed. The open-source pycrate [31] library is an efficient library that supports format manipulations of various cellular network signaling packets. In the present study, pycrate was used to parse the format of NGAP and NAS packets. Pycrate provides a runtime for encoding and decoding data structures, including CSN.1 and ASN.1 notations typically used in LTE and 5G network protocols. Moreover, we observed that performing black-box fuzzing directly against the entire packet often resulted in malformed packets that were dropped in the initial processing stages. Therefore, our protocol mutator decomposes each received packet into individual protocol fields, randomly selects protocol fields for mutation, and then reassembles the original and mutated protocol fields into a packet that appears valid. These steps considerably increase the likelihood that the core network components perform in-depth processing of the mutated packets.

To ensure that the mutator was independent of the core network implementation, we employed the open-source black-box fuzzing library Radamsa [32] in the proposed FitM. The operations in this library involve the mutation of input strings on the basis of a random seed. Radamsa differs from standalone fuzzers such as Sulley [13] or Boofuzz [12], which are specifically designed for fuzzing network protocols. The mutation operations in Radamsa are summarized in Table I. Throughout this paper, all available mutation operations were applied unless otherwise mentioned. The effect of each mutation strategy on performance was evaluated, and the results are provided in Section V-B.

TABLE I: Summary of Employed Mutation Strategies.

Class: Line-based mutation	
Operation	Description
`ld`	Delete a line
`lds`	Delete many lines
`li`	Copy a line nearby
`lis`	Insert a line from elsewhere
`lp`	Swap the order of lines
`lr`	Repeat a line
`lr2`	Duplicate a line
`lrs`	Replace a line with one from elsewhere
`ls`	Swap two lines
Class: Binary-based mutation	
Operation	Description
`sd`	Delete a sequence of bytes
`sr`	Repeat a sequence of bytes
`bd`	Drop a byte
`bed`	Decrement a byte by one
`bei`	Increment a byte by one
`ber`	Swap a byte with a random one
`bf`	Flip one bit
`bi`	Insert a random byte
`bp`	Permute some bytes
`br`	Repeat a byte
Class: Other	
Operation	Description
`num`	Try to modify a textual number
`fn`	Likely clone data between similar positions
`fo`	Fuse previously saw data elsewhere
`ft`	Jump to a similar position in the block
`ui`	Insert funny Unicode
`uw`	Try to make a code point too wide
`xp`	Try to parse XML and mutate it
`ab`	Enhance silly issues in ASCII string data handling

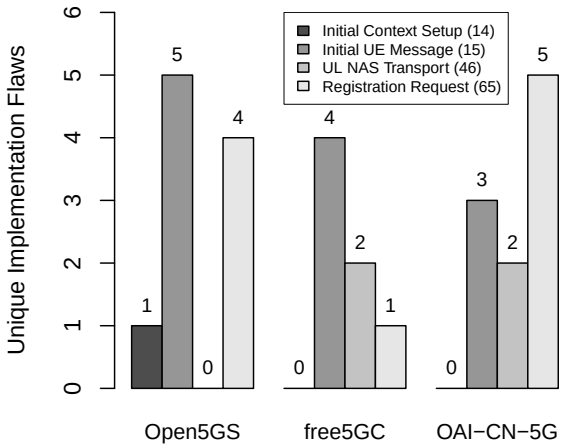

Fig. 12: Unique flaws identified for each project.

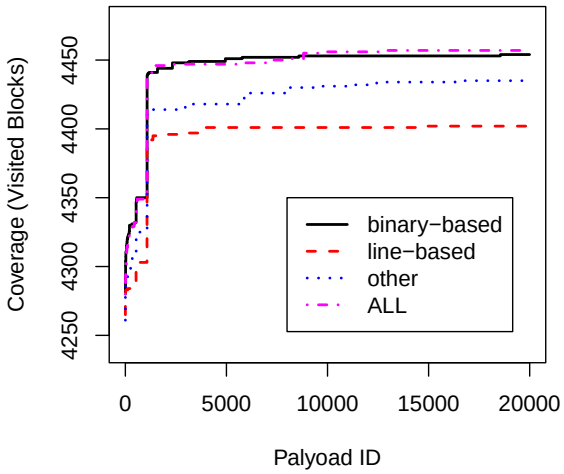

Fig. 13: Fuzzing coverage for each tested mutation strategy.

V. Evaluation

A. Environment Setup

We evaluated the proposed approach on three well-known open-source 5G core networking implementations: free5GC [33], Open5GS [34], and OpenAirInterface-CN-5G [35] (OAI-CN-5G). We deploy the selected core networks in docker containers [36]. Each core network was implemented in Docker containers [36] based on the provided official Dockerfiles and images of each network project. All software programs were installed and run on a single machine equipped with an Intel Core i7-7600 CPU having 16GB of RAM and running the Ubuntu 20.04.4 LTS operating system. Although the proposed approach can work with any target, including closed-source implementations without modifications, open-source projects were selected to enable better measurements of the effectiveness of the approach because these projects support using compiler features to collect runtime code coverage data. While the proposed approach is a black-box fuzzer, we try our best not to touch the default configurations of each target under testing unless necessary.

B. Mutation Strategy

We evaluated how each mutation strategy affected the overall fuzzing performance. Each mutation strategy involved a set of mutation operations and the detection of encrypted payloads. Ideally, a fuzzer should explore as many codes as possible in the targeted system. A basic block is the minimum control flow unit at the machine code level; a fuzzer that visits more blocks has better coverage and performs a more intensive test of the program. Therefore, the selected performance metric was the number of visited basic blocks.

Open5GS was the primary target for evaluating the mutation strategies because it is implemented in C; thus, many code instrumentation tools, such as AFL++ [37] with `afl-clang-lto` and LLVM [38] compiler framework, are available for measuring the number of visited blocks. For each experiment, the measured visited block coverage was stored in shared memory.

The Radamsa mutation strategies were classified as binary-based, line-based, and other; the operations in each class are listed in Table I. The efficacy of each class was tested for

TABLE II: Identified Network Function Implementation Flaws for Each Procedure (Code) in the NGAP and NAS Protocol.

CN	Ver.	Selected State	Comp.	Crashed Location / Reason	New
Open5GS-1	v2.6.4	InitialUEMessage (15)	AMF	amf_ue_set_suci: Assertion 'suci' failed	Yes
Open5GS-2	v2.6.4	Registration Request (65) *	UDM	ogs_supi_from_supi_or_suci: Expectation 'supi' failed	Yes
Open5GS-3	v2.6.4	Registration Request (65) *	AUSF	ausf_ue_add: Assertion 'ausf_ue' failed	Yes
Open5GS-4	v2.4.8	InitialContextSetup(14)	AMF	amf_gnb_add: Assertion 'gnb' failed	Yes
Open5GS-5	v2.4.8	InitialUEMessage(15)	AMF	amf_state_operational: Assertion 'OGS_FSM_STATE(&amf_ue->sm)	-
Open5GS-6	v2.4.8	InitialUEMessage(15)	AMF	amf_ue_find_by_message: Assertion 'suci' failed	-
Open5GS-7	v2.4.8	InitialUEMessage(15)	AMF	ogs_hash_set_debug: Assertion 'klen' failed	-
Open5GS-8	v2.4.8	InitialUEMessage(15)	AMF	udm_ue_add: Assertion 'udm_ue' failed	-
Open5GS-9	v2.4.8	Registration Request (65) *	AUSF	server_send_response: Assertion 'fd != INVALID_SOCKET' failed	-
Open5GS-10	v2.4.8	Registration Request (65) *	AMF	common_register_state: Assertion 'true == amf_ue_sbi_discover_and_send (OpenAPI_nf_type_AUSF, amf_ue, NULL, amf_nausf_auth_build_authenticate)' failed	-
OAI-CN-5G-1	v1.5.0	Any	AMF	In sctp::sctp_server::sctp_receiver_thread(void*)	Yes
OAI-CN-5G-2	v1.5.0	Registration Request (65) *	AMF	In nas::RegistrationRequest::decodefrombuffer(nas::NasMmPlainHeader*, unsigned char*, int)	Yes
OAI-CN-5G-3	v1.4.0	In InitialUEMessage(15)	AMF	In nas::_5GSMobilityIdentity::imeisv_decodefrombuffer(unsigned char*, int)	-
OAI-CN-5G-4	v1.4.0	In InitialUEMessage(15)	AMF	In nas::_5GSMobilityIdentity::getSuciWithSupiImsi(nas::SUCI_imsi_s&)	-
OAI-CN-5G-5	v1.4.0	In InitialUEMessage(15)	AMF	In nas::_5GSMobilityIdentity::suci_decodefrombuffer(unsigned char*, int, int)	-
OAI-CN-5G-6	v1.4.0	Registration Request (65) *	AMF	In nas::_5GSMobilityIdentity::_5g_s_tmsi_decodefrombuffer(unsigned char*, int)	-
OAI-CN-5G-7	v1.4.0	Registration Request (65) *	AMF	In nas::NAS_Message_Container::decodefrombuffer(unsigned char*, int, bool)	-
OAI-CN-5G-8	v1.4.0	Registration Request (65) *	AMF	In nas::EPS_NAS_Message_Container::decodefrombuffer(unsigned char*, int, bool)	-
OAI-CN-5G-9	v1.4.0	Registration Request (65) *	AMF	In nas::LADN_Indication::decodefrombuffer(unsigned char*, int, bool)	-
OAI-CN-5G-10	v1.4.0	UplinkNASTransport (46)	AMF	In nas::EAP_Message::decodefrombuffer(unsigned char*, int, bool)	-
OAI-CN-5G-11	v1.4.0	UplinkNASTransport (46)	AMF	In nas::Authentication_Response_Parameter::decodefrombuffer	-
free5GC-1	v3.3.0	InitialUEMessage(15)	AMF	In aper.GetBitString, "dstBytes[byteLen-1]" index out of range	Yes
free5GC-2	v3.3.0	InitialUEMessage(15)	AMF	In nasType.(*MobileIdentity5GS).GetSUCI, "bits.RotateLeft8(a.Buffer[4], 4)" index out of range	Yes
free5GC-3	v3.2.1	InitialUEMessage(15)	AMF	In gmm.HandleRegistrationRequest, "mobileIdentity5GSContents[0]" index out of range	-
free5GC-4	v3.2.1	InitialUEMessage(15)	AMF	Signal SIGSEGV: invalid memory address or nil pointer dereference	-
free5GC-5	v3.2.1	Registration Request (65) *	AMF	In nasMessage.(*RegistrationRequest).DecodeRegistrationRequest, slice bounds out of range	-
free5GC-6	v3.2.1	UplinkNASTransport (46)	AMF	In nasMessage.(*AuthenticationFailure).DecodeAuthenticationFailure, slice bounds out of range	-
free5GC-7	v3.2.1	UplinkNASTransport (46)	AMF	In nasMessage.(*AuthenticationResponse).DecodeAuthenticationResponse, slice bounds out of range	-

Note: Asterisks in the "selected state" column indicate that most flaws were identified through NAS fuzzing.

the NGSetup connection phase. FitM applied all mutation operations from the selected class to all recognized fields for fuzzing, and the coverage was measured. The results in Figure 13 indicate that the binary-based mutation operations resulted in the fuzzer visiting the most blocks; however, the line-based and other mutation operations also increased the coverage substantially. To ensure a comprehensive evaluation, all operations were considered by the fuzzer, and fuzzing operations were selected randomly from all classes for mutation.

The effect of the cryptographic module was evaluated for secure UL NAS transport messages. The FitM was applied with and without the cryptographic module, and the fuzzer's efficacy was again assessed in terms of the total number of visited basic blocks. Figure 14 reveals that the number of visited blocks only increased with the payload ID when the cryptographic module was activated. This phenomenon occurred possibly because the fuzzer without the cryptographic module might have corrupted the encrypted packets; such corrupted packets would have been ignored by the AMF. These results highlight the vital role of the cryptographic module in improving the effectiveness and efficiency of fuzzing by ensuring that more packets are processed by the core network.

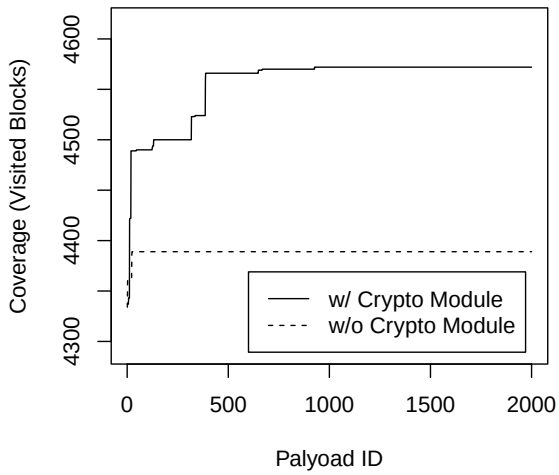

Fig. 14: Fuzzing coverage with and without the cryptographic module.

C. Comparison with Boofuzz

The performance of our proposed fuzzer and the open-source black-box fuzzer Boofuzz was compared [12]. Our fuzzer was not compared against the black-box fuzzer 5Greplay [18] because 5Greplay simply replays core network packets from a prerecorded packet trace file, corrupts the

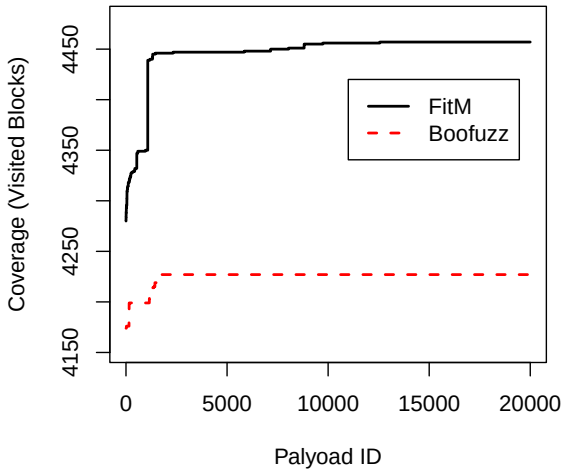

Fig. 15: Performance of FitM and Boofuzz.

packets, and repeatedly sends the packets to the AMF. It does not consider protocol stages and focuses only on the service availability of the AMF. Although sending many malformed packets can reveal NGAP parsing faults and DoS-based attacks, the scope of the testing coverage is limited. Therefore, 5Greplay is not directly comparable to FitM. The black-box fuzzer of He et al. [6] also performs field-aware mutation; it does so by assigning weights to selected fields. Although its fuzzing architecture is similar to that of Boofuzz, the aforementioned fuzzer was unavailable and could not be used for comparisons. Therefore, we selected Boofuzz for the comparison.

Boofuzz lacks support for the SCTP and does not recognize 5G core network protocols. Therefore, we implemented these two features in Boofuzz for the performance evaluations. Boofuzz requires the user to develop a customized packet format parser and re-assembler. We applied the same parsing method for the NGAP packets as for the FitM, and the extracted protocol fields were then passed to Boofuzz for mutations. Figure 15 depicts the performance of Boofuzz and FitM. The number of visited blocks for Boofuzz increased considerably more slowly than that for the proposed FitM. This result was attributed to the monotonous linear and deterministic mutation pattern employed in Boofuzz, which generates many ineffective test cases from scratch. Specifically, Boofuzz applies specific mutation rules to each field based on its type. For example, an integer field may be tested with edge-case values such as 0, 1, -1, `INT_MAX`, `INT_MIN`, or subjected to bit-level mutations like random bit flips. Consequently, even after extended fuzzing, the total number of visited blocks remains limited. Note that Boofuzz did not find any crashes in this evaluation experiment.

D. Summary of Recognized Implementation Flaws

For each selected configuration, we requested the fuzzer to generate 2 million mutated packets to evaluate the selected core network functions. The proposed solution successfully recognized implementation flaws for NGAP fuzzing in the InitialUEMessage, UplinkNasTransport, InitialContextSetup, and PDUSessionResourceSetup procedure codes and recognized flaws for NAS fuzzing in the Registration Request, Registration Complete, Authentication Response, Security Mode Response, and UL NAS transport procedures.

Figure 12 presents the unique implementation flaws recognized by the proposed approach for each core network. The details of each flaw are listed in Table II. The "Crashed Location / Reason" field presents the report of each recognized flaw collected by the logger. The "New" field in the table indicates a previously unknown implementation flaw recognized by our approach; we have reported these new flaws to the development team of the corresponding projects. Open5GS implements an anomaly self-detection feature (ref: Section V-E), which reports the origin component of an issue when abnormal behavior is detected (refer to Table II). For free5GC and OAI-CN-5G, abnormal behavior was detected by evaluating the behavior of the AMF; thus, all "Component" fields are labeled as "AMF" for these networks.

In the remainder of this section, we discuss several interesting test cases that revealed implementation flaws for each evaluated core network. We further discuss how the root cause of each flaw was identified. In general, the proposed system detects unexpected behavior by identifying whether the target program crashes or triggers an exception. We retrieved the backtrace from the corresponding crash dump or exception logs and analyzed these records to recognize the root cause of the issue.

E. Open5GS

Open5GS is a prominent open-source project that has emerged as a key player in the development and implementation of 5G core network solutions. This versatile platform provides a comprehensive and flexible framework for deploying and testing various elements of 5G networks, including the core network functions. Open5GS is mainly implemented in C, and its code has many strict assertions; if any assertion is violated, the network functions of Open5GS are shut down. Violations include an unexpected state, decoding errors for received packets, file descriptor errors, and length-too-long errors. These assertions are helpful for developers investigating the root cause of abnormal system states.

In our evaluation, we successfully reproduced seven known issues that have been discussed in the developer community. Furthermore, the proposed system recognized four new issues in its latest version (v2.6.4 as of writing). The recognized issues have been reported to the relevant developers.

The Open5GS-1 test case listed in Table II is discussed here as an example of issue recognition. The payloads that triggered the flaw can be mutated from the NGAP InitialUEMessage packet or the NAS Registration Request used in the attach procedure. The root cause of the selected issue is the use of

```
[amf]:[ERROR] : Cannot get the SUCI from Mobile Identity (../src/amf/context.c:1766)
    0000: 4100f110 00000000 21436587 59
          A.......!Ce.Y
[amf] [INFO]: [Added] Number of AMF-UEs is now 2 (../src/amf/context.c:1563)
[gmm] [INFO]: Registration request (../src/amf/gmm-sm.c:1061)
[nas] [ERROR]: Not implemented SUPI format [4] (../lib/nas/5gs/conv.c:86)
[amf] [FATAL]: amf_ue_set_suci: Assertion 'suci' failed. (../src/amf/context.c:1906)
[core] [FATAL]: backtrace() returned 12 addresses (../lib/core/ogs-abort.c:37)
./open5gs-amfd(+0x20f6d) [0x558c16721f6d]
./open5gs-amfd(+0x9d715) [0x558c1679e715]
./open5gs-amfd(+0x3f4f8) [0x558c167404f8]
./open5gs-amfd(+0x38adf) [0x558c16739adf]
/open5gs/install/lib/x86_64-linux-gnu/libogscore.so.2(ogs_fsm_dispatch+0x149) [0x7f19a8c6bde3]
./open5gs-amfd(+0x4f7f8) [0x558c167507f8]
/open5gs/install/lib/x86_64-linux-gnu/libogscore.so.2(ogs_fsm_dispatch+0x149) [0x7f19a8c6bde3]
./open5gs-amfd(+0x12bc7) [0x558c16713bc7]
/open5gs/install/lib/x86_64-linux-gnu/libogscore.so.2(+0x1c9dc) [0x7f19a8c599dc]
/lib/x86_64-linux-gnu/libpthread.so.0(+0x8609) [0x7f19a79a0609]
/lib/x86_64-linux-gnu/libc.so.6(clone+0x43) [0x7f19a78c5133]
```

Fig. 16: Backtrace of the Open5GS core network for the implementation flaw in case Open5GS-1.

an unsupported SUPI format. The TS 23.501 specification [22] currently defines that only 0 and 1 are valid values in the IMSI (0) and NAI (1) formats. Our mutator created a test case by using an undefined type of 4, which led to a subsequent parsing error in the packet that finally crashed the core network function. The crash log reports from Open5GS (Figure 16) were used to recognize the root cause. The log pinpoints the crashed line numbers and relevant filenames; this information is highly helpful for security engineers and developers attempting to fix the vulnerability.

We reported the detected issue to the Open5GS development team, and a workaround was provided in a recent commit by adding SUPI format checks in src/amf/context.c and src/amf/gmm-handler.c. The context.c code primarily manages the 5G AMF context with functionalities such as user registration, connection management, and resource allocation. The gmm-handler.c code implements parts of the GMM functionality in the 5G network, such as UE registration, status management, identity recognition, and mobility management, ensuring that users can communicate and move correctly within the 5G network. The fix is shown in Figure 17.

All other implementation flaws in Open5GS (Open5GS-2 to Open5GS-10) were related to the use of an unchecked NULL pointer. Although Open5GS employs many assertions in its codes to detect unexpected system conditions, the lack of appropriate handling may cause system instability in production networks.

```
mobile_identity_suci = (
    ogs_nas_5gs_mobile_identity_suci_t *)
    mobile_identity->buffer;
if(mobile_identity_suci->h.supi_format !=
   OGS_NAS_5GS_SUPI_FORMAT_IMSI) {
     ogs_error("Not implemented SUPI format [%d]",
         mobile_identity_suci->h.supi_format);
     return NULL;
}
```

Fig. 17: Open5GS: Check SUPI format in src/amf/context.c and src/amf/gmm-handler.c.

```
panic: runtime error: slice bounds out of range [:46] with length 13
goroutine 52 [running]:
github.com/.../nasMessage.(*RegistrationRequest).DecodeRegistrationRequest(0xc0006dc8f0, 0xc000643850)
/go/.../nas@v1.0.7/nasMessage/NAS_RegistrationRequest.go:196 +0x1b57
```

Fig. 18: Sample log for slice-bounds-out-of-range errors in free5GC version 3.2.1 and NAS version 1.0.7.

```
func (a *RegistrationRequest) DecodeRegistrationRequest(byteArray *[]byte){
  buffer := bytes.NewBuffer(*byteArray)
  ...
  case RegistrationRequestCapability5GMMType:
    a.Capability5GMM = nasType.NewCapability5GMM(ieiN)
    binary.Read(buffer, binary.BigEndian, &a.Capability5GMM.Len)
    a.Capability5GMM.SetLen(a.Capability5GMM.GetLen())
    binary.Read(buffer, binary.BigEndian, a.Capability5GMM.Octet[:a.Capability5GMM.GetLen()])
```

Fig. 19: Flawed DecodeRegistrationRequest function in free5GC versions up to version 3.2.1.

```
case RegistrationRequestCapability5GMMType:
  a.Capability5GMM = nasType.NewCapability5GMM(ieiN)
  if err := binary.Read(buffer, binary.BigEndian, &a.Capability5GMM.Len); err != nil {
    return fmt.Errorf("NAS ...: %w", err)
  }
  if a.Capability5GMM.Len < 1 || a.Capability5GMM.Len > 13 {
    return fmt.Errorf("invalid ...: %d", a.Capability5GMM.Len)
  }
  a.Capability5GMM.SetLen(a.Capability5GMM.GetLen())
  if err := binary.Read(buffer, binary.BigEndian, a.Capability5GMM.Octet[:a.Capability5GMM.GetLen()]); err != nil {
    return fmt.Errorf("NAS ...: %w", err)
  }
```

Fig. 20: Fixed DecodeRegistrationRequest function in free5GC version 3.3.0.

```
1  panic: runtime error: index out of range
       [18446744073709551615] with length 0
2  goroutine 15 [running]:
3  runtime/debug.Stack()
4  /usr/.../stack.go:24 +0x65
5  github.com/.../service.handleConnection.func1()
6  /go/.../service.go:184 +0x58
7  panic({0xc4f860, 0xc000414ab0})
8  /usr/.../panic.go:1038 +0x215
9  github.com/.../aper.GetBitString({0xc00002624a, 0
       xb7e540, 0x12c36a0}, 0x0, 0x0)
10 /go/.../aper@v1.0.4/aper.go:54 +0x247
```

Fig. 21: Sample log for index-out-of-range errors in free5GC version 3.3.0.

F. free5GC

free5GC is a noteworthy open-source project at the forefront of advancing 5G technology. Designed as a 5G core network framework, free5GC offers a versatile and accessible platform for developers, researchers, and network operators to explore, implement, and optimize 5G core network functionalities. With a commitment to open standards, free5GC enables seamless integration and interoperability within the evolving landscape of 5G networks. Its modular architecture and comprehensive feature set make it a valuable resource for those seeking to understand, experiment with, and contribute to developing robust and scalable 5G infrastructure. free5GC is implemented in the Go programming language, which is targeted toward developing scalable and secure systems. Runtime messages from goroutine threads were evaluated to identify three common classes of implementation flaws at the AMF: slice bounds out of range, index out of range, and invalid memory address or nil pointer dereference.

The class "slice bounds out of range" is often found in NAS decoding procedures. Interestingly, implementation flaws in this class are usually recognized by the NGAP fuzzing module instead of the NAS fuzzing module because the NGAP fuzzing module corrupts the format of the NAS message, whereas the NAS fuzzing module generates protocol-compliant messages. A sample log message reported for case free5GC-5 is shown in Figure 18. The logged information and corresponding source codes are shown in Figure 19. This figure indicates that a received packet is considered to be a NAS Registration Request if its message type is RegistrationRequestCapability5GMMType. The code attempts to load the NAS payload into the Capability5GMM.Octet byte array without checking the array size (line 7 in Figure 19). Therefore, if the NAS payload's length exceeds the array size, a slice-bounds-out-of-range error is triggered. This class of implementation flaws can be easily fixed through a length check before data are loaded into an array, as shown in Figure 20.

Index-out-of-range crashes are similar to slice-bounds-out-of-range errors. These crashes are also usually triggered by the NGAP fuzzing module. A sample log message reported for case free5GC-1 is shown in Figure 21. The logged information and corresponding source code shown in Figure 22 reveal that when the variable numBits has a value of 0, a byte array dstBytes of length 0 is created (lines 8-10). Consequently,

```
1  func GetBitString(srcBytes []byte, bitsOffset uint
       , numBits uint) (dstBytes []byte, err error){
2    bitsLeft := uint(len(srcBytes))*8 - bitsOffset
3    if numBits > bitsLeft {
4      err = fmt.Errorf("Get ...: %d, leftBits: %d",
           numBits, bitsLeft)
5      return
6    }
7    byteLen := (bitsOffset + numBits + 7) >> 3
8    numBitsByteLen := (numBits + 7) >> 3
9    dstBytes = make([]byte, numBitsByteLen)
10   numBitsMask := byte(0xff)
11   if modEight := numBits & 0x7; modEight != 0 {
12     numBitsMask <<= uint8(8 - (modEight))
13   }
14   for i := 1; i < int(byteLen); i++ {
15     dstBytes[i-1] = srcBytes[i-1]<<bitsOffset |
           srcBytes[i]>>(8-bitsOffset)
16   }
17   if byteLen == numBitsByteLen {
18     dstBytes[byteLen-1] = srcBytes[byteLen-1] <<
           bitsOffset
19   }
20   dstBytes[numBitsByteLen-1] &= numBitsMask
21   return
22 }
```

Fig. 22: GetBitString function in free5GC version 3.3.0.

subsequent accesses to the array always trigger an index-out-of-range error. This phenomenon was observed in free5GC version 3.2.1 (CVE-2022-43677) but was not fixed in the next release, namely version 3.3.0. We helped the development team by generating test payloads that can reproduce this CVE and fix the issue.

Finally, in every test state, "invalid memory address or nil pointer dereference" crashes were observed as originating from the ListenAndServe network function of the AMF (case free5GC-4). We concluded that free5GC exhibits a race condition issue when it handles high packet traffic and the high-frequency establishment of the monitor's SCTP connection to the AMF. If a racing thread unexpectedly releases a shared SCTP session, another thread trying to reuse the session causes the aforementioned type of error. The aforementioned three types of errors were revealed by the implementation flaws summarized in Table II. Overall, the aforementioned results demonstrate the effectiveness of our approach in finding problems within free5GC.

G. OpenAirInterface-CN-5G (OAI-CN-5G)

OAI-CN-5G is an open-source project focusing on evolving 5G core network solutions. This project provides a flexible and extensible platform for developing, testing, and deploying various components, contributing to the ongoing innovation and standardization of 5G technologies. OAI-CN-5G supports open standards, which foster collaboration and enable interoperability within the dynamic landscape of 5G networks. This project is mainly implemented in C++. It is worth noting that the container for OAI-CN-5G by default activates the AddressSanitizer feature to categorize recognized implementation flaws into decoding and network-associated issues. While Address Sanitizers are designed to detect a wide range of

```
#0  __GI_abort () at abort.c:107
#1  0x00007f934aed6837 in __libc_message (action=
    action@entry=(do_abort | do_backtrace),
    fmt=fmt@entry=0x7f934b003869 "*** %s ***: %s
        terminated\n") at ../sysdeps/posix/
        libc_fatal.c:181
#2  0x00007f934af81b5f in __GI___fortify_fail_abort
    (need_backtrace=need_backtrace@entry=true,
    msg=msg@entry=0x7f934b0037e6 "buffer overflow
        detected") at fortify_fail.c:33
#3  0x00007f934af81b81 in __GI___fortify_fail (msg=
    msg@entry=0x7f934b0037e6 "buffer overflow
    detected")
    at fortify_fail.c:44
#4  0x00007f934af7f870 in __GI___chk_fail () at
    chk_fail.c:28
#5  0x00007f934af81a6a in __fdelt_chk (d=<optimized
    out>) at fdelt_chk.c:25
#6  0x0000556c85476395 in sctp::sctp_server::
    sctp_receiver_thread(void*) ()
#7  0x00007f934d5bc6db in start_thread (arg=0
    x7f9343d84700) at pthread_create.c:463
#8  0x00007f934af6e61f in clone () at ../sysdeps/
    unix/sysv/linux/x86_64/clone.S:95
```

Fig. 23: Backtrace of the OAI5G-CN core network for the implementation flaws demonstrated in case OAI-CN-5G-1.

```
1  if(FD_ISSET(i, &read_fds)) {
2    if(i == ptr->getSocket()) {
3      if((clientsock = accept(ptr->getSocket(), NULL
           , NULL)) < 0) {
4        Logger::sctp().error(
5          "[socket(%d)] Accept() error: ...");
6        pthread_exit(NULL);
7      } else {
8        FD_SET(clientsock, &master);
9        if (clientsock > fdmax) fdmax = clientsock;
10     }
11   } else {
12     int ret = ptr->sctp_read_from_socket(i, ptr->
           app_->getPpid());
13     if(ret == SCTP_RC_DISCONNECT) {
14       FD_CLR(i, &master);
15       if(i == fdmax) {
16         while (FD_ISSET(fdmax, &master) == false)
               fdmax -= 1;
17 }}}}
```

Fig. 24: OAI-CN-5G: sctp_receiver_thread function.

memory errors, including those that may not immediately lead to a crash, the bugs identified in this study all result in crashes. As a result, the impact of employing Address Sanitizers for detecting non-crashing memory bugs cannot be fully assessed in this context.

Most OAI-CN-5G cases (from OAI-CN-5G-2 to OAI-CN-5G-11) were attributable to decoding errors. The NGAP fuzzing module corrupted the content of NAS messages, leading to inconsistencies in the retrieved and actual payload lengths following decoding. Consequently, memory copy operations based on the retrieved length and actual payload length led to buffer overruns and crashes.

Network issues were related to high data packet traffic and frequent establishment of SCTP connections between a monitor and the AMF (case OAI-CN-5G-1). During the testing of case OAI-CN-5G, regardless of which procedure code was selected for fuzzing, an AMF crash occurred after approximately 200 iterations. Because OAI-CN-5G is mainly implemented in C++, we could recognize the root cause in this case by using the function call backtrace generated from the crash core dump (Figure 23). Although an ABORT signal abnormally terminated the process, the backtrace shows that the crash originated from a function call from sctp_receiver_thread (#6) to __fdelt_chk (#5), which is a function used to check the validity of a passed file description in the C library.

On the basis of the clues in the crash dump, the critical codes that led to the aforementioned error were identified within the sctp_server::sctp_receiver_thread function (Figure 24). Upon investigation, the `clientsock` variable was discovered to continuously increase during testing, potentially exceeding the limit of FD_SETSIZE, which is 1024 by default in Linux. When this value is greater than or equal to 1024 and an FD_SET operation is attempted, an overflow issue occurs that ultimately causes the AMF to crash. This issue can be addressed and prevented by checking whether `clientsock` is less than FD_SETSIZE before executing the FD_SET operation. Moreover, when SCTP connections are being terminated or closed, a close operation must be explicitly performed on the file descriptor (fd) to release resources after calling FD_CLR. Suitably managing file descriptors would ensure that the value of `clientsock` returned from the `accept` system call does not continually increase.

VI. Conclusion

This paper presents a systematic fuzz testing approach for probing implementation flaws in the 5G core network. The proposed system uses a UE/RAN emulator to simulate the UE and a gNodeB for communication with a 5G core network using the NAS protocol and NGAP. The proposed FitM architecture mutates NAS and NGAP messages to trigger crash events. To explore additional test states in the protocol procedures, the FitM was modified to process encrypted NAS messages. Our approach was evaluated on three popular open-source 5G core networks, namely Open5GS, free5GC, and OAI-CN-5G; this fuzzer discovered novel implementation flaws in all three networks. Most exception events originated from "InitialUEMessage" in NGAP fuzzing and "Registration Request" in NAS protocol fuzzing. We successfully triggered crashes in various network functions of the AMF, AUSF, and UDM, discovered eight new crash events, and successfully reproduced 20 similar problems that were previously reported in the developer community. Our experimental results indicate that the proposed approach can considerably improve protocol integrity and network security.

References

[1] M. Cook, S. Quinn, D. Waltermire, and D. Prisaca, *Security Content Automation Protocol (SCAP) Version 1.3 Validation Program Test Requirements*. NIST IR 7511 Rev. 5, 2018.

[2] Y. Cao, Y. Chen, and W. Zhou, "Detection of pfcp protocol based on fuzz method. 2022 2nd international conference on computer," 2022.

[3] S. Potnuru and P. K. Nakarmi, "Berserker: ASN.1-based fuzzing of radio resource control protocol for 4g and 5g." 2021.

[4] 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.
[5] Y. Chen, Y. Yao, X. Wang, D. Xu, X. L. Chang Yue, K. Chen, H. Tang, and B. Liu, "Bookworm game: Automatic discovery of lte vulnerabilities through documentation analysis," 2021.
[6] 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.
[7] R. P. Jover and V. Marojevic, "Security and protocol exploit analysis of the 5g specifications," 2019.
[8] "UERANSIM, is the open-source state-of-the-art 5G UE and RAN (gNodeB) implementation." [Online]. Available: https://github.com/aligungr/UERANSIM
[9] M. Mahyoub, A. AbdulGhaffar, E. Alalade, E. Ndubisi, and A. Matrawy, "Security analysis of critical 5g interfaces," *IEEE Communications Surveys & Tutorials*, pp. 1–30, 2024.
[10] 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.
[11] J. Yang, S. Arya, and Y. Wang, "Formal-guided fuzz testing: Targeting security assurance from specification to implementation for 5g and beyond," *IEEE Access*, 2024.
[12] "Boofuzz: A fork and successor of the sulley fuzzing framework," 2017, accessed: 2024-03-01. [Online]. Available: https://github.com/jtpereyda/boofuzz
[13] "Sulley: A pure-python fully automated and unattended fuzzing framework." 2017, accessed: 2024-03-01. [Online]. Available: https://github.com/OpenRCE/sulley
[14] 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.
[15] S. Schumilo, C. Aschermann, A. Jemmett, A. Abbasi, and T. Holz, "Nyx-net: network fuzzing with incremental snapshots," in *Proceedings of the Seventeenth European Conference on Computer Systems*, 2022, pp. 166–180.
[16] J. Li, S. Li, G. Sun, T. Chen, and H. Yu, "Snpsfuzzer: A fast greybox fuzzer for stateful network protocols using snapshots," *IEEE Transactions on Information Forensics and Security*, vol. 17, pp. 2673–2687, 2022.
[17] R. Natella and V.-T. Pham, "Profuzzbench: A benchmark for stateful protocol fuzzing," in *Proceedings of the 30th ACM SIGSOFT international symposium on software testing and analysis*, 2021, pp. 662–665.
[18] Z. Salazar, H. N. Nguyen, W. Mallouli, A. R. Cavalli, and E. M. de Oca, "5greplay: a 5g networktraffic fuzzer - application to attack injection," 2021.
[19] 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.
[20] P. Chen and H. Chen, "Angora: Efficient fuzzing by principled search," in *2018 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2018, pp. 711–725.
[21] C. Aschermann, S. Schumilo, T. Blazytko, R. Gawlik, and T. Holz, "Redqueen: Fuzzing with input-to-state correspondence." in *NDSS*, vol. 19, 2019, pp. 1–15.
[22] "3GPP. TS 123 501 - V16.6.0 - 5G; System architecture for the 5G System (5GS)(3GPP TS 23.501 version 16.6.0 Release 16))."
[23] "3GPP. TS 129 502 - V15.9.0 - 5G; 5G System; Session Management Services; Stage 3(3GPP TS 29.502 version 15.9.0 Release 15))."
[24] R. T. Fielding, *Architectural styles and the design of network-based software architectures*. University of California, Irvine, 2000.
[25] "3GPP. TS 138 306 - V15.3.0 - 5G; NR; User Equipment (UE) radio access capabilities (3GPP TS 38.306 version 15.3.0 Release 15))."
[26] "3GPP. TS 138 401 - V15.2.0 - 5G; NG-RAN; Architecture description (3GPP TS 38.401 version 15.2.0 Release 15)."
[27] "3GPP. TS 133 501 - V16.3.0 - 5G; Security architecture and procedures for 5G Systeem(3GPP TS 33.501 version 16.3.0 Release 16))))."
[28] "3GPP. TS 124 501 - V16.5.1 - 5G; Non-Access-Stratum (NAS) protocol for 5G System (5GS); Stage 3 (3GPP TS 24.501 version 16.5.1 Release 16)."
[29] "3GPP. TS 138 413 - V16.2.0 - 5G; NG-RAN; NG Application Protocol (NGAP)(3GPP TS 38.413 version 16.2.0 Release 16)."
[30] P. Hintjens, *ZeroMQ: messaging for many applications*. O'Reilly Media, Inc., 2013.
[31] "Pycrate provides basically a runtime for encoding and decoding data structures, including CSN.1 and ASN.1. Additionally, it features a 3G and LTE mobile core network." [Online]. Available: https://github.com/P1sec/pycrate
[32] "A general purpose fuzzer radamsa." [Online]. Available: https://gitlab.com/akihe/radamsa
[33] "An open-source project for 5th generation (5g) mobile core networks(stage 3)." [Online]. Available: https://www.free5gc.org/
[34] "An implementation of 5gc and epc using c-language. and webui is provided for testing purposes and is implemented in node.js and react." [Online]. Available: https://open5gs.org/open5gs
[35] "A 3GPP-Compliant 5G Standalone (SA) CN implementation with a rich feature set." [Online]. Available: https://openairinterface.org/oai-5g-core-network-project/
[36] "A tool for defining and running multi-container docker applications." [Online]. Available: https://docs.docker.com/compose/
[37] A. Fioraldi, D. Maier, H. Eißfeldt, and M. Heuse, "{AFL++}: Combining incremental steps of fuzzing research," in *14th USENIX Workshop on Offensive Technologies (WOOT 20)*, 2020.
[38] C. A. Lattner, "Llvm: An infrastructure for multi-stage optimization," 2002.

APPENDIX A
5G KEY DERIVATION

The key derivation and distribution scheme presented in Figure 3, based on [27], outlines the procedures for deriving all keys within the 5G core network. The following keys are implemented in this paper:

- **Keys in the ARPF.** The ARPF stores the long-term key K, which must be 128 or 256 bits long.
- **Keys in the AUSF.** The AUSF derives the anchor key, K_{SEAF}, from the authentication key material received from the UDM during the authentication and key agreement process.
- **Keys in the SEAF.** Upon successful primary authentication, the SEAF receives K_{SEAF} from the AUSF and derives K_{AMF} from it. This key is then forwarded to the AMF.
- **Keys in the AMF.** The AMF receives K_{AMF} either from the SEAF or another AMF. It then derives K'_{AMF} from K_{AMF} for use in inter-AMF mobility, where the receiving AMF adopts K'_{AMF} as its own K_{AMF}. Additionally, the AMF derives K_{NASint} and K_{NASenc} to secure the NAS layer.