AMFuzz: Black-box Fuzzing of 5G Core Networks

Francesco Mancini
CNIT NAM Lab / Univ. of Rome Tor Vergata
Rome, Italy
francesco.mancini@cnit.it

Sara Da Canal
Univ. of Rome Tor Vergata
Rome, Italy
sara.da.canal@uniroma2.it

Giuseppe Bianchi
CNIT / Univ. of Rome Tor Vergata
Rome, Italy
giuseppe.bianchi@uniroma2.it

***Abstract*—As the shift to 5G continues, the growing emphasis on software within network architecture poses new testing hurdles. Testing becomes particularly challenging in diverse network environments involving multiple vendors, where access to the Core Network source code is limited. This study promotes 'protocol fuzzing', namely the assessment of how the network responds to malformed or unexpected protocol messages, as a viable and effective approach to tackle these challenges. Our approach involves the development of a black-box fuzzing tool serving as an intermediary between users and the Core Network. We specifically address the fuzzing of the Access and Mobility Management Function (AMF), owing to its accessibility from external vantage points (UE and gNB). The methodologies discussed here are readily adaptable to other 5G core network functions. Validation of the fuzzer is carried out by testing it on three open-source Core Network implementations, revealing a number of implementation bugs, thus proving its effectiveness.**

***Index Terms*—5G, mutation-based fuzzing, security assurance, AMF, black-box, feedback**

I. Introduction

The cellular network has experienced significant growth to meet users' increasing demand for connectivity. With the introduction of the 5G, new usage possibilities have been opened up, making security a crucial aspect. Although the 5G Core Network has been designed with a "security by design" approach, ensuring the presence of security features as an integral part of network function development, the area of "security assurance", i.e. trust in the actual implementation of security measures, remains a gap. It is widely recognised that security assurance tests must be conducted by parties outside the network manufacturer. As we shall see, this implies the need to carry out tests without access to the source code, introducing quite a few challenges. To address this aspect of security assurance, the Third Generation Partnership Project (3GPP), namely the organization which governs the standardization process of mobile cellular systems, developed a set of Security Assurance Specifications (SCAS), which represent a comprehensive set of tests to assess the correct implementation of security functions. These are positive tests: they check whether the network is behaving correctly with respect to the security specifications of the standard. This includes creating abnormal scenarios to uncover flaws in the handling of corner cases. However, this approach does not provide sufficient assurance of the security of the network. As stated by the European Union Agency for Cybersecurity (ENISA) [1]:

> *"Additional testing should be considered, which could involve negative tests and fuzzing of individual components, integrated systems and the network as a whole."*

In the case of negative testing, the tests analyze the reaction of the network to unexpected events not covered by the standard. In this context, fuzzing is a type of negative testing that identifies vulnerabilities by sending incorrect data (e.g., malformed messages, larger-than-expected input, etc.) to the network. Fuzzing can be classified based on how the mutated input is generated and whether the execution mode is black-box or white-box. This approach is widely used in various areas of the software world [2]–[4], with very advanced techniques to maximize findings (e.g., fuzzing with code coverage). The introduction of software as a predominant aspect of the architecture of 5G networks enables the inheritance of experience and tools for fuzzing. However, applying the standard strategies as they are does not yield satisfactory outcomes [5]. Moreover, software fuzzing usually relies on the assumption that source code is available, while we want to test commercial mobile networks, so we need to assume a black-box approach. This paper explores challenges in conducting a fuzzing campaign in a 5G Core Network, specifically focusing on the Access and Mobility Management Function (AMF), the vital component linking the Radio Access Network (RAN) to the control plane. We introduce a testing architecture to address these challenges, presenting a proof-of-concept validated on three open-source Core Network implementations.

The remainder of the paper is structured as follows. Section II provides a brief background on the 5G Core Network and its protocols, as well as some details on fuzzing methodologies. Moreover, it also places our contribution in the context of related works. In section III we motivate why standard fuzzing techniques are not effective in the context of 5G, highlighting what challenges a security tester must tackle. In Section IV we present our testing architecture and how it can overcome the challenges outlined. Section V discusses the proof-of-concept implementation and experimental results. Finally, Section VI presents the conclusions and future research directions.

This work was partially supported by the EU under the Italian National Recovery and Resilience Plan (NRRP) of NextGenerationEU, partnership "Telecommunications of the Future" (PE00000001 - program "RESTART") and "SEcurity and RIghts In the CyberSpace" (PE00000014 - program "SERICS").

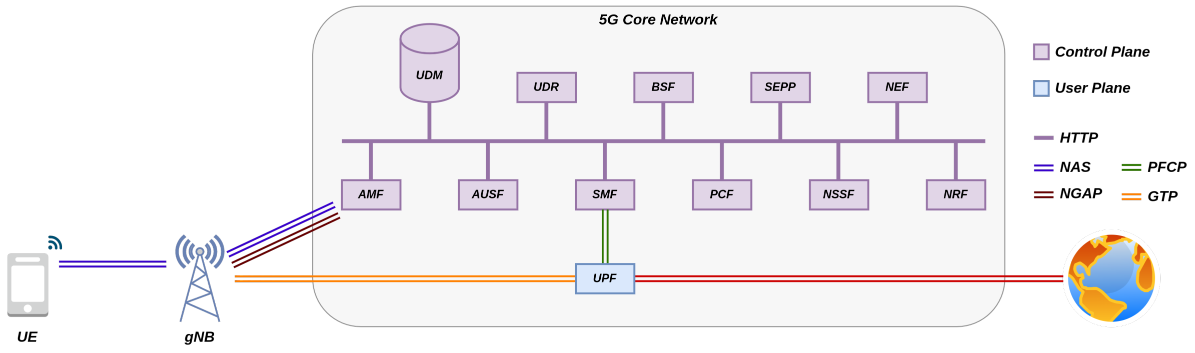

Fig. 1. 5G Core Network Architecture and Protocols

II. Background and Related Work

A. 5G Core Network

The increasing demand for flexibility in 5G networks has stimulated a transition from hardware components to network functions, adopting the paradigm of Software Defined Networks (SDN) and technologies like Network Function Virtualisation (NFV) to realise this softwarization of the network. The 5G Core Network has brought another major innovation – the service-based architecture – which enables the various network functionalities of the control plane to operate autonomously and intercommunicate using a standard API set. A 5G core is composed of numerous network functions, each responsible for a specific functionality. For instance, the Access and Mobility Management Function (AMF) manages user access to the network and handover between cells, the Session Management Function (SMF) is responsible for managing PDU sessions or the Authentication Server Function (AUSF) manages user authentication and autorization. Nevertheless, it must be highlighted that the interfaces between the control and user planes are still point-to-point, as illustrated in Figure 1.

B. Protocols in 5G Core Network

Interactions between network components are minutely defined in the 3GPP technical specifications. These provide for the following protocols:

- **HTTP**: communication between control plane components, within the Service-Based-Architecture (SBA),
- **Next Generation Application Protocol (NGAP)**: Communication between gNB and AMF,
- **Non Access Stratum Protocol (NAS)**: Communication between UE and AMF,
- **Packet Forwarding Control Protocol (PFCP)**: Communication between Session Management Function (SMF) and User Plane Function (UPF),
- **GPRS Tunneling Protocol (GTP)**: Communication between gNB and User Plane Function (UPF), or between two UPFs.

Network functions utilize the HTTP protocol to fulfill internal control plane functionalities, such as the generation of authentication vectors. In this new generation of Core Networks, a service-oriented approach is applied, in which the specification of network capabilities follows a systematic and well-defined structure. This methodology not only improves interoperability but also facilitates modular and agile network design. The gNB and AMF communicates via the NGAP protocol for overseeing mobility, session, and communication management. Structured as a message-based protocol, NGAP employs ASN.1 - a concise and standardized data representation format - and SCTP as transport protocol. The use of ASN.1 ensures a compact representation of data, optimizing bandwidth utilization. The NAS protocol governs signaling communications between UE and AMF. As these messages traverse the air interface, the gNB acts as a transparent relay, seamlessly forwarding them to the core side. In particular, these messages are encapsulated within NGAP messages. Furthermore, the NAS protocol enhances the security of these communications by providing both integrity and encryption. PFCP is the control protocol for establishing, modifying and releasing protocol data unit (PDU) sessions, as well as dynamically allocating and deallocating resources for user plane data streams. Finally, the GTP protocol takes care of encapsulating user data during transport from the gNB to the UPF and vice versa, thus allowing the UE to communicate with the data network (e.g., the Internet).

C. Fuzzing approaches

The term fuzzing was first used in 1988 by Professor Barton Miller at the University of Wisconsin. He was sending commands to a UNIX system over a highly disturbed dial-up connection. As a result, extra characters were frequently sent, and he noticed that this often led to program crashes [6]. Over time, fuzzing has gradually become more popular and more sophisticated, different tools have been developed to apply it, and different possible approaches to this testing method have been standardized.

Fuzzing techniques can be divided into three kinds: black box, white box, and gray box depending on how much information they require from the target program at run-time [7].

1) Black-box fuzzing: This approach assumes that no information of any kind is obtainable from the program under consideration; inputs are generated without having information related to internal implementation or coverage achieved, but

ISBN: 978-3-903176-61-4

with predefined random mutations, often relying on knowledge of which inputs are valid or not to define mutation rules.

2) White-box fuzzing: White-box fuzzing is a testing method that analyzes deeply a program's internal logic. It employs dynamic symbolic execution and heuristic search algorithms to thoroughly explore a program's execution paths. White-box fuzzing requires knowledge of the target program to guide test case generation.

3) Grey-box fuzzing: Grey-box fuzzing operates between black-box and white-box fuzzing, leveraging partial knowledge of the target program to uncover software errors effectively. A common technique in grey-box fuzzing is code instrumentation, providing real-time code coverage. This information guides mutation strategies, enhancing the creation of test cases to explore more execution paths or identify bugs efficiently [8].

Probably the most crucial aspect of a successful fuzzing campaign is creating good test cases that will interact with the target in a way likely to expose any defects. There are three main approaches to creating the test cases or inputs for use in a fuzzing campaign; mutation, generation and evolution.

- **Mutation-based fuzzing** starts from a seed of valid inputs, which are repeatedly modified or corrupted to generate new test cases,
- **Generation-based fuzzing** works by creating test cases based only on some kind of model that describes a valid input, like a grammar or a format specification,
- **Evolution-based fuzzing** also referred to as guided fuzzing, creates new test cases based on the response of the target program to previous test cases. This can be an extension of either mutation or generation fuzzing, as either technique can be guided to create the new test cases [9].

D. Related Work

Fuzzing is a growing testing technique in the field of software testing, and there are several fuzzers that operate in white-box mode, passing input directly to specific functions in the code, such as [10].

However, this approach is not scalable as it requires a significant effort to analyze each implementation. In fact, for each fuzzing campaign, it is necessary to thoroughly analyze the code to determine the functions to be tested and write ad-hoc tests that are unlikely to be generalized to other implementations. Moreover, in the case of closed source implementations, this approach is not feasible. Our approach overcomes these limitations by using an implementation agnostic approach, relying on what is defined by the standard.

However, this approach is less applicable for network testing, as network packets are the required input in such cases. Therefore, there has been the development of several fuzzers, which can send network packets, particularly for widely employed protocols [11]–[13]. Previous attempts tried to apply one of them, AFLnet, to NGAP testing, but were unsuccessful. The tool necessitates code instrumentation for mutation-based guided fuzzing, making it unsuitable for a purely black-box

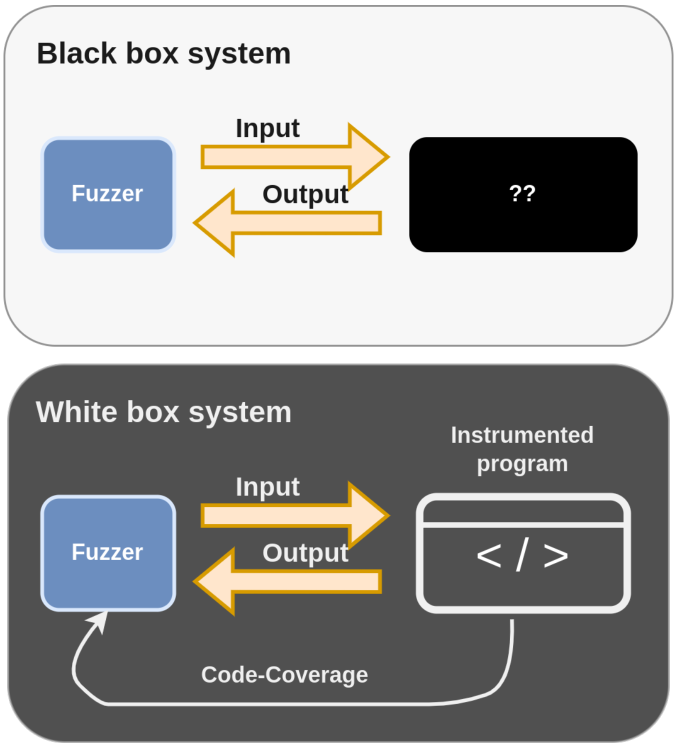

Fig. 2. White-box approach vs. Black-box approach to fuzzing

approach. It is better suited for protocols following a sequential request-response model, unlike our situation where we may encounter multiple sequential messages in the same direction, and not always have response messages.

However, applying generic network fuzzers to mobile networks entails several additional challenges and does not produce the desired results, as general-purpose fuzzers can only detect very basic bugs. We attempted to create our own tool to address this problem. A similar effort has been undertaken for LTE, although with a generation based approach instead of a mutation based one, and related studies are [14]–[16].

Regarding the fuzzing of the 5G Core Network, there are two interesting works, [17], [18], which complement our approach as they focus on the verification of the interface between the UE and the Radio Access Network, also using NAS messages, although the underlying protocol is different. Another related work is 5Greplay [19], [20], which focuses on testing the Core Network, including our own protocols, but employs a different approach. 5Greplay is indeed designed to replay network packets with and without modifications. This tool also allows the creation of specific test cases by replaying a pcap trace. However, the main functionality of 5Greplay is the modification of pcap traces by filtering or duplicating packets, and it has several limitations when it comes to fuzzing the message fields: it does not allow in-depth editing and is limited to reading and editing the packet header fields. We drew inspiration from this tool but extended our focus to include fuzzing the message body as well. OSS-Fuzz [21], a project which aims to deploy guided in-process fuzzing as a standard in open-source software development, contains some work related to Open5GS, a 5G Core Network open-source, that, while limited to open-source implementations, performs work on the NAS protocol that complements ours. Specifically, it focuses on fuzzing the decoding function, which we are not testing, as we propose correctly encoded messages.

III. Challenges in fuzzing 5G Core Network

Fuzzing the components of a 5G Core Network cannot be conducted simply by applying the approaches and practices employed in the software world. In this section we discuss the challenges that limit the feasibility of typical approaches and their effectiveness during testing. While some of this challenges are not specific to 5G, the overall complexity of the network means that many problems of different nature occur simultaneously, making them more difficult to deal with. Moreover, often software-fuzzing solution are tailored for a specific program, while we aim to realize a tool that can be generally applied to different core network implementations, adding another layer of complexity. We focus primarily on the AMF, the pivotal function of the control plane that manages the access and mobility aspects of UEs, as it presents all the obstacles that can be experienced. Furthermore, this network function exposes one of the few Core Network interfaces that can be accessed by the external world, namely interface N1 by which any UE can interact through the NAS protocol. What is presented in this section can be easily adapted and generalized for other network functions.

A. White-box approach

The approach commonly used in software fuzzing assumes that the source code is available, a specific function is identified for testing and the fuzzed parameters are passed as input to the function under test. The major problem in employing this method is the near impossibility of obtaining the source code of a commercial network, given the competitive nature of the market. But even assuming that the code is available, there are several other challenges:

1) External dependencies: The Core Network architecture resembles a microservices architecture, where various network functions collaborate. The AMF relies on interactions with other network functions for proper functioning. This complexity in testing arises because the entire Core Network must be active and available during test. Otherwise, it becomes necessary to simulate the responses that the AMF require. Both these solutions add complexity.

2) Implementation Challenges and Scalability Issues: It is quite difficult to pinpoint a specific area for fuzzing, due to the function complexity. Based on the analysis of open-source implementations, the AMF is generally an asynchronous function with numerous threads and message queues, so it is not easy to follow a specific path in the code. Moreover, identifying which function to fuzz is a challenge of itself, that needs to be repeated for every different Core Network implementation, considering that they probably do not share a common structure and common functions. This, together with the difference in programming used in different implementations means that it is not possible to use the same tools, which are often language-dependent, on multiple cores, posing a significant scalability issue.

3) State initialization: The AMF, being a stateful function, tailors its responses based on its current state. Running tests requires initializing this state. This again poses complexity and scalability problems, since it amounts to initializing complex, and often different between different implementations of the Core Network, data structures. Moreover, when a crash occurs, distinguishing whether it results from a bug or incomplete state initialization becomes challenging due to the complex nature of the initialization process.

B. Black-box approach

The black-box approach does not require access to the source code, but rather an access point to interact with the component under test. This approach is more suitable in this context. It happens very often, in fact, that vendors do not give direct access to their commercial implementations when contracting third-party services to test and debug their networks, and moreover, a black-box approach can be used to test a network from an outside perspective. The approach would seem simpler than the white-box approach and would be scalable across multiple implementations, since the interfaces are standard. However, pitfalls lie hidden that can nullify this methodology.

1) Protocol restrictions: The protocols employed in the 5G core have precise structures and encodings, as explained in subsection II-A. Therefore, sending messages that do not conform to the specifications will result in the interruption of processing at the initial stages, during the parsing/decoding phase. Although this methodology is excellent for stressing the component on message structure management aspects, on the other hand it is ineffective on its own for testing the remaining functionalities. Indeed, a fuzzer that is unaware of input structures is unlikely to be able to explore all possible processing paths of the component in a reasonable amount of time (in the vast majority of cases it will stop at the initial phases). Therefore, the basic fuzzing technique should be enriched with the use of structure-aware fuzzing [22]. However, the security aspects associated with the NAS protocol must be taken into account. Part of the messages, in fact, require encryption and integrity to be processed properly. These rely on specific session keys generated by the key agreement procedures provided by the standard (e.g. 5G-AKA). Alterations to the packet must not compromise integrity, as any such alteration would cause the network to reject the packet and consider it as an attempted attack. Similarly, modifications that alter the ciphertext could lead to corruption of the plaintext message structure, resulting in decoding/parsing errors.

2) Procedure restrictions: Communication in the network is strictly defined by a set of procedures. For each of these procedures, the entities involved and the sequence of messages to be exchanged are specified, covering possible alternatives and how to handle anomalies. It is therefore essential to

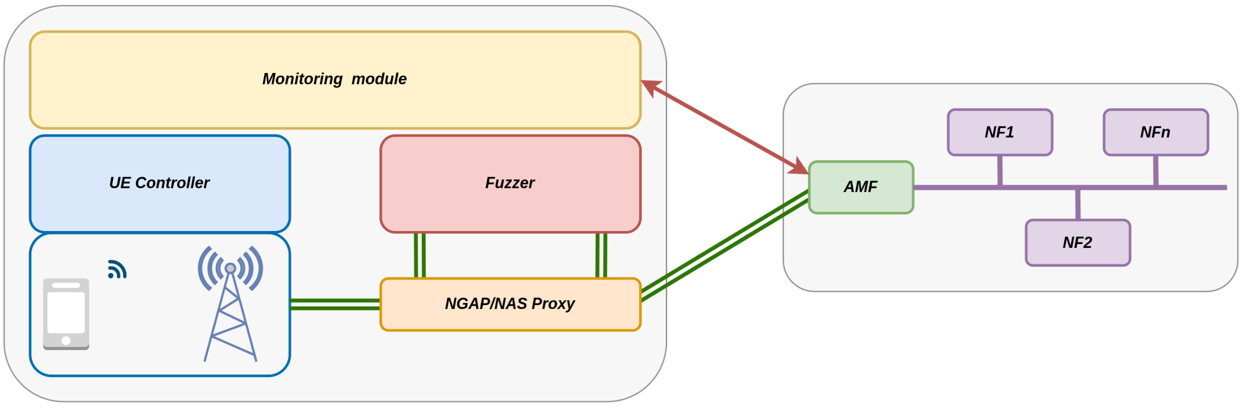

Fig. 3. Fuzzing framework modular system architecture.

adhere to the procedural aspects in order to explore further the behavior of the component under test. The employment of structure-aware fuzzing by itself in these circumstances may not be effective in performing a complete test.

IV. 5G Fuzzing Architecture

This section presents the proposed architecture for managing fuzzing in the 5G Core Network. This architecture is designed with a modular structure, strategically aimed at improving manageability and providing flexibility for seamless future extensions. Each module encapsulates specific functionality, enabling clear separation and simplifying maintenance and upgrades. This modular approach not only streamlines the management process, but also facilitates the integration of additional components without disrupting the existing structure. Figure 3 shows the system architecture and the connections between its primary modules, assuming the Core Network is deployed in a containerised environment.

A. Input generation

One of the main challenges in implementing a fuzzing system is the generation of inputs. Our solution is based on a mutation-based approach with adaptations to the system we are testing. Following this approach, the basic idea would be to use communication traces (e.g. pcap files) representing various scenarios (e.g., registration, deregistration, handover, etc.) as the starting seed corpus and let the fuzzer carry the alteration work. However, applying this approach in a raw way does not give the expected results. As a matter of fact, in order to reach a specific state to be tested, it is necessary to respond consistently to the messages sent by the AMF. As an example, let us consider the case where we want to test the network function for user session management aspects (i.e., PDUs, Protocol Data Unit Sessions). To achieve this state, the UE has to first perform the registration phase, with associated authentication. This requires a precise exchange of packets, the content of which must be specific and tailored to the needs of the AMF. Specifically, during the authentication and key agreement phase, the AMF sends a fresh challenge (i.e. never sent before) and the UE must reply with a result strictly linked to that challenge. A possible solution to this scenario would be to disable the verification aspects. However, this would involve access to the source code, thus invalidating the initial assumptions. Furthermore, this identical approach would have to be applied to all similar situations in the standard, which requires effort and expertise.

The approach we selected involves creating messages at runtime, intercepting them and modifying them before sending them back to the AMF. A proxy component was developed to act as an intermediary between the AMF and the RAN. Basically, this component transparently forwards NGAP/NAS messages between the gNB and the AMF. Hooks can be added to this component to process each message that passes through it. In this scenario we register our fuzzing module as a hook. The proxy is part of the ScasDK development kit; more details are available in the dedicated paper [23]. In order to have a modular system, we avoided including the code responsible for simulating RAN and UE in the proxy, and instead used generic simulators. Using this approach, we can make sure that the message being considered is consistent with the state of the network and contains the required information, and then fuzz it. Decoupling the proxy, fuzzers and simulators has several advantages: we can perform the same message exchange with both modified and unmodified messages, to be sure that the exchange without fuzzers works properly. We can also easily replace the chosen simulators with updated versions or with different simulators in order to have an updated system without modifying proxy and fuzzer.

B. Fuzzer

The fuzzer was also designed as a modular system, including modules responsible for generating the seed, mapping the seed to specific changes in the packet, and managing encryption and integrity. The processing of each packet is done sequentially over three steps: *pre-processing*, *fuzzing*, and *post-processing*. During the pre-processing phase, the packet is prepared so that the fuzzer can process it properly. This includes decoding the traffic according to ASN.1 specifications and, if necessary, a decryption step in case the packet includes a NAS message. Furthermore, there is a specific module for generating session keys. Indeed, session keys are strictly dependent on data known a priori by the UE and the network (e.g. OPC, subscriber key, etc.) and data exchanged during authentication (e.g. RAND value, encryption and integrity algorithms). The former are configured in the module at startup, the latter are acquired by transparently extracting them from the communication. When the module acquires all the necessary data, it derives the encryption and integrity keys, which are then employed by other modules. During the fuzzing phase, the dedicated module can introduce various modifications to the network packets. The fuzzer decides how to modify each packet based on a sequence of bytes, called a seed. For each change to be made, the fuzzer requests the seed from an appropriate generation module. Both NAS and NGAP packets consist of a header followed by a series of information elements (IEs). Based on the input seed, a particular IE is selected in the message and its fields are modified, mapping the seed to the allowed values. This modification assigns a specific value to the IE taking into account the correct data type for each field. Different message types correspond to different IEs in the message, so the mutation process must be aware of the message type in question, and the same random seed is mapped to make different changes to different message types. Some seeds are alternately mapped to make changes to the header, which consists of procedure code, message type, implemented security type, and message authentication code, if necessary. Other seeds are used to send the packet without internal changes, but with some delay or repeatedly. Finally, the post-processing phase deals with preparing the modified message so that the network can process it properly. This includes encrypting the message and generating the new integrity signature, in the case of a NAS message, and encoding it according to the ASN.1 standard.

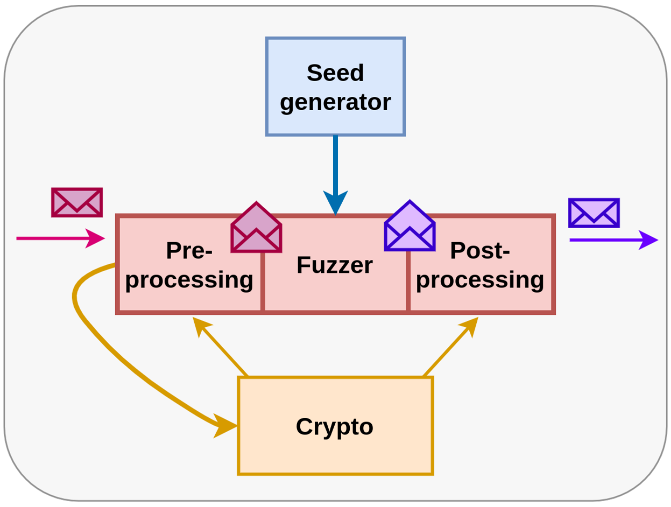

Fig. 4. Fuzzer module internal architecture.

C. Monitoring and Feedback

There are various ways to accomplish this task. For a purely black-box approach, a heartbeat system can be implemented using another gNB that consistently attempts to communicate with the core and verify its response. Alternatively, a simpler solution is viable if we are willing to compromise with a grey-box system. In this case, if the AMF and the monitor operate on the same hardware, the monitor can continuously assess the AMF's status using the PID or a container-specific API in the case of a containerized environment. While the monitor does not require access to the Core Network's code, it is not totally decoupled from the core due to this constrain.

Guided fuzzing brings an additional responsibility for the monitoring system: gathering feedback from the AMF to determine whether the applied seed is promising for future changes. This can be achieved with instrumentation to obtain code coverage, following an approach similar to the one proposed by AFL [24], although this approach involves having at least code binaries. If we want a completely black-box system, code coverage is not an applicable metric, and the proposed approach involves intercepting the AMF response and establishing criteria to discern whether a particular message type indicates positive or negative feedback.

D. Post Analysis and Test Repeatability

The verification of a previously observed finding is important. Being able to verify a finding provides some guarantee that it is not product of chance but that the observed phenomenon is stable [25]. Fuzzing is a testing method that relies on random changes, so we need to capture the introduced random element to allow the same test to be repeated. As a first approach to support the analysis phase, we save a pcap trace of the communication for each execution. This simplifies the root cause analysis phase and, considering that reading messages from a pcap trace and sending them to a network interface is a relatively simple process, it may seem like the solution for performing repeatability tests. However, this approach present several critical point:

- In a 5G Core Network, authentication is based on a challenge-response method: a random sequence of bytes is sent by the AMF, encrypted by the user and sent back, to prove that they know the encryption key of the user they claim to be. If a new registration procedure is started, a different challenge needs to be encrypted, so simply replaying old message will not work. This problem can be solved by going to recompute the authentication response message each time instead of replaying a past message, but we may lose information about how the message fields were fuzzed, making the test invalid.
- Similarly, for each message, it would be necessary to recompute integrity and encryption. However, there is a risk of losing information if the fuzzing process has altered the header fields related to these calculations.
- In the testing process, the timing of sending is crucial, but the pcap trace does not retain information such as the delay before sending a specific message. As communication is not solely based on a request-response pattern, determining when to send a message for the test to be accurately reproduced poses a challenge.

ISBN: 978-3-903176-61-4

For all these reasons, we opted for a different approach. Replication of the test is achieved by saving the random seed used for modifications and conducting a new test where messages are generated by UE and RAN but modified using the same random seed. In this situation we keep as much of the same information as possible between tests.

V. Proof-of-concept implementation and results

A. Implementation

The fuzzing framework, implemented in Python, utilizes pycrate [26] - a preexisting library facilitating the encoding and decoding of messages in the ASN.1 format. This library also offers additional functionalities for encoding, decoding, encryption, and MAC code generation (assuming knowledge of cryptographic keys) for NAS messages, and can be used to obtain network packets' structure. We employed Ueransim [27], an open-source project offering UE and RAN components along with a CLI for command transmission, to simulate these elements. Sending different command through the CLI we are able to trigger different procedure and have a wider range of fuzzable messages. To seamlessly integrate with our system, we created a Python controller to interact with Ueransim, which is the fourth main module showed in Figure 3.

The fuzzer needs to be aware of the NAS or NGAP message type being processed, as each message type corresponds to different information elements. Currently, the fuzzer focuses on a subset of possible messages related to registration, setup, update, and release of communication sessions, as well as deregistration. Messages related to cell handover have not been considered at this stage, although we aim to integrate all possible NAS and NGAP procedures in the future. However, we have implemented a class that takes a sequence of random bytes as a seed and can apply it to any information element. With a nearly complete list of information elements, expanding to integrate other message types only requires configuring the appropriate combination of IEs according to the appropriate structure.

The monitoring system was implemented as a grey-box monitor, designed to observe the state of the AMF container. To align with the validation architecture, we utilized APIs tailored for monitoring Docker containers. However, the modular structure of the system provides flexibility in replacing the monitoring component as needed, especially when conducting tests on networks deployed in a different mode. Additionally, this flexibility allows the integration of a feedback mechanism for mutation-based guided fuzzing.

It is possible to display on screen all messages sent to and from the AMF during the framework's execution, aiding in debugging in case of a crash. Moreover, to ensure persistent tracking, a module in the proxy has been implemented to save intercepted data onto a pcap trace, generated using the Python Scapy library.

B. Validation and Results

1) Testing Methodology: The validation process involved testing three different open-source 5G Core Networks, Open5GS, free5GC e OAI 5GC [28]–[30]. All Core Networks have been deployed using a Docker container for each network function, orchestrating them with Docker Compose. The versions considered for each core are respectively 2.6.4 for Open5GS, 3.3.0 for Free5GC and 1.5.1 for OAI.

Tests were initially designed to uncover bugs during the registration phase. Subsequently, the approach shifted to allowing a smooth registration process, enabling further testing of session setup phases and deregistration. Forty tests were performed on each core for each of these test cases. Each test was deemed concluded when the monitoring function detected a crash or when the fuzzing epoch ended without causing issues in the AMF. A fuzzing epoch was considered finished if the proxy did not intercept any messages from the AMF or the RAN for over a minute. The wait time depends on the network's speed, favoring smaller values for efficient testing, yet avoiding premature epoch endings with excessively small values. Overall, this testing method is quite slow. Different tests have vastly different execution time, depending on the seed considered and which changes are introduced. On average, a single tests takes approximately from a couple of minutes to half an hour.

2) Preliminary results: Our framework successfully uncovered bugs in each tested core, identifying some that were already flagged as issue to resolve, and also unveiled previously unknown bugs. The known issues served as benchmarks to assess the effectiveness of our automated approach. We discovered a total of 7 bugs. Bugs seems more commonly related to NAS instead of NGAP fuzzing, probably due to the added complexity of NAS specification, which are not in ASN.1 format. The objective of these tests was to validate the framework, and the underlying causes of the newly discovered bugs have not been investigated yet. Table I shows our findings.

Core Network	Registration	Session setup	Deregistration
Open5GS 2.6.4	2	0	2
Free5GC 3.3.0	1	0	0
OAI 5GC 1.5.1	2	1	0

TABLE I
Bugs Found

3) Analysis of a known bug: To provide an example of the types of bugs identified and to illustrate why detecting them would be challenging without this form of testing, we showcase a known bug in Open5GS. The identified bug pertains to the handling of an Information Element containing the 5G Mobily Identity, specifically the UE identifier, which can be of various types. This identifier is included in different message types, with the registration request being a primary example. Due to flawed validation checks, a specific type of identifier, the SUPI (Subscription Permanent Identifier), is mishandled, resulting in an AMF crash. The SUPI, although typically based on IMSI (International Mobile Subscriber Identity), can also rely on other implementation-specific parameters. It is in this nonstandard scenario that the crash occurs. Additionally, executing identical tests with the same random seeds on all Core Networks exposed variations in execution between OAI and the other two cores. We will not delve into the causes of these differences, our primary focus was on validating the framework rather than investigating the root causes of distinct bugs, but this difference may mean that one of the networks is not conform to 3GPP specification. In general, confronting executions between multiple networks offers an interesting way to discover bugs which do not result in an immediate crash but cause the network to not respond properly, so we aim to integrate a module for automatic trace comparison.

VI. Conclusion

The analysis on different Core Networks demonstrated the effectiveness of this method in detecting implementation bugs, particularly useful for unexpected situations not considered during development. Analysis on several Core Networks also allowed us to evaluate the performance of our tool. Specific challenges emerge in handling NAS messages versus NGAP messages, with crashes concentrated when fuzzing NAS PDU. The complexity of 3GPP specifications for NAS messages, implemented outside the ASN.1 coding rules, could contribute to this complexity. It is crucial to carefully evaluate the execution parameters to optimize test time and results, considering the significant execution time of these types of tests.

Many improvements evaluated during design and then not included in this first iteration of development can be introduced:

- Insertion of a feedback mechanism to improve the mutation system, making it no longer solely random but driven by coverage and state.
- Increasing the tested area by inserting new types of NAS PDUs and NGAP PDUs.
- Increasing the area of Core tested, for example by performing fuzzing on 5GSM packets (NAS packet used in session handling and forwarded from the AMF to the SMF) to test other network functions.

References

[1] ENISA, "Security in 5G Specifications - Controls in 3GPP," European Union Agency for Cybersecurity (ENISA), Tech. Rep., 02 2021.

[2] J. Krupp, I. Grishchenko, and C. Rossow, "AmpFuzz: Fuzzing for amplification DDoS vulnerabilities," in *31st USENIX Security Symposium (USENIX Security 22)*. Boston, MA: USENIX Association, Aug. 2022, pp. 1043–1060.

[3] T. Yin, Z. Gao, Z. Xiao, Z. Ma, M. Zheng, and C. Zhang, "KextFuzz: Fuzzing macOS kernel EXTensions on apple silicon via exploiting mitigations," in *32nd USENIX Security Symposium (USENIX Security 23)*. Anaheim, CA: USENIX Association, Aug. 2023, pp. 5039–5054.

[4] H. Green and T. Avgerinos, "Graphfuzz: Library api fuzzing with lifetime-aware dataflow graphs," in *2022 IEEE/ACM 44th International Conference on Software Engineering (ICSE)*, 2022, pp. 1070–1081.

[5] M. Tedman, "The challenges of fuzzing 5g protocols," October 2021. [Online]. Available: https://research.nccgroup.com/2021/10/11/the-challenges-of-fuzzing-5g-protocols/

[6] B. Miller, L. Fredriksen, and B. So, "An empirical study of the reliability of unix utilities." *Commun. ACM*, vol. 33, pp. 32–44, 12 1990.

[7] E. Jääskelä, "Genetic algorithm in code coverage guided fuzz testing," Master's thesis, E. Jääskelä, 2016.

[8] H. Liang, X. Pei, X. Jia, W. Shen, and J. Zhang, "Fuzzing: State of the art," *IEEE Transactions on Reliability*, vol. 67, no. 3, pp. 1199–1218, 2018.

[9] J. Fell, "A review of fuzzing tools and methods," *PenTest Magazine*, 2017.

[10] 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)*. USENIX Association, Aug. 2020.

[11] V. Pham, M. Böhme, and A. Roychoudhury, "Aflnet: A greybox fuzzer for network protocols," in *Proceedings of the 13rd IEEE International Conference on Software Testing, Verification and Validation : Testing Tools Track*, 2020.

[12] 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*, ser. EuroSys '22, 2022.

[13] A. Andronidis and C. Cadar, "Snapfuzz: high-throughput fuzzing of network applications," *Proceedings of the 31st ACM SIGSOFT International Symposium on Software Testing and Analysis*, 2022.

[14] 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)*, 2019, pp. 1153–1168.

[15] 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)*. Boston, MA: USENIX Association, Aug. 2022, pp. 1325–1342.

[16] 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*, 2014, pp. 323–332.

[17] 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)*, 2021, pp. 295–300.

[18] 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*, 2022, pp. 86–92.

[19] Montimage EURL, "5Greplay," 2021. [Online]. Available: https://github.com/montimage/5greplay

[20] Z. Salazar, H. N. Nguyen, W. Mallouli, A. R. Cavalli, and E. M. de Oca, "5greplay: a 5g network traffic fuzzer - application to attack injection," *Proceedings of the 16th International Conference on Availability, Reliability and Security*, 2021.

[21] K. Serebryany, "OSS-Fuzz - google's continuous fuzzing service for open source software." Vancouver, BC: USENIX Association, Aug. 2017.

[22] "Structure-aware fuzzing with libfuzzer," September 2019. [Online]. Available: https://github.com/google/fuzzing/blob/master/docs/structure-aware-fuzzing.md

[23] F. Mancini and G. Bianchi, "Scasdk - a development kit for security assurance test in multi-network-function 5g," in *Proceedings of the 18th International Conference on Availability, Reliability and Security*, ser. ARES '23. New York, NY, USA: Association for Computing Machinery, 2023. [Online]. Available: https://doi.org/10.1145/3600160.3605044

[24] "Afl technical details," July 2019. [Online]. Available: https://github.com/google/AFL/blob/master/docs/technical_details.txt

[25] O. S. Gómez, N. Juristo, and S. Vegas, "Replication, reproduction and re-analysis: Three ways for verifying experimental findings," *1st International Workshop on Replication in Empirical Software Engineering Research (RESER'2010)*, 05 2010.

[26] P1Sec, "Pycrate," 2016. [Online]. Available: https://github.com/P1sec/pycrate

[27] Aligungr, "Ueransim," 2021. [Online]. Available: https://github.com/aligungr/UERANSIM

[28] Sukchan Lee, "Open5GS," 2017. [Online]. Available: https://open5gs.org

[29] National Yang Ming Chiao Tung University, "Free5GC," 2021. [Online]. Available: https://free5gc.org

[30] Open Air Interface, "OAI 5GC," 2020. [Online]. Available: https://openairinterface.org/oai-5g-core-network-project/