PERFORMANCE EVALUATION OF 5G REPLAY ON 5G NETWORKS

A Project

Presented to the

Faculty of

California State Polytechnic University, Pomona

In Partial Fulfillment

Of the Requirements for the Degree

Master of Science

In

Computer Science

By

Dhruv Jigneshbhai Patel

2024

COMMITTEE MEMBERSHIP

PROJECT: PERFORMANCE EVALUATION OF 5G REPLAY ON 5G NETWORKS

AUTHOR: Dhruv Jigneshbhai Patel

DATE SUBMITTED: Fall 2024

Department of Computer Science

Dr. Mohammad I Husain
Project Committee Chair
Professor of Computer Science

Dr. Tingting Chen
Professor of Computer Science

ACKNOWLEDGMENTS

I want to express my profound gratitude to my Project advisor, Dr. Mohammad Husain, for their exceptional advice, guidance, support, and expertise during my project. Dr. Husain's insightful feedback, constant encouragement, and constructive criticism have not only shaped the direction of my work but have also been crucial to its success. I am deeply thankful for their unwavering commitment, patience, and dedication, which provided me with essential resources and mentorship.

I am also immensely grateful to my committee member, Dr. Tingting Chen, for their invaluable contributions. Dr. Chen's expertise, thoughtful suggestions, and critical evaluations have greatly enhanced the quality and depth of my research. Their thorough review, insightful perspectives, and assistance in refining my ideas have been greatly appreciated. Furthermore, I wish to acknowledge the collective support and encouragement of the entire committee. Their involvement and guidance have been key to my project's overall development and success.

Additionally, I would like to extend my appreciation to our university and department. The resources, facilities, and support offered have been fundamental in realizing this research endeavor.

Lastly, I am profoundly thankful to Dr. Husain and Dr. Chen for their support, expertise, and guidance. Their contribution has been instrumental in my academic and personal growth, and I am honored to have the opportunity to work with them.

ABSTRACT

The 5G network is evolving with a various number of new security challenges, and Fuzzing techniques came into the scenario with finding the vulnerabilities and testing the implemented system. The development of an effective fuzzer is a complex task, requiring much more than simply setting it up and running it against a network.

In most of the cases, Positive testing is carried out, which means validating software what it is designed to do but minimizing negative testing, i.e., checking whether the software or system can accept the unexpected input without showcasing unusual behavior or warnings. The goal of this project is to measure the performance of Fuzzer 5Greplay based on its capabilities to generate traffic in the network. It will be implemented on the two available 5G Core Network Implementation we have i.e., Open5GS Simulator and 5Gcore (Super Micro System). Moreover, from the above one of the solutions is commercial grade i.e. 5Gcore from Supermicro.

We will be testing the expected input/output and unexpected input on the system using 5Greplay and gather the data results and behavior of fuzzer on the above-mentioned 5G cores we have. The project aims to experiment on the different 5G cores using 5Greplay and analyze the system's behavior.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1 : INTRODUCTION

The evaluation of the 5G network comes with numerous testing challenges and objectives. 5G deployment introduces a new set of technologies such as network function softwarization enabled by software-defined networking (SDN) and Network Function Virtualization (NFV), Mobile Edge Computing (MEC), and Network Slicing (NS) (Salazar et al., 2021). The Fuzzers are used to find security vulnerabilities that can be missed by unit or system testing. Traditional fuzzers primarily focus on testing software by generating inputs through command lines or input files. Recently, some widely used continuous integration frameworks, such as GitLab, have begun incorporating fuzzing into their build pipelines (*Aki Helen / Radamsa · GitLab*, n.d.).

Fuzzing Network protocols is a different scenario and requires sending input with the help of network ports. Communication usually involves multiple network protocols, which are stacked on top of one another (nccgroup, n.d.). Another problem with fuzzing the 5G network is getting access to a 5G core component. There are two open-source solutions Free5GC and Open5GS (Open5gs, n.d.) (*Free5GC*, n.d.). Neither of these implementations are commercial grade but they do give a reasonable target to fuzz the system for free.

Additionally, there are different types of open-source network protocol fuzzers such as Fuzzowski 5GC, Frizzer, AFLNet, and 5Greplay. Firstly, Fuzzowski 5GC is a template-based generational fuzzer, which means that the format of the input is specified, and the values defined within the format are mutated using selected algorithms depending on the data type. It can fuzz the sequence of messages and Correcting checksums and length fields helps prevent early parsing errors (Salazar et al., 2021). Fuzzowski is a Black Box fuzzer as it has no knowledge of the System Under Test (SUT) other than the network API. Secondly, Frizzer is a black box-guided mutational fuzzer and it uses input and randomly mutates it using Radamsa (*Aki Helen / Radamsa · GitLab*, n.d.) It uses Frida to dynamically instrument the System under

Test (SUT) to provide code coverage feedback. While a fuzzer AFLNet is a guided mutational-based fuzzer. It uses example input and randomly mutates part of the input based on different mutation algorithms with no knowledge of the input data format. It uses state feedback from the network messaging to guide the fuzzing process. It is a grey box fuzzer as it uses source code instrumentation to generate the code coverage feedback. Lastly, the 5Greplay tool provides a feature to test 5G virtual network function and Intrusion Detection Systems (IDSs) by allowing the forwarding of network packets from one Network Interface Card (NIC) to another with or without modification of packets.

The 5Greplay tool supports the implementation of test cases by giving functionalities to create specific scenarios using PCAP files that contain 5G network Traffic data and execute them on a target network. The tool allows users to alter 5G packets in a very flexible way to perform tests and attack scenarios. All the fuzzers were tested against Access and Mobility Management Function (AMF) as there is more possibility of finding vulnerability in AMF. Also, it is stated by ENISA Organization (Salazar et al., 2021) that AMF can be vulnerable to replay attacks of Non-Access Stratum (NAS) signaling message between the User Equipment (UE) and Access and Mobility Management Function (AMF) on the N1 Interface. Protocols tested to find the vulnerabilities using Fuzzer were Next Generation Application Protocol (NGAP), GPRS Tunneling User Data Protocol (GTPU), Packet Forwarding Control Protocol (PFCP), and DIAMETER protocol.

CHAPTER 2 : LITERATURE REVIEW

Flaws in the implementation of network protocols are some of the most serious security problems. A security flaw would allow a malicious user to attack a vulnerable system. Approximately 85% of all vulnerabilities reported by the National Vulnerability database in the last 3 years can be exploited remotely on the internet (Gorbunov & Rosenbloom, 2010).

Fuzzing: It is a security testing approach based on injecting invalid or random inputs into a program to obtain an unexpected behavior and identify errors and potential vulnerabilities. There are many definitions of fuzzing but from (Sutton et al., n.d.) (Takanen et al., 2008) above is the summarized one.

The key idea of fuzzing is to generate relevant tests and be able to crash target and choose the most appropriate tools to monitor the process. A protocol fuzzer can be classified as Smart or dumb depending on its knowledge of the network protocol implemented by its target. 'Dumb' fuzzers are easy and immediately applicable to any protocol but it is measured to be 50% less effective than smart fuzzing. One of the dumb fuzzer example is Proxy Fuzz which is a man-in-middle nondeterministic network fuzzer. Second fuzzers which are known as 'Smart' fuzzers understand their message syntax, structure and field type and use this to efficiently fuzz the target. An example of such smart fuzzers is 5Greplay which has protocol awareness (NAS, NGAP), replay traffic attack, and targeted mutation by altering the message field. Disadvantage of 'smart' fuzzers include their reliance on the availability of a protocol's specification documents and the degree to which target implementation conforms to the published specification (Gorbunov & Rosenbloom, 2010).

The test cases that were tested on 5Greplay on the simulator were I) to determine whether the altered packets are accepted by the 5G core network, ii) To verify the traffic injection in the network works correctly, iii) To test the scalability of the tool. The test cases were tested based on four scenarios in the experiment: Scenario – 1, Modifying and injecting network traffic

offline where modifying packets from a pre-existing PCAP file and replaying it on a specific network interface card (NIC) and checking whether altered packets are processed by an Intrusion Detection System (IDS). Scenario - 2, Sending malformed packets against open-source 5G core network in real-time, and in this 5Greplay was configured to detect NGAP protocol messages sent by User Equipment (UE) during authentication exchange.

Scenario – 3, it evaluates the use of 5Greplay to perform a security test by modifying and injecting network traffic into a specific target, and it is reported by ENISA that Access and mobility Management Functions (AMF) are vulnerable to replay attack of the NAS Security Mode Control (SMC) procedure message. Scenario – 4: High bandwidth traffic generation, the main objective of this scenario is to check the scale of 5Greplay. For this, the tool was configured with libpcap and Data plane development kit (DPDK) to replay the traffic as much as possible. The experiment achieved the full rate of 9.56Gbps and 1.28 Mpps (Salazar et al., 2021).

2.1 5G Architecture

5G network is made up of major components such as User Equipment (UE), Radio Access Network (RAN), and 5G core. Here, the 5g core plays a key role in enabling the connectivity. It is responsible for many functions that, including Network Slicing, Control plane where, Access and Mobility Management Function (AMF), and Session Management Function (SMF) are part of it and AMF & SMF interact with each other using an interface called N11. SMF connects with other functions in the network such as User Plane Function (UPF), and Policy Control Function (PCF), to establish and manage connection.

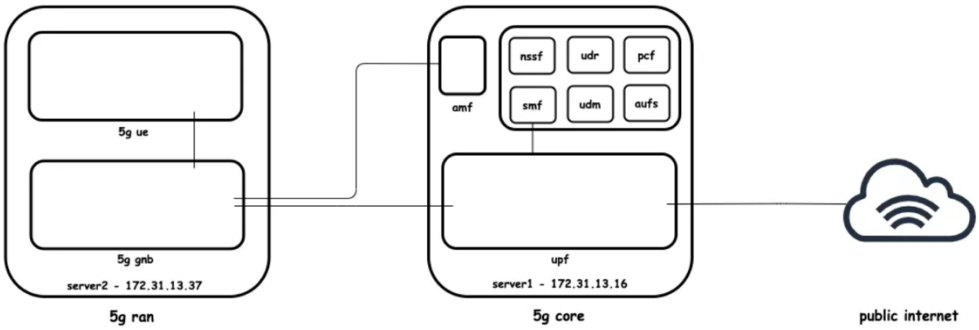

Figure 1 - 5G Architecture (Open5GS-Simulator)
(Adapted from "Deploying 5G Core Network with Open5GS and UERANSIM," 2024)

Open5GS is an open-source implementation of a 5G core and EPC written in C language. Open5GS was chosen to emulate the 5G core as it is freely available and actively being maintained. Due to it not being a commercial product and written in C, it makes an ideal target for fuzzing as it is unlikely to be as thoroughly tested. It is also more likely to be focused primarily on functionality, rather than security (nccgroup, n.d.).

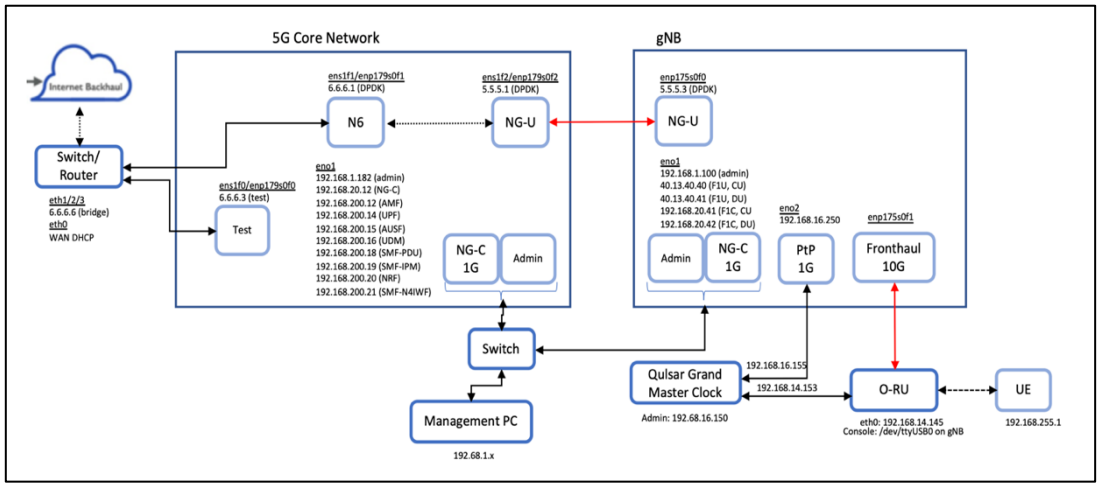

Figure 2 - 5G Architecture – Actual Network (Adapted from "Super Micro System")

The above Figure 2 above can be used to understand the different components and their IP addresses. The main components of the system are as follows:

- Core Network Server (CN) – Provides the 5G standalone core network function.
- Base Station Server (gNB) – Provides the 5G CU/DU functions.

- Radio Unit (RU) – Provides the radio communication to the wireless devices.
- Cell phone or User Equipment (UE) – 5G handset terminals for mobile access
- Internet Router – Provides NAT/Firewall services enabling a UE to access the public Internet.
- Management Access Point – Provides WIFI and/or Ethernet access to the management PC.
- Management PC or Laptop - Provides web and SSH access for server management.

From the above list of components, there are 4 main components as follows:

1) CN (Core Network): The 5G Core Network is the heart of 5G technology, handling essential functions like user connectivity, authentication, data management, and session control. It ensures devices stay securely connected, verifies user identities, and applies rules for data usage and quality. The core network also plays a key role in managing mobility, allowing for smooth transitions and uninterrupted communication. It's built to support a wide variety of use cases, from high-speed internet to low-latency applications such as autonomous vehicles and industrial automation. With its flexible and scalable design, the 5G Core Network is a cornerstone of the 5G system, enabling its efficiency and adaptability for future needs.
2) CU (Central Unit): The Central Unit (CU) is a key logical unit in the 5G Radio Access Network (RAN) that manages higher-layer, non-real-time network functions. It handles Radio Resource Control (RRC), which oversees radio connections and handovers, ensuring seamless communication for User Equipment (UE). Additionally, it manages the Packet Data Convergence Protocol (PDCP), responsible for encryption and header compression to improve security and data efficiency. Acting as an interface with the 5G Core, the CU manages session and mobility tasks while optimizing network resources for efficient operations, making it vital for reliable 5G performance.

3) DU (Distributed Unit): The Distributed Unit (DU) is responsible for low-latency, real-time processing in the 5G Radio Access Network (RAN). It handles Medium Access Control (MAC), which schedules radio resources and manages uplink/downlink data, and Radio Link Control (RLC), which ensures retransmission and error correction for reliable communication. Positioned closer to the physical radio, the DU connects to the Central Unit (CU) via the F1 interface and communicates with the Radio Units (RU) for data transmission. Its decentralized role enhances response times and resource efficiency, making it essential for 5G's real-time capabilities.
4) RU (Radio Unit): The Radio Unit (RU) manages radio frequency operations and interfaces with antennas in the 5G Radio Access Network (RAN). It converts digital signals from the Distributed Unit (DU) into radio waves for transmission and processes received radio waves back into digital signals, a process known as Radio Frequency (RF) Transceiving. The RU also performs Beamforming, directing radio signals toward specific User Equipment (UE) to improve signal quality and network performance. Located near the antenna site, it minimizes signal loss and supports scalable deployments by allowing multiple RUs to connect to a single DU, ensuring efficient and flexible network operations.

2.1.1 5G Core

The 5G core is the central part of the 5G network architecture, designed to handle data and service management between RAN and external networks i.e. internet. It is highly flexible, and cloud-native, allowing dynamic scaling, network slicing and improved network efficiency. It is built on cloud architecture to meet the performance and scalability requirements of modern 5G networks. It allows resources to handle varying traffic loads efficiently, ensuring optimal performance during peak use of resources. Virtualized infrastructure reduces cost of hardware and enables flexible deployment and updates, supporting rapid innovation and adaptation to new use cases.

The following are key Functions in the 5G core network:

1) AMF (Access and Mobility Management Function): AMF is a critical component of the 5G Core Network that manages user authentication, registration, mobility, and signaling. Its primary role is to ensure seamless connectivity for User Equipment (UE) as it moves across different network regions. The AMF authenticates users by identifying and verifying their credentials before granting access to the network. It establishes and maintains signaling connections between the UE and the core network, serving as the central communication hub for these interactions. Additionally, the AMF handles mobility management by tracking the UE's location and ensuring continuity of sessions during handovers between cells or regions. It also informs the UE about incoming data or connection requests when the device is in an idle state, ensuring efficient resource utilization and uninterrupted service. By performing these functions, the AMF plays a vital role in maintaining secure and reliable connectivity within the 5G network. AMF is prone to vulnerability as per ENISSA, so for our project we will evaluate fuzzer on AMF (nccgroup, n.d.).
2) SMF (Session Management Function): The SMF is a vital component of the 5G Core responsible for managing and orchestrating data sessions between User Equipment (UE) and external data networks. Its name highlights its primary role: the creation, modification, and deletion of Protocol Data Unit (PDU) sessions, which facilitate communication between the UE and the network. The SMF also determines the optimal routing path for data, ensuring efficient transmission between the UE and the data network. It plays a critical role in enforcing Quality of Service (QoS) rules, prioritizing traffic based on service requirements, such as giving precedence to voice calls or latency-sensitive applications over less critical traffic. By performing these functions, the SMF ensures seamless

connectivity, efficient resource allocation, and adherence to network policies, enabling an enhanced user experience in the 5G environment.

3) UPF (User Plane Function): The UPF is a crucial component of the 5G Core network responsible for managing user plane traffic. It ensures efficient routing and forwarding of data packets between User Equipment (UE) and external data networks, serving as the key interface for data transport. The UPF operates under policies defined by the Session Management Function (SMF), directing traffic in accordance with predefined rules and service requirements. It also performs advanced traffic management tasks, such as inspecting data flows to ensure compliance with network policies and optimizing performance. In scenarios where network congestion is a concern, the UPF can offload traffic to local networks or edge computing resources, reducing latency, and alleviating the load on the core network. This functionality is essential for maintaining high-speed, reliable data transmission and supporting diverse 5G use cases, including low-latency applications and high-bandwidth services.
4) PCF (Policy Control Function): The Policy Control Function (PCF) in the 5G Core is a key network function tasked with managing and enforcing policies throughout the 5G network. It ensures efficient allocation of network resources and compliance with Quality of Service (QoS) requirements. The PCF applies policies related to data usage, QoS, charging, and other service parameters based on rules defined by the network operator. It also handles user-specific policies tailored to subscription details, preferences, and device capabilities. Additionally, it collaborates with the charging function to enforce appropriate billing and usage rules based on service types or data plans. As part of the 5G Core's service-based architecture, the PCF seamlessly integrates with other functions, such as the Access and Mobility Management Function (AMF) and Session Management Function (SMF).

As the protocol specifications are the same for both open-source and commercial products, the network message formats should be representative of a real 5G core network.

2.1.2 5G RAN

The 5G Radio Access Network (RAN) includes both the physical hardware and the software protocols that enable wireless communication between User Equipment (UE) and the 5G core network. It plays a key role in ensuring network coverage and high-speed data transmission. The 5G RAN introduces technologies like MIMO (Multiple-Input Multiple-Output), which increases data throughput and improves signal quality, and network slicing, which allows the creation of virtual networks tailored to specific service needs. These advancements enhance the overall performance of the network in terms of speed, capacity, and latency, making the 5G RAN essential for modern, high-performance communication.

Components of RAN:

1) User Equipment (UE): It encompasses devices like smartphones, IoT devices, and laptops that connect to 5G networks. It establishes connections to the network via the gNB (5G base station) and communicates with the 5G Radio Access Network (RAN) over the air interface to access services such as data transfer, voice calls, and internet browsing. In the RAN, UE interacts with the gNB for control plane signaling, which manages connections and mobility, and for user plane data transmission, which handles data streaming and other user-specific activities. The UE plays a vital role in enabling seamless access to 5G services and applications.
2) gNodeB (gNB): It is the 5G base station that facilitates wireless communication between User Equipment (UE) and the 5G Core Network. It provides network coverage and processes radio signals, ensuring efficient connectivity for UEs. The gNB handles both control plane tasks, such as connection management and signaling, and user plane functions, like data transmission. Leveraging advanced technologies like MIMO for

increased capacity and beamforming for improved signal quality, the gNB ensures high-performance communication. As part of the Radio Access Network (RAN), it connects to the core network via the NG interface, playing a critical role in 5G operations.

Key Functions of 5G RAN:

1) Network Slicing: Network slicing allows the creation of multiple virtual networks over a shared physical infrastructure, with each slice optimized for a specific use case. For instance, one slice could be allocated for high-speed internet access, while another could be designed for low-latency applications such as autonomous driving. This enables operators to efficiently manage resources, offering tailored services for various needs. By utilizing technologies like virtualization and SDN, network slicing ensures flexible, high-performance communication for a wide range of applications, from everyday internet usage to mission-critical services.
2) Mobility Management: It is responsible for ensuring uninterrupted service as User Equipment (UE) moves between network areas. It handles tasks like handover, where a UE switches from one cell to another without losing connection, and location tracking, which enables the network to monitor the UE's position for efficient service routing. This process ensures continuous, high-quality service even as users move across the network, supporting a seamless experience for mobile users.

All in all, UE and gNB enable communication in 5G RAN, offering high-speed internet and enhanced connectivity across a wide range of use cases.

2.2 5Greplay Fuzzer

5Greplay is a tool specifically developed to capture, manipulate, and replay network packets, making it an essential utility for testing and analyzing traffic within a 5G network. It offers testers the ability to simulate various scenarios and evaluate the performance and resilience of the 5G core network under different conditions. This includes analyzing how the

system responds to specific packet flows, identifying potential vulnerabilities, and executing fuzzing scenarios to test its robustness against unexpected or malformed inputs. The tool operates by pre-defined rules and configuration files to control how captured network packets are processed. These configurations allow testers to specify the exact behavior they wish to replicate, such as modifying packet headers, altering payload data, or injecting anomalies. Once prepared, these packets can be replayed into the network to observe the system's response in a controlled and systematic manner.

By facilitating both positive testing ensuring the 5G core handles expected traffic patterns correctly and negative testing-evaluating its ability to test invalid or unexpected inputs-5Greplay provides a comprehensive testing environment. For example, testers can simulate scenarios like heavy traffic bursts, malformed protocol messages, or replay attacks to assess the network's ability to maintain functionality and security under stress. Through these capabilities, 5Greplay enables the in-depth evaluation of 5G core networks, helping identify flaws, improve system resilience, and ensure the network is prepared to handle real-world challenges effectively.

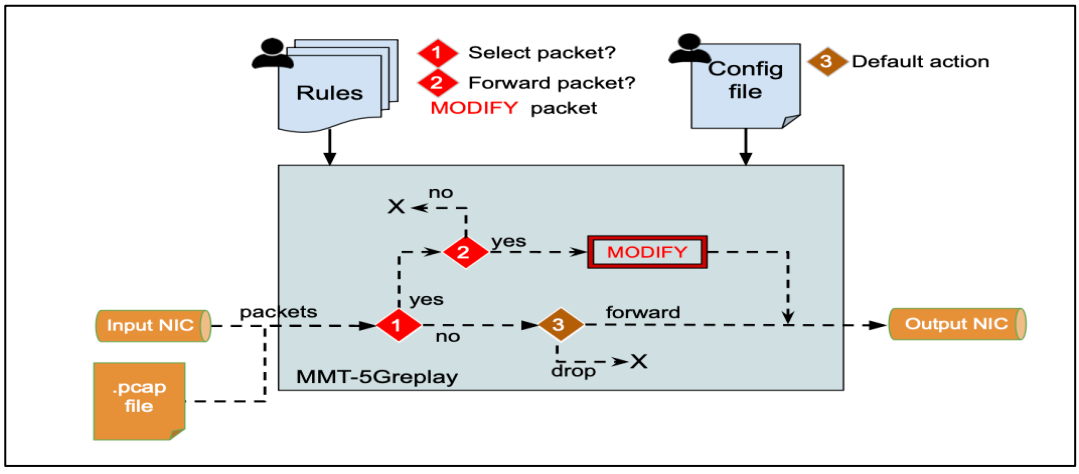

Figure 3 - 5Greplay architecture (Adapted from Salazar et al., 2021).

Working Behavior:

1) Input Source: the packets to be processed can either be captured live from a network interface card (NIC) or supplied via a .pcap file, which is pre-captured of network traffic using Wireshark or tcpdump.
2) Rules and Config File: It provides the logic for selecting packets and determining which one to forward or modify. The config file defines the default action to be taken if no specific rules apply to the packet, such as whether to forward or drop it.
3) Tool Flow: Firstly, it will examine each incoming packet and apply the rule to decode whether the packet should be selected for further processing. Moving forward, if a packet is selected the next step determines whether the packet should be forwarded or modified. The modification includes alteration to the packet based on the rules, such as changing its content or headers to simulate different traffic attacks.
4) Output: The modified or forwarded packets are sent to the output NIC for injection into the network. The packet modification rule depends on the user-defined and configuration, making it flexible for different testing scenarios (Salazar et al., 2021).

2.3 Wireshark

It is an open-source network protocol analyzer used for monitoring and troubleshooting network traffic in real-time. It allows users to capture and inspect data packets being transmitted over a network and making it an essential tool for capturing traffic from certain interfaces.

Wireshark captures packets from a live network or a saved file. It then provides a detailed breakdown of these packets, showing exactly what data is being sent and received, which helps in analyzing network issues, mainly regarding some security issues.

The following are use cases of Wireshark:

1) Capture Network Traffic: Wireshark can be set up to capture all packets of data flowing through a network interface such as Ethernet, Wi-Fi, etc. It can capture from both types of networks like live traffic in real-time or from saved files like .pcap files to analyze historical data and it captures all low-level activity in the network, including all the information like source, destination, protocol, and data sent or received.
2) Filter and Analyze Data: Wireshark provides a wide range of filters to narrow down specific types of network traffic. For instance, you can filter based on the IP address, particular protocol, or ports. In analyzing network packets, it shows detailed information about each packet which also includes timestamp, IP addresses, packet size, and data payloads this information makes it easy to detect any vulnerability or unexpected behavior in the network (*Wireshark · About*, n.d.-b)

Fuzzed data generation can be performed in following three ways:

They can be generated randomly by modifying correct data without requiring any knowledge of the application details this is known as Blackbox fuzzing and it was the first concept. Moreover, Whitebox fuzzing concepts in generating tests assume a complete knowledge of the application code and behavior of the system. The third type of fuzzing is gray box Fuzzing which stands in between of Blackbox and Whitebox fuzzing and aims to take advantage of both where it uses only a minimal knowledge of the target's behavior. It is thus the most appropriate method according to the purpose of the project. Detail explanation of workflow of network protocol fuzzing starting from 5G core deployment information going through multiple stages and checking of the system crash has been displayed in Figure 4.

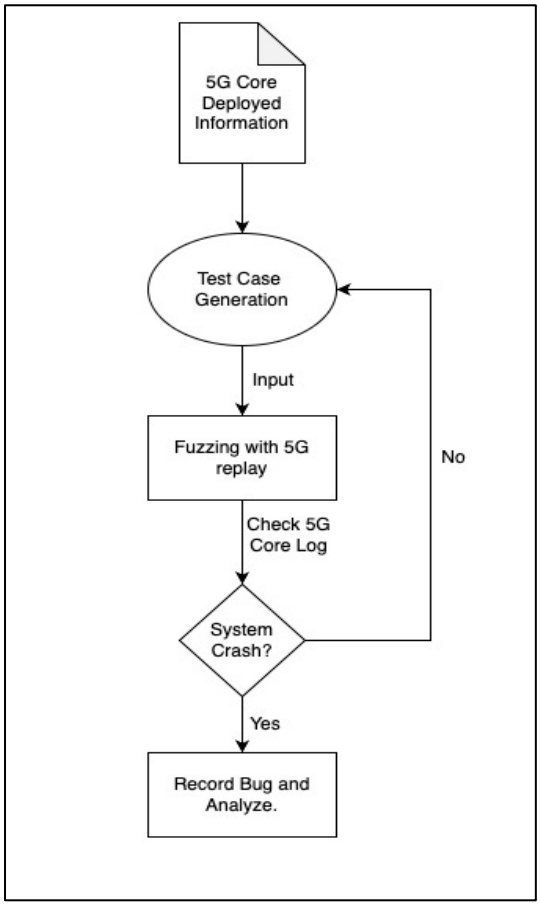

Figure 4 - Workflow of network protocol fuzzing

CHAPTER 3 : METHODOLOGY

3.1 Fuzzing Technique

Fuzzing is a software testing technique originally developed at the University of Wisconsin–Madison in 1989 by Professor Barton Miller and his team. It was designed as a method to uncover potential implementation vulnerabilities within software systems, which could be exploited by attackers. By injecting unexpected or random data into a target system, fuzzing exposes flaws such as bugs, crashes, or security weaknesses that traditional testing methods might overlook. This makes fuzzing a valuable tool for enhancing system robustness and security. For our purposes, we utilized the 5Greplay fuzzer, a tool designed to evaluate vulnerabilities in 5G network systems. This fuzzer operates by introducing random or malformed data into the system under test, targeting components to detect abnormal behavior or crashes. Such targeted injection helps identify flaws in the implementation of protocols or other system components, contributing to a more secure and stable environment (*Fuzzing Introduction: Definition, Types and Tools for Cybersecurity Pros | Infosec*, n.d.).

When applying fuzzing techniques to network protocols**,** specific operating system requirements must be considered. For example, fuzzing protocols like SCTP (Stream Control Transmission Protocol) require support from the underlying system. SCTP is integral to the operation of 5G core components and is used in protocols like NGAP (Next Generation Application Protocol) and NAS (Non-Access Stratum). Due to its unique requirements, SCTP-based fuzzing can only be effectively conducted in a Linux environment, where the protocol stack is fully supported (*NGAP/NAS Server*, n.d.). To begin fuzz testing, the target system must first be identified, followed by selecting an appropriate fuzzer and preparing a suitable input file for the system. A key aspect of fuzzing is monitoring system behavior, logging activity, and identifying crashes or warnings.

Following are the types of Fuzzing:

1) Protocol Fuzzing: Fuzzing is widely used in network protocol testing, where protocols define the rules for how different software components communicate over a network. The code responsible for interpreting protocol messages often serves as a potential attack surface. Fuzzing is highly effective for uncovering unknown vulnerabilities in protocol-handling code. Network software testing can be further categorized based on the roles and components involved. Many protocols operate using a client-server model, where the client initiates connections, and the server responds. In server testing, the fuzzer's role is straightforward: establish a connection with the target server and deliver test cases (*Black Duck*, n.d.).
2) Application Fuzzing: Any input, such as URLs, parameters, forms, cookies, and more, can be fuzzed using various character sets and payloads, all with the same objective: to crash the system and exploit weaknesses in its implementation (*Fuzzing Introduction: Definition, Types and Tools for Cybersecurity Pros | Infosec*, n.d.).
3) File Format Fuzzing: A file format fuzzer generates multiple malformed samples and opens them one by one. If the program crashes, the debug information is recorded for further analysis. Although less common, this type of fuzzer is still used occasionally today.

When it comes to the benefit of fuzzing, because of random nature of fuzz testing, experts say it's the methodology most likely to find bugs missed by other tests (GitLab, 2023). There are various benefits in ensuring the reliability, security, and robustness of the system. By generating unexpected or malformed inputs, fuzzing helps uncover hidden bugs and edge cases that traditional testing methods may overlook. It is particularly valuable for identifying security vulnerabilities, such as unhandled exceptions, which could be exploited by attackers. This technique is adaptable to various systems, including operating systems, web applications, and network protocols, making it a versatile technique for stress testing under real-world

conditions. Incorporating fuzzing into testing not only helps organizations meet industry standards but also helps in continuous improvement, resulting in more secure and resilient system.

3.2 Positive and Negative Testing

Traditionally, the primary focus of software testing has been on validating the functionality of a system—ensuring that it performs as intended under ideal conditions. This process, commonly known as functional testing, involves designing test cases and frameworks that provide valid inputs to the system and verifying that the outputs match expected results. Functional testing is a form of positive testing that serves as a cornerstone of quality assurance, ensuring the software meets its design specifications and behaves predictably during normal operation.

In conventional software development methodologies, the design phase generates a comprehensive list of requirements outlining the system's desired functionality. These requirements act as a blueprint for testing. The test development team's role is to interpret these requirements and translate them into structured test cases to evaluate whether the software conforms to the specified behavior. For instance, if the design requires that a calculator program adds two numbers, functional testing ensures the program produces accurate sums when valid inputs are provided. While functional testing is essential for confirming correct behavior with valid inputs, it is inherently limited. It focuses solely on expected scenarios, leaving the system vulnerable to failure under unpredictable or adverse conditions. Real-world software rarely operates in such controlled environments. It often encounters invalid inputs, unexpected user behavior, or interactions with other software that could lead to errors, crashes, or security breaches.

The real world is unpredictable and filled with unexpected conditions and malformed inputs. Software systems must handle interactions with users or other programs that may

provide invalid data, perform actions out of sequence, or misuse the system in ways not anticipated during development. To address these challenges, negative testing is introduced. Negative testing involves intentionally supplying incorrect, malformed, or unexpected inputs to the software to assess its ability to handle these situations gracefully. By doing so, it identifies vulnerabilities, failures, or behaviors that might compromise reliability and security.

Negative testing complements functional testing by simulating adverse conditions, such as excessively long inputs, unexpected file formats, or malformed network packets. For example, while functional testing ensures that a login system works correctly with valid usernames and passwords, negative testing evaluates how the system responds to invalid data, such as SQL injection attempts or excessively large input strings. The goal is to determine whether the software can resist such scenarios without crashing, exposing sensitive data, or becoming unresponsive.

It's important to note that various negative testing tools may yield different results for the same system, as each tool employs unique methodologies to test different types of malformed inputs. This diversity in tools allows for various exploration of potential vulnerabilities and ensures that the system can handle a wide range of unexpected interactions. To achieve robust software, modern testing practices integrate both positive and negative testing strategies. By combining these approaches, developers can assess not only whether the software meets its functional requirements but also whether it remains secure, reliable, and resilient under unpredictable conditions. This comprehensive testing methodology ensures that software is equipped to handle the complexities of real-world environments. (*Black Duck*, n.d.).

3.3 Condition-Based Classification for Testing

Network protocol fuzzing is a critical method for testing and identifying vulnerabilities in systems that manage network communication. It can be classified into three main categories- black-box**,** white-box, and gray-box**,** depending on the level of understanding and access to the protocol's implementation. This implementation refers to the software or hardware components that handle the management of network protocols and the processing of protocol messages (Zhang et al., 2023). Here's a detailed explanation of each type:

1) Black-box Fuzzing: The approach described, commonly known as random testing or black-box fuzzing**,** is a fundamental technique in software and system testing. It operates without any insight into the internal workings of the System Under Test (SUT)**,** treating the system as a "black box." This means the tester, or the fuzzing tool has no access to, or understanding of the source code, architecture, or underlying processes. Instead, it focuses entirely on the external behavior of the system by providing various inputs and analyzing the corresponding outputs to deduce its functionality and identify vulnerabilities. (Salazar et al., 2021).
2) White-box Fuzzing: This approach, often referred to as white-box fuzzing, involves a deep understanding of the internal structure and logic of the target program. Unlike black-box fuzzing, which treats the system as a "black box" and generates random test cases without any insight into its inner workings, white-box fuzzing relies on detailed knowledge of the system's internal design. The fuzzing tool actively collects and analyzes information about the system's code, runtime behavior, and processing mechanisms to craft highly targeted test cases. In the realm of network protocol fuzzing, this method leverages specific details about the protocol's implementation, such as its source code, message-handling routines, and runtime operations. By utilizing this information, white-box fuzzing can systematically explore various execution paths and identify potential vulnerabilities that might remain

hidden in a less informed approach. This method was initially proposed by Godefroid et al. (2008) to address the inherent limitations of black-box fuzzing, which relies solely on blind and random testing and often fails to uncover deep issues within a system. White-box fuzzing aims to provide broader and more precise coverage by systematically analyzing the code and generating inputs that target specific paths or conditions within the system. (Zhang et al., 2023).

3) Gray-box Fuzzing: Positioned as a middle ground between black-box and white-box fuzzing, gray-box fuzzing strikes a balance by leveraging limited feedback from the System Under Test (SUT) to guide test case generation. Unlike black-box fuzzing, which operates blindly without any insight into the internal workings of the system, gray-box fuzzing uses runtime feedback - such as code coverage metrics, system conditions, and state changes - to intelligently adapt its testing strategy. This feedback-driven approach enables the fuzzing tool to target a wider range of execution paths, enhancing its ability to uncover hidden vulnerabilities and errors. Despite not having full access to the system's source code or internal logic, as required by white-box fuzzing, gray-box fuzzing can still achieve significant coverage and precision. By analyzing dynamic behaviors during execution, the method identifies which areas of the system remain unexplored and directs test cases accordingly. This process often involves techniques like instrumentation or lightweight monitoring to collect feedback during runtime (*Finding Software Vulnerabilities by Smart Fuzzing*, 2011) (Zhang et al., 2023).

3.4 Classification Based on Test Case Generation Methods

In network protocol fuzzing, test cases typically refer to properly formatted protocol data packets with extensive body content. Various fuzzing techniques employ different methods to generate these test cases, which are then transmitted to the protocol implementation under test through mechanisms like sockets to uncover vulnerabilities. The method used for

test case generation plays a crucial role in the classification of network protocol fuzzing techniques. Broadly, these methods fall into two categories: mutation-based and generation-based approaches, detailed below (Zhang et al., 2023).

1) Mutation-Based Approach – This technique begins with valid data that is properly formatted and structured. It then alters this data using various methods to produce semi-valid test cases. Four key methods used in this approach are bit flipping, arithmetic mutation, block-based mutation, and dictionary-based mutation: Bit Flipping – Specific bits within the data packet are toggled, converting 0s to 1s and 1s to 0s.

Arithmetic Mutation – A byte sequence is interpreted as an integer, subjected to arithmetic operations, and the resulting value is inserted back into the sequence. Block-Based Mutation – A defined byte sequence is treated as a block, the fundamental unit of a data packet. Operations such as adding, removing, replacing, or rearranging blocks are performed. Dictionary-Based Mutation – This method targets specific semantic fields within the data. It replaces weighted fields or other elements with predefined values, such as numbers or strings.

2) Generation-Based Approach – This method involves creating semi-valid data using predefined specifications or templates available at the outset. These templates can either be created by testing personnel or come preloaded within the fuzzer.

3.5 Protocols Prone to Vulnerabilities

NGAP

The Next Generation Application Protocol supports services for both UE-associated and non-UE-associated scenarios. It encompasses operations such as configuration, UE context transfer, PDU session resource management, and mobility procedure support. In our testing, this protocol is utilized on the N2 interface between the AMF and gNB. (nccgroup, n.d.).

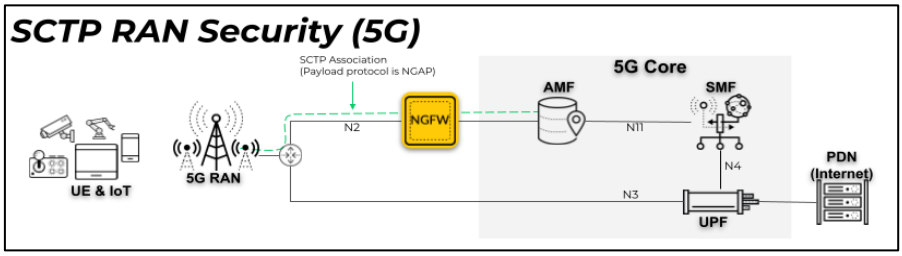

Figure 5 - SCTP is used by NGAP at N2 interface (Adapted from SCTP Use Cases, n.d.).

The green dashed line in Figure 5 represents an SCTP association over the N2 interface between the 5G RAN and the AMF. This SCTP connection is used to transport the NG Application Protocol (NGAP) (*SCTP Use Cases*, n.d.).

The NGAP architecture is structured into the following layers:

1. Application Layer: This layer contains NGAP protocol entities responsible for generating and processing NGAP messages.
2. Transport Layer: It facilitates the reliable delivery of NGAP messages between the gNB and AMF, typically using the Stream Control Transmission Protocol (SCTP).
3. Security Layer: This layer ensures the security of NGAP messages by providing authentication and integrity protection, utilizing the Transport Layer Security (TLS) protocol (TechLTE World, 2024).

NGAP facilitates seamless onboarding, efficient handovers between stations, and effective resource management, all while maintaining robust security. It is essential for enabling the high-speed connectivity, low latency, and diverse services that define 5G networks. The following are the important Features of NGAP: NAS (Non-access stratum) Signaling, Security Mechanism, User Equipment Management, Radio resource Management, and Mobility Management.

PFCP

The Packet Forwarding Control Protocol (PFCP) is used to manage data sessions within the User Plane Function (UPF) in the 5G core network. It enables the creation, modification, and termination of these sessions, facilitating data flow between User Equipment (UE) and the external network. PFCP messages include various rules, such as Packet Deletion Rules (PDR) for discarding data, QoS Enforcement Rules (QER) for prioritizing traffic, and Buffering Enforcement Rules (BAR) for managing data buffering. These rules are sent from the control plane to the user plane to manage data traffic. In the network, PFCP operates on the N4 interface between the UPF and Session Management Function (SMF), enabling dynamic management and efficient traffic handling. (nccgroup, n.d.).

GTPU

GPRS Tunneling Protocol User Data (GTP-U) is an IP-based tunneling protocol designed to transport user plane data between network nodes, enabling secure and efficient data flow. It allows multiple tunnels between endpoint pairs and encapsulates user data for transmission across various network elements. In 5G, GTP-U is used to tunnel data between the gNB and the UPF, ensuring that user data, such as voice and video, can be reliably sent across different network parts. This protocol is implemented on the N3 interface, which is vital in supporting the high-speed data transfer and low latency demands of 5G networks. (nccgroup, n.d.).

DIAMETER

DIAMETER is an advanced application-layer protocol used for Authentication, Authorization, and Accounting (AAA) in telecommunication networks. It enhances scalability and security compared to its predecessor, RADIUS, and can be extended with additional commands and attributes to meet evolving network needs. On the S6a interface, DIAMETER facilitates communication between the Mobility Management Entity (MME) and the Home

Subscriber Server (HSS). It is essential for exchanging subscriber profiles, location data, and authentication information, enabling the MME to manage subscriber services effectively. This protocol ensures secure and efficient subscriber management, making it integral to the operation of 4G and 5G networks.

SCTP

Stream Control Transmission Protocol (SCTP) is a robust transport layer protocol designed for reliable, message-oriented communication, making it ideal for signaling traffic in complex networks like 5G. It supports multi-homing, ensuring redundancy by using multiple network paths for a single connection, and multi-streaming, which prevents message loss in one stream from affecting others. In 5G, SCTP is crucial for transporting signaling messages between the 5G Core Network (5GC) and the Radio Access Network (RAN). Its ability to handle redundancy, load balancing, and efficient message delivery makes it a cornerstone for managing the dynamic and high-speed data flows of 5G systems. (Juniper Networks, n.d.).

NAS

The Non-Access Stratum (NAS) Protocol in 5G facilitates control-plane communication between User Equipment (UE) and the Core Network (CN). It manages key operations such as registration, enabling devices to connect to the network, and mobility management, allowing users to move between cells without losing connectivity. NAS also oversees the establishment, maintenance, and termination of data sessions and implements security protocols like authentication, encryption, and integrity protection. These functions ensure seamless connectivity and secure communication, making NAS a vital component of 5G network operations.

Following are the Protocol Fuzz supported by 5G replay:

- NGAP (Next Generation Application Protocol)
- SCTP (Stream Control Transmission Protocol)

- GTPv2 (GPRS Tunneling Protocol Version 2)
- NAS (Non-access Stratum) Protocol
- DIAMETER

3.6 Types of Failure in Fuzzing

Software or System fails when it acts in ways unintended or unanticipated by its creators. In traditional fuzzing, failure modes generally fall into four categories:

- Crashes
- Endless loops
- Resource leaks or shortages
- Unexpected behavior

These failure modes can vary depending on factors such as the type of software or system under test, the operating environment, and more. A crash might simply disrupt the software or escalate to a denial-of-service attack, performance degradation, information leakage, security breaches, or other adverse outcomes. The impact largely depends on the software's purpose, function, and operating context.

CHAPTER 4 : EXPERIMENTAL SETUP AND RESULTS

4.1 ENVIRONMENT SETUP

5G replay Installation (WSL or Ubuntu)(Eurl n.d.)

Following Commands were used to install 5Greplay.

```
sudo apt update && sudo apt install -y wget

    # Download 5Greplay version 0.0.1

Wget https://github.com/Montimage/5GReplay/releases/download/v0.0./5greplay-
0.0.1_Linux_x86_64.tar.gz

    # Decompress 5Greplay

Tar -xzf 5greplay-0.0.1_Linux_x86_64.tar.gz

    # View 5Greplay Parameter

Cd 5greplay-0.0.1

./5greplay replay -h
```

Open5GS Installation

Following commands were used to install open5GS simulator in ubuntu:

```
Sudo apt update && sudo apt install software-properties-common

Sudo add-apt-repository ppa: open5gs/latest

Sudo apt update

Sudo apt install open5gs
```

Following is 5G Network startup from the Lab Manual at Polysec Lab:

5G Core startup – Supermicro Systems

Step 1: Connect a laptop or PC to the provided Netgear access point via WiFi or Ethernet to gain SSH console access to the 5G servers. The WiFi SSID and password can be found on the device, while Ethernet connections require no additional setup.

Step 2: Ensure that a local IP address is assigned (e.g., 192.168.1.x), or configure a static IP within the 192.168.1.x network. Avoid using the following IP addresses, as they are already assigned to the 5G system: 192.168.1.3, 192.168.1.4, 192.168.1.5, 192.168.1.100, and 192.168.1.182.

Step 3: Launch a web browser and navigate to https://192.168.1.3 and https://192.168.1.4. These URLs provide access to the remote management systems for the CN and gNB servers, enabling server control without physical access. This allows you to power cycle the servers and view the server console without requiring a local monitor connection.

Step 4: Log in to the IPMI management console for each server using the credentials specific to your environment (refer to the passwords section). Keep in mind that the IPMI interface appearance may vary depending on your environment.

For the Supermicro based system:

Username: ADMIN

Password: specific to each environment, see passwords section

Step 5: In the IPMI console, navigate to the Power Control section and select the option to power on each server. The web interface will prompt you for confirmation and then update to indicate the powered-on state.

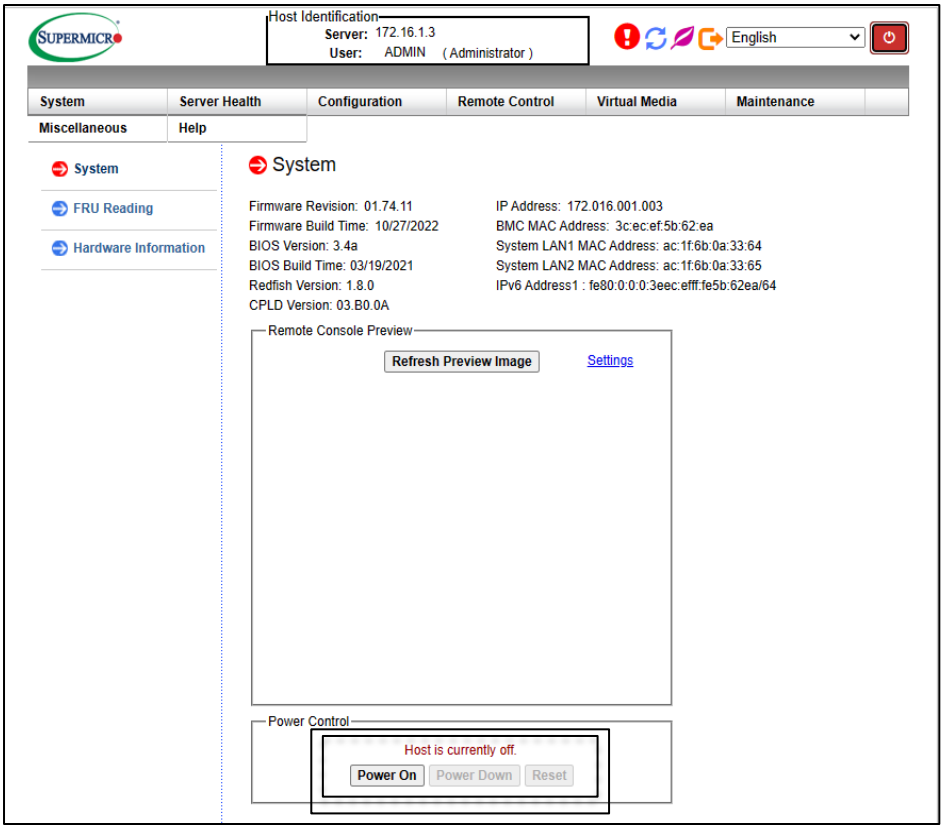

Figure 6 - Console to control 5G Network (Adapted from Startup Manual).

Step 6: The servers will take a few minutes to boot and initialize the 5G software. While the boot process can be monitored using the remote console preview option in the IPMI web interface, this step is optional as the system starts automatically.

Step 7: After the servers have booted, launch an SSH terminal program (e.g., MobaXterm) to establish console connections to each server. Open multiple SSH sessions as follows: one session for the CN and four sessions for the gNB, using the 'vnc' account. This will automatically log you in as the vnc user and start a Bash command shell.

Step 8: In each active session, switch from the vnc user to the root user by using the rootme command or sudo su–. This grants the administrator privileges required to run the 5G programs.

Step 9: In the CN server session tab, restart the core network Docker services to clear any residual state from a previous shutdown. This involves stopping all Docker services and then restarting them. Use the following commands to stop the core services, and wait for the process to complete.

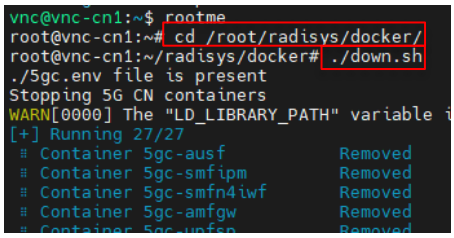

Figure 7 - First Shutdown previous sessions.

After all services have stopped, as indicated by the 'Removed' labels, enter the following commands to restart them. Monitor the status as it transitions from 'stopped' to 'healthy' and 'started'.

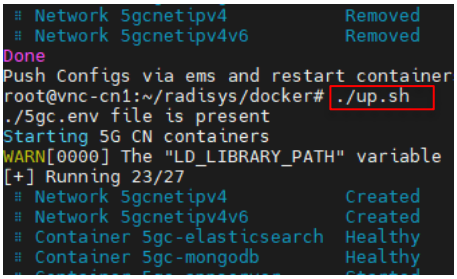

Figure 8 - Run ./up.sh after shutdown complete.

Step 10: In the CN server session, verify that the 5G services are running and healthy. Enter show_status.sh to display the container status information. This will initiate a display that updates every second with the latest status details. Monitor the container status information accordingly.

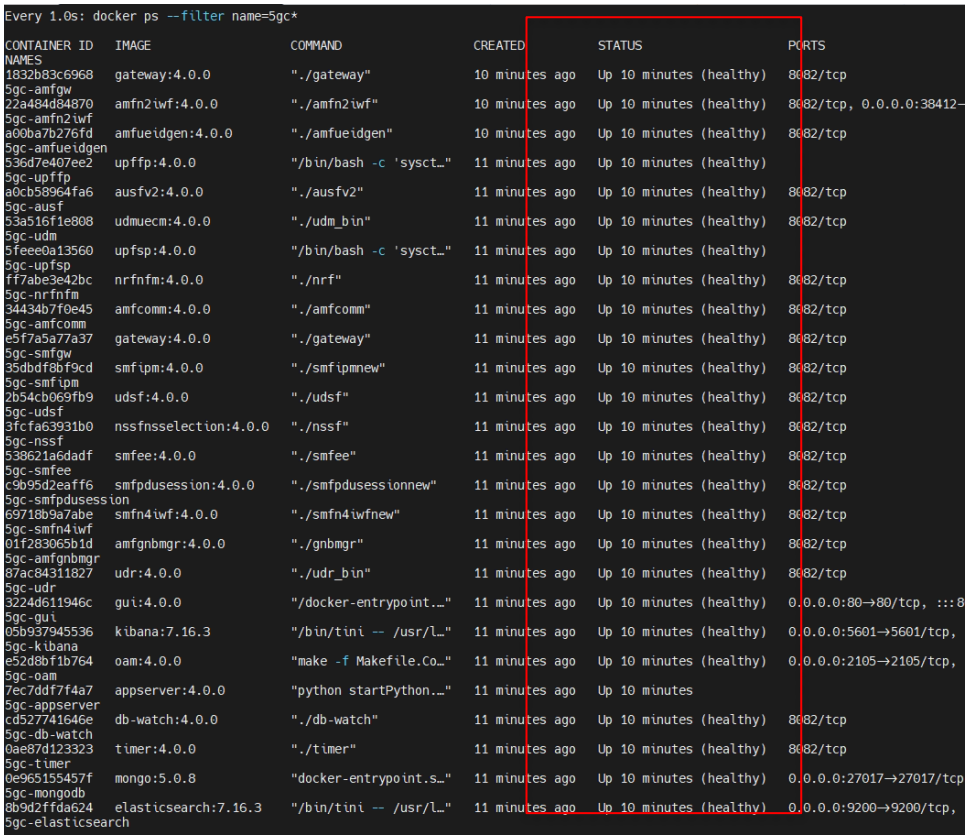

Figure 9 - 5G Core Up and Running Healthy

Step 11: Ensure there are no containers marked as (unhealthy) in the STATUS column. It is normal for the containers to show varying health statuses during startup, but they will stabilize and show as healthy once the system has fully started.

Step 12: Once all the containers display as healthy, the CN is fully operational and ready to connect to the gNB server.

Step 13: Open four separate SSH sessions to the gNB server as outlined earlier. Each session will be used to start a different process, running in the foreground.

Step 14: In each SSH terminal, switch from the vnc user to the root user by using the rootme or sudo su – commands.

Step 15: First, confirm that the PTP time synchronization process has been completed successfully by running check_ptp.sh and checking for the "PTP is VALID" output. If it shows "INVALID," ensure that the Master Offset value is approaching '0'. If there is no change after

a minute, try restarting the PTP service using the restart_ptp.sh command. If PTP still does not converge to VALID, there is an issue with GPS reception and timing, which must be resolved before proceeding.

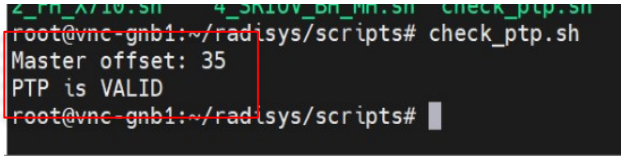

Figure 10 - Checking if PTP is valid in gNB server.

Step 16: Once the time is valid, you can proceed with starting each of the four gNB services (RU, CU, L1, and DU) by following the steps below:

Start RU

1. In the first SSH terminal, start the radio unit (RU) by running ru_console.sh and wait for the terminal to open. Once it opens, press Enter and verify that the radio terminal is running. If it is not, check the power LED on the top of the unit to ensure the radio unit is powered on.
2. Power on the radio unit. The serial connection output will appear as following.

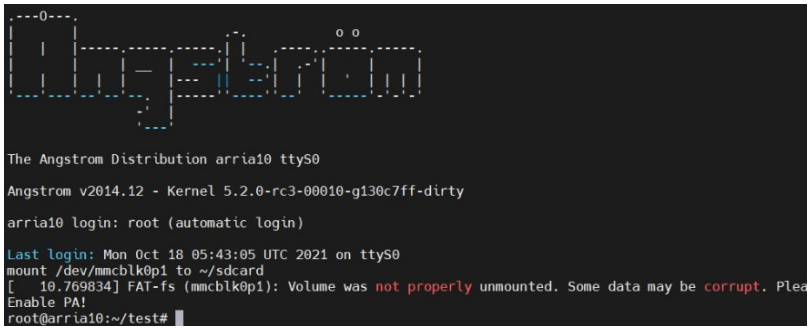

Figure 11 - RU Booting UP Console.

3. If the radio unit is already powered on, press Enter on the keyboard to access the command shell prompt. You should then see a terminal where you can enter commands to run the RU.
4. Once the shell prompt appears, start the radio by typing ./run_ru.sh and pressing Enter.

5. The RU will begin initializing and, after a few minutes, will display a continuous stream of status values similar to the example below. Verify that the command reaches “ad9025 init…” as shown. At this point, it will pause for several minutes to calibrate the radio.
6. After several minutes, the RU initialization will be complete, and the status information will be displayed. Ensure that the output changes to state=1 before proceeding.

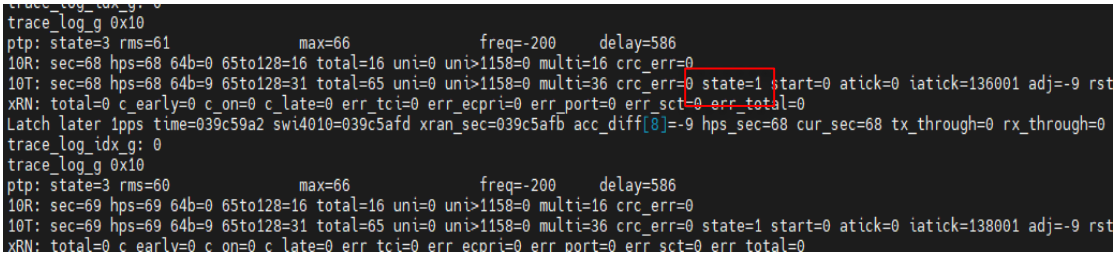

Figure 12 - Validating State = 1 in RU

7. This radio unit console window is for viewing status information only. It will be used later to confirm that the DU has connected to the radio once all other services have been started.

Start CU

1. In the second SSH terminal, start the CU service by typing run_cu.sh in the shell and pressing Enter. The following text should appear after a few seconds.

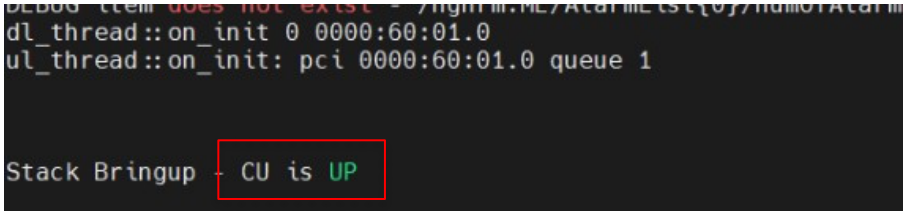

Figure 13 - CU is UP and running.

2. Keep the console window open. It will update to display the status once all services are started and running. You can use it to monitor the attached UE devices on the network.

Start L1

1. In a third SSH terminal start the L1 service. Type ‘run_l1.sh’ in the shell and hit enter.
2. The following text should appear after few seconds including ‘welcome to the application console’.

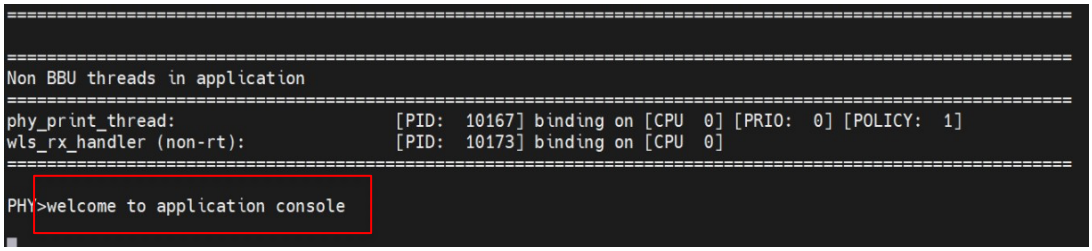

Figure 14 - L1 Welcome message when booted up.

3. Leave the console window open. This will update to a status display once all the services are started and running. It can be used to monitor the layer 1 packets being sent to the radio unit.

Start DU

1. In a fourth SSH terminal start the DU service. Type ‘run_du.sh’ in the shell and hit enter.
2. The following text should appear after several seconds. It is important that the ‘PHY Start Success’ is present indicating that the gNB has registered properly with the CN.

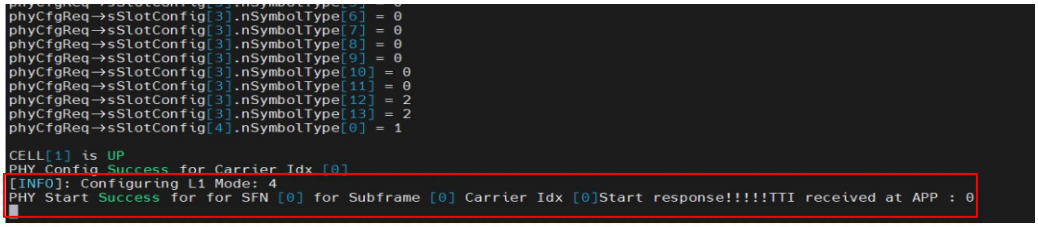

Figure 15 - Success message from DU setup console.

3. Keep the console window open. It will update with a status display once all the services are started and running and can be used to monitor the attached UE devices on the network.

Validating 5G core Startup

Once all the 5G services have been started using the previous startup procedure, the 5G system is initialized. After starting the RU, CU, L1, and DU programs, the 5G network is live and ready to connect UEs. The following steps can be followed to verify that the startup was successful:

1. Check the radio unit's operation by reviewing the console information in the RU shell. Ensure that the state=2, which indicates the RU is communicating with the gNB.

Figure 16 - Validating RU is in State = 2

2. The open SSH consoles on the gNB for the CU/DU/L1 services will continuously update with network status information. Below are examples of the output from each.

CU Status Information

The CU console provides information about the established PDU session connections between a UE and the core network. The display below shows one device attached (UE_ID=65543) with two active sessions.

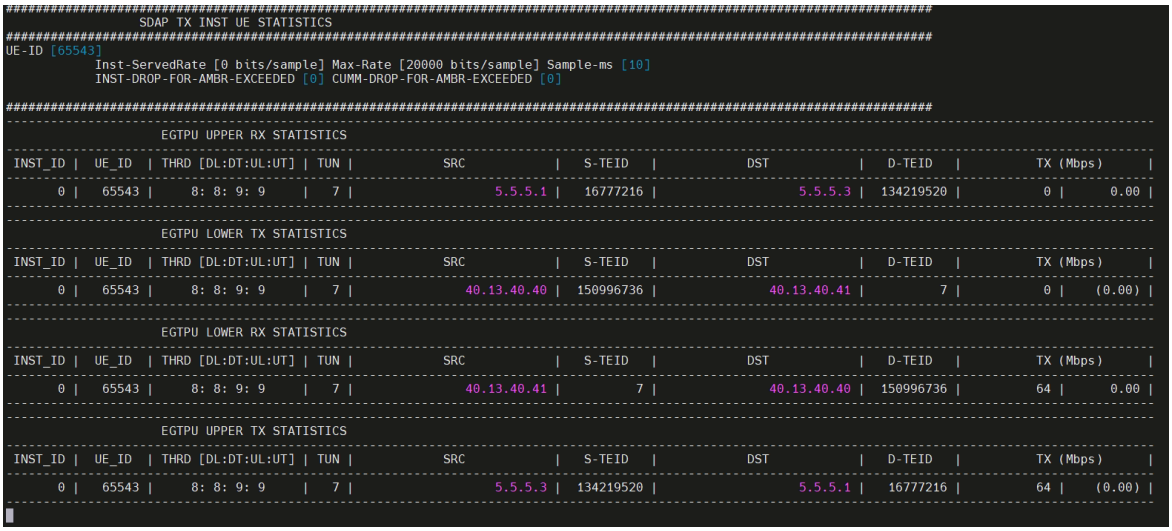

Figure 17 - Validating CU is running Healthy.

L1 Status Information

The L1 console provides information about the types of radio packets being transmitted as well as the signal quality to the attached devices. In the status below there is one cell established with near zero block error rate (BLER). Each of the 4 antennas are active and sending PUSCH packets.

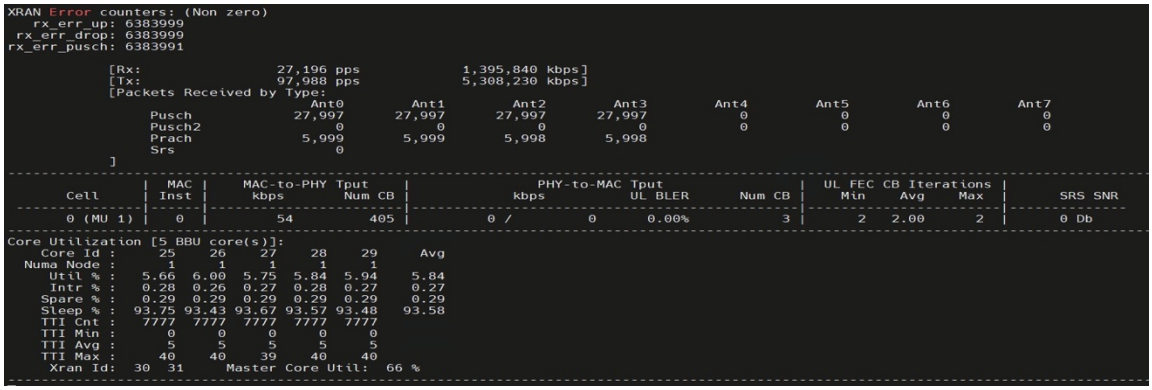

Figure 18 - Validating L1 console.

DU Status Information

The DU console displays information about the individual UEs attached to the gNB, as well as the transmission performance of each UE for both uplink (UL) and downlink (DL) directions. In the status report below, one UE is connected, and its UL and DL packets are being reported.

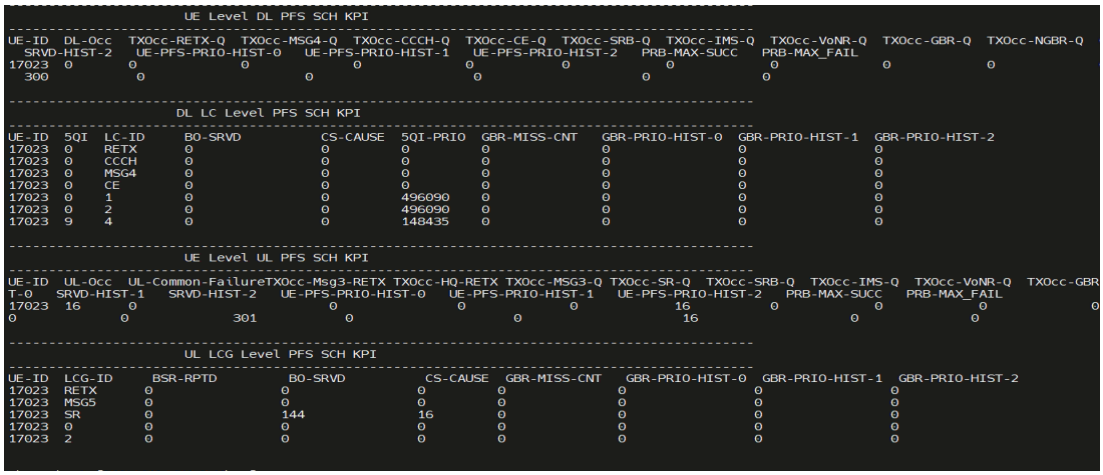

Figure 19 - Validating DU Startup.

Rule to Forward Packet

```
 GNU nano 7.2                                                forward-localhost.xml
beginning>

<property property_id="103" type_property="FORWARD"
    description="Inject only SCTP packets from UE -> Core but not inversed direction">
    <event description="From UE and NAS-5G packets"
           boolean_expression="( (sctp.dest_port == 38412 ) &amp;amp;&amp;amp;(sctp.ch_type == 0))"/>
</property>

<property property_id="104" type_property="FORWARD"
    description="Inject only UDP packets from UE -> Core but not inversed direction">
    <event description="From UE and GTP packets"
           boolean_expression="( (udp.dest_port == 2152 ) )"/>
</property>

</beginning>
```

Figure 20 - Rule ‘103’ forward-localhost.xml file to Fuzz 5Gcore

Configuration (mmt-5greplay.conf)

```
dpatel@DESKTOP-1P7R1BR:~/5greplay-0.0.1$ cat mmt-5greplay.conf
# option only when using DPDK to capture packets or to forwarding packets
dpdk-option = "--syslog=syslog --log-level=5 -c 0x55555555555"

#root stack of network protocol
# for Ethernet: 1
# for ieee802154: 800
# for Linux cooked capture: 624
stack-type = 1

input {
        # in case for PCAP the input mode can be ONLINE or OFFLINE, however for DPDK it's only ONLINE
        mode = ONLINE

        # input source for PCAP online mode (interface name) and for offline mode (pcap name),
        #   however for DPDK its interface port number
        #   in DPDK mode, MMT supports also multi-port inputs,
        #     - e.g., source="0,1" will tell MMT to capture packets on port 0 and 1
        #     - MMT will aggregate traffic on these 2 ports, thus 2 packets of one flow can be received on 2 different ports
        source = "lo"

        # maximal size of a packet
        snap-len = 65535 #
}

output {
        enable = true
        output-dir = "./" # Location where files are written:
        sample-interval = 5 #a new sample file is created each x seconds given by output.cache-period
        report-description = true # true to include rule's description into the alert reports,
             # otherwise it will be excluded (thus rules's descriptions will be an empty string in the reports)
             # Excluding rules's descriptions will reduce the size of reports.
}

engine {
        thread-nb     = 0   # the number of security threads per one probe thread , e .g . , if we have 16 probe threads and thread-nb = x ,
        # then x*16 security threads will be used .
        # If set to zero this means that the security analysis will be done by the threads of the probe .

        exclude-rules = ""  # Range of rules to be excluded from the verification
        rules-mask    = ""  # Mapping of rules to the security threads:
                # Format: rules-mask = (thread-index:rule-range);
                # thread-index = a number greater than 0
                # rule-range = number greater than 0, or a range of numbers greater than 0.
                # Example: If we have thread-nb = 3 and "(1:1,2,4-6)(2:3)" ,
                # this means that:
                # thread 1 verifies rules 1 ,2 ,4 ,5 ,6;
                # thread 2 verifies only rule 3; and
                # thread 3 verifies the rest
                # Note: if we have thread-nb = 2 and "(1:1)(2:3)", then only rules 1 and 3 are verified (the others are not)

        ip-encapsulation-index = LAST   # If traffic is ip-in-ip, this option selects which IP will be analysed.
                # - FIRST: first ip in the protocol hierarchy
                # - LAST:  last ip in the protocol hierarchy
                # - i: i-th ip in ther protocol hierarchy.
                # For example, given ETH.IP.UDP.GTP.IP.TCP.VPN.IP.SSL,
                #   - FIRST, or 1, indicates IP after ETH
                #   - LAST, or any number >= 3, indicates IP after VPN
                #   - 2 indicates IP after GTP
                # NOTE: this option will be ignored in non ip-in-ip traffic

        # number of fsm instances of one rule
        max-instances = 100000
}

# A mem_pool contains several pools. Each pool stores several blocks of memory
```

Figure 21 - Configuration_1 File used in 5Greplay (mmt-5greplay.conf)

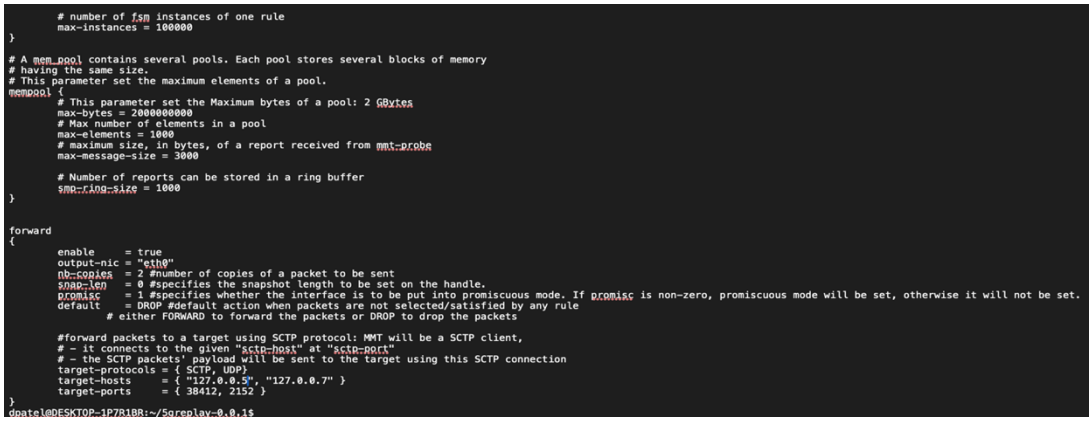

Figure 22 - Configuration_2 File used in 5Greplay (mmt-5greplay.conf)

Moving forward, we can proceed with the experiments only if an SCTP server is active at the specified IP address and port. Otherwise, the errors illustrated in Figures 23 and 24 will occur. During our experiments, we also encountered these errors. In our 5G core (SuperMicro system), the SCTP server is accessible at IP address 0.0.0.0 on port 38412. In the WSL environment, it is available at IP address 127.0.0.5 on port 38412.

```
dpatel@DESKTOP-1P7R1BR:~/5greplay-0.0.1$ sudo ./5greplay replay -c 5greplay-sctp.conf -t ue_authetication.pcapng > log.t
xt 2>&1
dpatel@DESKTOP-1P7R1BR:~/5greplay-0.0.1$ cat log.txt
mmt-5greplay: 5Greplay v0.0.1-d9f4cef using DPI v1.7.0.0 (a8ad3c2) is running on pid 1626
mmt-5greplay: Ignore duplicated rule id 103 (Inject only packet from UE -> Core but not inversed direction)
mmt-5greplay: [_sctp_connect:49] Cannot connect to 0.0.0.0:38412 using SCTP
mmt-5greplay: Interrupted by signal 6
dpatel@DESKTOP-1P7R1BR:~/5greplay-0.0.1$ nano 5greplay-sctp.conf
dpatel@DESKTOP-1P7R1BR:~/5greplay-0.0.1$ sudo ./5greplay replay -c 5greplay-udp.conf -t ue_authetication.pcapng > log.txt 2>&1
dpatel@DESKTOP-1P7R1BR:~/5greplay-0.0.1$ cat log.txt
mmt-5greplay: 5Greplay v0.0.1-d9f4cef using DPI v1.7.0.0 (a8ad3c2) is running on pid 1630
mmt-5greplay: Ignore duplicated rule id 103 (Inject only packet from UE -> Core but not inversed direction)
mmt-5greplay: MMT-5Greplay 0.0.1 (d9f4cef - Sep  9 2021 10:20:49) is verifying 1 rules having 2 proto.atts using the main thread
mmt-5greplay: Analyzing pcap file ue_authetication.pcapng
          13 packets received
          13 messages received
           1 alerts generated
```

Figure 23 - SCTP improper Configuration error-5Greplay

```
5greplay: [_sctp_connect:49] Cannot connect to 192.168.49.3:38412 using SCTP
5greplay: Interrupted by signal 6
```

Figure 24 - SCTP error (Adapted from Eurl (n.d.))

Following experiments commands were implemented with the help of Eurl (n.d.-a)

4.2 EXPERIMENT 1 - Open5GS simulator core (AMF) crash in Positive Testing

In the initial experiment on the Open5GS simulator, a pre-built packet capture file (5g-sa.pcap) was used to verify the results previously submitted by the 5Greplay team. Using the same configuration, rules, platform (Docker), and packet capture file, the experiment led to a crash of the AMF in the 5G core of the simulator. This experiment was performed to check the feasibility of the 5Greplay fuzzer.

Command - sudo ./5greplay replay -t 5g-sa.pcap -Xforward.nb-copies=2000 -Xforward.default = FORWARD > log.txt 2>&1

```
# sudo ./5greplay replay -t 5g-sa.pcap -Xforward.nb-copies=2000 -Xforward.default=FORWARD > log.txt 2>&1
# sudo tail -f /var/log/open5gs/amf.log
01/12 01:21:39.138: [app] ERROR: Signal-NUM[20] received (Stopped) (../src/main.c:82)
01/12 01:21:46.651: [sbi] WARNING: [b25c0f2c-b0e7-41ee-aaf5-af6a483bea55] Retry registration with NRF (../lib/sbi/nf-sm.c:183)
01/12 01:21:46.652: [sbi] WARNING: [7] Failed to connect to 127.0.0.200 port 7777: Connection refused (../lib/sbi/client.c:698)
01/12 01:21:46.652: [sbi] WARNING: ogs_sbi_client_handler() failed [-1] (../lib/sbi/path.c:64)
01/12 01:21:48.347: [app] ERROR: Signal-NUM[20] received (Stopped) (../src/main.c:82)
01/12 01:21:49.431: [app] ERROR: Signal-NUM[20] received (Stopped) (../src/main.c:82)
01/12 01:21:50.566: [app] ERROR: Signal-NUM[20] received (Stopped) (../src/main.c:82)
01/12 01:21:57.663: [sbi] WARNING: [b25c0f2c-b0e7-41ee-aaf5-af6a483bea55] Retry registration with NRF (../lib/sbi/nf-sm.c:183)
01/12 01:21:57.663: [sbi] WARNING: [7] Failed to connect to 127.0.0.200 port 7777: Connection refused (../lib/sbi/client.c:698)
01/12 01:21:57.663: [sbi] WARNING: ogs_sbi_client_handler() failed [-1] (../lib/sbi/path.c:64)
```

Figure 25 - Open5Gs Simulator Crash experiment log

4.3 EXPERIMENT 2 - 5G core crash with same parameters in Docker

In this experiment, we will be testing with the minor changes to configuration i.e. changing IP of sctp in forward section of configuration and output to “en10” which is ethernet connected to the 5G core and header stack type to 1 for ethernet connection.

```
root@7567a792bcbb:/workdir/5greplay-0.0.1# sudo ./5greplay replay -t 5g-sa.pcap -Xforward.nb-copies=2000 -Xforward.default=FORWARD > log.txt 2>&1
mmt-5greplay: 5Greplay v0.0.1-d9f4cef using DPI v1.7.0.0 (a8ad3c2) is running on pid 13502
mmt-5greplay: Overridden value of configuration parameter 'forward.nb-copies' by '2000'
mmt-5greplay: Overridden value of configuration parameter 'forward.default' by '0'
mmt-5greplay: Ignore duplicated rule id 103 (Inject only packet from UE -> Core but not inversed direction)
mmt-5greplay: [_sctp_connect:49] Cannot connect to 127.0.0.5:38412 using SCTP
mmt-5greplay: Interrupted by signal 6
```

Figure 26 - Connection didn’t establish from 5greplay log file.

After testing the SCTP connection from the Docker environment using sctp_test, connectivity was verified using different IP addresses. The IP address was referenced from the architecture section of the supermicro system startup manual.

command - sudo sctp_test -H 192.168.200.12 -P 38412 -l

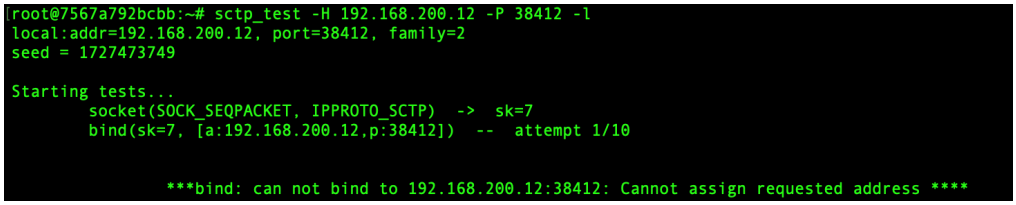

Figure 27 - SCTP binding failed for the given IP.

To proceed, different IP addresses were tested to access the core, but binding to the system failed. To address this error in Docker, minor modifications were made to Docker's configuration JSON file. It was later discovered that SCTP is not supported in Docker on macOS, as `libsctp` (or `libsctp-dev`) must be installed on a Linux system, and macOS does not support SCTP. Due to the lack of SCTP protocol support in the Docker environment, the platform was switched to WSL for further experiments.

Figure 28 - SCTP module not found (MacOS docker system).

4.4 EXPERIMENT 3 - 5G core crash test in WSL

In this experiment, we switched to WSL (Windows Subsystems for Linux) platform due to a lack of sctp support in docker environment. In first command, tried to get the IP address in the environment using netstat command and made sure that open5gs core is not in active running state. After that tried to check the firewall permission to check on there is no rules or restriction on input, output, and forward chains.

command - netstat -an | grep 38412.

The above command performs 2 actions i.e. “netstat -an” lists all network connections and listening ports on the system where -a shows all connections and -n shows address and ports. Where “grep 38412”, filters the output to only show the lines that include the port 38412. This results in a specific IP address 127.0.0.5 in which the service is listening.

command - sudo systemctl status open5gs.amfd.service, this command states that core of open 5Gs is not running and not in an active state.

command - iptables -L, shows that there is no restriction rules on the connection

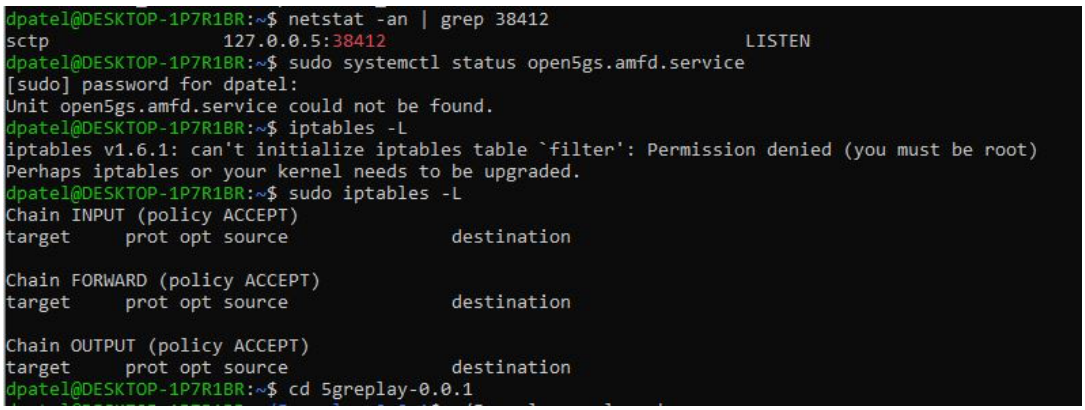

Figure 29 - result for command to check on the platform.

By adjusting minor parameters and some settings from the previous results into the configuration file, 5Greplay successfully forwarded packets to the 5G core. However, no packets were observed while monitoring in Wireshark during the test, despite no warning messages appearing in the 5G core logs, indicating that all components were operational and healthy. Unusual behavior was recorded in the system’s packet receiving and transmitting logs.

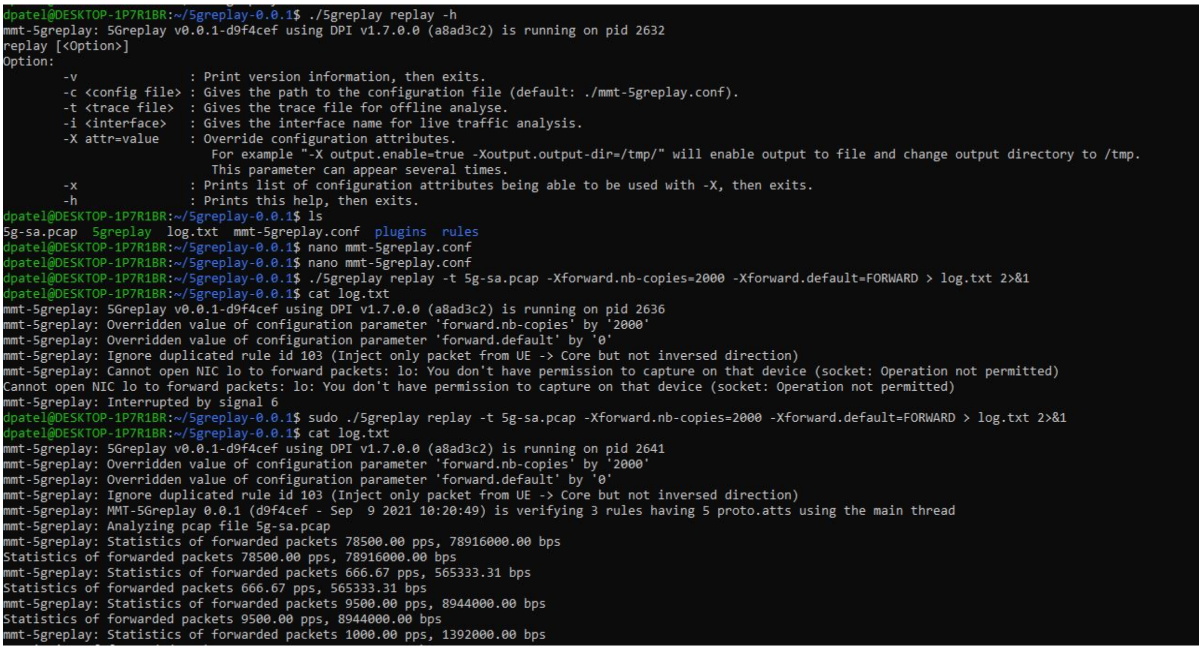

Figure 30 - 5Greplay successfully forwarded packets to core.

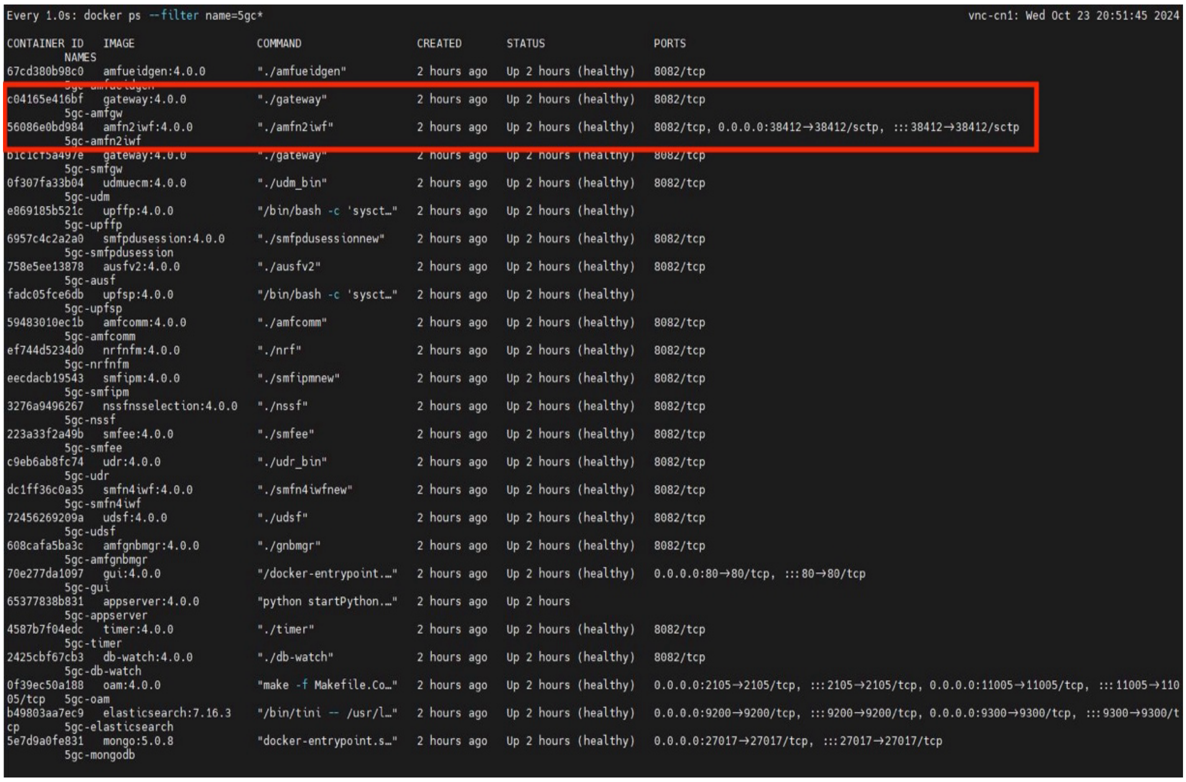

Figure 31 - 5g Core log Healthy and running without any warning.

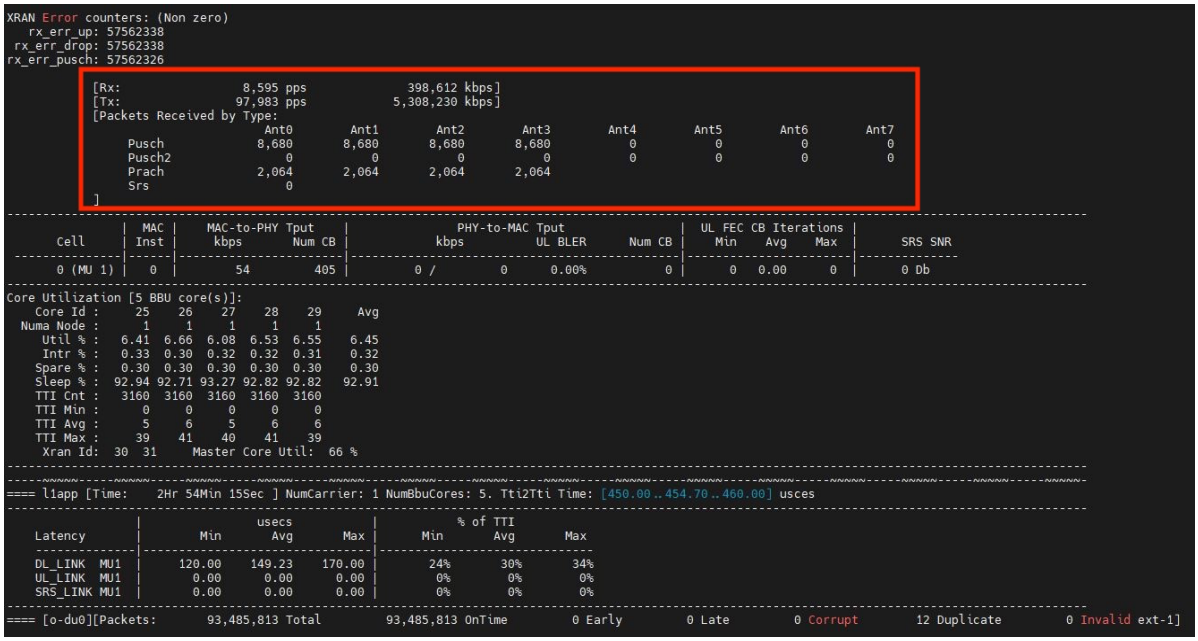

Figure 32 - Rx i.e. receiving rate of packets increased gradually (L1 console).

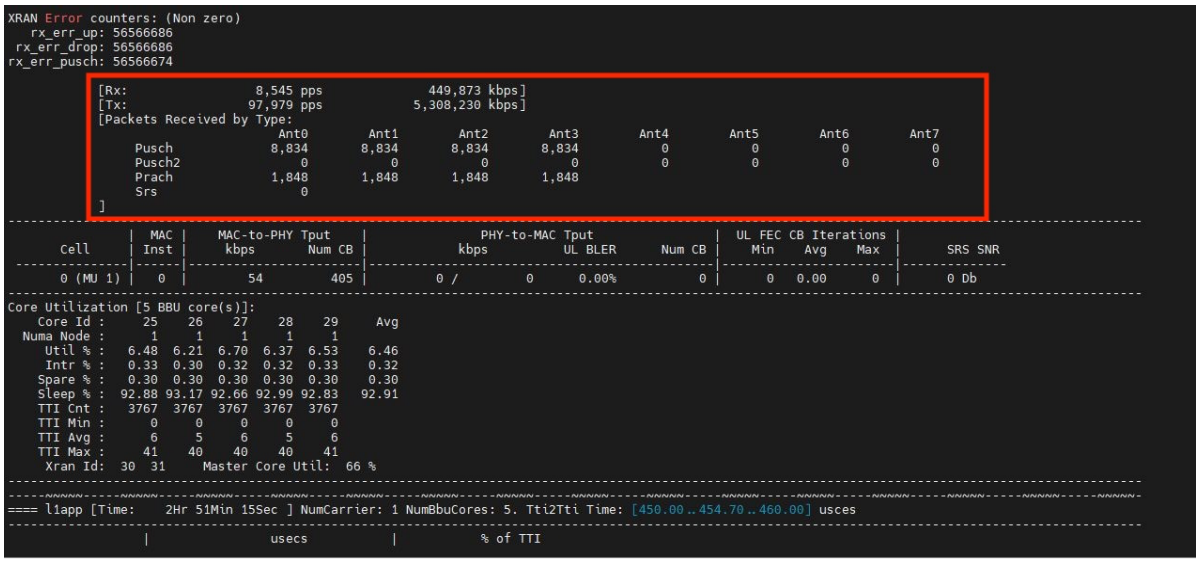

Figure 33 - Receiving of packets in 5G core increased (L1 Console).

Following this the experiment was stopped, and packets were replayed again to observe any increase in receiving packet log on L1 console in 5G system. Where Rx went up till 9766 pps & 499,069 kbps rate.

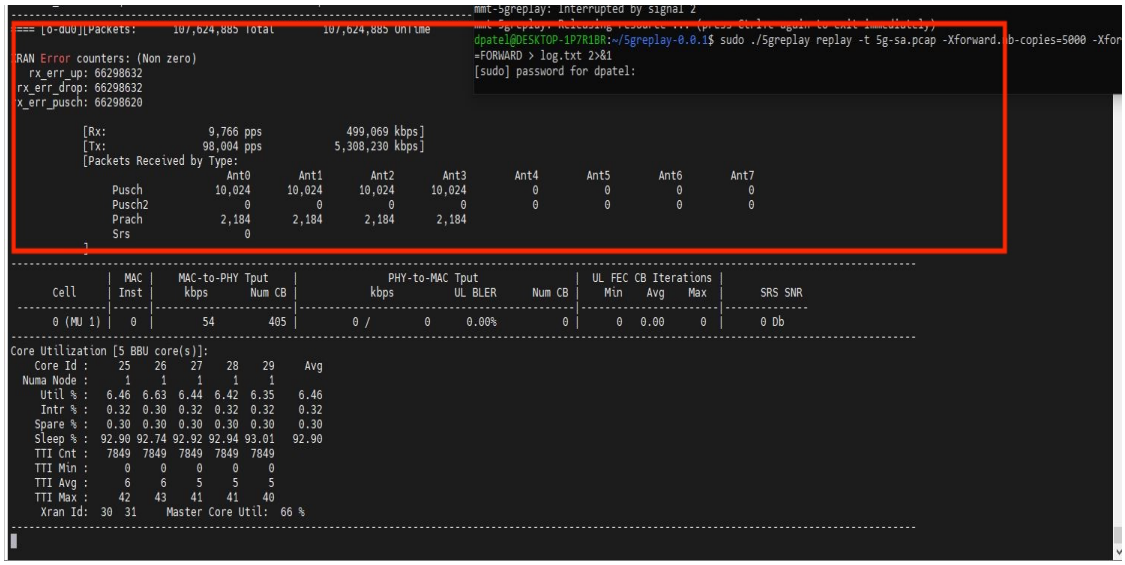

Figure 34 - An increase in received packet as tool replayed packets (L1 console).

Moreover, 5Greplay was forwarding packets at an average rate of 500pps to 600pps (Packets Per Second) which was using offline mode using built-in malformed pcap, more efficient and faster packets can be transferred to the output NIC using an external adapter in online mode as mentioned in Salazar et al. (2021). The tool reached the full rate of 9.56Gbps and 1.28Mpps when tested using Intel Ethernet Network Adapter X710 (Salazar et al., 2021). Additionally, when we started the stress attack initially Rx was at 7500pps and received packet size was at 398,612 kbps as in figure 32, eventually it went up till 9766pps and size of the packets went up 499,069 kbps as in figure 34 in few seconds, this states that a stress attack was done as large chunk of packets were recorded and no warning were generated in the 5Gnetwork.

4.5 EXPERIMENT 4 - Experiment to capture the Test packets in Wireshark.

Goal of this experiment was to capture the test packets sent from specifically from the fuzzer by changing the configuration of IP address stated in Wireshark. As per the experiment the packets were captured on the Wireshark ethernet interface (eth0).

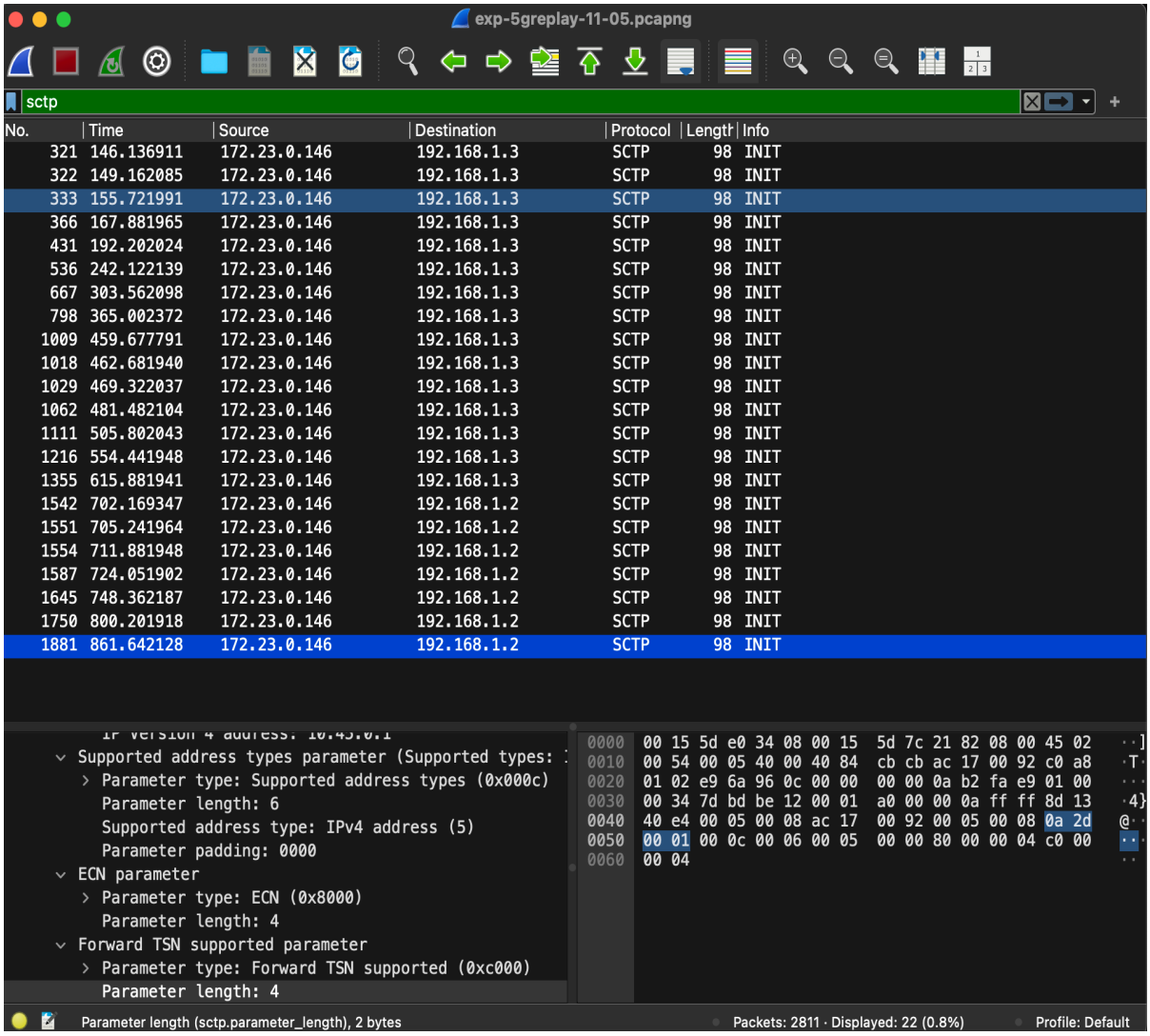

Figure 35 - Test Packets were captured on Wireshark.

4.6 EXPERIMENT 5 - NAS-5G SMC Attack using 5Greplay

Salazar et al. (2021) mention that, as per ENISA, AMFs are vulnerable to replay attacks of NAS (non-access stratum) SMC (Security mode control) procedure messages. This attack is initiated by the AMF after a successful authentication of the user equipment to establish a security context that enables encrypted communication between the UE and AMF. Here, AMF could send a NAS SMC message in an already existing security header to change the security in use. In this attack before the establishment of the security context and NAS messages are not encrypted, an intruder with access to the NAS traffic in the N1 interface, could able to access NAS SMC security mode command clear message sent from the AMF and UE, whereas

intruder can copy their NAS sequence number (NAS SQN), and use it to build a NASSMC security mode complete message that is replayed to the AMF as mentioned in figure 34.

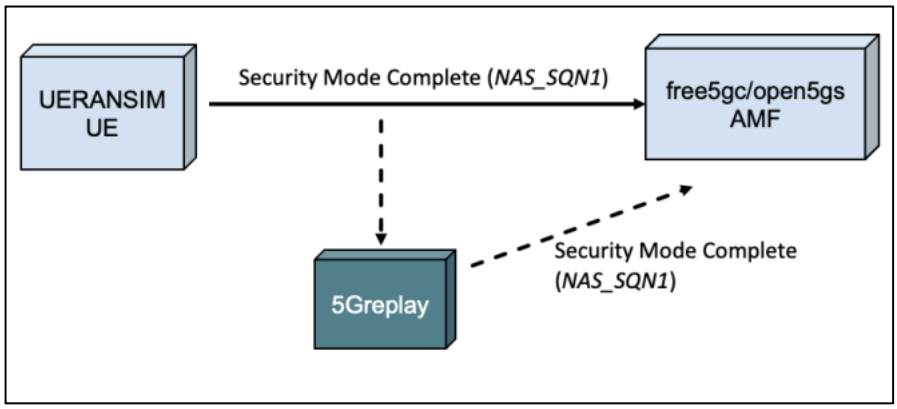

Figure 36 - NAS-5G SMC Replay Attack (Attached from Salazar et al.)

In the experiment conducted by Salazar et al. (2021) on a simulator core using Free5GC, the system's behavior was analyzed under a specific scenario. The results demonstrated that the Free5GC core successfully mitigated the attack by identifying that the packets did not belong to the same NGAP security context. These packets were determined to be replayed by 5Greplay. The study concluded that Free5GC had been tested against this type of replay attack, and it was proven that the Free5GC core is protected against such attacks.

This experiment was conducted on a 5G core following the tutorial guidance provided by Eurl (n.d.-b). The attack was executed in three main steps:

1. Designing a rule to filter the desired packets to be replayed, as shown in Figure 37.
2. Modifying the 5Greplay configuration file to specify the output NIC for the traffic and define the rules to be applied when forwarding packets as shown in Figure 38[5greplay-udp.conf] & Figure 39[5greplay-sctp.conf].
3. Running and analyzing the replayed traffic on Wireshark & 5Greplay logs.

While the first two steps were executed successfully, the third step yielded partial success. Although 5Greplay logs were generated and warning packets were captured in Wireshark, there were no visible warnings or indications of vulnerabilities in the 5G core console.

```
<beginning>
<!-- Property 4: Forwarding NAS security mode COMPLETE that answers to NAS security mode COMMAND.-->
<property value="THEN"  delay_units="ms" delay_min="0" delay_max="10" property_id="4" type_property="FORWARD"
    description="Forwarding NAS security mode COMPLETE that answers to NAS security mode COMMAND " >
    <event value="COMPUTE" event_id="1"
           description="NAS Security mode COMMAND"
           boolean_expression="(nas_5g.message_type == 93)"/>
    <event value="COMPUTE" event_id="2"
           description="NAS Security mode COMPLETE"
           boolean_expression="(nas_5g.security_type == 4)"/>
</property>
</beginning>
```

Figure 37 - Rule-NAS-5G SMC Attack

The rule is defined by two events with specific timing constraints: a minimum delay of 0ms when both events occur in the same packet, and a maximum delay of 10ms when they occur in separate packets. In the first event, the rule identifies a NAS Security Mode Command message by checking if the NAS message type equals 93. In the second event, a NAS Security Mode Complete message is expected to follow the first event. This second message is identified by a NAS security type value of 4.

After defining the rules, we need to update the configuration file to specify how and where the packets will be forwarded. For this experiment, we will use two different configuration files: 5greplay-sctp.conf and 5greplay-udp.conf**.** Both files include the general content found in the mmt-5greplay.conf file, as shown in Figure 21. Additionally, Figures 36 and 37 illustrate the modifications made to the forward section of these configuration files.

In 5greplay-sctp.conf file we modified, the default action to DROP, so only the NAS security mode complete packets filtered by property 90 will be forwarded and added 2 in nb-copies of forward section to send only 2 copies of packet to the destination.

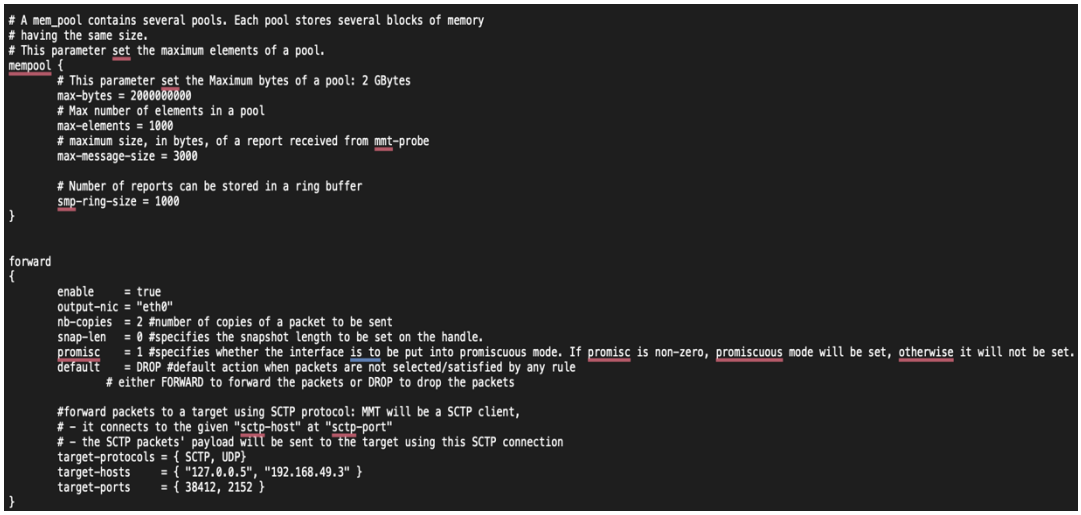

Figure 38 - Modification in Forward section of 5greplay-sctp.conf file

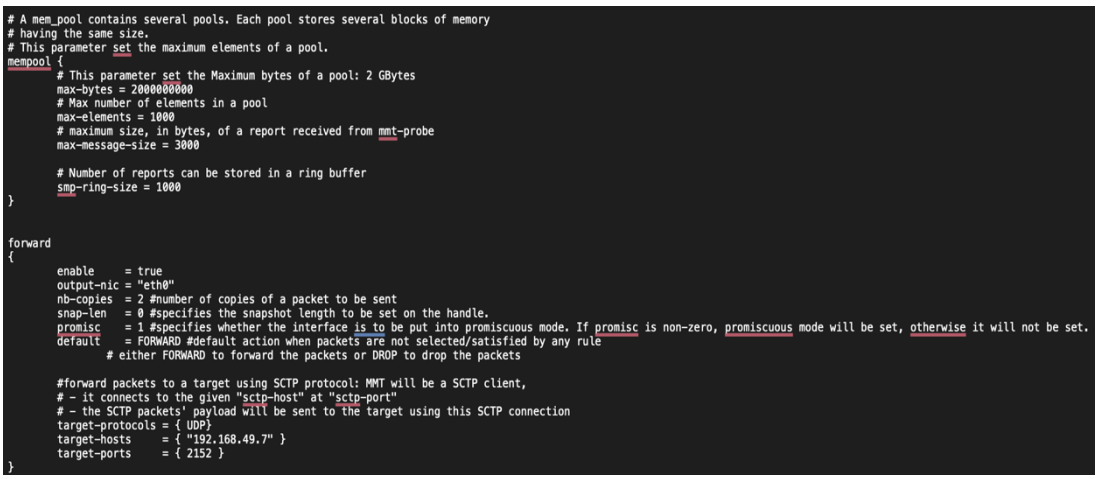

Figure 39 - Forward section of 5greplay-udp.conf file

In the 5greplay-udp.conf file, the forward section specifies the output-nic as eth0, which serves as the destination interface for sending traffic. The destination IP address and port for the transport layer are configured as 192.168.49.7:2152, using the UDP protocol. Any packet that does not meet property 90 will be forwarded to the destination by default as part of the "forward" action.

After modifying rules and configuration files, we ran 5Greplay and tested if everything worked as needed. In this test we processed an offline pcap file named ue_authetication.pcapng which contains the authentication process between UE and AMF, so packet NAS Security mode complete packet will be present, to see if rules actually filter it or if it is forwarded to the desired destination depending on the configuration file used.

Command - sudo ./5greplay replay -c 5greplay-udp.conf -t ue_authetication.pcapng, after running above command using 5greplay-udp.conf and ue_authentication.pcapng we need to observe “eth0” interface in Wireshark to see the forwarded packets as a “forward” was the default action in this configuration file. As a result, Wireshark captured warning packets as mentioned in Figure 40 and in the 5greplay log analyzer reported forwarded and dropped packets as per in Figure 41.

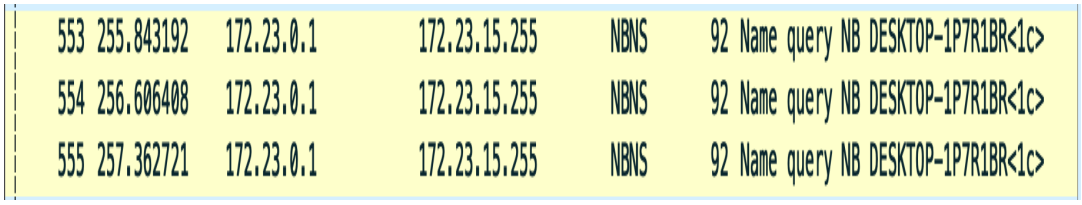

Figure 40 - Wireshark captured forwarded packets

The results obtained in Wireshark and the 5Greplay logs for this step are consistent with the outcomes demonstrated in the tutorial by Eurl (n.d.-b).

Figure 41 - 5Greplay log used with 5greplay-udp.conf file

When executing the command: sudo ./5greplay replay -c 5greplay-sctp.conf -t ue_authentication.pcapng, we observed the "eth0" interface in Wireshark; however, no packets were visible at this step. Only two packets were forwarded by 5Greplay, with the NAS Security Mode Complete message being filtered and forwarded, while the default action was set to "drop." Additionally, the 5Greplay log reflected the same output as reported in the tutorial Eurl (n.d.-b) in Figure 42.

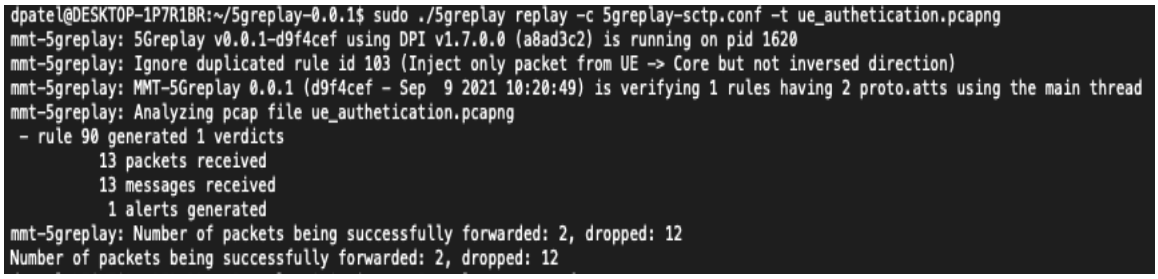

Figure 42 - 5Greplay log used with 5greplay-sctp.conf file

CHAPTER 5 : CHALLENGES

Some of the main challenges faced when looking to implement fuzz testing are the setup and the second one is to record the bug or warning and analyze it. Setting up fuzz testing can be challenging as it often requires complex testing harnesses, which can become even more difficult to create if the fuzz testing is not conducted within the same existing environment. (GitLab, 2023). If working across a simulator is easy to configure as a fuzzer or tool and the network simulator platform are in the same environment and easy to communicate among each other on the same platform. It is challenging when it comes to communicating with the fuzzer and the network protocol on different environments and platforms.

In this project, the setup of the 5Greplay fuzzer was challenging as you need to access the network from different environments, and you need to make sure with the wireshark or other monitoring tool that the fuzzer is communicating with network protocol by sending test packets. Second, we faced a challenge to log the behavior of the network when a stress attack was implemented in the network and analyze if any warning or crash occurred or if any other part of the system which is other than the target showed any unusual behavior or warnings. moreover, fuzz testing can tend to generate false positive data, so it is to ensure that everything is correctly coupled and a bit hard to document as the fuzzing process is stated as feeding random input, different platforms. For this reason, there is no surety that the fuzz technique will generate the same result or the same coverage every time (GitHub, 2024). Fuzzing may not cover every possible input, code path, or scenario within the system under test and can potentially overlook rare or deeply hidden bugs and vulnerabilities that lead to incomplete or insufficient results (GitHub, 2024).

Additionally, there is a limitation in the fuzzer we choose as well. For example, for our project, 5Greplay has its own shortfall and limitations. Firstly, the proper input where the IP address and Port number should be given for the input protocol stream i.e. SCTP or UDP to

send the packet payload. If the input is incorrect the tool will show a connection interrupted error. So, the port and IP address should be open, and the system should be running to overcome this error. Also, the tool uses SCTP to send the packet payload and what if the platform doesn't support the SCTP protocol. This kind of challenge occurs when it comes to using 5G replay as a fuzz tool for fuzzing network protocol.

CHAPTER 6 : CONCLUSION

Fuzzing with 5Greplay requires familiarity with the Linux environment and some basic knowledge of XML programming. The input rules used with the tool are developed in XML. Packets traffic was generated, forwarded, and recorded on the radio unit (L1) of the 5G core as part of negative testing, without prior knowledge of the expected output.

The results were captured on the radio unit (L1) but not in Wireshark, indicating that while the packets were forwarded within the system, they failed to reach the target where core AMF is located and crashed the system as 5G core was running healthy without stating any warnings and 5Greplay forwarded traffic at the average rate of 500pps to 600pps. Additionally, test packets sent to the core over the network were successfully captured during Wireshark monitoring. The packet capture file used in these experiments should be larger than 5 MB and contain approximately 40,000 to 50,000 packets. Initial tests on the simulator with smaller pcap files failed to produce definitive results.

Furthermore, when the NAS-5G SMC attack was performed on the 5G core following the tutorial, the results were partially successful. While only a few warning packets were observed in Wireshark, the experiment successfully demonstrated that 5Greplay was able to forward packets and generate logs. This indicates that the tool successfully transmitted the packets to the system, which was tested and found to be protected against such attacks. The captured system behavior will be utilized for further fuzz testing on the 5G core across various protocols and scenarios.

CHAPTER 7 : FUTURE WORK

In the future, additional scenarios described by Salazar et al. (2021) can be tested using the 5Greplay fuzzer. Various types of pre-built rules are available, and custom user-defined rules can also be developed to conduct in-depth testing on the 5G core. The 5Greplay tool was tested on a 5G core of commercial grade, and all captured behaviors were based on experiments tested on simulator. These different scenarios can also be applied to testing on a commercial grade 5G core other than the above experiment tested. Further testing may include scenarios such as modifying offline packets and injecting them into network traffic or sending malformed packets to the 5G core to observe whether it accepts or rejects the forwarded packets in different ways.

Additionally, there are various fuzzers that should be tested where they fuzz 5G protocols. Using 5Greplay stress testing can be done with an external adapter such as intel X710 Network adapter as it got the highest forwarding rate of 1.29Mpps and 9.5Gbps in online mode to see if the AMF crashes or not. So, if the traffic is forwarded at this rate, then there might be a chance to crash AMF at this rate.

REFERENCES

Aki Helin / radamsa · GitLab. (n.d.). GitLab. https://gitlab.com/akihe/radamsa

Deploying 5G Core Network with Open5GS and UERANSIM. (2024, March 17). *Medium*. https://medium.com/rahasak/5g-core-network-setup-with-open5gs-and-ueransim-cd0e77025fd7

Eurl, M. (n.d.-a). *Command usage*. 5Greplay.https://5greplay.org/docs/references/commands/

Eurl, M. (n.d.). *Replay 5G traffic against open5Gs*. 5Greplay https://5greplay.org/docs/tutorial/replay-open5gs/

Eurl, M. (n.d.-b). *Filtering and replaying NGAP/NAS-5G Security Mode Complete messages*. 5Greplay. https://5greplay.org/docs/tutorial/smc-message/

Finding software vulnerabilities by smart fuzzing. (2011b, March 1). IEEE Conference Publication | IEEE Xplore. https://ieeexplore.ieee.org/document/5770635

Free5GC. (n.d.). https://free5gc.org/

Gorbunov, S., & Rosenbloom, A. (2010). AutoFuzz: Automated Network Protocol Fuzzing Framework. *IJCSNS International Journal of Computer Science and Network Security, VOL.10 No.8, August 2010*. http://paper.ijcsns.org/07_book/201008/20100836.pdf

GitHub. (2024). *What is fuzzing and fuzz testing?* GitHub. https://github.com/resources/articles/security/what-is-fuzz-testing

GitLab. (2023, April 14). *What is fuzz testing? | GitLab*. GitLab. https://about.gitlab.com/topics/devsecops/what-is-fuzz-testing/

Juniper Networks. (n.d.). *SCTP Overview | Junos OS | Juniper Networks*. https://www.juniper.net/documentation/us/en/software/junos/gtp-sctp/topics/topic-map/security-gprs-sctp.html

nccgroup. (n.d.). The challenges of fuzzing 5G protocols. *nccgroup*. https://www.nccgroup.com/us/research-blog/the-challenges-of-fuzzing-5g-protocols/#tegtpu

Open5gs. (n.d.). *GitHub - open5gs/open5gs: Open5GS is a C-language Open Source implementation for 5G Core and EPC, i.e. the core network of LTE/NR network (Release-17)*. GitHub. https://github.com/open5gs/open5gs

Salazar, Z., Nguyen, H. N., Mallouli, W., Cavalli, A. R., & De Oca, E. M. (2021b). 5Greplay: a 5G Network Traffic Fuzzer - Application to Attack Injection. *ARES 2021, August 17–20, 2021, Vienna, Austria*. https://doi.org/10.1145/3465481.3470079

Sutton, M., Greene, A., & Amini, P. (n.d.). *Fuzzing: Brute Force Vulnerability Discovery*. Pearson Education.

Takanen, A., DeMott, J., & Miller, C. (2008). *Fuzzing for software security testing and quality assurance*. Artech House.

Wireshark · about. (n.d.-b). Wireshark. https://www.wireshark.org/about.html

Zhang, Z., Zhang, H., Zhao, J., & Yin, Y. (2023). A survey on the development of network protocol fuzzing techniques. *Electronics*, *12*(13), 2904. https://doi.org/10.3390/electronics12132904

SCTP use cases. (n.d.). https://docs.paloaltonetworks.com/service-providers/10-1/mobile-network-infrastructure-getting-started/sctp/sctp-use-cases

[White Paper] What is fuzzing? | Black Duck. (n.d.). https://www.blackduck.com/resources/white-papers/what-is-fuzzing.html

Fuzzing introduction: Definition, types and tools for cybersecurity pros | Infosec. (n.d.). https://www.infosecinstitute.com/resources/penetration-testing/fuzzing-introduction-definition-types-and-tools-for-cybersecurity-pros/

NGAP/NAS Server. (n.d.). https://www.blackduck.com/fuzz-testing/defensics/protocols/ngap-server.html

TechLTE World. (2024, July 18). *NGAP (NG Application Protocol) in 5G-NR*. https://www.linkedin.com/pulse/ngap-ng-application-protocol-5g-nr-techlte-world-wilvc/