Fuzz Testing Architecture Used for Vulnerability Detection in Wireless Systems

Stephen R. Mayhew

Keywords: Fuzzing, LTE, 5G, UE, eNB, gNB

Fuzz Testing Architecture Used for Vulnerability Detection in Wireless Systems

Stephen R. Mayhew

(ABSTRACT)

The wireless world of today is essential to the everyday life of millions of people. Wireless technology is evolving at a rapid pace that's speed outmatches what the previous testing can handle. This necessitates the need for smarter and faster testing methods. One of the recent fast and efficient testing methods is fuzz testing. Fuzz testing is the generation and injection of unexpected input called ”fuzzed” input for a system by slightly changing a base input hundreds or even thousands of times and introducing each change into a system to observe its effects. In this thesis, we developed and implemented a fuzz testing architecture to test 5G wireless system vulnerabilities. The proposed design uses multiple open-source software to create a virtual wireless environment for testing the fuzzed inputs' effects on the wireless attach procedure. Having an accessible and adaptable fuzzing architecture to use with wireless networks will help against malicious parties. Due to 5G simulation technology still being developed and the cost of ready-made 5G testing equipment, the architecture was implemented in an LTE environment using the srsRAN LTE simulation software, the Boofuzz fuzzing software, and Wireshark packet capture software. The results show consistent effects of the fuzz testing on the outputs of the LTE eNB. We also include a discussion of our future suggestions to improve the proposed fuzzing architecture.

Fuzz Testing Architecture Used for Vulnerability Detection in Wireless Systems

Stephen R. Mayhew

(GENERAL AUDIENCE ABSTRACT)

The persistence of the cellular network is essential to the everyday life of millions of people. Cell phones and cell towers play an important role in business, communication, and recreation across the globe. The speed of advancements made in phones and cell towers technology is outpacing the speed of security testing, increasing the possibility of system vulnerabilities and unexplored back-doors. To cover the security testing gap, different automated testing models are being researched and developed, one of which is fuzz testing. Fuzz testing is the generation and injection of unexpected input called ”fuzzed” input for a system by slightly changing a base input hundreds or even thousands of times and introducing each change into a system to observe its effects. The fuzzing architecture proposed in this thesis is used to test for security flaws in wireless cellular networks. We implemented our fuzz testing model in a simulated 4G cellular network, where the results show the effectiveness of the model on tracing network vulnerabilities. The results of the experiment show consistent effects of the fuzz testing on a wireless system. A discussion of how the proposed model can be further improved for future work is added to the end of this thesis.

Dedication

...to my parents, Ruth and David Mayhew, and my brothers, Alex and Austin Mayhew

Acknowledgments

I would like to thank the many people who guided me in this journey. Dr. Jeff Reed, Dr. Carl Dietrich, and Dr. Ying Wang were all amazing sources of knowledge and encouraged me while I worked towards the completion of my degree. Dimitri Dessources and Daniel Setareh have been incredible helpful, and working alongside them was a pleasure. Thank you to the following research sponsors, listed in chronological order: UT Battelle LLC (Contract Number: 4000170832), Commonwealth Cyber Initiative, National Science Foundation (Contract Number: CNS-1642873), Deloitte Touche LLP (Contract Number: AT-81748), Air Force Research Laboratory (Contract Number: FA8750-20-2-0504).

Contents

List of Figures

List of Tables

List of Abbreviations

3GPP	3rd Generation Partnership Project
5G	5th Generation
5GNR	5th Generation New Radio
eNB	Evolved Node B or E-UTRAN Node B
FCC	Federal Communications Commission
GAN	Generative Adversarial Network
gNB	next Generation Node B
HSS	Home Subscriber Service
KPI	Key Performance Indicator
LTE	Long-Term Evolution
M-TMSI	MME-Temporary Mobile Subscriber Identity
MAC	Medium Access Control
MME	Mobility Management Entity
MMEC	Mobility Management Entity Code
NAS	Non-Access Stratum
PDCP	Packet Data Convergence Protocol

PDN	Packet Data Network
PDU	Protocol Data Unit
PGW	PDN Gateway
RLC	Radio Link Control
RRC	Radio Resource Control
SGW	Serving Gateway
UE	User Equipment

Chapter 1

Introduction[1]

1.1 Motivation

Wireless Networks make up a significant part of modern-day life; people spend their free time and professional time on their phones and other mobile devices. Enhancing network security is important to avoid harmful attacks that may exploit user data and disrupt network connections.

Identifying vulnerabilities prevents malicious entities from exploiting users. These bad actors can use the vulnerabilities to track users or disconnect users from the network [3]. In cellular networks, fuzzing is not uncommon when testing the UE side, and for this research we explore the utility of the approach for base station testing. Base station attacks, when successful, can have a much more pronounced effect on the network as the malicious attack likely affects more than one user's experience. A single user affected by an attack is already an unwelcome occurrence, but a compromised base station may impact hundreds of people. Fuzz testing can uncover these vulnerabilities in networks especially at the physical and MAC layers (layers 1 and 2, respectively). Once the vulnerabilities are known, steps can then be taken to fix the vulnerabilities so that malicious entities cannot take advantage of them.

[1]The research for this thesis was done together with Dimitri Dessources and Daniel Setareh. Dimitri worked together with the author of this thesis on the planning, theorizing, and implementation of the fuzz testing in 4G and 5G environments. Daniel Setareh assisted with writing code and implementing it into srsRAN.

The goal of this thesis is to design a fuzzing architecture that detects vulnerabilities in a wireless system. The architecture developed for this thesis was designed for use in future wireless generations to detect vulnerabilities. The author pursued this goal because of the uniqueness of fuzz testing and his interest in wireless technology.

1.2 Significance

The rapid advancements in wireless technology necessitate faster vulnerability detection. To accommodate these advancements and elevate network security, a method of security vulnerability testing with minimal knowledge of the system is essential. In this thesis, we developed a testing framework by combining open source wireless simulation software with fuzzing software in order to test for vulnerabilities. This novel approach is designed to test network connections and their standards rather than user equipment. The standards are a guide for wireless connections, but the standards have flaws and can be exploited. The implementation of a wireless network has weaknesses that are overlooked by the standards. Past research on this topic concentrated on vulnerabilities in user equipment. The focus on UE security is because customers of a wireless network are protective of their own devices. The significance of this project is the development of an architecture to test the security of a network connection. This architecture is designed to be easily implemented, uncover vulnerabilities, and be adaptable to future wireless networks.

Fuzzing is a testing method similar to a brute force attack, which sends countless input variations to monitor the system for negative responses. When the system reacts negatively to the fuzz test, the tester can take that knowledge to the appropriate authority such as the standards committee or product manufacturer to responsibly report the vulnerability. The fuzz testing that is being used in this thesis has a straightforward implementation comprised of open-source software. A consequence of this architecture is that a malicious entity could establish a similar system and use these testing methods as a way to probe for vulnerabilities.

The methodology designed to find the vulnerabilities in a wireless network was co-designed by this author. The testing results of our proposed framework show an efficient and successful performance in finding vulnerabilities in the system under test.

1.3 Overview

Chapter 2 provides the background of fuzz testing and wireless systems, reviews similar methods of fuzz testing, and compares other types of attacks on wireless systems. Chapter 3 discusses the roadblocks of building a 5G system. The architecture of the 5G network has some similarities and some important differences to the 4G network. Two notable differences in the 5G architecture are the transformed core network, and the addition of a Non-Standalone(NSA) mode. The challenges of building an end to end emulated system for testing are discussed in this section. Chapter 4 discusses the ideal architecture for fuzz testing a wireless network. A high level description of the fuzz testing architecture is presented and explained. Chapter 5 discusses the design of the fuzzing system. The software used and the architecture of the system is discussed in addition to the advantages and disadvantages to the architecture. Chapter 6 presents preliminary results of this experiment and discusses the possible vulnerabilities found during the testing. Chapter 7 makes concluding remarks about the experiment and discusses how this research can be expanded in the future.

The research concluded that there are possible weaknesses in either the 3GPP standards or the code of the srsRAN software. When tested, the response of the system was measured using specific key performance indicators(KPIs). The response revealed a weakness: delay of data being sent. The research was performed using open-source LTE simulation and fuzzing software.

Chapter 2

Review of Literature

2.1 Relevant Fuzzing Work and Fuzzing Background

Fuzzing is categorized into multiple sub-types based on the target application structure, the input format, and the input generation. The application structure formatting has three categories: blackbox fuzzing, whitebox fuzzing, and greybox fuzzing. These three types of fuzzing are shown in Figure 2.1. Blackbox fuzzing is when the fuzzer has no knowledge of the code or protocol that it is fuzzing. Whitebox fuzzing is when the fuzzer knows the complete structure of the code and uses this knowledge while fuzzing. Greybox fuzzing is when the fuzzer has moderate knowledge of the code or protocol that it uses when fuzzing. Greybox can be thought of as halfway between whitebox and blackbox fuzzing with some of the benefits as well as downsides to each. Blackbox and greybox are easier to implement than whitebox fuzzing as they require less knowledge, but whitebox can fuzz more complicated systems than either blackbox or greybox [4] [5].

Input format has two categories: smart and dumb. This naming convention can be easily misunderstood, but a smart fuzzer is not necessarily better than a dumb fuzzer; they are simply different tools used in different scenarios. A smart fuzzer knows the format for the input and restricts the fuzzing so that it will fit this format [6] [7]. A dumb fuzzer on the other hand has no knowledge of the input format and fuzzes without any restrictions. A smart fuzzer could be used in a case that the input has specifications such as the size of a

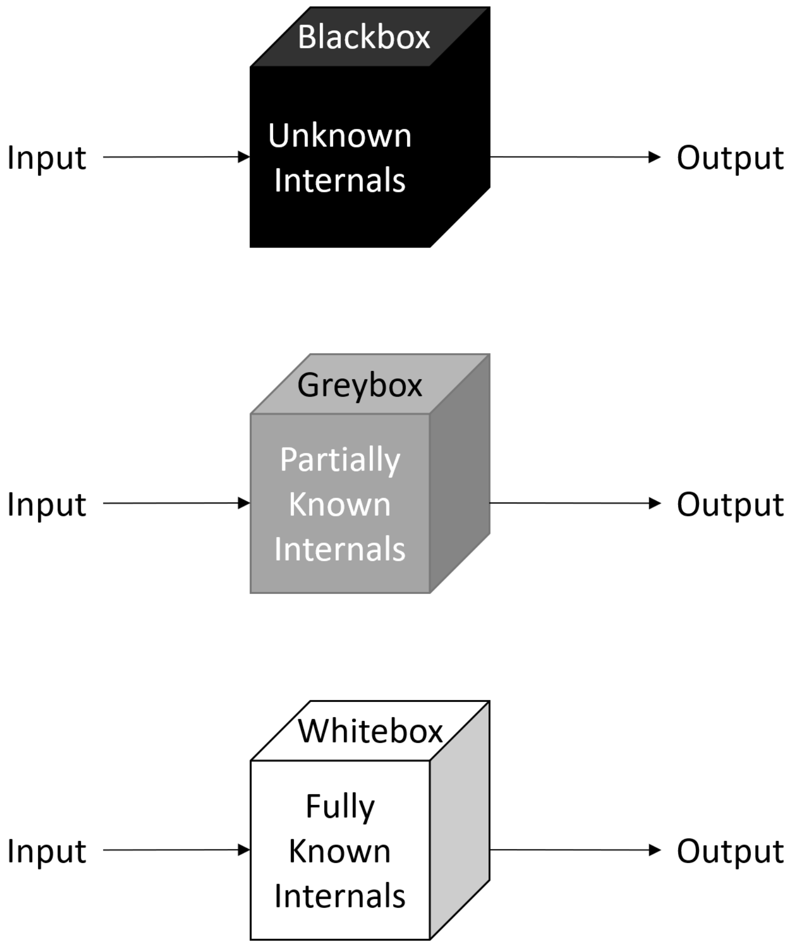

Figure 2.1: Blackbox, Greybox, and Whitebox Fuzzing Diagram[1]

message. A dumb fuzzer can be used in most situations where a format is not required and the input can be completely random [7]. The authors of [6] demonstrate the use of a smart fuzzing method to test file transfer protocol implementations.

Input generation can be split into two categories as well: generation and mutation. A generation based fuzzer takes no prior input and generates the fuzzed input without any basis. A mutation based fuzzer uses prior input and modifies it in various ways to create fuzzed input. A mutation based fuzzer can mutate the prior input in a variety of ways such as swapping bits, flipping bits, appending bits, and removing bits. The way mutations are generated depends on the fuzzing software being used. The author of [8] uses a genetic algorithm to mutate the inputs, but other fuzzers can do this by cycling through a list of possible mutations [9] [10].

In [11] the author uses Peach framework to fuzz REDHAWK communication software. They state that one of the benefits to blackbox fuzzing is the detection of edge case errors, which they themselves discovered during their tests. Although the target for the fuzz testing in their paper is different, the purpose is the same as ours. The author uses fuzzing to test for edge cases in a wireless system similarly to the fuzzing used for this research. Peach fuzzer [12] is used to fuzz test REDHAWK, while the fuzz testing in this work is performed using Boofuzz [13]. At the time of writing this paper, Peach is a fuzzing tool owned by Mozilla, but when [11] was written, Peach was an open source fuzzer similar to Boofuzz. Each of these testing environments was developed to uncover flaws in a wireless related system.

The authors of LTEFuzz [14] developed a semi-automated testing tool for LTE networks using open-source LTE software. They analyzed the LTE security standards to identify possible vulnerabilities. Using the analysis of the standards, different properties were designated based on handling of the messages and compliance with 3GPP standards. The standards were then used to generate test cases for each property and used fuzz testing to assess these test cases. The results of the test cases were used to identify negative responses from the system. The authors uncovered a variety of negative behaviors such as denial of service and de-registration of a UE. The authors uncovered 36 new vulnerabilities in the LTE network using this method. While [14] uses in-depth analysis of the standards to find properties to test, our straightforward approach negated the need for this analysis.

The author of [8] uses a physical environment to implement their fuzz testing. They used a genetic algorithm to fuzz the data rather than Boofuzz because their genetic method had better compatibility with their physical environment. The author states that due to the way that Boofuzz generates inputs, it would be difficult to make the inputs compatible with the equipment they were using. The set up for the testing environment discussed later in this thesis is held in a virtual environment, which allowed the generation of more test cases faster compared to a physical environment. The fuzzing proposed in this thesis focuses more on the 3GPP technical standards and correctly implementing them in order to see whether the standard itself is exploitable, in comparison to the fuzz testing of [8] which seeks to evaluate fuzz testing with specific equipment in LTE.

In [15] the author performs fuzz testing on the physical layer of a Wi-Fi access point. The author uses a fuzzer to modify drivers responsible for the Physical Layer Convergence Protocol (PLCP) and Protocol Data Unit (PDU). It was concluded that physical layer fuzzing is possible, but there were concerns about the meaning of the results that they produced because of a lack of insight into the system's response. Similar to this thesis, [15] is also concerned with fuzzing a wireless connection. However, they are focused on the physical layer and Wi-Fi implementation whereas this thesis focuses on fuzzing the network layer of 4G and 5G networks.

The authors of [16] perform fuzz testing on MAC Layer protocols. They use mutation based fuzzing alongside an analysis algorithm called Relationships Analysis and Testing case Marking (RATM). RATM is a model for analyzing the relationships between multiple fields in a protocol. The mutation based fuzzing is used on MAC layer protocols and RATM is used to analyze the effects of the fuzzing. The authors uncover multiple vulnerabilities using this method including crashing the server used in their testing. [16] has a similar goal to this thesis. The focus of [16] is on the MAC layer whereas this thesis focuses on the network layer. Their use of an analysis algorithm to observe the effects also contrasts to the single field focus of this thesis.

There is different fuzzing software that can be used for testing listed in Table 2.1. Boofuzz [13] is an open-source fuzzing software that has a straightforward implementation, extensive documentation, and a variety of useful features. It generates exportable response data from the system and monitors the testing target for possible crashes. An arguable downside to Boofuzz is the unsophisticated fuzzing that it performs. Peach [12] is another fuzzer that was previously open-source, but was purchased by Mozilla in 2020. The previous open-source code for Peach is available, but it is no longer maintained or updated. Because it is now commercial it has the benefits of having professional support. One can only use it if they purchase the software. AFL [17] is another open-source fuzzing software with extensive documentation and many useful features. AFL was initially considered for use in this project, but it could not be used because AFL requires a binary file to interact with the fuzz target. Sulley [18] was also considered, but it is an antiquated version of Boofuzz so it was discarded. Another fuzzer that was researched was LibFuzzer [19]. LibFuzzer is a fuzzing software used to fuzz libraries rather than programs or inputs so it was not considered for use. Boofuzz was chosen to be the best program to use for this project.

There is debate about the best method of fuzz testing. Techniques applied to one system could be less effective when applied to a different system. A technique such as RATM used in [16] would not be applicable to a system with only one input, but RATM was highly effective when it was used by the authors of [16]. Recent efforts in fuzz testing have included researchers developing there own fuzzers as well as incorporating machine learning into the fuzz testing process [20] [21] [22] [9].

Table 2.1: Fuzzing Software

Fuzzing Software	**Pros**	**Cons**	**Reference**
Boofuzz	Open-source, straight-forward implementation, extensive documentation, python code	Unsophisticated fuzzing	[13]
Peach	Extensive documentation, commercial support and maintenance	Commercial, not open-source	[12]
AFL	Open-source, extensive documentation	Requires binary files, not compatible with this architecture	[17]
Sulley	Open-source, straight-forward implementation	Antiquated version of Boofuzz, lacks documentation	[18]
LibFuzzer	Open-source, extensive documentation	Made to fuzz libraries, not compatible with this architecture	[19]

2.2 Relevant Wireless Work and Wireless Background

2.2.1 LTE and 5G Stack

The LTE protocol stack between the UE and the eNB, shown in Figure 2.2, can be sectioned into three layers. Starting from the bottom the layers are split into the physical layer, the data link layer, and the network layer. The physical layer transmits information over the air interface between the UE and the eNB. It is also in charge of cell search, cell synchronization, and encoding and modulation schemes.

The data link layer is in charge of linking the physical and network layers and is split into three subsections. From the bottom the subsections are the Media Access Control(MAC) protocol, the Radio Link Control(RLC) protocol, and the Packet Data Convergence Protocol(PDCP). At a high level, the MAC protocol is in charge of controlling radio resources for allocation. The RLC protocol is in charge of error detection, error correction, and transfer of network layer PDUs. The PDCP manages transfer of user and control plane data to the network layer as well as encryption and integrity protection.

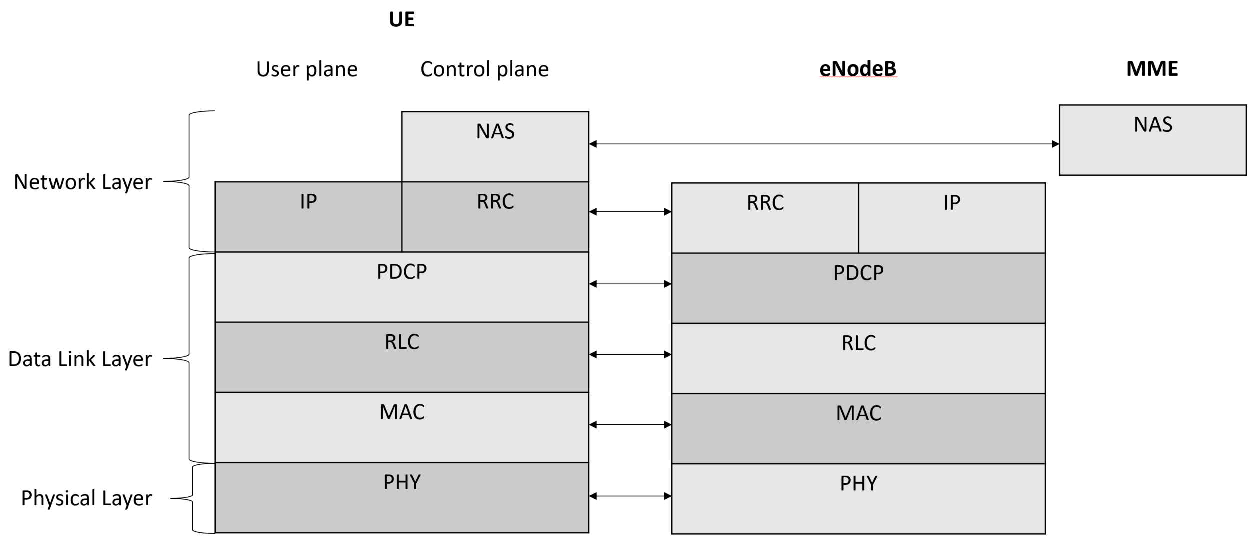

Figure 2.2: LTE Stack Architecture[1]

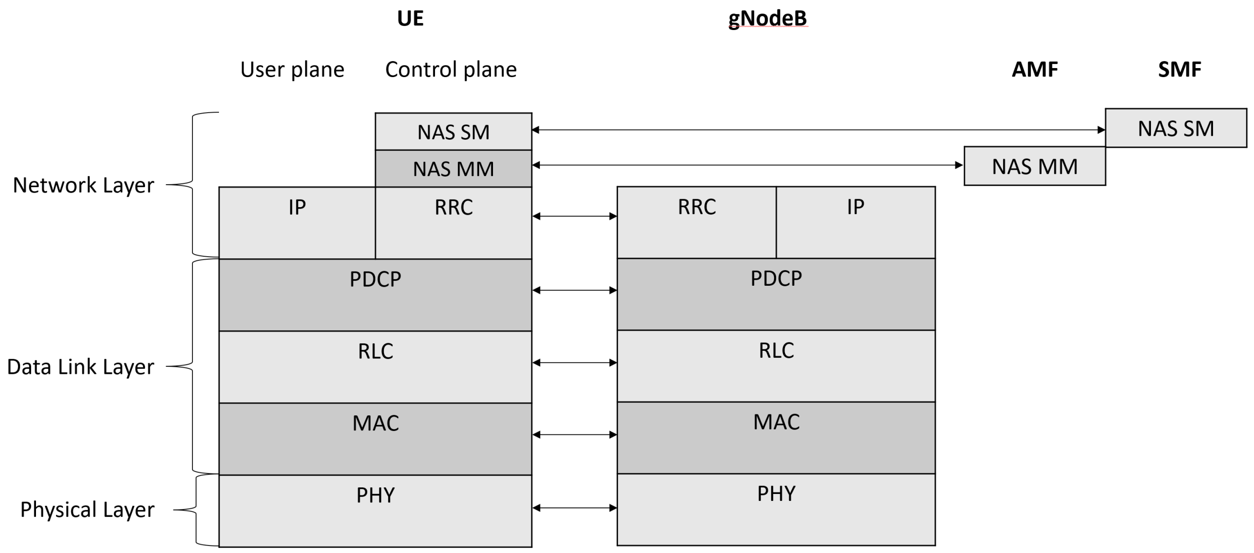

Figure 2.3: 5G Stack Architecture[1]

The network layer, similarly to the data link layer, is split up into three subsections. The network layer subsections are the Non-Access Stratum(NAS), the Radio Resource Control(RRC), and Internet Protocol(IP). The NAS is tasked with establishing and maintaining continuous communications between UEs and core networks. The RRC by contrast is in charge of communications between UEs and eNBs. The IP manages messages to the transport layer and connections to the internet.

The 5G protocol stack is similar to the LTE protocol stack with a few key differences. A diagram for the 5G stack can be seen in Figure 2.3. Due to differences in the core network that are discussed later, the NAS interaction between the UE and the MME has been split up into the NAS Mobility Management (NAS-MM) and the NAS Session Management (NAS-SM) with the AMF and SMF, respectively. The NAS-MM is responsible for communication between the UE and the AMF of management functionality as well as ciphering and integrity protection of the NAS signalling. The NAS-SM is responsible for the session management between the UE and the SMF. The rest of the core network is functionally the same as the EPC, with the physical and data link layers being comprised of the same elements with the same functions as the EPC [23] [24] [25].

RRC

The experiment in this thesis primarily deals with the RRC protocol. At its core the RRC is in charge of communicating between the UE and the eNB in the RAN over what is called the RRC connection. More specifically the RRC handles specifics like the broadcasting of system information and characteristics of the entities such as the Temporary Mobile Subscriber Identity(TMSI) that lets the eNB identify who or what is trying to connect. In addition the RRC handles activating the security mode during the connection procedure. Security mode handles the ciphering and integrity protecting of Signalling Radio Bearer(SRB) and Data Radio Bearer(DRB) messages. The RRC also handles handover procedures when transferring from one eNB to another. The handover procedure also accounts for transferring from an eNB to a base station from a newer or older mobile network like 5G or 3G.

The RRC has two states that it can be in LTE: idle mode(RRC_IDLE) or connected mode (RRC_CONNECTED). In RRC_IDLE, the eNB is not connected to any UE and it remains in this state until the RRC connection complete message has been sent. In RRC_CONNECTED, the eNB is actively connected to a UE. While in RRC_CONNECTED the UE and eNB transmit and receive traffic and signaling data from each other. To enter connected mode the UE and eNB have to go through the LTE attach procedure which is discussed in a later section.

RRC in 5G has been generally unchanged since LTE, but there are differences that need to be mentioned. In LTE, RRC switches between idle and connected mode, but in 5G a new state called inactive mode(RRC_INACTIVE) exists as a middle-ground between idle and connected mode. Inactive mode takes down the radio bearers but keeps the signalling connection and the UE's connection to the core network. This allows smartphone apps to keep running in the background without completely disconnecting.

2.2.2 LTE and 5G Core Networks

The focus of the testing performed for this thesis was on the connection between a UE and eNB, but understanding how these interface with the core network is important. The core network of an LTE network, also commonly called the Evolved Packet Core(EPC), is composed of four main components. These components are the Mobility Management Entity(MME), the Home Subscriber Service(HSS), the Serving Gateway(SGW), and the PDN Gateway (PGW). The MME is responsible for control plane functions and session management. The HSS is in charge of user identification information for authentication and supports the mobility management of the MME. The SGW and PGW are both responsible

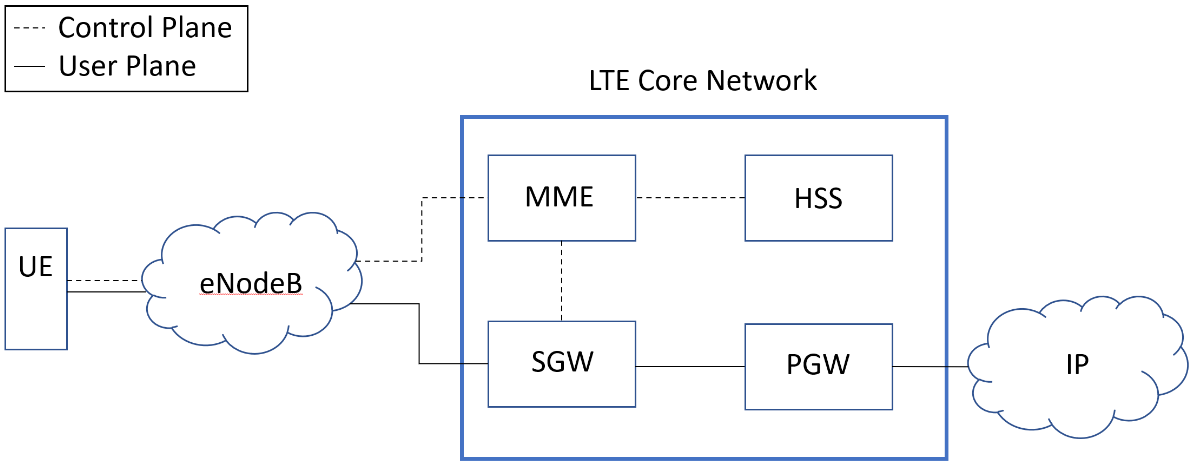

Figure 2.4: LTE Core Network Diagram[1]

for user plane functions. The SGW handles the connection between the RAN and the EPC by routing IP packets both coming and going from the EPC. The PGW handles the connection between the EPC and external IP networks. One part of the EPC not shown in the figure is the Policy and Charging Rules Function Server(PCRF). The PCRF manages the service policy and quality of service information for user sessions as well as charging rules selection for a given service.

The 5G Core Network (5GC) has the same overall function as the 4G core network, but it has a service based architecture. The MME, HSS, SGW, and PGW's functionality has been split up into different core network functions. The ten network functions of the 5GC that can be seen in Figure 2.5 are listed below.

- Access and Mobility Function (AMF): Handles NAS ciphering and integrity protection, mobility management, and registration management on top of other functions so it has part of the MME's functionality.

- Session Management Function (SMF): supports session management, UE IP address allocation, and traffic configuration for the UPF so it has parts of the MME and PGW’s functionality.

- User Plane Function (UPF): handles packet routing and forwarding as well as quality of service handling and external PDU session point of connection to the Data Network which means it has parts of the SGW and PGW’s functionality.
- Policy Control Function (PCF): handles the unified policy framework and providing rules to the control plane functions so it has part of the PCRF’s functionality.
- Authentication Server Function (AUSF): an authentication server like the HSS in an LTE network.
- Unified Data Management (UDM): generates authentication and key agreement credentials and handles user identification and subscription management like the HSS does in an LTE network.
- Application Function (AF): supports accessing the NEF, application’s influence on traffic routing, and interactions with policy framework for the sake of policy control.
- Network Exposure Function (NEF): handles access to exposed network services and their capabilities.
- NF Repository Function (NRF): provides a record of the network functions available in a given area.
- Network Slice Selection Function (NSSF): handles network slicing. Network slicing is a new capability to 5G that allows for virtualized end-to-end ”slices” of a network that can be configured for specific uses.

The NEF, NRF, and NSSF all have unique functionality to the 5G network that cannot be compared to LTE core network functionality. The NSSF especially brings something unique to the system, as network slicing is a new capability that 5G boasts [26] [27] [28] [29] [30].

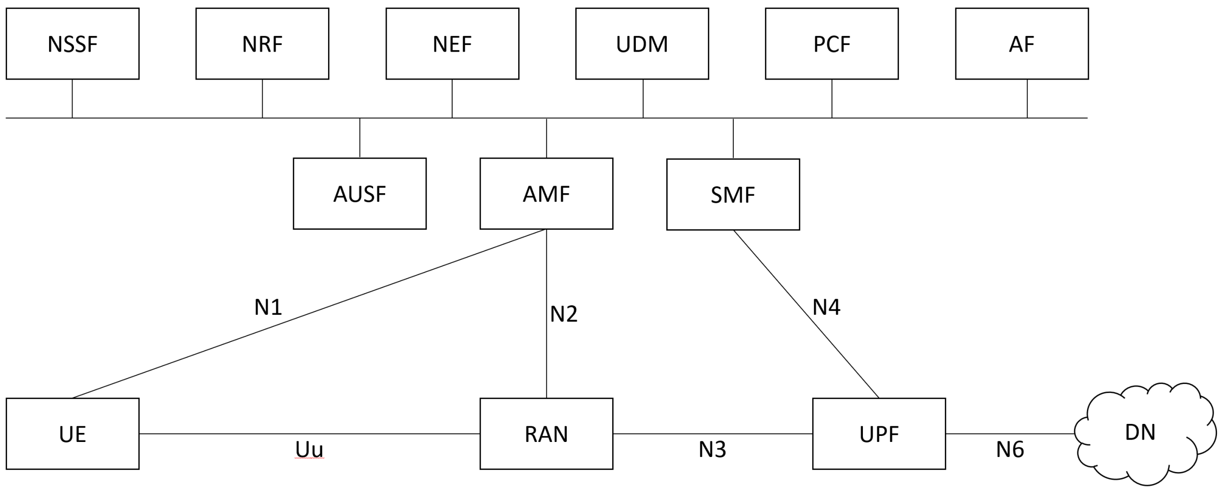

Figure 2.5: 5G Core Network Architecture

2.2.3 LTE and 5G Attach Procedures

The LTE attach procedure, shown in Figure 2.6, follows a strict sequence of messages and responses between the UE, eNB, and EPC. The research conducted focuses on the first three messages in the initial attach procedure: the network acquisition, the random access preamble, and the RRC connection setup. The network acquisition step is when the UE and eNB send information to each other about specifications and capabilities of the network. The random access preamble is a Zadoff-Chu sequence to establish synchronization between the UE and the eNB. The RRC connection request is used to request an RRC connection with the eNB from the UE. The next message, the RRC connection setup, is the eNB's response to the RRC connection request from the UE to initiate an RRC connection. The RRC connection complete is an acknowledgement from the UE to the eNB that the RRC

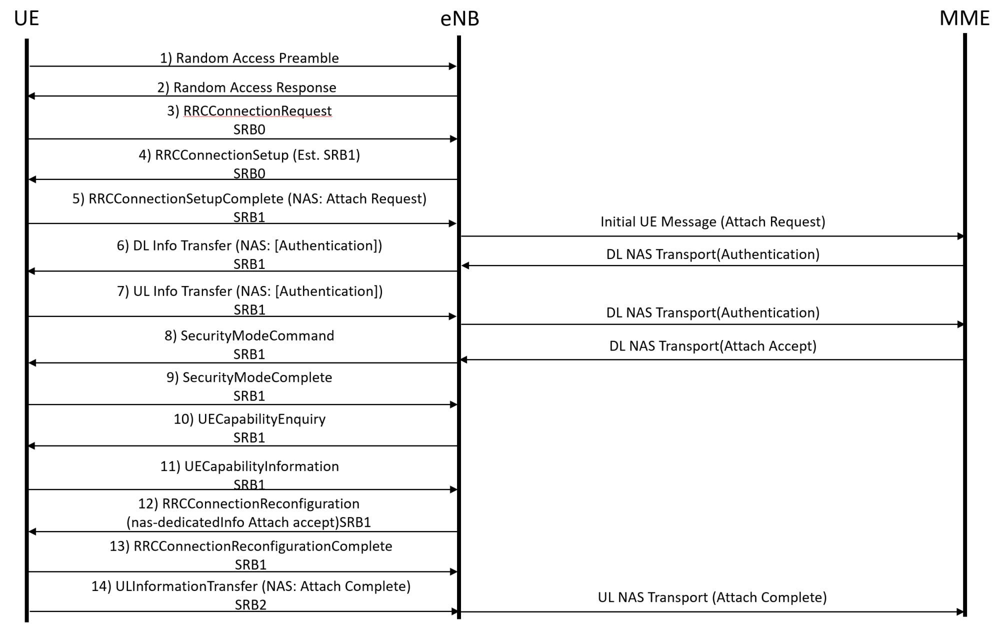
UE
eNB
MME
1) Random Access Preamble
2) Random Access Response
3) RRCConnectionRequest SRB0
4) RRCConnectionSetup (Est. SRB1) SRB0
5) RRCConnectionSetupComplete (NAS: Attach Request) SRB1
Initial UE Message (Attach Request)
6) DL Info Transfer (NAS: [Authentication]) SRB1
DL NAS Transport(Authentication)
7) UL Info Transfer (NAS: [Authentication]) SRB1
DL NAS Transport(Authentication)
8) SecurityModeCommand SRB1
DL NAS Transport(Attach Accept)
9) SecurityModeComplete SRB1
10) UECapabilityEnquiry SRB1
11) UECapabilityInformation SRB1
12) RRCConnectionReconfiguration (nas-dedicatedInfo Attach accept)SRB1
13) RRCConnectionReconfigurationComplete SRB1
14) ULInformationTransfer (NAS: Attach Complete) SRB2
UL NAS Transport (Attach Complete)

Figure 2.6: Diagram of LTE attach procedure[1]

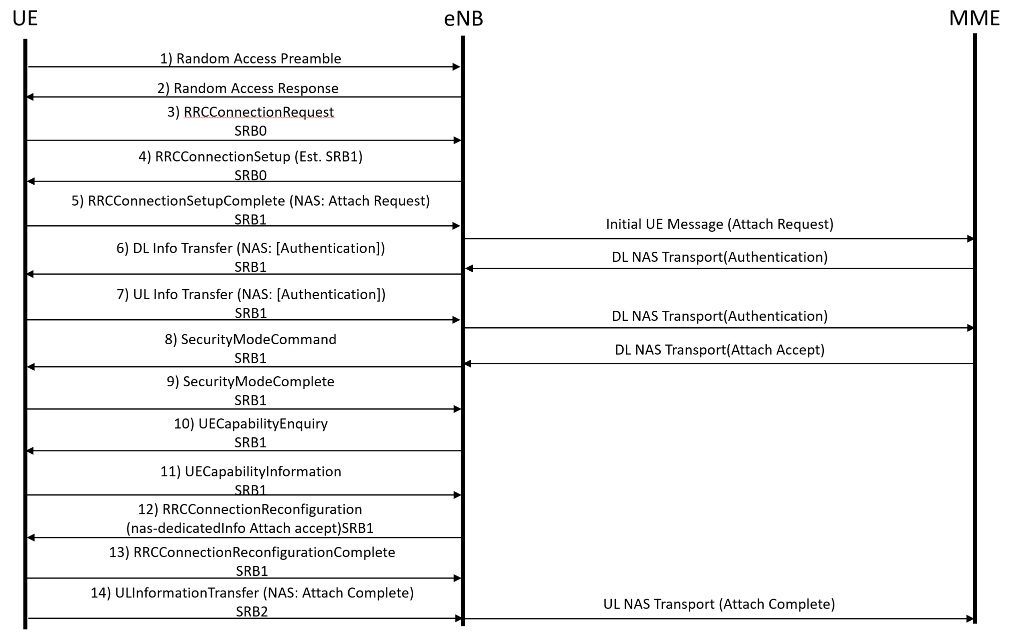
UE
eNB
MME
1) Random Access Preamble
2) Random Access Response
3) RRCConnectionRequest SRB0
4) RRCConnectionSetup (Est. SRB1) SRB0
5) RRCConnectionSetupComplete (NAS: Attach Request) SRB1
Initial UE Message (Attach Request)
6) DL Info Transfer (NAS: [Authentication]) SRB1
DL NAS Transport(Authentication)
7) UL Info Transfer (NAS: [Authentication]) SRB1
DL NAS Transport(Authentication)
8) SecurityModeCommand SRB1
DL NAS Transport(Attach Accept)
9) SecurityModeComplete SRB1
10) UECapabilityEnquiry SRB1
11) UECapabilityInformation SRB1
12) RRCConnectionReconfiguration (nas-dedicatedInfo Attach accept)SRB1
13) RRCConnectionReconfigurationComplete SRB1
14) ULInformationTransfer (NAS: Attach Complete) SRB2
UL NAS Transport (Attach Complete)

Figure 2.7: Flow chart of 5G SA attach procedure

connection has been established. Once the RRC connection has been established the eNB communicates with the MME to start an attach request to the core network. After the MME has confirmed the attach request, the UE and eNB initiate the security mode of the RRC. Once the security mode has started the UE shares its capabilities, the RRC connection is reconfigured, and the attach complete is sent to the MME. Messages sent after the security mode complete are encrypted and integrity protected.

One aspect to consider in the design of a 5G network is the difference between stand alone(SA) and non-stand alone(NSA) deployments. The attach procedure in a 5G stand alone network is for the most part the same as the LTE attach procedure with the eNB replaced with a gNB and the RAN interacts with the AMF rather than the MME.

However, the 5G non-stand alone attach procedure differs due to the difference in structure, with the non-stand alone using an LTE core with 5G radio. Non-stand alone is a shorter term solution for 5G since it uses 4G infrastructure that is already in place. Both setups can be used to test for vulnerabilities, but since the non-stand alone procedure uses both an LTE eNB and 5G gNB the messages that can be tested differ. The beginning of the 5G non-stand alone attach procedure still uses the LTE attach procedure to establish a connection between an LTE and 5G capable UE and an LTE eNB, but after that the procedure differs with messages between the LTE eNB and 5G gNB. This could mean that any vulnerabilities uncovered with this experiment may work in a non-stand alone environment, but until the 5G simulation software becomes available that is just speculation.

2.2.4 Related Wireless Work

Jamming and fuzzing are methods of attacking a network with vastly different methods. With a fuzzing attack, the attacker sends legitimate bits to get accepted by the receiver and then sends the fuzzed message that may or may not negatively affect the network. Jamming differs in that it requires consistent use of resources to harm a network while fuzzing is a tool used on a network to find a specific message that can be used to harm the network. Fuzzing does differ in that although during testing it is continuously sending fuzzed input to get a response, once the harmful sequence has been found it would only take one message to harm the system rather than the countless messages it takes for a jammer to be effective. 5G is expected to be used for low-latency high-bandwidth services such as emergency services and military. Any vulnerabilities during these operations can be extremely costly. A few milliseconds of lag in a low-latency signal used for emergency medical services could be the difference between life and death for a patient. The authors of [31] talk about jamming and how it can affect a 5G network. They go over several types of jamming attacks. Regular jammers inject signals continuously to occupy the transmission channel and make it harder for legitimate users to send data. Delusive jammers continuously inject legitimate sequences of bits into the communication channel so that the receiver waits for messages that will not arrive. Random jammers act as regular or delusive jammers, but they alternate between active and idle to conserve power.

Wireless networks have the nature of being open and shared which makes them vulnerable to many kinds of attacks. A Generative Adversarial Network(GAN) uses a receiver and a transmitter pair to learn the pattern of another legitimate transmitter receiver pair and fool the legitimate receiver into thinking the malicious transmitter is legitimate. In [32], the authors discuss spoofing attacks using GANs. When GANs are used to generate the signals for a spoofing attack the probability of success increases compared to random or replayed signals. The probability of success can be increased by using multiple antennas to attack. This architecture is similar to fuzz testing; the adversary interacts with and measures the response of the system under test. More specifically, a GAN synthesizes data samples similar to real data samples. This contrasts with fuzz testing, which produces inputs that can differ greatly from a real sample. GANs and fuzzing can be used together to test a network protocol more efficiently. The authors of [33] use a GAN to learn the protocol grammar. The results from the GAN are then used to help generate better fuzzed inputs. GAN and fuzzing used together in this case helped produce better test cases for the researchers.

Chapter 3

Building a Representative 5G Testing System

The desirable characteristics of a 5G testing system are an end-to-end standalone simulation and emulation of a 5G network and compatibility with various testing equipment. The constraints and challenges of building a 5G testing system are a lack of available software and cost of 5G equipment.

3.1 Desirable Characteristics

When testing in a 5G environment there are many factors that a tester would desire depending on their goals. In a perfect environment, a fully emulated end-to-end standalone 5G system would be available. Such technology is not readily available at this time, but certain aspects can be highlighted for future testing purposes. The first desirable aspect would be for the system to have an end-to-end connection between the 5G UE, gNB, and core network. To achieve a full end-to-end system, a stand alone system would be desirable. A testing environment with 5G stand alone is desired because the full capabilities of a 5G network require 5G SA mode. A non-stand alone system can also be used to to achieve an end-to-end system, but it comes with its own limitations which were discussed in Chapter 2.

Another desirable characteristic would be the ability to use hardware like USRPs to connect with the simulated core network. The ability to use a USRP and connect it to a COTS 5G capable UE would be useful for testing the real world applicability of any testing that needs to be done. Emulating the UE has benefits such as lowered cost and not needing to work with real wireless signals, but it is just an imitation of a commercial deployment.

3.2 Constraints and Challenges

The main constraint in designing a 5G system is the lack of available software that can support it. 5G COTS UEs started becoming available in the middle of 2020, but due to the destructive goals of this experiment, there were risks that had to be mitigated. The experiment could not be conducted on a 5G phone or on a cellular provider's network as it would go against FCC guidelines. The design instead used software simulations of the wireless network to perform experiments. When this experiment started, srsRAN had not released the 5G side of their software, so tests could not be run in a virtual 5G environment. srsRAN has released a non-standalone version of the UE and eNB by the time this paper has been written, so future tests can be run in 5G NSA mode and even SA mode once srsRAN releases it.

Another constraint is the cost of 5G hardware, which was a reason the virtual route was taken for testing. Testing a network with 5G hardware can be prohibitively expensive, but 5G hardware can have high capability and mimic a real radio environment better than other testing environments. Physical environments can be simulated and emulated in a hybrid test where a virtual eNB and EPC can be run on a computer with srsRAN. srsRAN is linked to a USRP which can connect to another machine running a UE with srsRAN or a COTS UE with 5G capability. This setup comes with some advantages and disadvantages compared to the virtual implementation. Physical environments can provide more accurate testing when compared to an actual network, but the equipment needed can still cost thousands of dollars and is more difficult to set up. Also for the purpose of this experiment, the UE and eNB connected tens of thousands of times which would take significantly longer in a physical realization. The third and final option is a fully virtual route. This option has the lowest cost and the easiest setup, but is the furthest from a real-world environment. A virtual environment does mimic an actual wireless network, but a simulation cannot fully recreate a commercial deployment. These three environments are shown in Figure 3.1.

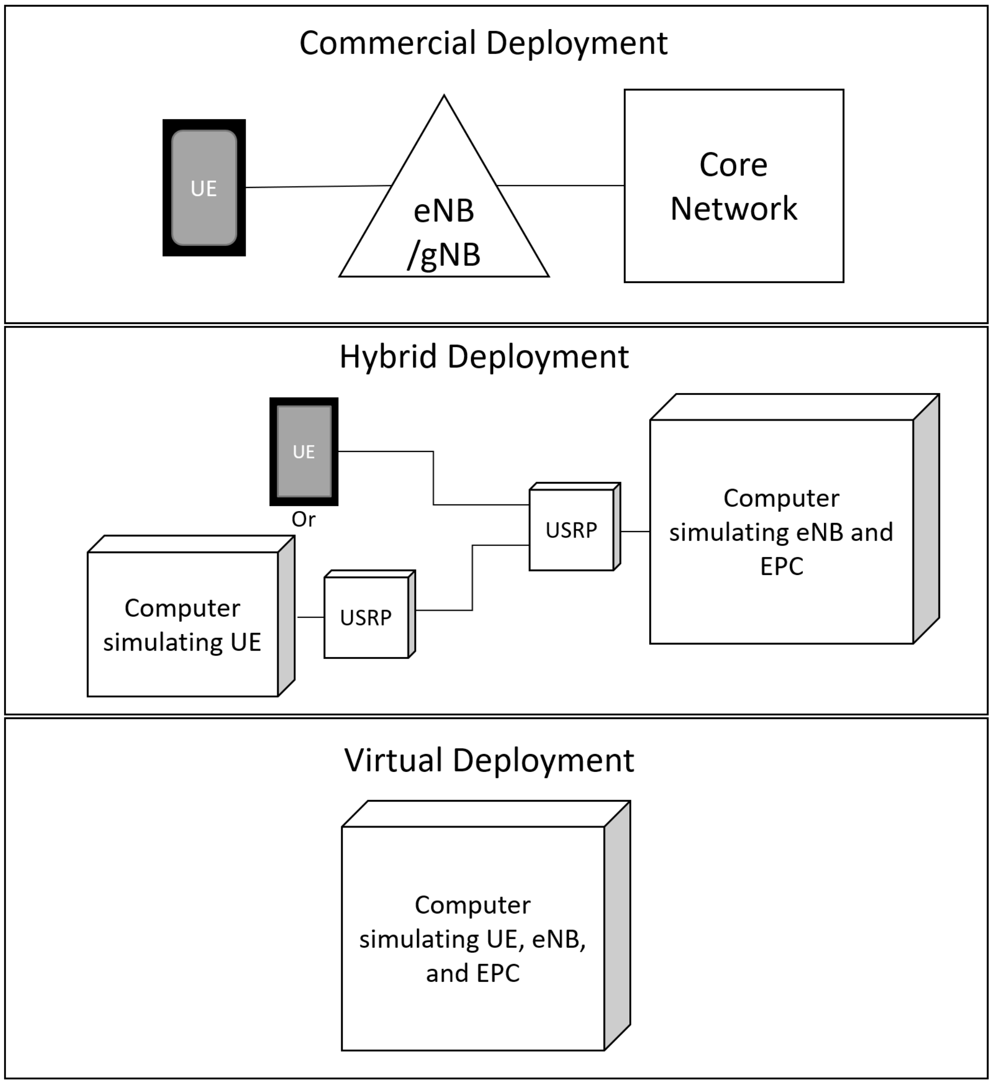

Figure 3.1: Diagram of Different Testing Environments[1]

3.3 Architectural Considerations

An aspect to consider when testing a 5G network is the end-to-end connectivity needed for testing purposes. Testing on a single 5G capable UE or 5G gNB can be done, but testing each component individually could produce different results than when tested together. In addition, the tests performed on an isolated UE or gNB would not have the same communications, so testing the attach procedure would not be possible. The ideal fuzz testing architecture would be implemented inside a 5G network. It is the next big innovation in wireless networks because of its capabilities.

Chapter 4

Fuzz Testing Architecture Description

4.1 High Level Description

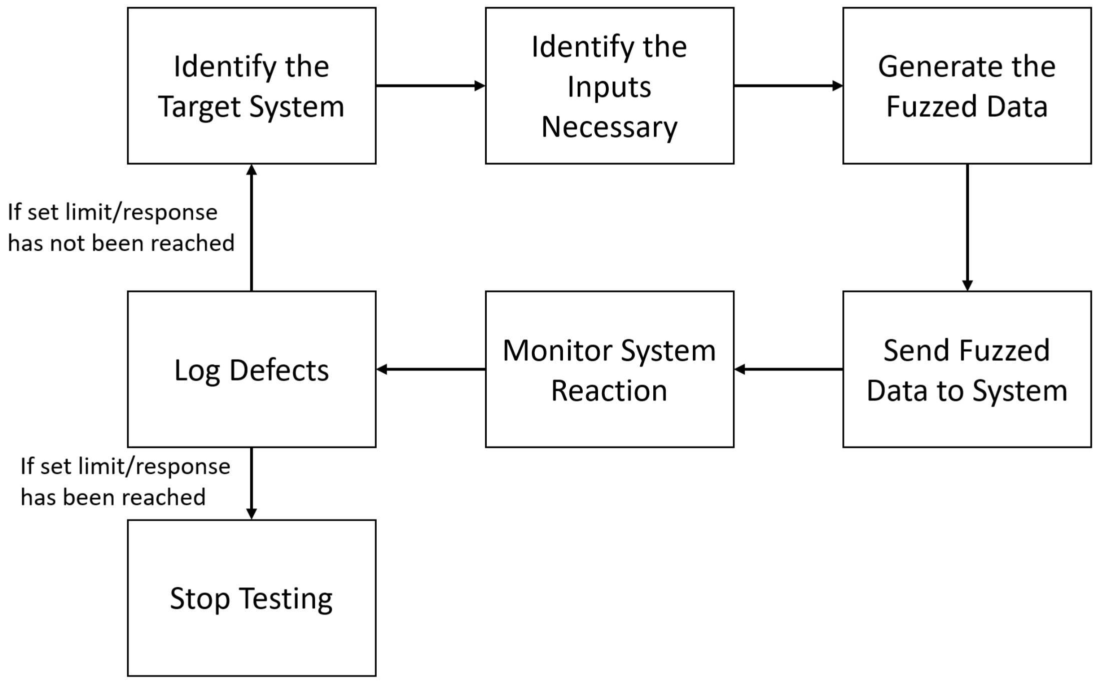
Identify the Target System
Identify the Inputs Necessary
Generate the Fuzzed Data
If set limit/response has not been reached
Log Defects
Monitor System Reaction
Send Fuzzed Data to System
If set limit/response has been reached
Stop Testing

Figure 4.1: Fuzz Testing Block Diagram[1]

At a high level, fuzz testing follows the circular pattern shown in Figure 4.1. The target system is identified and examined in order to accomplish the next step, identifying the inputs necessary. Once the inputs are identified, the fuzzed data is generated. This can be accomplished using one of the previously mentioned fuzz data generation techniques: generation or mutation. After the fuzz data is generated, it is sent to the system. The response from the system is monitored and logged, then the whole process is repeated. In actual implementations the order of the steps can be rearranged to accommodate characteristics of the system under test. For example, if the system has strict timing constraints the fuzzed data can be generated all at once, then sent one at a time.

The fuzz testing architecture used in this thesis follows this high level approach with minor changes. First, the wireless attach procedure in 4G and 5G is selected as the target system. Second, the RRC Connection Request is selected as the target input for fuzzing because it is early in the attach procedure. The RRC Connection Request's early occurrence in the attach procedure means that the time required for each run is reduced, and the RRC Connection Request is before the encryption is activated. All of the fuzzed data is generated before it is introduced to the system because of timing constraints. Once the fuzzed data has been generated, it is introduced into the system using modified wireless simulation software. The system is monitored using packet capture software, and logs are generated using the packet capture software. The logs of the wireless software are then examined to see the effects of the fuzz testing.

4.2 Ideal Fuzz Testing Architecture

The ideal fuzz testing architecture follows the same high level approach, but with improvements at various steps of the process. Figure 4.2 shows the ideal fuzz testing block diagram. Intelligently selecting fuzzing inputs can increase the impact of fuzz testing. Intelligent selection of the fuzz testing parameters can be done by studying the system and using analytic techniques such as RATM to measure the impact of changing the input. Generating the

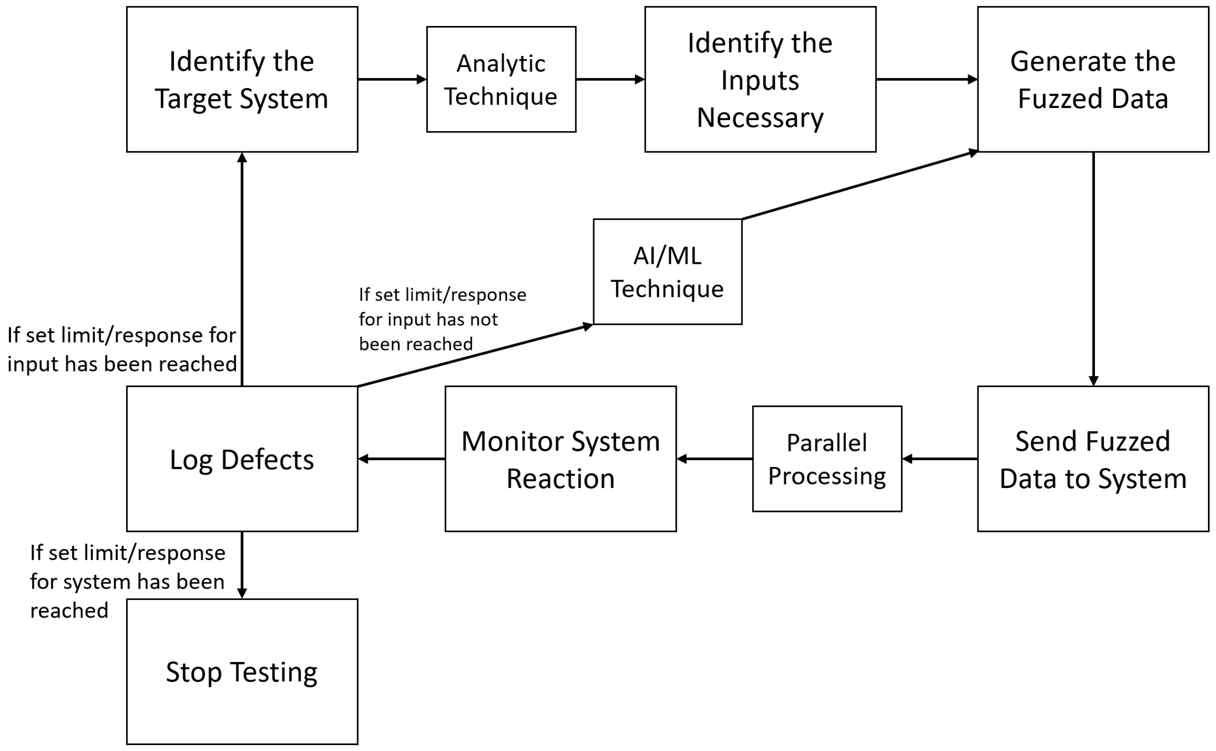
Identify the Target System
Analytic Technique
Identify the Inputs Necessary
Generate the Fuzzed Data
AI/ML Technique
If set limit/response for input has been reached
If set limit/response for input has not been reached
Log Defects
Monitor System Reaction
Parallel Processing
Send Fuzzed Data to System
If set limit/response for system has been reached
Stop Testing

Figure 4.2: Ideal Fuzz Testing Block Diagram[1]

fuzzed inputs can be improved by using machine learning to refine the fuzzed data being produced. The speed of the system can be improved using a fast time scale and parallel processing. A fast time scale can be used to reduce the amount of time for a simulated wireless network to complete a testing run. Parallel processing can be used to have runs executing simultaneously. With parallel processing, the fuzz data generated would need to be divided among the different processes to avoid overlap in the fuzzed inputs sent to the system.

In reality each of these implementations has disadvantages that limit their usefulness. Intelligent selection of fuzz testing parameters takes time to accomplish. Getting machine learning to give back useful changes to the fuzz testing input also takes time to optimise. Fast time scale and parallel processing require more computing power to accomplish, which increases the cost of implementing the fuzz testing.

Chapter 5

Fuzz Testing Architecture Implementation

5.1 Architecture Description

The current system is an amalgamation of a variety of software, each with a different purpose and use. srsRAN is the base software used for this project and is an LTE simulation software [34]. srsRAN simulates the UE, eNB, and EPC each in a different instance. This set up can be used on multiple computers with USRPs with one computer running srsUE while the other runs srseNB and srsEPC. srsRAN can also be run on a single computer using zeromq drivers to emulate the network. The setup that was used for a majority of the testing used zeromq so that tests could be run on a single computer. This version of srsRAN has been edited to allow resetting the connection for quick testing. A stop has been put in the source code for srsUE so that when srsUE gets to the message that we want to fuzz, the running of the program stops so that our fuzzed input can be injected. The system then carries on and generates a pcap file when srsUE disconnects from srsENB. This pcap file can then be analyzed to learn how that particular input affected the system.

For the first incarnation of the test, the RRC connection request was chosen as the target for the fuzz testing. The RRC connection messages are early enough in the attach procedure that security mode commands are not initiated yet or completed yet which leaves them more open to tampering. The early nature of the RRC connection request in the protocol exchange also means that testing is faster because not as many messages need to be completed to get to the target fuzzed message. This time to insertion of the fuzzed message would be negligible in a single trial, but due to the repetitive nature of fuzz testing with thousands of tests executed the time expended is very long.

The fuzzing software used in the system is Boofuzz [13]. Boofuzz is a fuzzing software which was created as a successor to Sulley, another fuzzing framework, after it had fallen out of maintenance. Boofuzz is a dumb, greybox, mutation-based fuzzer. This means it gets input that it knows nothing about, changes it, and injects it into the system targeted for fuzz testing. Boofuzz uses Python code for implementation. In our architecture Boofuzz is given an initial value that could be used by srsUE. Boofuzz then modifies and sends the value numerous times.

This combination of software is used in the VT CORNET [35] servers, running on multiple virtual machines running in parallel so that many tests can be run simultaneously. To analyze the resulting pcap files from srsRAN, different tools have been used. Originally the pcap files were analyzed using Wireshark packet capture software, but with the volume of data that needs analyzing manually, looking at the data was not efficient. Instead, Pyshark was used to convert the pcap files to json files to analyze them.

The design of the implemented fuzzing architecture is summarized by the following steps with Figure 5.1.

1. Boofuzz is used to generate a number of byte sequences whose elements correspond to the values of the fields specific to the network message to be fuzzed. Each individual sequence generated serves as one fuzz test case. The number of test cases is determined by the number of fields specific to the message of interest.

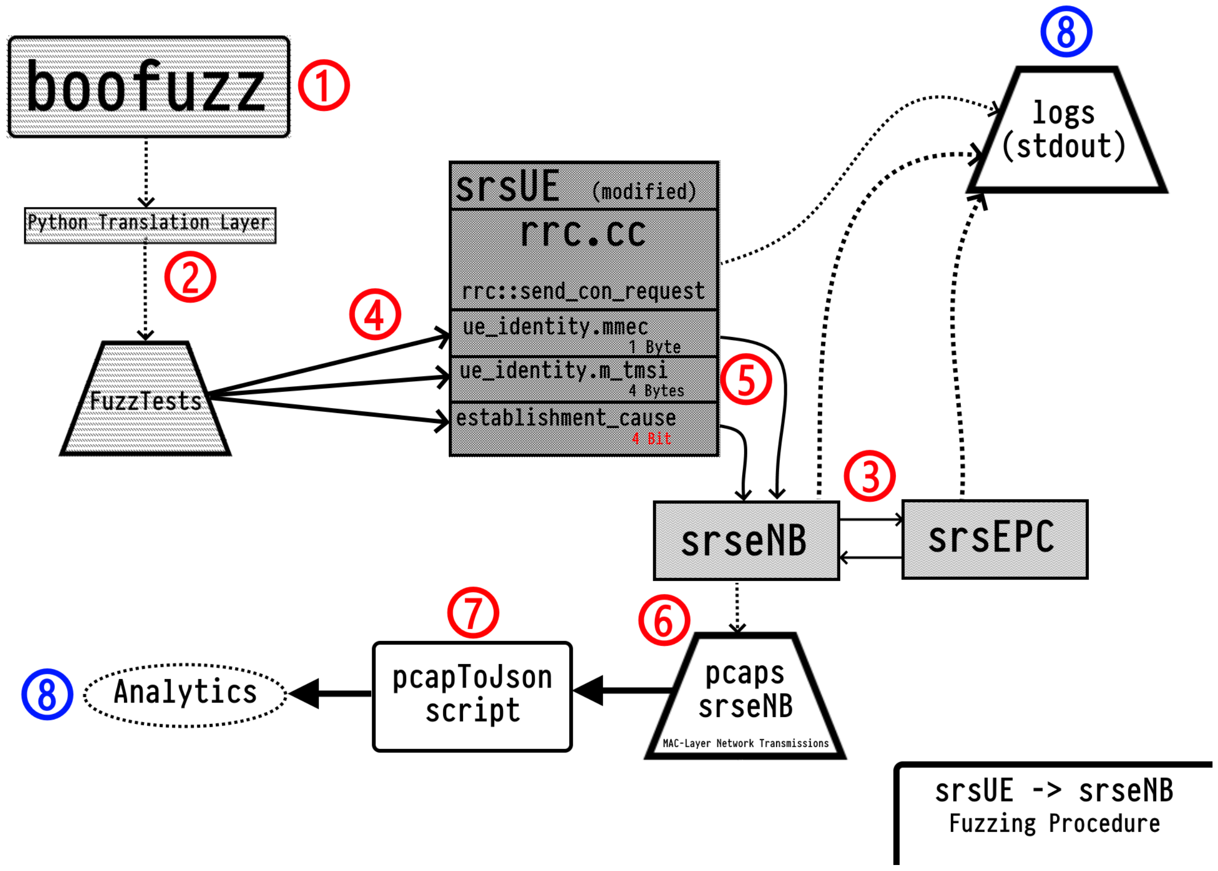

Figure 5.1: Diagram of RRC connection fuzzing setup

2. Python code external to Boofuzz is then run to scrape the byte sequences generated from step 1 and concatenates them all to a 'test-case list' file.

3. The srsRAN executables: *srsEPC* which serves as the LTE core network and *srsENB*, which serves as the LTE eNodeB, are initiated, and their respective outputs are sent to a logfile.

4. The *srsUE* executable, which serves as the UE transmitting the fuzzed message, is then initiated. Here, the srsUE executable has been modified to replace the field values in the message of interest with the fuzzed values of the current test case. The new fuzzed message is then sent from UE to eNB.

5. The srsRAN stack is then taken down upon completion, and srsENB saves MAC-Layer network transmissions to a pcap file upon termination.

6. Steps 3 to 5 are repeated 100 times for every fuzz test case, and the pcap and the log files are all saved to separate folders. The repetition is done for probability analysis of eNB reponse outputs found in the pcap files.

7. The pcap files generated by srsENB are fed through a python script utilizing ‘tshark’ to generate ‘plaintext’ json data.

8. Analytics of generated pcaps are performed to determine results by comparing the outputs of the fuzzed data to the outputs of non-fuzzed data.

5.2 Advantages and Features

A major advantage to this setup is the fact that all of the software used in the design is open-source. This reduces the cost of the experiment since no licenses or software had to be purchased to use the software. In addition this allows recreation of this experiment process for other researchers or for verification of the results of this experiment. The open source nature of srsRAN was an important attribute of our implementation. The open-source nature of srsRAN was a critical aspect because it allowed us to introduce fuzzed data by editing the srsRAN code.

Another advantage to our architecture is the ease of implementation. Once the sections of code that need to be edited are found implementing the fuzz testing is simple. A snippet of the default srsUE code is shown in A.1 while the modified srsUE code is shown in A.2. Both of these code snippets show the ue_identity implementation in srsUE, which is the target of the fuzz testing. The code did not require much change. The default values are assigned new values that were generated by Boofuzz. B shows the Boofuzz code that can be used to fuzz the RRC Connection message. The code goes through the message sections down to the ue-Identity level where it signifies the Mobility Management Entity Code (MMEC), MME Temporary Mobile Subscriber Identity (M-TMSI), and establishmentCause as fuzzing targets. The ue-Identity is included in the RRC Connection Request to facilitate contention resolution. The mmec and m-tmsi comprise the ue-Identity. The MMEC and M-TMSI are used to identify a unique Mobility Management Entity(MME) within a group of MME. The establishmentCause is a value that describes one of 15 establishment cause values in the srsRAN code. Adding the fuzz test to the srsRAN code is uncomplicated with this implementation.

5.3 Downsides

The most apparent downside to this setup is the computing power or time required to run the tests. The first setup that was implemented was on a commercial laptop PC. This PC had a third generation Intel i7 CPU, an AMD FirePro M4000 Mobility Pro GPU, and 32GB of RAM. This computer took a month to complete 250,000 runs while the VT CORNET setup took a day to do the same amount. The VT CORNET server used to run the fuzz tests had an Intel Xeon E5-2687W CPU, and 128 GB of RAM with each VM using 2 CPU cores and 4GB of RAM. A significant amount of computing power is required. Multiple commercial computers cuts down on the time required, but that increases the testing cost. Cloud computing was considered, but then discarded because it increased the difficulty of implementation.

Another limitation of the implementation is the dependency on srsRAN rather than another wireless simulation software. While srsRAN is a well functioning software, it also has limitations. It could be possible that other RAN and core network simulation could be used in its place, but whether it will work the same as this implementation is not known. Furthermore, the software execution needs to stop for the fuzzed data to be introduced to the system. This increases the time it takes for each run, which adds a great deal to the overall time if numerous tests are conducted, and distorts the response time of the system. The stop put in the code was necessary to change the fuzzed values, but there may be other more efficient methods of introducing the fuzzed data. This approach that predefined the fuzzed states does make it awkward to find sequences of fuzzed values that could result in security vulnerabilities.

Chapter 6

Results

As mentioned earlier, the goal of this thesis is to show a proper fuzzing architecture that indicates the presence of security vulnerabilities. Security vulnerabilities can take the form of any number of negative responses from the system, such as a crash or a memory leak. Although nothing as significant or obvious as a crash was achieved with this fuzz testing framework, other more subtle effects that provide clues to potential vulnerabilities on the system were found. These Identified effects could be used to help identify combinations of sequences that may lead to serious issues. However, examining such sequence combinations is computationally infeasible for our resources.

To obtain these results, 250,000 fuzzed inputs were generated. Each of these inputs is considered a test case. Each test case was run 100 times for a total of 25,000,000 runs. The histograms presented in this chapter show 22 of the 100 runs from test case #40200 compared to 22 runs without fuzz testing implemented.

The eNB response values obtained from the pcap files were placed into the json files. There are over 1300 eNB response fields that were extracted and from those, 42 were considered to be of interest. These values are shown in Table 6.1 and were selected for specific reasons. First, many of the fields extracted were categorical rather than numerical. A categorical field has a word as its output rather than a number. For example, the rlc_lte_context_12 field predominantly contains the value ”Context.” The issue with categorical fields is that manually examining the significance of hundreds of fields, each with thousands of values, is inefficient. The effects of fuzzing can be determined much more simply using numeric fields. Numerical values can be statistically quantified and displayed. Another factor for choosing the fields was noticeable deviation from the benchmark. The benchmark was obtained by running the experiment without implementing fuzz testing. Many of the fields' values were static values that never changed from the benchmark by the fuzzed tests. These selection metrics resulted in 42 fields potentially showing significant effects from the fuzz testing.

Table 6.1: Outputs of Interest

eNB Response Field Function	**eNB Response Field Name**	**Packet Number**
Code for Identifying Home Network	e212 mnc	3
Length of GSM Signal	gsm a len	3, 8
RRC Connection Setup Message	lte rrc c1	12, 14, 15
RRC Establishment Cause	lte rrc establishmentCause	1
UE Identity	lte rrc ue Identity	1
Msg3 sent during RAP for contention resolution	mac lte control ue contention resolution msg3	2
Length of frame	mac lte length	5, 8
Padding length	mac lte padding length	3, 8, 15
Synchronization Channel length	mac lte sch length	3, 8, 11
System frame number	mac lte sfn	2
Subframe number	mac lte subframe	0, 1, 4, 5, 7, 8, 10, 11, 14, 16
NAS Key set identifier	nas eps emm nas key set id	3
EPS Identity has odd or even numbered digits	nas eps emm odd even	3
Message type for EMM	nas eps nas msg emm type	6, 8
Encryption and Integrity protection used	nas eps security header type	3, 9
Length of octet string value	per octet string length	3, 5, 8, 9
Length of RLC PDU	rlc pdu length	3, 5. 11

These 42 fields were then observed in arbitrarily chosen test cases to observe any effects that the fuzz testing had. One test case is described below to showcase the results of the fuzz testing. These effects are shown and discussed in the next section.

6.1 Histograms

The output of three fields are shown here to demonstrate the effects of fuzz testing. Figures 6.1, 6.3, and 6.4 each show outputs from the fuzzing test case #40200 compared to the non-fuzzed default case.

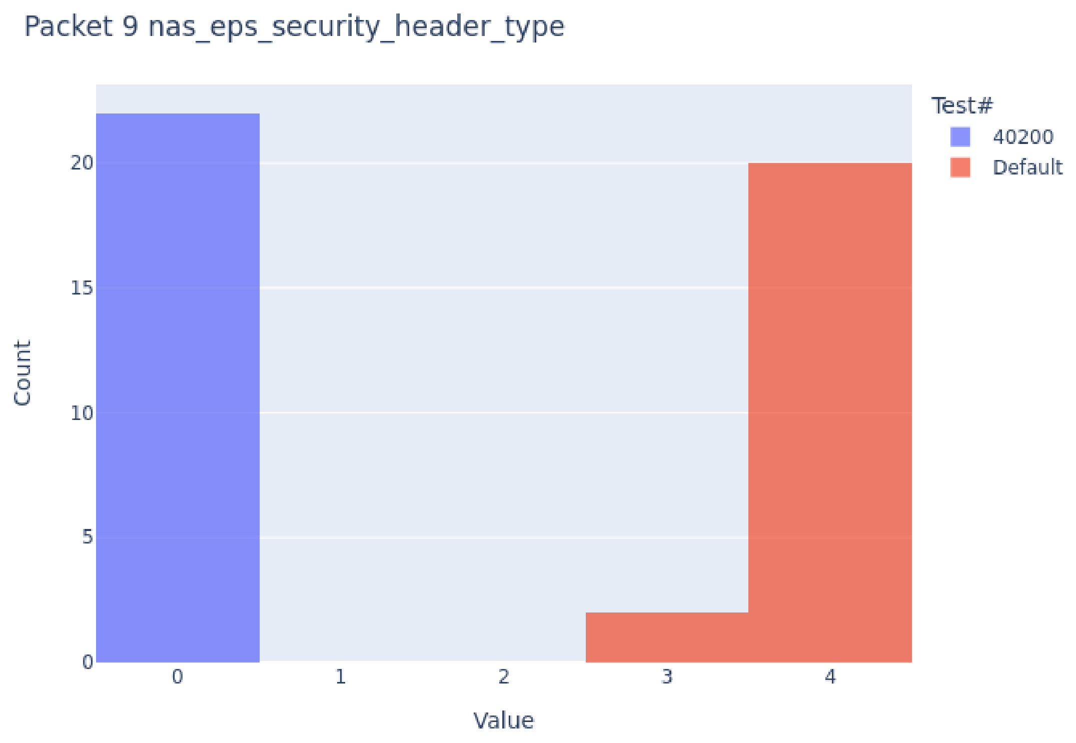

Figure 6.1: Histogram of the security header type

For example, Figure 6.1 shows a histogram of the fuzz case #40200 compared to the non-fuzzed default case for the nas_eps_security_header_type value of packet 9. The fuzzing appears to have affected the nas_eps_security_header_type with the fuzzed tests all outputting a value of 0 while the default case outputs a value of 3 or 4. From Table 6.2 from the 3GPP standard 24.301, a value of 0 for the nas_eps_security_header_type means that there is no encryption on what is being sent. A value of 3 means that the message has integrity protection and a value of 4 means that the message has both integrity protection and ciphering. It is suspected the encryption did not to appear in this message because the fuzz testing delayed an earlier message from being sent. Figure 6.2 shows a theoretical example. This delay of encryption can be utilized to extend the time frame available to send harmful messages to the system.

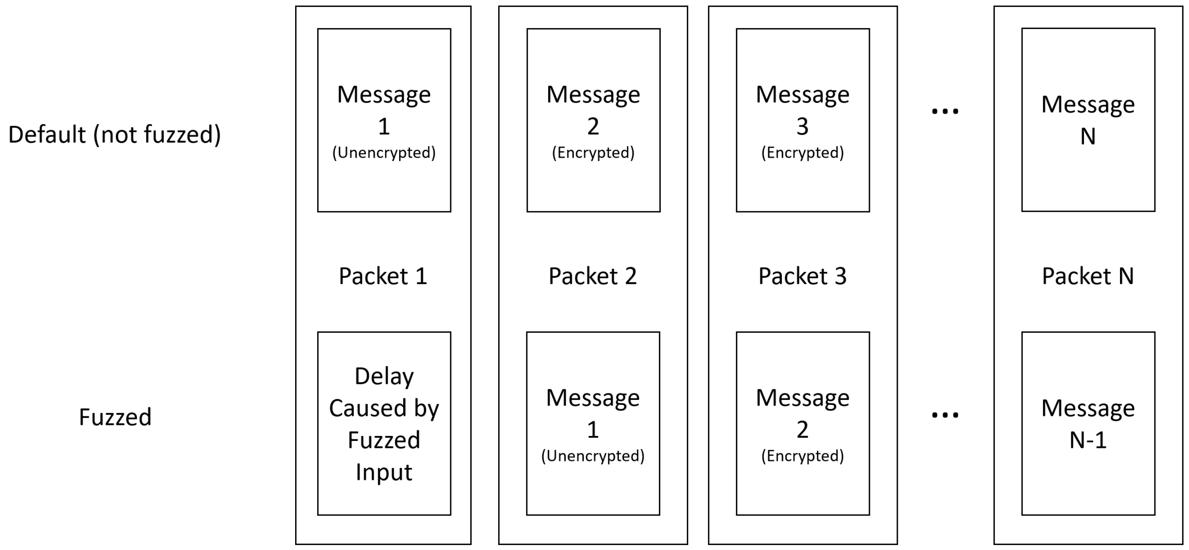

Figure 6.2: Diagram of Delay Caused by Fuzz Testing [1]

Table 6.2: Security header type [2]

Bits(Binary)	Bits(Decimal)	Security Protocol
0 0 0 0	0	Plain NAS, not security protected
0 0 0 1	1	Integrity protected
0 0 1 0	2	Integrity protected and ciphered
0 0 1 1	3	Integrity protected with new EPS security context
0 1 0 0	4	Integrity protected and ciphered with new EPS security context
1 1 0 0	12	Security header for SERVICE REQUEST message

Figure 6.3 shows the histogram of the fuzz case #40200 and the default case of the Message

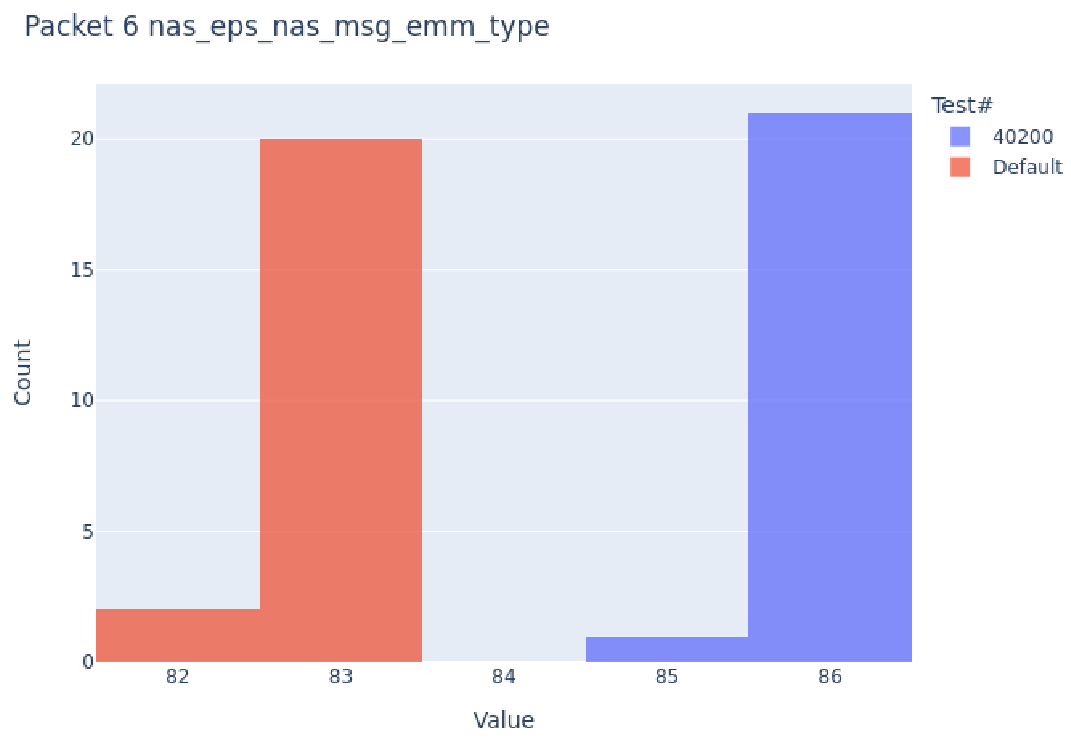

Figure 6.3: Histogram of the Message type for EPS Mobility Management

type for the EPS Mobility Management in the 6th packet. In the default case this is shown to be a value of 83 with a few outputs of 82. The fuzzed output shows a value of 86 for most cases and a value of 85 for a couple. The meaning of these values can be understood by referencing Table 6.3 from 3GPP standard 24.301 [2]. The value of the fuzzed output is consistently an identity response or request message as compared to an authentication response or request that occurs for the default case. Similar to what is shown in Figure 6.2, the message being sent is being delayed. In the non-fuzzed case the identity response or request is being sent in packet 6, but with the fuzz testing the authentication response or request is sent in packet 6. In the wireless attach procedure, the identity response and request are sent during the random access procedure, and the authentication response and request are sent during the DL and UL Info Transfer messages. This suggests that the fuzzing caused an earlier message to get delayed and that effect is propagating to this packet being sent. The fuzzed RRC Connection Request is only sent once during the attach procedure, and the system would discard a repeated one. This prevents repeating the message to further propagate the delay would not be possible. However, the Random Access Preamble, RRC Connection Complete, or another pre-Security Mode message can be fuzzed to attempt to create a delay. If other fuzzed messages are found to produce a delay, messages could be sent together to compound the delay in packets being sent. With enough delays to the system, a crash could be created due to the strict limitations of wireless systems.

Table 6.3: Message types for EPS mobility management [2]

Bits(Binary)	**Bits(Decimal)**	**EPS Mobility Management Messages**
0 1 0 1 0 0 0 0	80	GUTI reallocation command
0 1 0 1 0 0 0 1	81	GUTI reallocation complete
0 1 0 1 0 0 1 0	82	Authentication request
0 1 0 1 0 0 1 1	83	Authentication response
0 1 0 1 0 1 0 0	84	Authentication reject
0 1 0 1 1 1 0 0	92	Authentication failure
0 1 0 1 0 1 0 1	85	Identity request
0 1 0 1 0 1 1 0	86	Identity response
0 1 0 1 1 1 0 1	93	Security mode command
0 1 0 1 1 1 1 0	94	Security mode complete
0 1 0 1 1 1 1 1	95	Security mode reject

Figure 6.4 shows the value of subframe 0 for the fuzzed and non-fuzzed case. This shows that the fuzz testing caused the value of subframe 0 to stay the same throughout the runs. Most of the outputs of the default case have a value of 6 with a few having a value of 8 or 0. Concurrently, case #40200 outputs a value of 6 every time. Fuzz testing did not change the median value and it kept the output consistent. This is shown as a comparison to the previous histograms as separate effect of the fuzz testing. The other histograms show a clear separation of the outputs for the fuzzed test and the default while this one shows an overlap in the outputs. The darker red in Figure 6.4 shows the overlap of test case #40200 and the

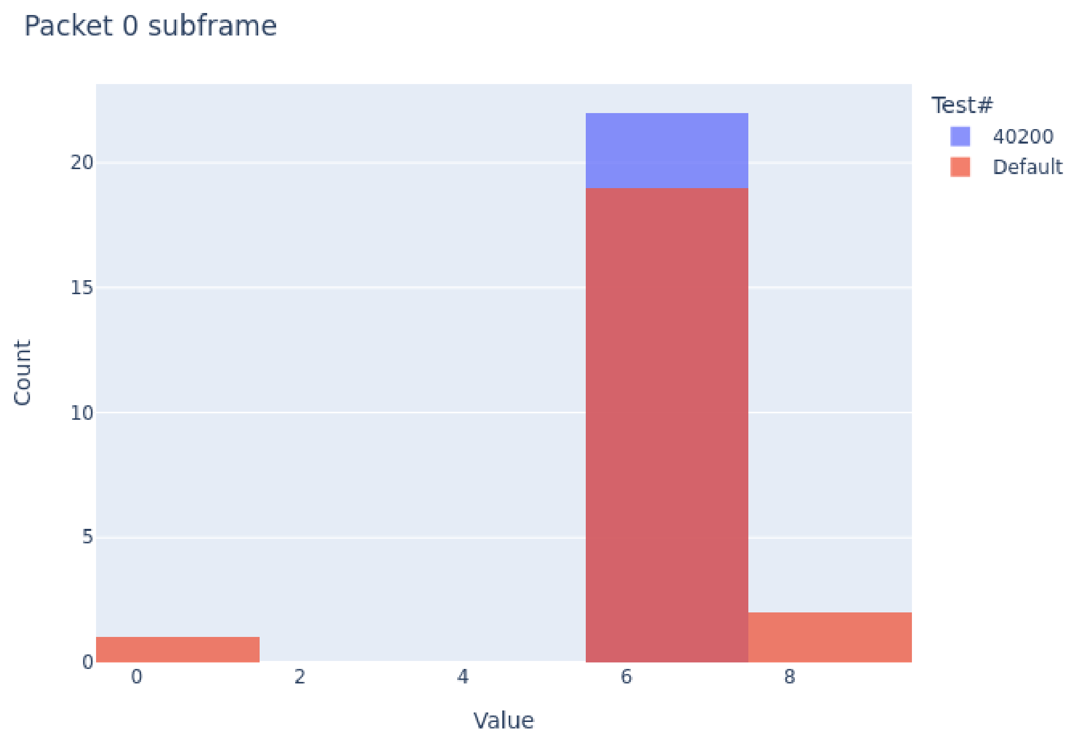

Figure 6.4: Histogram of Subframe value of Packet 0

non-fuzzed test case while the lighter red is the non-fuzzed test case.

Chapter 7

Summary

A novel fuzz testing architecture is presented in this thesis. The architecture was designed for use with 5G systems to test for vulnerabilities. The architecture is the process of using a fuzzer within a wireless network to test for vulnerabilities. The architecture uses open-source software for its implementation. The state of fuzz testing and similar efforts in this field were discussed. Different fuzzing tactics are shown to be effective in different environments. Due to the state of 5G software and the cost of 5G hardware, the architecture had to be implemented in a 4G environment. The 4G implementation of the fuzz testing was designed to be adaptable to 5G after simulation software for 5G network components becomes available. The 4G implementation used Boofuzz to generate fuzzed data, srsRAN to simulate the 4G network, and Wireshark to enable analysis. A VT CORNET server utilized for wireless testing was used while implementing the fuzz testing architecture with the mentioned software. The 4G fuzz testing implementation uncovered security risks. The test case that was presented shows a message without encryption being sent by the eNB. This lack of encryption is suspected to appear because of a delay in messages. The delay causes a message without encryption to be sent when a separate message with encryption would normally be sent. This shows that the fuzz testing causes a delay in the messages sent to the system.

7.1 Conclusion

The conducted experiments successfully showed the effectiveness of fuzz testing using the proposed architecture produced significant results. The fuzz testing was able to uncover unencrypted messages being sent by the eNB. It also showed consistent impact on the responses sent by the eNB in separate messages. Moreover, the fuzz testing caused messages to be sent later than they were sent without fuzz testing. These results are significant because they prove the effectiveness of the fuzz testing architecture. Our proposed architecture can be used in future research to help find vulnerabilities in wireless networks. While none of the effects observed caused the system to crash, they provide clues to possible issues in the software or protocol standard.

7.2 Future Work

This research can be expanded on once 5G simulation software becomes available. 5G simulation software would allow the ideal architecture to be implemented. This implementation could be used to test 5G networks and the 5G simulation software once they become available. Another improvement that can be implemented is the use of AI and Machine Learning techniques with the fuzzing process. AI can be used to refine the mutations made by a fuzzer by measuring the response of the system. The response's reaction to the refined mutations can then be used by the AI ad infinitum to generate more effective fuzzed data. The fuzz testing architecture could also be used in conjunction with known wireless attacks to test its effectiveness. Fuzz testing has been proven to be effective with other vulnerability testing methods such as GANs; combining fuzz testing with more testing methods could have a synergistic effect for finding previously undiscovered vulnerabilities.

Bibliography

[1] Stephen Mayhew. Created by Author.

[2] 3GPP. TS 24.301 Non-Access-Stratum (NAS) protocol for Evolved Packet System (EPS); Stage 3, . URL https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=1072.

[3] David Rupprecht, Katharina Kohls, Thorsten Holz, and Christina Popper. Breaking LTE on Layer Two. In *2019 IEEE Symposium on Security and Privacy (SP)*, pages 1121–1136. IEEE, 5 2019. ISBN 978-1-5386-6660-9. doi: 10.1109/SP.2019.00006.

[4] Patrice Godefroid. Fuzzing. *Communications of the ACM*, 63(2):70–76, 1 2020. ISSN 0001-0782. doi: 10.1145/3363824.

[5] Patrice Godefroid, Hila Peleg, and Rishabh Singh. Learn&Fuzz: Machine Learning for Input Fuzzing. 2017.

[6] Serge Gorbunov and Arnold Rosenbloom. Autofuzz: Automated network protocol fuzzing framework. *IJCSNS*, 10(8), 2010.

[7] Jun Li, Bodong Zhao, and Chao Zhang. Fuzzing: a survey. *Cybersecurity*, 1(1):6, 12 2018. ISSN 2523-3246. doi: 10.1186/s42400-018-0002-y.

[8] Anna Pestrea. *Fuzz testing on eNodeB over the air interface.* PhD thesis, 2021.

[9] George Klees, Andrew Ruef, Benji Cooper, Shiyi Wei, and Michael Hicks. Evaluating fuzz testing. In *Proceedings of the ACM Conference on Computer and Communications Security*, pages 2123–2138. Association for Computing Machinery, 10 2018. ISBN 9781450356930. doi: 10.1145/3243734.3243804.

[10] Jie Liang, Mingzhe Wang, Yuanliang Chen, Yu Jiang, and Renwei Zhang. Fuzz testing in practice: Obstacles and solutions. In *2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pages 562–566. IEEE, 3 2018. ISBN 978-1-5386-4969-5. doi: 10.1109/SANER.2018.8330260.

[11] Shereef Sayed, Jeffrey H Reed, Carl Dietrich, and Cameron D Patterson. *Black-Box Fuzzing of the REDHAWK Software Communications Architecture*. PhD thesis, 2015.

[12] Peach. URL https://github.com/MozillaSecurity/peach.

[13] boofuzz. URL https://github.com/jtpereyda/boofuzz.

[14] Hongil Kim, Jiho Lee, Eunkyu Lee, and Yongdae Kim. Touching the Untouchables: Dynamic Security Analysis of the LTE Control Plane. In *Proceedings - IEEE Symposium on Security and Privacy*, volume 2019-May, 2019. doi: 10.1109/SP.2019.00038.

[15] Steven Heijse. *IEEE 802.11 Physical Layer Fuzzing Using OpenWifi*. PhD thesis.

[16] Xing Han, Qiaoyan Wen, and Zhao Zhang. A mutation-based fuzz testing approach for network protocol vulnerability detection. In *Proceedings of 2nd International Conference on Computer Science and Network Technology, ICCSNT 2012*, pages 1018–1022, 2012. ISBN 9781467329644. doi: 10.1109/ICCSNT.2012.6526099.

[17] Michal Zalewski. AFL. URL https://github.com/google/AFL.

[18] Sulley. URL https://github.com/OpenRCE/sulley.

[19] LibFuzzer. URL https://llvm.org/docs/LibFuzzer.html.

[20] Jochem Hoes. *Rise of the Machines On the Security of Cellular IoT Devices*. PhD thesis, 2021.

[21] Srinath Potnuru and Prajwol Kumar Nakarmi. Berserker: ASN.1-based Fuzzing of Radio Resource Control Protocol for 4G and 5G. 7 2021. URL http://arxiv.org/abs/2107.01912.

[22] Gary J Saavedra, Kathryn N Rodhouse, Daniel M Dunlavy, and Philip W Kegelmeyer. A Review of Machine Learning Applications in Fuzzing. 6 2019. URL http://arxiv.org/abs/1906.11133.

[23] 3GPP. TS 36.321 Evolved Universal Terrestrial Radio Access (E-UTRA); Medium Access Control (MAC) protocol specification, . URL https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=2437.

[24] 3GPP. TS 36.322 Evolved Universal Terrestrial Radio Access (E-UTRA); Radio Link Control (RLC) protocol specification, . URL https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=2438.

[25] 3GPP. TS 36.323 Evolved Universal Terrestrial Radio Access (E-UTRA); Packet Data Convergence Protocol (PDCP) specification, . URL https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=2439.

[26] 3GPP. TS 33.512 5G Security Assurance Specification (SCAS); Access and Mobility management Function (AMF), . URL https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=3445.

[27] 3GPP. TS 33.513 5G Security Assurance Specification (SCAS); User Plane Function (UPF), . URL https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=3446.

[28] 3GPP. TS 33.514 5G Security Assurance Specification (SCAS) for the Unified Data Management (UDM) network product class, . URL https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=3447.

[29] 3GPP. TS 33.515 5G Security Assurance Specification (SCAS) for the Session Management Function (SMF) network product class, . URL https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=3448.

[30] 3GPP. TS 33.516 5G Security Assurance Specification (SCAS) for the Authentication Server Function (AUSF) network product class, . URL https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=3535.

[31] Youness Arjoune and Saleh Faruque. Smart Jamming Attacks in 5G New Radio: A Review. Technical report.

[32] Yi Shi, Kemal Davaslioglu, and Yalin E. Sagduyu. Generative Adversarial Network in the Air: Deep Adversarial Learning for Wireless Signal Spoofing. *IEEE Transactions on Cognitive Communications and Networking*, 7(1):294–303, 3 2021. ISSN 23327731. doi: 10.1109/TCCN.2020.3010330.

[33] Zhicheng Hu, Jianqi Shi, YanHong Huang, Jiawen Xiong, and Xiangxing Bu. GANFuzz. In *Proceedings of the 15th ACM International Conference on Computing Frontiers*, pages 138–145, New York, NY, USA, 5 2018. ACM. ISBN 9781450357616. doi: 10.1145/3203217.3203241.

[34] srsRAN. URL https://github.com/srsran/srsRAN.

[35] CORNET. URL https://cornet.wireless.vt.edu/.

Appendices

Appendix A

srsUE Code

A.1 Default srsUE Code

```
        srsran::to_asn1(&rrc_conn_req->ue_id.s_tmsi(), ue_identity); //
            default ue_identity set
                rrc_conn_req->establishment_cause = (establishment_cause_opts
                    ::options)cause; // establishment cause set
                send_ul_ccch_msg(ul_ccch_msg); // message queued
```

A.2 Modified srsUE Code

```
        ue_identity.mmec = (uint8_t)fuzzedMMEC; // set mmec to the bytes
            loaded from file
                ue_identity.m_tmsi = (uint32_t)fuzzedTMSI; // set tmsi to the
                    bytes loaded from file
                rrc_conn_req->establishment_cause = (establishment_cause_opts
                    ::options) fuzzedCause; // set cause to the bits loaded
                    from file
                srsran::to_asn1(&rrc_conn_req->ue_id.s_tmsi(), ue_identity);
                    // modified ue_identity set
                send_ul_ccch_msg(ul_ccch_msg); // message queued
```

Appendix B

Boofuzz Code

```
from boofuzz import *

def main():

    session = Session(
        target=Target(connection=UDPSocketConnection("127.0.0.1", 57005)),
        sleep_time=0)

    req = Request("UL-CCCH-Message",
                  children=(Block("c1", children=(Bytes("buffer", size=6)))))

    req1 = Request("UL-CCCH-Message",
                   children=(Block(
                       "c1",
                       children=(Block(
                           "rrcConnectionRequest",
                           children=(Block(
                               "criticalExtensions",
                               children=(Block(
                                   "rrcConnectionRequest-r8",
                                   children=(Block(
                                       "ue-Indentity",
                                       children=(Block(
                                           "s-TMSI",
                                                            children=(Byte("mmec"),
                                                                      DWord("m-TMSI"))))),
                                                              Byte("establishmentCause",
                                                                   max_num=15)))))))))))

    session.connect(req1)
    session.fuzz()

if __name__ == "__main__":
    main()
```