Linköping University | Department of Computer and Information Science
Master's thesis, 30 ECTS | Computer science
2021 | LIU-IDA/LITH-EX-A--21/020--SE

Fuzz testing on eNodeB over the air interface

– Using fuzz testing as a means of testing security

Fuzz testing på eNodeB genom luftgränssnittet

Anna Pestrea

Supervisor : Felipe Boeira
Examiner : Simin Nadjm-Tehrani

External supervisor : Magnus Öberg

LINKÖPING UNIVERSITY

Linköpings universitet
SE–581 83 Linköping
+46 13 28 10 00 , www.liu.se

Abstract

In modern society, security has become an increasingly important subject, as technology has become an integrated part of everyday life. The security of a system can be tested with the help of fuzzing, where incoming messages to the system are altered. In this thesis, a fuzzer was developed targeting an E-UTRAN Node B (eNB) in the Long-Term Evolution (LTE) landscape. The eNB is current prototype and is from the company Ericsson. The fuzzer is particularly designed for testing the Medium Access Control (MAC) layer of the eNB. The fuzzer uses a genetic method where all of the fuzzer's flags (the R, F2, E, LCID, F and L flags) are triggered during the fuzzing period. Depending on the output of the first generation of fuzzed values, new values are generated either by choosing a value close to the original value, or by choosing a value that belong to the same subgroup as the original value. Four test cases are made, where first test case is the base line of the program and the other three test cases fuzzes the eNB, using different parts of the fuzzer.

The results show that depending on which parts of the fuzzer are used, the connection becomes different. For test two and three, the connection became increasingly unstable and more data was present in the connection. Test case four did not however deviate so much from the baseline, if compared to test two and three.

Acknowledgments

Thanks to Simin Nadjm-Tehrani and Felipe Boeira from Linköping University for undertaking the roles of examiner and supervisor for this thesis. Thanks for Magnus Öberg for acting as corresponding supervisor from the Ericsson company.

Additional thanks to Niklas Johansson for helping with the creation of the thesis opportunity, to Lyle Li for helping with the technical difficulties and to all Ericsson employee who provided their knowledge and experience for this thesis.

Contents

List of Figures

List of Tables

1 Introduction

Although it is easy to realise that a high level of system security is needed, it is harder to know what higher security is and if it is already implemented or not. One way of testing security is through fuzz testing. This type of testing evaluates a system's security by changing the originally valid input, and then study how the system reacts to the altered input. There are several ways to so. One way is to change the length of the input, which might cause the system to crash due to memory overflow. Fuzz testing can also be used to assess the quality of the system, for example by discovering coding errors [15]. If an unintended behaviour is not discovered by fuzz testing and the causes not removed, then a person with malicious intent could take advantage of this. Attackers can, for example, get access to sensitive data about the user or they could infect the user's device with malware. Both of these examples would not only compromise the security of the user, but also other people, as data leaks have a tendency of affecting a large number of users.

This thesis is done in the context of software and base stations owned and produced by the company Ericsson. Ericsson has tools to test the Long-Term Evolution (LTE) protocols on a base station (E-UTRAN Node B) over the cable interface, but does not have a corresponding tool for the air interface. All communication that is transmitted through the air is said to be transmitted through the air interface. The protocols for the air interface are different, compared to the protocols used when communicating through the cable interface. The LTE protocol stack consists of the following protocols for the air interface; the Non Access Stratum (NAS), the Radio Resource Control (RRC), the Packet Data Convergence Protocol (PDCP), the Radio Link Protocol (RLC), the Medium Access Control (MAC) and the physical layer [14, 5].

For this thesis, only one out of the six existing protocols will be of interest, the MAC protocol. However the MAC protocol is connected to and codependent of other protocols, which include RRC, PDCP and RLC. Figure 1.1 shows the protocol stack with the interesting protocol right above the Physical layer, which is at the bottom.

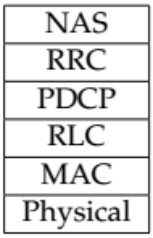


Figure 1.1: An overview of the protocol stack

Some of the functions of the RRC are to establish a connection between the User Equipment (UE) and the E-UTRAN Node B (eNB), broadcast system information and handle the RRC connection previously made. It also receives and sends messages through the PDCP protocol below it. The PDCP provides its services to the upper layer RRC. It is responsible for transferring control and user plane data, header compression, ciphering and integrity protection. The RLC transfers Service Data Unit (SDU) to the upper layers and the Protocol Data Units (PDU) to the layers below. All SDU and PDU have a subheader, which consist of different header fields, also called flags. The flags are indicators about the characteristics of the packet. The MAC is situated between the physical layer and the RLC layer. Data that passes the MAC layer often end up in the RRC, PDCP or the RLC layer. All layers and the LTE components, including the Evolved Packet Core (EPC), are illustrated in Figure 1.1 and 1.2.

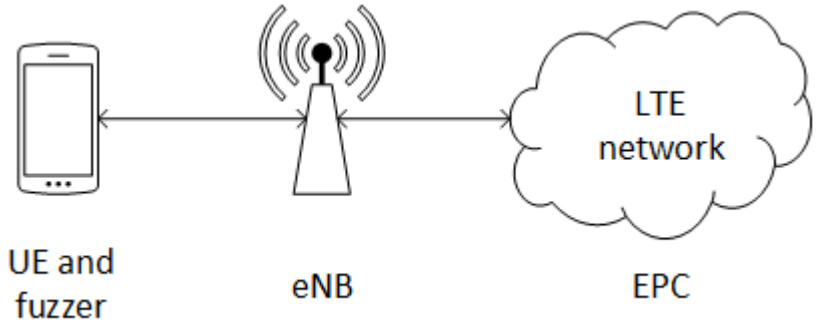


Figure 1.2: An overview of the LTE components

1.1 Motivation

By evaluating the implementations of the protocols using fuzz testing, it is expected that unknown bugs are discovered. This will in turn help the development of the protocols, as the first step to fix a bug is to discover and understand it. Protocols are continuously updated when new features are added or new defects are discovered. Creating a fuzz testing program will enable Ericsson to test the protocols and it gives a foundation for a fuzz tester, that in the future might be further developed.

The four protocols are chosen because these protocols have not been thoroughly fuzz tested before, as previous implementations have only fuzz tested the protocols shallowly, according to Ericsson employees. Therefore chances are higher that these protocols contain hidden bugs or unwanted behaviour. According to the Ericsson employees the MAC protocol, which has the highest priority of the four protocols (see Section 1.3), also has a history of protocol-related problems in the past. Therefore only this protocol is of interest in this thesis.

1.2 Aim

The purpose of this thesis is to investigate how fuzz testing can be deployed towards equipment developed by Ericsson. The fuzz testing is to be directed at the LTE protocol operating at the eNB, a part of the LTE landscape, more specifically at the flags of the MAC protocol. The aim is to understand the benefits of fuzz testing the air interface of LTE and its limitations.

1.3 Research questions

In order to reach the goal of the thesis, the following research questions are answered:

1. Which fuzzing techniques are applicable and suitable to use on the eNB over the air interface?
2. Do these techniques test the eNB in a satisfying way so that all flags are covered by the tests and are able to pass the MAC layer on the eNB side?

1.4 Delimitations

The thesis is limited to the products of Ericsson. Since the equipment is located in another city, the access to the parts needed for the testing part is also limited. Some details of the findings are considered to be sensitive information and is not included in the thesis.

The fuzzer is built inside of the program named srsLTE [20], which creates a virtual LTE network. The network consist of an UE, an eNB and an EPC.

1.5 Overview of the approach

Here an overview of the approach is presented. The fuzzing strategy of the fuzzer, the metrics and the evaluation method of the fuzzer are briefly explained.

1.5.1 Fuzzing strategy

From relevant theory and related work, a fuzzing strategy for the MAC layer is designed. The strategy aims to test all values, per flag, that should cover all possible outcomes. However the strategy does not test all possible combinations of the different flags, or all possible versions of a header. Trying to implement this exhaustive testing would drain too much resources and might not even find any bugs in the end, as there might not even exist any bugs. Therefore a more intelligent fuzzing strategy is developed by including a genetic algorithm in the fuzzing strategy.

1.5.2 Metrics

In order to be able to evaluate the successfulness of the fuzzer, metrics are needed. After studying related works it is clear that the most common metric for evaluating a fuzzer is the number of discovered bugs. This metric is, despite its common use, not a preferable metric for this thesis. If the fuzzer is unable to discover any bugs, information about the successfulness of the fuzzer must still exist. Since the fuzzer is run in a black-box environment, this also affects the testing, as it is not possible to directly observe the result of the fuzzing. Therefore the metrics are affected by this. The following metrics are selected instead:

- The coverage of the flags
- The proportion of packets that are able to reach the RLC layer and above on the eNB side

The metric concerning the proportion of accepted packets tells how many packets are successful in passing the eNB side, which in turn indicates which flag combinations are a good choice when fuzzing.

After discussion with the Ericsson supervisor, the proportion of packets is set to a minimum of 50% of the total number of packets in the connection.

1.5.3 Evaluation

Before the fuzzer is tested against the eNB, a baseline is established. By having a baseline it is possible to make a comparison of how the fuzzer affects the packet flow. By repeating this step the baseline becomes more reliable, as the packet flow sometimes creates one-time occurrences, which in turn could affect the accuracy of the result. The fuzzer is evaluated by comparing the metrics between the baseline and the fuzzing test case. It is also tested to only use parts of the fuzzer, to determine which parts of the fuzzer causes the eNB to react.

2 Background

This chapter contains relevant theory. The chapter presents different fuzzing techniques, what kind of vulnerability types there exist and background information about the MAC layer. A detailed overview of the different MAC packets and their flags is also presented.

2.1 Fuzzing techniques

Fuzz testing is a way of testing a system and can be used for either assessing the quality or testing the security of the system. The testing is done by sending in altered input to a system and study how the system reacts. Fuzz testers can be of different types and can either be categorized very general or very detailed. The more general categorization classifies the fuzzers after how aware they are of the targeted systems structure. This general approach divides fuzzers into three groups: black-box fuzzers, white-box fuzzers and grey-box fuzzers. The more detailed categorization classifies the fuzzers after preprocess, scheduling, input generation, input evaluation and configuration updates. Since some of the categories contains duplicate mentions of the same type of fuzzers, the preprocess, input evaluation and configuration updates-categories will not be presented [15].

2.1.1 Black-box fuzzing

Black-box testing is when the internal structure of the system is unknown. This is also one of the advantages of the black-box testing approach, as it does not need to consider the structure of the system when fuzzing [21].

2.1.2 White-box fuzzing

White-box testing is the opposite of black-box testing. The internal structure is known and the fuzzing technique is focused on reaching one-hundred percent of some kind of coverage (for example code-coverage). The advantage of white-box fuzzing is that it is more likely to expose bugs hidden deeper inside of the code [21].

2.1.3 Grey-box fuzzing

The last type of fuzzing, grey-box fuzzing, is a combination of black-box fuzzing and white-box fuzzing. This makes grey-box fuzzing have all of the advantages of the two other fuzzing techniques [21].

2.1.4 Scheduling

Here fuzzers are categorized after how the configuration is selected for the next iteration. It can be divided into either black-box, white-box or grey-box fuzzing. For the grey-box approach the genetic algorithm, or evolution algorithm, is included [15].

2.1.4.1 Genetic algorithm

A genetic algorithm is an algorithm for selecting and generating new individuals from a group of individuals. The algorithm is based on Charles Darwin's theory of evolution. First the algorithm selects one individual from the original group and tests if it is "fit". If it passes the test, the algorithm marks it and selects the next individual. This process is then repeated until only the fittest individuals are left, which is also the advantage of this fuzzing technique. After the selection is finished the algorithm generates new individuals by taking the characteristics of the fittest individuals into consideration. In this context the term "individuals" refers to the value of the MAC header [22] [9].

2.1.5 Input Generation

If fuzzers are to be categorized after how the input is generated, it can be divided into either mutation- or generation-based fuzzing.

2.1.5.1 Mutation-based fuzzing

For mutation-based fuzzing the input derives from valid input that is mutated and fed into the system. A few examples of mutating methods are substitution of data, addition of data, removal of data and bit flipping. One advantage of this fuzzing technique is that little needs to be known about the input format, unlike generation-based fuzzing.

2.1.5.2 Generation-based fuzzing

Generation-based fuzzing considers the format of the input and generates semi-valid input with this in mind. This is also one of the advantages of the fuzzing technique, as consideration of input format gives input closer to the expected input. Unlike mutation fuzzing the generation fuzzing technique generates input from scratch [21] [10].

2.2 Security attacks

This section contains theory about different security attacks. The reason these attacks are chosen is that the likelihood of these being successful is higher.

2.2.1 Denial-of-Service

Denial of Service is an attack that makes a service become unavailable due to a vulnerability in the system or program. The users that are affected by this attack are legitimate users [7].

2.2.2 Information disclosure

This attack tricks the system or program to reveal information that normally would not have been disclosed, to the attacker [7].

2.2.3 Memory corruption

This attack makes the memory of the system or program become corrupted in some way. It can simply be that the wrong value is stored or it can for example be that the value is too big to be stored [7].

2.3 Software

The software named srsLTE [20] creates a virtual LTE network that consist of an UE, an eNB and an EPC. The software is distributed freely and it has been used in previous experiments. One of these experiments were executed by Gomez-Migueles et al. [8]. In the experiments it is tested to expand the srsLTE program. The expanded version made it possible for the srsLTE to run normally while also running WiFi in the environment. Besides this, the extended code made the srsLTE able to transmit in unlicensed bands. The authors also use the successful expansion as a way to show the potential of the srsLTE library for future work.

2.4 The MAC layer

In this section theory about the MAC protocol and the MAC packet is presented. The subsection about the MAC protocol contains information about the structure of the MAC protocol and the different channels. The subsection about the MAC packet contain detailed information about the structure of the MAC packet, different packet versions and the flags of the packet.

2.4.1 The MAC protocol

The MAC protocol [1] is the lowest layer of the previously mentioned protocols. The purpose of the MAC protocol is to transfer data between the physical layer, through the transport channels, and the RLC layer, through the logical channels. Its functions include mapping of logical and transport channels, priority handling for different logical channels, error correction with hybrid automatic repeat request (HARQ), multiplexing and demultiplexing of transferring data. An overview of the MAC layer is presented in Figure 2.1. Since there are many acronyms a glossary is provided as an appendix.

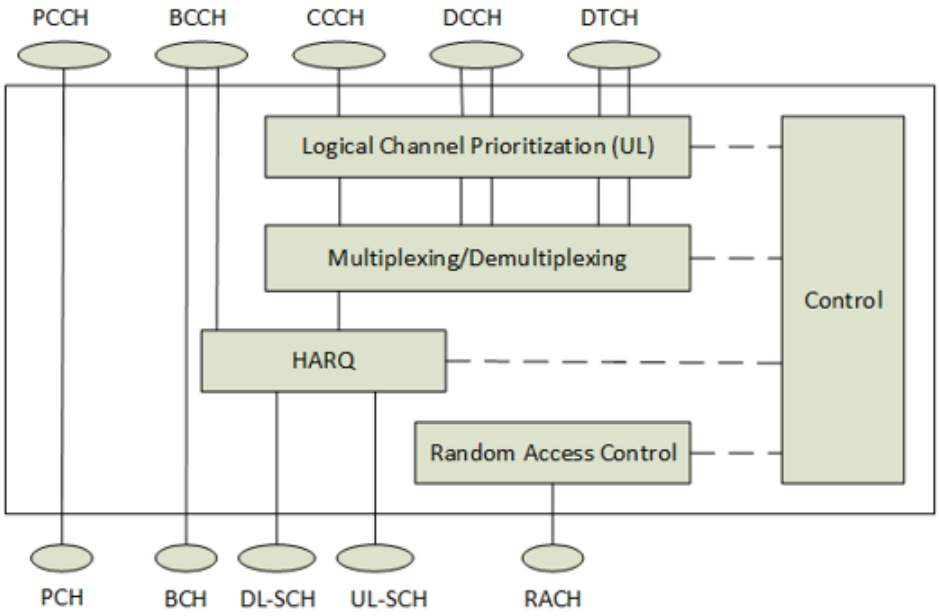


Figure 2.1: Structure of the MAC layer

The MAC protocol has nine transport channels and twelve logical channels mapped to each other. Since the fuzzer is fuzzing incoming data to the eNB, uplink channels are of interest while downlink channels are not. In telecommunications the words uplink and downlink have a different interpretation than in other computer-related areas. The term uplink is referring to all communication that travels ”up” in the communication channel, i.e. from the UE to the eNB. The term downlink refers to all communication travelling ”down” from the eNB to the UE.

The sidelink channels are also discarded since the set-up specifies a connection between an UE and an eNB, not between an UE and an UE. The interesting channels are so forth decreased to five channels in total, two transport channels and three logical channels. The channels and their mapping are presented in Figure 2.2.

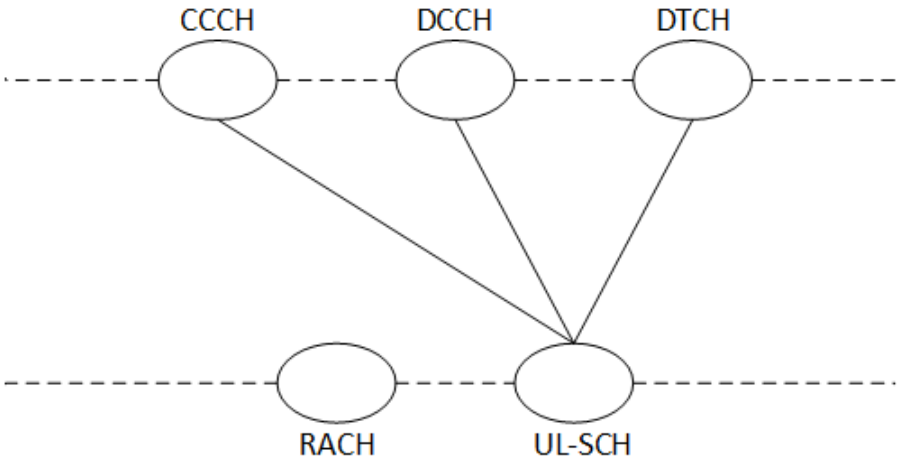


Figure 2.2: Mapped logical channels and transport channels for the uplink MAC protocol.

The Common Control Channel (CCCH), Dedicated Control Channel (DCCH) and the Dedicated Traffic Channel (DTCH) are logical channels connected to the RLC above the MAC protocol. The Uplink Shared Channel (UL-SCH) and Random Access Channel (RACH) are connected to the physical layer below the MAC protocol. The RACH is used before the RRC connection establishment process. It is a channel used by the UE to initiate contact with the eNB. The UL-SCH carries most of the signalling messages and data that is on its way to higher layers. When the UE moves from state RRC_IDLE to state RRC_CONNECTED, the messages for Signalling Radio Bearer 0 are transported over the CCCH. The DCCH also carries signalling messages. Unlike the previous two logical channels the DTCH does not carry signalling messages but instead carries data from an UE [5].

2.4.2 The MAC packet

As mentioned in Section 2.4.1, the UL-SCH and RACH channels are of interest. Depending on the channel the MAC packet has different structures.

For the UL-SCH the MAC packet has a header and a payload. The header is composed of one or more subheaders. The subheaders represents either a MAC SDU, a MAC control element or padding in the MAC payload [1], see Figure 2.3.

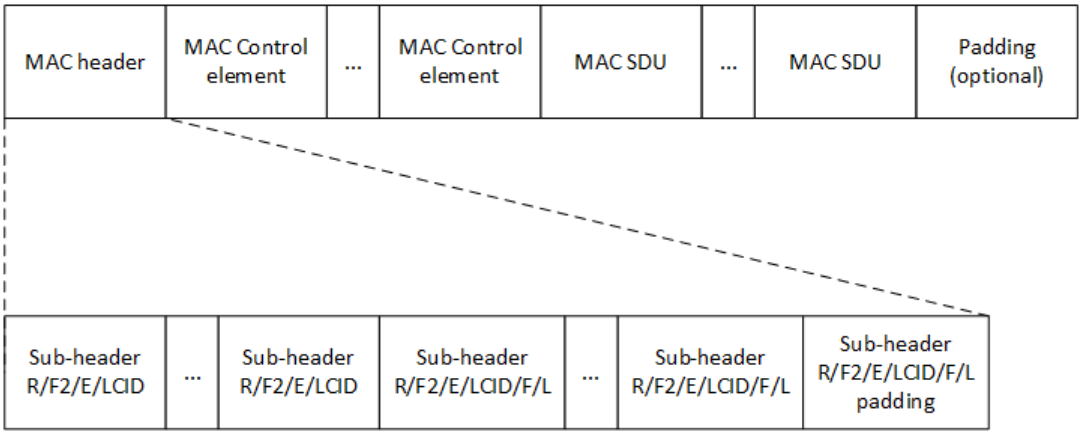


Figure 2.3: Structure of the MAC header and payload

The subheaders can have two different sets of flags in three different combinations, see Figure 2.4. There are six possible flags for the MAC subheaders that are used in the srsLTE program, see Table 2.1.

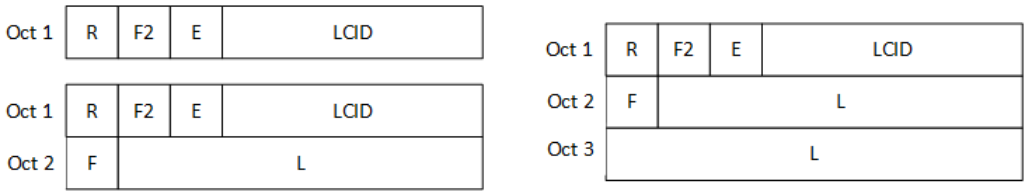


Figure 2.4: The different versions of the MAC header

Full name and abbreviation	Length (in bits)	The flag indicates ...
Reserved bit (R)	1	...that the bit is reserved. For uplink the Rserved bit is set to 0.
Format2 (F2)	1	”... the size of the Length field”
Extension bit (E)	1	”... if more flags are present in the MAC header or not. The E flag is set to "1" to indicate another set of at least R/F2/E/LCID flags. The E flag is set to "0" to indicate that either a MAC SDU, a MAC control element or padding starts at the next byte”
Logical Channel ID (LCID)	5	”... the logical channel instance of the corresponding MAC SDU or the type of the corresponding MAC control element or padding”
Format bit (F)	1	”...the size of the Length field”
Length (L)	7	”... the length of the corresponding MAC SDU or variable-sized MAC control element in bytes.”

Table 2.1: The MAC flags and their description [1]

Three of the flags affect each other. In other words depending on the value of two of the flags, F2 and F, the length of the third flag, the L flag, varies. The exact combinations and their consequences for the L flag are illustrated in Table 2.2.

Value of the F2 flag	Value of the F flag	The length of the L flag
0	0	7
0	1	15
1	-	16

Table 2.2: The connected MAC flags F2, F and L.

The total number of combinations for the MAC packet depends on the length of the packet. For a MAC packet with eight bits, the total number of combinations is two hundred fifty-six (since $2^8 = 256$). For a MAC packet with sixteen bits the number of possible combination is increased to sixty-five thousand five hundred thirty-six (since $2^{16} = 65536$). This gives the fuzzer a maximum of either two hundred fifty-six or sixty-five thousand five hundred thirty-six values to test. However not all combinations of the flags needs to be tested. One example of this is the LCID flag. Some of the numbers can be grouped together as their implication are the same, for example number fourteen (0 1110) and fifteen (0 1111) both are reserved, see Table 2.3.

Value of LCID (in 5 bit)	**Implication**
0 0000	CCCH
0 0001 - 01010	Identity of logical channel
0 1011	CCCH
0 1100	CCCH
0 1101	CCCH and Extended Power Headroom Report
0 1110 - 0 1111	Reserved
1 0000	Extended logical channel ID flag
1 0001	Reserved
1 0010	AUL confirmation (4 octets)
1 0011	AUL confirmation (1 octet)
1 0100	Recommended bit rate query
1 0101	SPS confirmation
1 0110	Truncated Sidelink BSR
1 0111	Sidelink BSR
1 1000	Dual Connectivity Power Headroom Report
1 1001	Extended Power Headroom Report
1 1010	Power Headroom Report
1 1011	C-RNTI
1 1100	Truncated BSR
1 1101	Short BSR
1 1110	Long BSR
1 1111	Padding

Table 2.3: Values for LCID and the implication of them

3 Related Work

In this chapter work related to this thesis is presented. The work includes other researchers' fuzz tests, possible fuzzing strategies and an evaluation of the most common metrics used in fuzzing.

3.1 Fuzz testing

Under this section other researchers' fuzz tests are presented. The papers test different systems and protocols with varying methods. Possible methods for the fuzzing strategy is also presented.

3.1.1 Fuzz testing a LTE network

Kim et al. [12] present a fuzzer named *LTEFuzz* that fuzzes a network using open-source LTE software. The work is restricted to the control plane, but wider in the sense that the target network includes an UE, eNB and Mobility Management Entity (MME). The authors found 36 new and 15 already known vulnerabilities. These vulnerabilities were caused by the incorrect management of:

1. "unprotected initial procedures"
2. "crafted plain requests"
3. "messages with invalid integrity protection"
4. "replayed messages"
5. "security procedure bypass"

After discovering these vulnerabilities the authors also exploited these bugs and attacked the LTE network. The attacks were conducted according to specific ethical guidelines that the authors set up. The guidelines made sure that test cases that could potentially cause interference with other legitimate users in the network, were executed separately. This way other users were not affected by the attacks. By attacking the network the authors were able to deny services, spoof messages, manipulate and eavesdrop the communication.

The authors did address countermeasures for the attacks against the UE, eNB and MME, but were sometimes not able to give more than a possible mitigation for the attack. In the conclusion the authors state that the fuzzer is not to be made available to the public, as it can be used for malicious goals.

The authors also published details regarding the result of LTEFuzz. The result showed that eNB had trouble with the RRCConnectionRequest and RRCConnectionSetupComplete-messages, more precisely for uplink. These messages are specification problems, that allowed for a malicious party to exploit plain messages. This makes the eNB exposed to denial-of-service attacks and spoofing [12].

3.1.2 Fuzz testing the NAS protocol

Fuzzing the NAS protocol is out of scope for this thesis, however fuzz testing has been done on this protocol, which could give some insights in a fuzz tester for the LTE protocol. Johansson et al. [10] created a fuzz tester for the purpose of testing the robustness of a System Under Test (SUT).

The fuzz tester, named T-Fuzz in the paper, had many benefits. Some of the benefits were that T-Fuzz is easily adapted to different (telecommunication) protocols, reaches full coverage of the protocol and is user friendly. The T-Fuzz is a generation-based fuzzer. The SUT needed a stimulated test environment and the framework Testing and Test Control Notation Version 3 (TTCN-3) was integrated with T-Fuzz for that purpose.

The T-Fuzz consists of the following parts: the Model Extractor, the Fuzz Engine and the Observer. The Model Extractor extracts the TTCN-3 models received from the TTCN-3 compiler. The Model Extractor is also able to distinguish between the mandatory and the optional flags of the model. Later on the models are used as input to the Fuzz Engine. The Fuzz Engine generates instances of previously mentioned models. The randomization of the flags are decided with the help of a fixed seed, which allows the fuzzer to be able to run repeatable experiments, and by recognizing which structure flags that are to be randomized. The Observer monitors the SUT. The input received from the SUT is used as feedback to guide the fuzz testing. The implementation of the Observer depends on the SUT, unlike the other two components of the T-Fuzz.

When the T-Fuzz was tested, Johansson et al. outlined four test cases that would test the NAS protocol. They collected statistics of how much "unwanted behaviours"[10] that was found in each test case. The statistics were as following: 0% found in the Initial Phase, 0.6% found in Phase 1, 43.4% found in Phase 2 and the remaining 56% was found in Phase 3. Since the Initial Phase only tried to fuzz the SUT when the UE was not attached to the SUT, it found nothing since the connection was aborted when an error was discovered.

Three out of the five authors were employed by Ericsson at the time, which is an indicator that similar fuzz testing has been done for Ericsson in the past and has succeeded.

3.1.3 Fuzz testing the RLC protocol

Cui et al. [6] presents an interesting conclusion after using format-aware mutation fuzzing on the RLC protocol. After testing in a NS-3 simulation platform the authors were able to find a defect in the RLC protocol. The defect was triggered by the fuzzer changing the parameters of the protocol. The following seven parameters was used in their paper (out of the total seventeen[2]):

1. D/C (Data/Control) flag
2. RF (Re-segmentation) flag
3. P (Polling bit) flag
4. FI (Framing Info) flag

5. SN (Sequence Number) flag
6. E (Extension bit) flag
7. LI (Length Indicator) flag

In their paper Cui et al. shows that it is possible to fuzz test the RLC protocol. They also list all of the parameters that were altered, which gives a hint to how a fuzz tester for RLC can be built.

3.1.4 Fuzz testing with a genetic algorithm

In the work of Jääskelä [9] a genetic algorithm is used to balance the mutators of a fuzzing program. The conclusion of the paper was that optimizing a fuzzer using this approach have potential, as the code coverage of the fuzzer increased.

The use of genetic algorithm was also used in the creation of fuzzing techniques by Spandan et al [22]. The techniques were then tested with the help of a prototype fuzzer, named IFuzzer. With the help of IFuzzer the authors was able to find bugs in an old JavaScript interpreter. The authors state that the generic operation's parameters are not optimal and that fine-tuning them would improve the process.

3.1.5 Fuzz testing with the BooFuzz program

The program *BooFuzz* [16] is a free fuzzing program that generates fuzzed data. The input for the program is the structure and format of the message that will be fuzzed. For the input structure of a MAC packet, the fuzzed output data can contain binary numbers (0,1), letters (A, B, C, ...) and numbers (1, 2, 3, ...).

3.1.6 Earlier experimentation

3.1.6.1 System

According to Taqi Raza and Lu [3] standardized LTE test are lacking in efficiency. In their work they present guidelines that, according to them, reduces up to 70% of LTE testing. This is done by eliminating the repetitive steps of the LTE testing and excluding incorrect output, which would not exist in a connection handled by the protocols.

3.1.6.2 Hardware

Earlier experimentation has proven that eNB can crash with the help of a Software-Defined Radio (SDR) implementation. Byrd et al. [4] demonstrates in their paper that with the help of open-source LTE, it is possible to find security flaws in a LTE network. In this case a certain alteration made it possible to launch a DoS attack against an eNB. The tool that Byrd et al. created is restricted to testing a radio access network.

3.2 Previous vulnerabilities and attacks in LTE

In this section other researchers' work are presented, regarding found vulnerabilities in the protocol stack.

3.2.1 The impersonation attack IMP4GT

Rupprecht et al. [18] presents a cross-layer attack, which enables a person to impersonate either a user in a LTE network, or a LTE network. The attack, named IMPersonation attacks in 4G neTworks (IMP4GT), takes advantage of the missing integrity protection of the user

plane. However, here it must be noted that the location of the vulnerability and that of the attack mechanism are different. The PDCP layer holds the vulnerability, as it only provides integrity protection for the control plane and not the user plane, while the attack mechanism resides in the IP stack.

Knowing this, the authors successfully launched a downlink and uplink attack against the UE and the LTE network. After their successful attack the authors stresses the importance of countering this vulnerability, as it could have devastating effects for the victim. For example the attack can be used to forge internet traffic. If the attacker also forges illegal traffic, which would be a subject for prosecution in the future, it make the victim appear responsible for the crime.

Although the effects of a possible IMP4GT attack is severe, it does have possible flaws for a real-world implementation. The stealth of the attack can be questioned, since the user of the UE will notice a internet connection loss for a brief time. For the LTE network case however, the internet connection will be lost for four seconds, which the authors felt was a justifiable time. Another flaw is that interference from other devices are to be taken into consideration.

Rupprecht at al. have in their paper shown that vulnerabilities and possible attacks does not always reside in the same location. This can also be said for the opposite case, a attack on one layer can trigger a vulnerability in an different layer.

3.2.2 Rogue base stations and tracking the UE

In the paper of Roger Piqueras Jover [11] studies rogue base station. The author used a USRP B210 and a customized version of openLTE in his experiments. From his experiments it was discovered that a rouge base station can deceive the UE that it is not allowed to connect to a specific network. This will make the UE stop trying to connect to the network. This attach can however be reverted simply by rebooting the UE, or by switching airplane mode on and off on the UE. Except from this, the author was also able to track the UE using the Radio Network Temporary Identifier for Physical layer (RNTI PHY) id.

3.2.3 Using the ReVoLTE attack to recover part of an encrypted VoLTE call

Rupprecht et al. [17] presents here the ReVoLTE attack, in addition to the previously mentioned attack IMP4GT mentioned in Section 3.2.1. The ReVoLTE is an attack that takes advantage of the fact that the same keystream is used when two calls are made during the same radio connection. The calls are here part of a over-the-air transmission for LTE, also known as voice over LTE (VoLTE). By connecting to the UE right after the first call, the attacker is able to recover the encrypted VoLTE call.

3.3 Evaluation of metrics for fuzzing

By evaluating other researcher's evaluations of different fuzzers, Klees et al. [13] was able to find problems with the evaluations, deriving from inaccurate statements. Two examples of problems are that varying performance was not taken into consideration and measuring of the performance was done in number of unique crashes. These problems can lead to misleading conclusions and the authors proposed some guidelines to counteract this. One of those guidelines was to run multiple tests when running the fuzzer and not only a single test. The authors discovered that fuzzers can vary in performance and that many evaluations did not take this into consideration. Another guideline was to measure the fuzzer's performance by counting the detection of known bugs and not in the number of unique crashes. This is because the number of bugs can be over-counted and some bugs might disappear if the crashes are linked to wrong bugs.

However Shudrak and Zolotarev [19] do not agree with the fact that fuzzing should be counted in the number of discovered bugs, when the effectiveness of the fuzzer is measured. The reason why the number of discovered bugs is not an adequate metric because it will not give any knowledge about the quality of the input data unless it actually triggers a bug. Neither do the authors believe that code coverage a good match for a metric. In code coverage the complexity of the code is not taken into account, which can lead to misleading results.

4 Method

This chapter describes among other things the setup environment of the fuzzer and the selection of protocol, fuzzing method, metrics and evaluation method. At the end of the chapter the test cases are presented, along with the predefined counters from the Ericsson lab environment.

4.1 The lab environment

The setup of the fuzzer and its environment is visualized in Figure 4.1. The setup is made up of a fuzzer, a SDR, an eNB and a LTE core network.

The fuzzer consists of a monitor, a test case part, an intelligence part and a simulated UE LTE stack. The monitor will register the mutated message sent from the fuzzer and it will register the effect of the message on the eNB. The monitor is connected to the UE and either the eNB or the interface between the SDR and the eNB (in the figure it is connected to the eNB). The test case part will hold all test cases that the fuzzer will execute. The intelligence part will, with the monitor as input, change the test cases and try to fuzz more intelligently. The LTE stack for the UE is simulated with srsLTE, an open software code that can simulate a LTE network and its components. The srsLTE can simulate LTE release 10 [20]. Since the protocol has been updated since then there might be a chance that found bugs are already discovered and fixed. There exist other software for simulating the UE and LTE stack, e.g. Amarisoft and openAir, but the srsLTE is the easiest (code-wise) and most well-documented of the possible choices.

The model of the SDR is Ettus USRP B210 and is provided by Ericsson. This model has previously been tested with the srsLTE, according to the srsLTE documentation [20].

The interface between the SDR and eNB can either be an RF-cable or antennas. If antennas are used the implementation will require a Faraday cage around the setup, so that signals from the environment will not disturb the testing. Initially, it was not known which interface would be used. However, in the end it was decided that RF-cables would be used and therefore there was no need for a Faraday-cage.

The eNB and the LTE core network are provided by Ericsson. If the core network cannot be provided, a simulated core network can be used, for example Ericsson's core-sim or srsLTE.

The Test Case and the Intelligence part of the fuzzer is coded by the author of this thesis. The UE LTE stack is provided with the help of the srsLTE program. The monitor part on the other hand is provided by Ericsson's internal services.

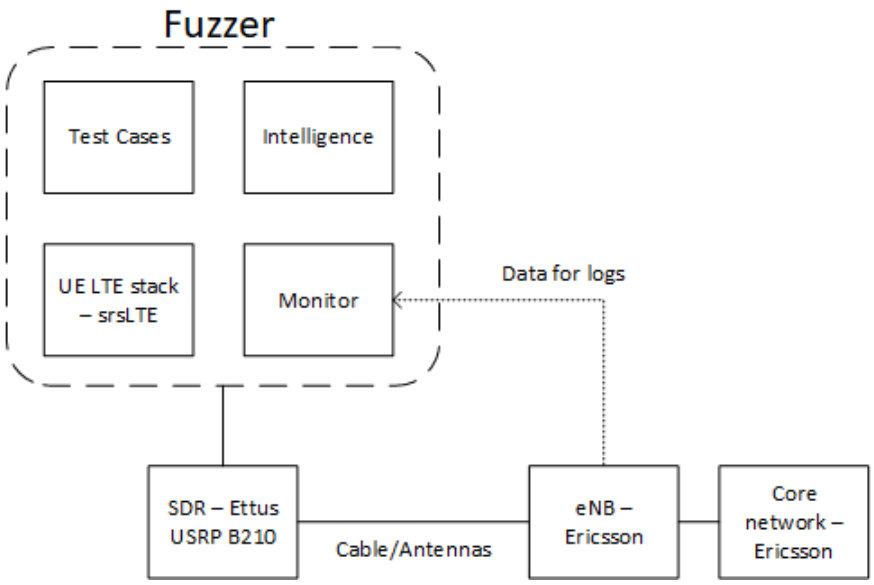


Figure 4.1: The setup

4.2 Fuzzing methods

This chapter describes the decisions about the method of the thesis and how the fuzzer is implemented. The initial plan is to combine srsLTE with one of the two fuzzers, BooFuzz or Genetic. The decision of fuzzer is based on preliminary implementation attempts for both approaches.

4.2.1 Method BooFuzz

In this section, the BooFuzz method is presented.

4.2.1.1 The strategy for fuzzing the connection and the packets

It can be learned from Section 3.1.6.1 that when testing LTE it would be best to minimize the number of repetitive steps. When implementing the fuzz tester this information can help with the efficiency of the test, as lower testing time gives additional time to the schedule, that could be used if an unfortunate situation arises. This indicates that the test cases for the fuzzer should not be based on previous repetitive steps, for example like establishing a connection each time something will be tested.

Section 3.1.2 stressed the importance of how the test varies when testing the initial phase of a connection or when testing an established connection. When testing the initial phase of a connection an error will likely tear down the connection, making the devices spend unnecessary amount of resources on the re-establishment of the connection.

4.2.1.2 Selecting and including fuzzing techniques

An implementation of this strategy was created in the MAC-layer of the UE, for downlink specifically. The downlink implementation worked since the UE needs to receive permission from the eNB to send packets. Sending a packet downlink made the UE believe that the packet came from the eNB and answered with a packets of its own.

For this method, black-box fuzzing is deemed suitable. This is because the other fuzzing techniques, grey-box and white-box fuzzing, is not implementable in the Ericsson eNB.

BooFuzz is used to generate inputs to the fuzzer. Because of this, generation-based and genetic algorithms are unsuitable, as the packets are mutated to match the output of BooFuzz. Mutation-based fuzzing is however suitable for use and the benefit of using mutation-based fuzzing is demonstrated in Section 3.1.4.

4.2.2 Method Genetic

Here the method used when making the Genetic method is presented. An overview of the genetic method can be seen in Figure 4.2. Note here that Round 2 in the figure will loop as long as the fuzzer is able to generate new data.

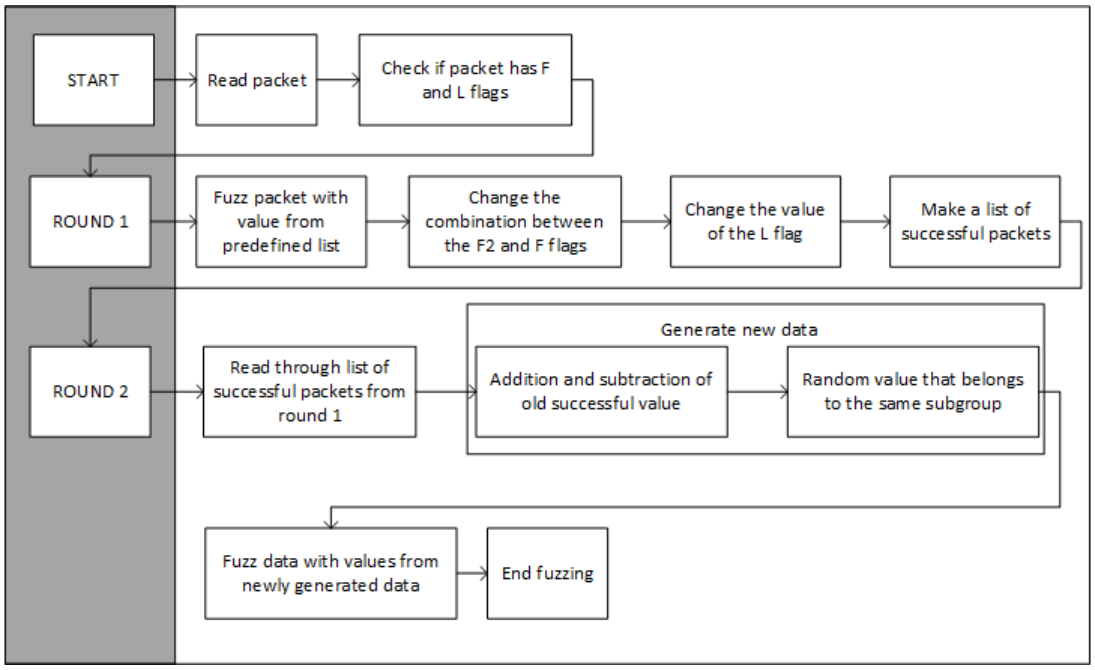


Figure 4.2: An overview of the parts of the fuzzer

4.2.2.1 The strategy for fuzzing the connection and the packets

From Section 3.1.2 it could be seen that testing the initial phase of a connection was different than testing with an established connection. If an error occurred during the initial phase then the connection would be torn down, unlike the other test where the connection would remain open. Additionally Section 3.1.6.1 stated that removing repetitive steps, such as the initial phase, would increase the efficiency of the testing.

Because of the structure of the MAC headers and the characteristics of the connection, when no fuzzing is done, the target of the fuzzer needs to be limited. The messages relating to establishing a connection (including RRCConnectionRequest) are excluded, as there is a difference between testing an established connection and an connection that is establishing.

For this strategy only the already existing packets is fuzzed, unlike the BooFuzz method which creates new packets and then fuzz them.

Firstly a MAC header contains either one or more subheaders, see Figure 2.3. This gives the fuzzer the options to either fuzz one subheader at a time, or to fuzz all subheaders. The second option would also lead to another issue, which is if all of the subheaders are fuzzed with the same values, or with new ones. To keep things as simple as possible, and to make it easier to backtrack possible causes for errors, the second option is discarded.

The subheaders can have two possible structures, see Figure 2.4 for an overview. The R, E and LCID flag are set independently from each other. The R and E flag is each one bit long, having only two possible values (0 or 1). For the LCID flag the fuzzer will go a bit more towards the test-all-approach and test all possible values for LCID, with the exception of some. All values between 01110 and 01111 are reserved and values between 00001 and 01010 describes the identity of the logical channel, see Table 2.3. It is not of interest to test all possible values from these two subgroups. Instead only one value from each group will be tested, see values in Table 4.1 marked with an asterisk inside a parenthesis (*) under the LCID column. This gives us a total of twenty-two test cases for LCID. Since all possible combinations for R and E is not important, these test cases are fused into the LCID test cases.

However the rest of the flags (F2, F and L) affect each other. As they do not affect any of the other flags in the subheader, these test cases can also be fused into previous mentioned test cases. For these flags it would be interesting to test all possible combinations, which in this case would be all possible combinations between the F2 and F flag. Something important to note is that only the size of the L flag is altered by the F2 and F flag. In addition to this, the fuzzer needs to be able to fuzz the content of the L flag. As the L flag contains the length of the MAC SDU or MAC control element, the value can be changed in four ways; to something higher than the original value, to something lower than the original value, to the highest possible value or to the lowest possible value (zero). Since the fuzzer will only change the F2 flag if the F and L flags already exist, the data under the F2 column in Table 4.1 might change and is therefore marked with an asterisk (*).

The strategy of the fuzzer can be summarized by saying that the fuzzer will try all possible values for each flag, but not all possible combinations between them. It will then use a genetic algorithm to find the values that worked best, i.e. the values that is able to pass the MAC layer on the eNB side, and create new values from this set.

Value in decimal format	Value in binary format			
	R	F2	E	LCID
32	0	0 *	1	0 0000
33	0	0 *	1	0 0001 (*)
43	0	0 *	1	0 1011
44	0	0 *	1	0 1100
45	0	0 *	1	0 1101
46	0	0 *	1	0 1110 (*)
48	0	0 *	1	1 0000
49	0	0 *	1	1 0001
50	0	0 *	1	1 0010
51	0	0 *	1	1 0011
52	0	0 *	1	1 0100
149	1	0 *	0	1 0101
150	1	0 *	0	1 0110
151	1	0 *	0	1 0111
152	1	0 *	0	1 1000
153	1	0 *	0	1 1001
154	1	0 *	0	1 1010
155	1	0 *	0	1 1011
156	1	0 *	0	1 1100
157	1	0 *	0	1 1101
158	1	0 *	0	1 1110
159	1	0 *	0	1 1111

Table 4.1: Input data for the first round

4.2.2.2 Selecting and including fuzzing techniques

This approach seems at this point too simple and plain, so it is decided to use one of the fuzzing techniques from Section 2.1. White-box fuzzing is automatically discarded, even though it is an appropriate fuzzing technique (as seen in Section 3.1.2). This is because it is not possible to measure the code coverage of the Ericsson eNB. Since grey-box fuzzing is a combination of black-box and white-box, it is also discarded. Therefore the black-box fuzzing technique is suitable for use.

Even if the fuzzer uses black-box fuzzing, i.e. being unaware of the internal structure of the system under test, it can still be generation-based, mutation-based or use a genetic algorithm. As previously mentioned the fuzzer will not create any new packets. Because of this the generation-based fuzzing technique is excluded, as generation-based fuzzing would create the data from scratch.

Both mutation-based fuzzing (as seen in Section 3.1.3) and using a genetic algorithm (as seen in Section 3.1.4) are suitable approaches for the fuzzer. The two techniques are therefore integrated into the fuzzing strategy. Mutation-based fuzzing is used as the first option when generating new values, while a genetic algorithm is used for the second option of generating new values.

4.2.3 Choosing between the BooFuzz method and the Genetic method

The BooFuzz method is not suitable for use. When new MAC packets are created the packet flow is changed, which in turn leads to more inconsistency between the different runs. Having more inconsistency will create a fragile baseline and since the packets vary from run to run, the probability of triggering the same behaviour from the eNB is lowered. In conclusion,

this approach would be unreliable and conclusions regarding the result from this method would not be trustworthy enough.

It is also noted that for the Genetic method it is impossible to send data in another type than byte, for example like a number or a letter, as the headers are stored in the type *uint8_t* which only accepts one byte. The program that would generate the input, BooFuzz, has letters and numbers in its output. This makes it impossible to use BooFuzz together with srsLTE. Additionally it is noted that the selection of fittest values, from the Genetic method, is not implementable in the Ericsson lab, since the lab software does not allow the fuzzer to see which packets were the successful ones and which were the unsuccessful ones.

Unlike the BooFuzz method the second method, the Genetic method, is more reliable and the results from this method is much more trustworthy. Therefore the Genetic method is used in this thesis.

4.3 Selection of protocol, metric and software location

In this section, the method behind selecting the protocols, metrics and software location is presented.

4.3.1 The decision of only implementing the MAC protocol

It is decided to implement only the MAC layer since resources, in the form of time and testing opportunity, is limited and is not enough for the implementation of other protocols. This decision is affected by the fact that the MAC layer is the highest prioritized layer. As the MAC layer has previously had problem in the past, making a fuzzer for the layer is more important then making a fuzzer for protocols which are known to be relatively stable, compared to the MAC protocol.

4.3.2 Selecting metrics for measuring the success of the fuzzer

Most of the related work in Chapter 3 used the number of found bugs or vulnerabilities as a metric for how successful the tested fuzzer was. However in Section 3.3 using this as a metric was discouraged, as the results might become misleading or inflated. Therefore different metrics were chosen.

4.3.2.1 Metric 1: Covered flags

A metric that can be measured even if there if no errors or bug is desirable. One metric is how well the input values represent the total number of possibilities that should give a different output. This measure should be able to indicate how many of the separate flags are covered by the test.

4.3.2.2 Metric 2: Accepted packets

A second metric would be how many packets are able to pass the filters on the eNB side and reach the other layers in the eNB, e.g. the RLC layer. It can also be expanded to see how many layers the packets is able cross in total. Since the UE has an RRC connection with the eNB, the highest layer would be the RRC layer. Additionally, choosing this metrics shows which flag combinations are good to use for fuzzing. As a measure of success a level of 50% for packets passing the MAC layer is set.

4.3.3 Selecting code location for the fuzzer

When implementing the fuzzer it needs to be implemented in code belonging to the UE and not affecting the eNB. There are four important folders in the srsLTE program that contain the code needed for the program to work: lib, srsue, srsenb and srsepc. The srsepc are not of interest to us. The lib folder is important. It is here that all of the subheaders are written, specifically in the file */srsLTE/lib/src/common/pdu.cc*. The lib folder is however not suitable as it is a used by both the UE and the eNB. Implementing code here will make sure that the same code is used by both UE and eNB. In addition to this it would be very hard to differentiate if it is the UE or the eNB that calls the code. Then the only folder left would be the folder for the UE (*/srsLTE/srsue*).

From here the next important folder would be */srsLTE/srsue/src/stack/mac*. A problem that arises here is that there are multiple files which could be suitable to use: *mac.cc, mux.cc, demux.cc, dl_harq.cc and ul_harq.cc*. The dl_harq and demux file can be eliminated as candidates as they handle data travelling downlink instead of uplink. Out of the three files left the mux file is the file closest "code-wise" to *pdu.cc*, which in turn creates the subheaders that will be fuzzed. This makes the mux file suitable for implementing the code.

4.4 Evaluation method

The evaluation of the fuzzer is made with the help of the previous mentioned metrics in Section 4.3.2. In order to be able to properly distinguish the effect of the fuzzer, a baseline will be created. For an accurate baseline it is important to differentiate one-time occurrences from the normal behaviour. Therefore the program should be run multiple times with no fuzzing when establishing a baseline, as suggested in Section 3.3. After a baseline is set and the eNB have been fuzzed, the results between these two are compared and the fuzzer is evaluated according to the metrics. The metrics are measured with the help of predefined counters from the Ericsson lab environment.

4.4.1 Test cases for Ericsson eNB

In order to see the impact of the different parts of the fuzzer, the fuzzing is split into several test cases. By doing this it is clear which parts of the fuzzer are important and which are less important. The fuzzer is considered to have two parts; the first part is run when the MAC packet has four flags (R, F2, E, LCID) and one, and the second part is run when the MAC packet has six flags (R, F2, E, LCID, F, L). All four test cases that will be run on the Ericsson eNB are described below. The tests were run once per test case, with the exception of the baseline test case.

4.4.1.1 Test case 1: Baseline for Ericsson eNB

The first test case is the baseline for when a connection is established with the Ericsson eNB.

4.4.1.2 Test case 2: Fuzz Ericsson eNB

The second test case has a connection that is fuzzed according to the Genetic method, described earlier in section 4.2.2. For this test case, all parts of the fuzzer are used.

4.4.1.3 Test case 3: Fuzz Ericsson eNB without additional flags

The third test case is a connection that is fuzzed according to the Genetic method, but without fuzzing the additional flags. The additional flags are the F2, F and L flags. The F2 flag is included in here since it is connected to the F and L flags. Even if the fuzzer receives a packet with F and L flags, it will fuzz it according to the Genetic method. By running the fuzzer without the additional flags, the importance of the first part of the method is demonstrated.

4.4.1.4 Test case 4: Fuzz Ericsson eNB by only fuzzing the additional flags

The fourth test case has a connection that is fuzzed according to the Genetic method, but is limited to only fuzzing the additional flags (F2, F and L). By only fuzzing the additional flags, the importance of the second part of the fuzzer is demonstrated.

4.4.2 Explanation of counters

In this subsection the counters and their overall description are presented. The definition of the description and the condition is from Ericsson's internal website. A glossary for the acronyms is provided as an appendix.

4.4.2.1 pmRrcConnEstabSucc

The description and condition of counter *pmRrcConnEstabSucc* is presented in this subsection.

Description	Condition
The total number of successful RRC Connection Establishments	Stepped at reception of RRC message RRC Connection Setup Complete

Table 4.2: The description and condition for the *pmRrcConnEstabSucc* counter

4.4.2.2 pmRrcConnMax

The description and condition of counter *pmRrcConnMax* is presented in this subsection.

Description	Condition
Peak number of UEs in RRC_CONNECTED mode, excluding bandwidth-reduced UEs	This counter is based on an internal level counter maintained by eNB. The internal level counter maintains the current number of UEs in RRC_CONNECTED mode. pmRrcConnMax is updated at end of ROP with the max value during the ROP period

Table 4.3: The description and condition for the *pmRrcConnMax* counter

4.4.2.3 pmRrcConnSamp

The description and condition of counter *pmRrcConnSamp* is presented in this subsection.

Description	Condition
Number of times corresponding Sum counter accumulates new samples. Associated ACC pmCounter is pmFlexRrcConnSum	Stepped when corresponding sum counter has accumulated a new sample

Table 4.4: The description and condition for the *pmRrcConnSamp* counter

4.4.2.4 pmRlcArqUlAck

The description and condition of counter *pmRlcArqUlAck* is presented in this subsection.

Description	Condition
The total number of successful RLC PDU transmissions (ACKs) in the uplink direction	Continous measurement for Radio Bearers aggregated to cell level

Table 4.5: The description and condition for the *pmRlcArqUlAck* counter

4.4.2.5 pmRlcPollRetxDl

The description and condition of counter *pmRlcPollRetxDl* is presented in this subsection.

Description	Condition
The number of sending downlink AM PDU retransmissions due to the poll retransmit timer expiry	Pegged when the poll retransmit timer expiry triggers a downlink AM PDU retransmission

Table 4.6: The description and condition for the *pmRlcPollRetxDl* counter

4.4.2.6 pmMacHarqDlAckQpsk

The description and condition of counter *pmMacHarqDlAckQpsk* is presented in this subsection.

Description	Condition
The total number of successful HARQ transmissions in the downlink direction using a QPSK modulation	To decide if it was successful is based on the HARQ ACK from the UE

Table 4.7: The description and condition for the *pmMacHarqDlAckQpsk* counter

4.4.2.7 pmMacHarqFail

The description and condition of counter *pmMacHarqFail* is presented in this subsection.

Description	Condition
Number of complete HARQ failures in UL. When a HARQ transmission reaches maximum number of retransmissions, not retransmitted again and regarded as a scheduling activity complete HARQ failure	Incremented at each scheduling activity complete HARQ failure in UL

Table 4.8: The description and condition for the *pmMacHarqFail* counter

4.4.2.8 pmMacHarqUlDtx16Qam

The description and condition of counter *pmMacHarqUlDtx16Qam* is presented in this subsection.

Description	Condition
The total number of occasions when an uplink grant was meant for HARQ transmission of a transport block using 16QAM modulation in the uplink direction, where DTX is considered the reason for no reception of HARQ in uplink in the eNB	To decide if it was DTX is based on not receiving a granted uplink transmission using 16QAM modulation

Table 4.9: The description and condition for the *pmMacHarqUlDtx16Qam* counter

4.4.2.9 pmMacHarqUlSucc16Qam

The description and condition of counter *pmMacHarqUlSucc16Qam* is presented in this subsection.

Description	Condition
The total number of successful HARQ transmissions in the uplink direction using a 16QAM modulation	To decide if it was successful is based on the CRC check, not based on if RBS sends HARQ ACK(RBS can use the ACK even if the transport block was not successfully decoded in a way to control the HARQ)

Table 4.10: The description and condition for the *pmMacHarqUlSucc16Qam* counter

4.4.2.10 pmMacHarqUlSuccQpsk

The description and condition of counter *pmMacHarqUlSuccQpsk* is presented in this subsection.

Description	Condition
The total number of successful HARQ transmissions in the uplink direction using a QPSK modulation	To decide if it was successful is based on the CRC check, not based on if RBS sends HARQ ACK (RBS can use the ACK even if the transport block was not successfully decoded in a way to control the HARQ)

Table 4.11: The description and condition for the *pmMacHarqUlSuccQpsk* counter

5 Results

The results are presented in this chapter. First the results for the different test cases are presented, where the results are divided into different tables depending on the layer they belong to. The different layers are RRC, RLC and MAC. The results connected to the RRC and RLC layers are an outcome of the fuzzing of the MAC layer. At the end a summary of all of the results are presented in order to gain a better overview of the results.

5.1 Test case 1: Baseline for Ericsson eNB

The result for test case 1 is presented in this section. In test case 1 a baseline was established where only the srsLTE program was run. The fuzzer was not run in this test case. First the counters regarding the RRC layer are presented, then the counters for the RLC and lastly for the MAC layer.

Name of counter	Value of counter
pmRrccConnEstabSucc	1
pmRrcConnMax	1
pmRrcConnSamp	6

Table 5.1: The result for test case 1, limited to the counters regarding the RRC layer

Name of counter	Value of counter
pmRlcArqUlAck	6
pmRlcPollRetxDl	9

Table 5.2: The result for test case 1, limited to the counters regarding the RLC layer

Name of counter	Value of counter
pmMacHarqDlAckQpsk	6
pmMacHarqFail	1
pmMacHarqUlDtx16Qam	4
pmMacHarqUlSucc16Qam	1
pmMacHarqUlSuccQpsk	9

Table 5.3: The result for test case 1, limited to the counters regarding the MAC layer

5.2 Test case 2: Fuzz Ericsson eNB

The result for test case 2 is presented in this section. In test case 2 the fuzzer is run against the eNB. First the counters regarding the RRC layer is presented, then the counters for the RLC and last for the MAC layer.

Name of counter	Value of counter
pmRrccConnEstabSucc	1
pmRrcConnMax	1
pmRrcConnSamp	61

Table 5.4: The result for test case 2, limited to the counters regarding the RRC layer

Name of counter	Value of counter
pmRlcArqUlAck	1
pmRlcPollRetxDl	0

Table 5.5: The result for test case 2, limited to the counters regarding the RLC layer

Name of counter	Value of counter
pmMacHarqDlAckQpsk	7
pmMacHarqFail	23
pmMacHarqUlDtx16Qam	18
pmMacHarqUlSucc16Qam	1
pmMacHarqUlSuccQpsk	1

Table 5.6: The result for test case 2, limited to the counters regarding the MAC layer

5.3 Test case 3: Fuzz Ericsson eNB without additional flags

The result of test case 3 is presented in this section. Test case 3 have the same basics as test case 2, as the fuzzer is run against the eNB, but the part of the fuzzer that fuzzes additional flags is not used.

First the counters regarding the RRC layer is presented, then the counters for the RLC and last for the MAC layer.

Name of counter	Value of counter
pmRrccConnEstabSucc	0
pmRrcConnMax	1
pmRrcConnSamp	103

Table 5.7: The result for test case 3, limited to the counters regarding the RRC layer

Name of counter	Value of counter
pmRlcArqUlAck	1
pmRlcPollRetxDl	0

Table 5.8: The result for test case 3, limited to the counters regarding the RLC layer

Name of counter	Value of counter
pmMacHarqDlAckQpsk	1
pmMacHarqFail	32
pmMacHarqUlDtx16Qam	45
pmMacHarqUlSucc16Qam	1
pmMacHarqUlSuccQpsk	1

Table 5.9: The result for test case 3, limited to the counters regarding the MAC layer

5.4 Test case 4: Fuzz Ericsson eNB by only fuzzing the additional flags

The result for test case 4 is presented in this section. The fuzzer only fuzzes the additional flags in this case.

First the counters regarding the RRC layer is presented, then the counters for the RLC and lastly for the MAC layer.

Name of counter	Value of counter
pmRrccConnEstabSucc	1
pmRrcConnMax	1
pmRrcConnSamp	12

Table 5.10: The result for test case 4, limited to the counters regarding the RRC layer

Name of counter	Value of counter
pmRlcArqUlAck	6
pmRlcPollRetxDl	9

Table 5.11: The result for test case 4, limited to the counters regarding the RLC layer

Name of counter	Value of counter
pmMacHarqDlAckQpsk	6
pmMacHarqFail	5
pmMacHarqUlDtx16Qam	0
pmMacHarqUlSucc16Qam	1
pmMacHarqUlSuccQpsk	9

Table 5.12: The result for test case 4, limited to the counters regarding the MAC layer

5.5 Summary of results

The summary of the counters is presented in this section.

Name of counter	Value for test case			
	Baseline	All flags	Without additional flags	Only additional flags
pmRrcConnEstabSucc	1	1	0	1
pmRrcConnMax	1	1	1	1
pmRrcConnSamp	6	61	103	12
pmRlcArqUlAck	6	1	1	6
pmRlcPollRetxDl	9	0	0	9
pmMacHarqDlAckQpsk	6	7	1	6
pmMacHarqFail	1	23	32	5
pmMacHarqUlDtx16Qam	4	18	45	0
pmMacHarqUlSucc16Qam	1	1	1	1
pmMacHarqUlSuccQpsk	9	1	1	9

Table 5.13: The summarized result of the counters

6 Discussion

This chapter discusses the results and method of this thesis. For the result, each test case is discussed in different sections. For the method, the fuzzing method, the overall method of the thesis and the metrics are examined and reviewed. At the end, the work is discussed in a wider context.

6.1 Results

A higher value in the results indicate a higher deviation from the baseline. Therefore it is considered as a good sign of a good fuzzing strategy.

6.1.1 Test case 1: Baseline for Ericsson eNB

From the results of this test is can be seen that the connection is successfully established between the UE and the eNB, as the counters *pmRrcConnEstabSucc* and *pmRrcConnMax* both having the value one. Something interesting to note is that the counter *pmMacHarqFail* is not zero but one, meaning that at least one packet fails at the MAC layer of the eNB. This could however be quite reasonable as it is not uncommon for UE connections to lose packets. It must however be taken into consideration when analysing the results for the other test cases.

6.1.2 Test case 2: Fuzz Ericsson eNB

For this test case it was expected that the flag combination would have the highest rate of success, since all parts of the fuzzer are used and each part of the fuzzer should increase the effectivity of the fuzzer. However the results contradicted this, as test case three had higher values of the counters, implying that the fuzzer worked better for that specific test case.

The counter *pmRrcConnSamp* is increased more than ten times compared to the base line. Since the value of the counter is the number of times the Sum counter creates new samples, it is logical to say that the number of samples has also increased. More samples should also mean more data is present in the connection. This is also supported by the fact that the counter *pmMacHarqUlDtx16Qam* is more than four times higher in test case two than in test case 1. If the value of the counter is higher, then that means that more uplink grants was

present in the connection. It can also be said that more uplink grants equals more data in the connection.

This could be dangerous, as some attacks have taken advantage of the fact that systems can't handle too much data. However the eNB will only listen to the UE for certain assigned time slots, meaning that even if all of the time slots are full, the eNB still have capacity to listen to more UEs. If multiple UE's were connected to the eNB and all of them ran the fuzzing program, so that all of the time slots of the eNB are occupied, then it might be possible that the eNB crashes under a Denial-of-Service attack.

The counter *pmMacHarqFail*, which holds the number of HARQ failures in the uplink, is only twenty-three compared to the value sixty-one of the counter *pmRrcConnSamp*. This shows that even though twenty-three of the messages fails to pass the HARQ in the MAC layer, the other counters can be affected much more. However if this number is compared to test case one, which has the value one, one must also start to reason whether or not it was actually twenty-two messages that failed, instead of twenty-three. Even when considering this potential change in the total number of stopped packets, the fact that most packets are stopped by the eNB still remains.

It is however surprising to see that the connection is able to established, considering the values of the counters *pmRlcArqUlAck* and *pmMacHarqUlSuccQpsk*. The counter *pmRlcArqUlAck*, whose value represents the number of successful PDU on the RLC layer, drops from the value six to one. The value of the counter *pmMacHarqUlSuccQpsk*, which drops from nine to one, shows that the number of successful HARQ transmissions decreases, although only when using a 16QAM modulation. The decrease in successful packets is interpreted as that the UE and eNB is struggling to establish a connection.

6.1.3 Test case 3: Fuzz Ericsson eNB without additional flags

The expectation of this test case was that it would perform worse than test case two, but still better than test case four. The values of the counters does however imply that the fuzzer worked better when fuzzing without the additional flags.

It was noted that the counter *pmRrccConnEstabSucc* was set to zero while the counter *pmRrcConnMax* had the value one. This is very interesting, as the description of *pmRrcConnMax* states that the counters value is the number of UEs in the RRC_CONNECTED mode, yet the other counter *pmRrccConnEstabSucc*, which contains the number of successful established RRC connections, have a different value. This raises the question of how the UE can be in the RRC_CONNECTED mode and not have established a successful connection to the eNB.

There are possible explanations for this occurrence. The description of the counter *pmRrcConnMax* mentions that the counter is updated at the end of Reporting Out-put Period (ROP). It could be that the UE was connected before the new ROP started, and therefore no "new" connections have been established.

Another possible explanation is that even though some parts of the connection is established, it is not a one-hundred percent finished connection. The UE have different layers which all needs to be verified at the eNB at some point. Just because some layers are able to function without problem, does not mean that the others layers are the same.

This test case had the highest value of all test cases for the counter *pmRrcConnSamp*. Since the fuzzer is used in this test case, it was expected that the value of the counter would be higher than the baseline, but not that fuzzing without the additional flags would increase the value by almost twice. Additionally it can be seen that the counters *pmMacHarqFail* and *pmMacHarqUlDtx16Qam* also raised in value. This pattern is visible for the first three test cases, as all three counters are raised over time.

This also shows that fuzzing the packets without fuzzing the additional F and L flags, creates more data in the connection. This could, as already mentioned before, be used by a malicious party to carry out a denial-of-service-attack, which would overload the eNB with connections, if enough UE's are fuzzing the eNB. However a malicious party can also gain

something from this, even without carrying out a denial-of-service attack. Unstable initial connections have previously shown to be a flaw by Rupprecht et al. [18] in Section 3.2.1 and by Kim et al. [12] who lists ”unprotected inital procedurs” as a vulnerability in Section 3.1.1.

6.1.4 Test case 4: Fuzz Ericsson eNB by only fuzzing the additional flags

For the last test case it was expected that the performance would be worst, out of all the test cases. This was expected since only a small part of the packet is fuzzed, which should therefore give less efficiency for the fuzzer. The results also indicates that this test case had the lowest efficiency, as the values of the counters only were slightly above the values of the base line.

Unlike the other two previous test cases, test case four does not give as high value for *pmRrcConnSamp*. This is seen as a indication that the connection is faster established and does not struggle as much compared to test case one and test case two. This can also be said for the MAC layer, where the counter *pmMacHarqFail* is very low and the counter *pmMacHarqUlSuccQpsk* is high. The fact that the counters *pmMacHarqUlSuccQpsk* and *pmMacHarqDlAckQpsk* has remained unchanged compared to test case one, could be a indication that the connection is unaffected by this fuzzing method, for the specific case of QPSK modulation. Another counter that remains unchanged is the counter *pmMacHarqUlSucc16Qam*. The value of the counter remains unchanged while the counter *pmMacHarqUlDtx16Qam* is set to zero. This interpretation of this is that the part of the connection modulated with 16QAM is not affected negatively by the fuzzer. Nevertheless it must not be forgotten that the value of *pmMacHarqUlDtx16Qam* could be a one-time-instance.

Since not all packets have the additional flags, it was expected that the values would be dropped, as not as many packets were fuzzed as for previous test cases. For this case it can be seen that, if compared to the baseline, not as many counters changed. While most of the counters changed value for test case 2 and test case 3, only four counters were altered for this test case: *pmRrcConnSamp*, *pmRlcArqUlAck*, *pmMacHarqFail* and *pmMacHarqUlDtx16Qam*.

It is interesting to note here that all counters regarding the RLC layer (*pmRlcArqUlAck* and *pmRlcPollRetxDl*) have remained unchanged . This indicates that the RLC layer have not been affected by the fuzzing of the additional flags. For the MAC layer the number of HARQ failures have increased, as indicated by the counter *pmMacHarqFail*. This means that more packets are stopped at the MAC level of the eNB, which is considered logical as packets are being fuzzed and not all of them might make it through. Another notable thing is that there are no occasions when a uplink grant was meant for HARQ for the 16QAM modulation, according to the counter *pmMacHarqUlDtx16Qam*, but the number of successful uplink HARQ transmissions for the same modulation is still at the value one.

6.2 Method

In this section, the methods are discussed. First is the fuzzing method, then the method of this thesis and last the metrics used.

6.2.1 The fuzzing method

Despite the advantages of the fuzzer’s method, there still exist some disadvantages. First of all, the fuzzer is based on genetic algorithm, so if the fuzzer does not find anything during the first round of fuzzing, it will not continue to fuzz. This might make the fuzzer miss important fuzzing combinations that might have appeared if a genetic algorithm had been used. Secondly, the fuzzer never checks the newly generated values after having generated the second round of data, so duplicate data might appear after the second round. Additionally the input data of the fuzzer could have been improved. One example is that the combinations of different flags could have been tested more thoroughly, instead of just triggering each flag.

Something that could be a disadvantage is the fact that the fuzzer does not fuzz the RAR/RACH and RRCConnectionSetupRequest packet after the first round of fuzzing. The reason these two packets are not fuzzed was because of suggestions made from Ericsson, who argued that the connection would become too unstable and that it would become harder to get concise results. But an unstable connection could potentially yield more promising results, as it has already been noted that "unprotected initial procedures" are listed as a vulnerability in Section 3.1.1.

6.2.2 Lacking logs

The method of this thesis could have been improved. For example a more detailed logging could have been made in order to get more detailed results when running the fuzzer. However while a more detailed logging would have helped, it would also have taken more time, as the logger either would have had to be built from scratch or integrated into Ericsson's internal system.

6.2.3 Metrics

The number of packets that was stopped in the HARQ varied depending on test case, as seen in Table 5.13 on the line of the counter *pmMacHarqFail*. The result for test case one is not of interest, as this test case is only the baseline and does not reflect anything on how well the fuzzer performed. For test case two, it is seen that twenty-four packets are changed by the fuzzer and that twenty-two of these are stopped at the eNB side. That makes eight percent of the packets successful in passing the MAC layer, which is far from the fifty percent that was asked previously. Despite this, one must also remember that the eNB is a current prototype and built to be as reliable as possible, in order to give the best service to the customers.

As seen in Table 4.1, all of the individual flags were covered by the test, as all of the flags are tested in the first round of fuzzing. The F2 flag, which is dependent on the F and L flag, is tested even more in test case four, when only the additional flags are fuzzed.

6.3 The work in a wider context

This thesis has shown that setting up an UE and fuzzing the data towards the eNB does have an effect on the eNB. While no vulnerabilities were discovered within the eNB, which is quite positive, this information can still be used by malicious parties. The fact that the eNB does change behaviour means that it is possible to affect the eNB using fuzzing.

Moreover this thesis provides potential malicious parties with information about what does not work on a current prototype eNB, which means that an attacker can simply skip these test cases if he or she wishes to fuzz an eNB.

Another aspect to think about is the fact that this thesis might inspire malicious parties to fuzz base stations, which might not have been the case if this thesis was not written. However, legitimate penetration testers might also be inspired by this thesis, or find useful information, which could have a positive effect for society in the future.

7 Conclusion

Building the fuzzer was quite difficult. The implementation of the fuzzer was not difficult and did not take long, but gaining the understanding and knowledge to do this was harder. Most of the time was spent on the fuzzer was to understand the the srsLTE code and seeing how the data moved between different parts. Additionally the fuzzer needed to be implemented across different parts of the program, meaning that there was multiple parts to keep track of at the same time. For example the test case part, seen in Figure 4.1, is implemented in another part than the intelligence part of the fuzzer. It was also a challenge to adapt the fuzzer to the Ericsson lab environment. As a matter of fact, adapting the fuzzer to work with the Ericsson environment was the second most time consuming part of building the fuzzer.

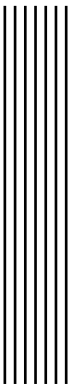
The results in Chapter 5 showed that depending on which part of the fuzzer that was used, the eNB responded differently. When the fuzzer only fuzzed the additional flags (test case four) the connection did not deviate too much from the baseline (test case one). Although when the basic functions of the fuzzer were running (test case two and three) the results started deviating much more compared to test case four.

Despite the fact that the test cases did or did not deviate too much from the baseline, the connection still became more unstable when the fuzzer was running. Therefore it can be considered a success, as the unstable connections have been considered to be unreliable.

The first research question is answered in the thesis. As white-box and grey-box is not implementable for the Ericsson environment, these fuzzing techniques discarded and black-box fuzzing is considered a good match. Generation-based fuzzing, which generates input from scratch, is also discarded while mutation-based, which mutates already valid data, is not. Additionally using a genetic algorithm is also considered to be a appropriate technique.

The second research question has more than one answer to it. First of all, it does cover all of the flags in a successful way, as all flags are triggered at least once during the connection. Regarding our selected success criteria of at least 50% of packets passing the air interface, no test case was able to do this. Therefore the anticipated success in section 4.3.2.2 was not accomplished.

Another conclusion of this thesis is that Ericsson now has the possibility to test their eNB equipment in the Ericsson lab. This test set-up can in the future be extended to test more equipment than just eNB, or it can be extended to run different software and security testing tools than just fuzz testing. This has already, and will in the future, improve Ericsson's understanding of their own equipment.



Bibliography

[1] *3gpp archive MAC protocol, version 15.8.0*. URL: https://www.3gpp.org/ftp/Specs/archive/36_series/36.321 (visited on 01/26/2020).

[2] *3gpp archive RLC protocol, version 15.3.0*. URL: https://www.3gpp.org/ftp/Specs/archive/36_series/36.322 (visited on 01/26/2020).

[3] *A Systematic Way to LTE Testing | The 25th Annual International Conference on Mobile Computing and Networking*. EN. URL: https://dl.acm.org/doi/abs/10.1145/3300061.3300134 (visited on 01/27/2020).

[4] Thomas Byrd, Vuk Marojevic, and Roger Piqueras Jover. "CSAI: Open-Source Cellular Radio Access Network Security Analysis Instrument". In: *2020 IEEE 91st Vehicular Technology Conference (VTC2020-Spring)*. ISSN: 2577-2465. May 2020, pp. 1–5. DOI: 10.1109/VTC2020-Spring48590.2020.9129373.

[5] Christopher Cox. *An introduction to LTE LTE, LTE-advanced, SAE, VoLTE and 4G mobile communications*. Chichester, West Sussex, United Kingdon ; Hoboken, New Jersey: John Wiley and Sons, Incorporated. ISBN: 9781118818039.

[6] Baojiang Cui, Shengbo Feng, Qinshu Xiao, and Ming Li. "Detection of LTE Protocol Based on Format Fuzz". In: *2015 10th International Conference on Broadband and Wireless Computing, Communication and Applications (BWCCA)*. Nov. 2015, pp. 187–192. DOI: 10.1109/BWCCA.2015.42.

[7] James Forshaw. *Attacking Network Protocols : A Hacker's Guide to Capture, Analysis, and Exploitation*. 2018. ISBN: 9781593277505.

[8] Ismael Gomez-Miguelez, Andres Garcia-Saavedra, Paul D. Sutton, Pablo Serrano, Cristina Cano, and Douglas J. Leith. "srsLTE: An Open-Source Platform for LTE Evolution and Experimentation." In: (2016). URL: https://arxiv.org/pdf/1602.04629.pdf.

[9] Esa Jääskelä. *Genetic algorithm in code coverage guided fuzz testing*. en. 2016. URL: http://jultika.oulu.fi/Record/nbnfioulu-201601151058 (visited on 05/04/2021).

[10] William Johansson, Martin Svensson, Ulf E. Larson, Magnus Almgren, and Vincenzo Gulisano. "T-Fuzz: Model-Based Fuzzing for Robustness Testing of Telecommunication Protocols". In: *Verification and Validation 2014 IEEE Seventh International Conference on Software Testing*. ISSN: 2159-4848. Mar. 2014, pp. 323–332. DOI: 10.1109/ICST.2014.45.

[11] R. Jover. "LTE security, protocol exploits and location tracking experimentation with low-cost software radio". In: *ArXiv* abs/1607.05171 (2016).

[12] Hongil Kim, Jiho Lee, Eunkyu Lee, and Yongdae Kim. "Touching the Untouchables: Dynamic Security Analysis of the LTE Control Plane." In: *2019 IEEE Symposium on Security and Privacy (SP), Security and Privacy (SP), 2019 IEEE Symposium on* (May 2019), pp. 1153–1168. ISSN: 978-1-5386-6660-9.

[13] George Klees, Andrew Ruef, Benji Cooper, Shiyi Wei, and Michael Hicks. "Evaluating Fuzz Testing." In: (2018). URL: https://www.cs.umd.edu/~mwh/papers/fuzzeval.pdf.

[14] *LTE Radio Protocol Architecture - Tutorialspoint*. URL: https://www.tutorialspoint.com/lte/lte_radio_protocol_architecture.htm (visited on 02/03/2020).

[15] V.J.M. Manes, H. Han, C. Han, S.K. cha, M. Egele, E.J. Schwartz, and M. Woo. "The Art, Science, and Engineering of Fuzzing: A Survey." In: *IEEE Transactions on Software Engineering* (2019). ISSN: 19393520. URL: https://ieeexplore.ieee.org/document/8863940.

[16] Joshua Pereyda. *Boofuzz*. original-date: 2015-11-26T00:42:48Z. Apr. 2020. URL: https://github.com/jtpereyda/boofuzz (visited on 04/02/2020).

[17] D. Rupprecht, K. Kohls, T. Holz, and C. Pöpper. "Call Me Maybe: Eavesdropping Encrypted LTE Calls With ReVoLTE". In: *USENIX Security Symposium*. 2020.

[18] David Rupprecht, Katharina Kohls, Thorsten Holz, and Christina Poepper. "IMP4GT: IMPersonation Attacks in 4G NeTworks". In: *Network and Distributed System Security Symposium*. Jan. 2020. DOI: 10.14722/ndss.2020.24283.

[19] Maksim Shudrak and Vyacheslav Zolotarev. "Improving Fuzzing Using Software Complexity Metrics". In: 9558 (2016), pp. 246–261. DOI: 10.1007/978-3-319-30840-1_16. URL: http://arxiv.org/abs/1807.01838.

[20] *srsLTE/srsLTE*. original-date: 2013-12-06T13:53:04Z. Feb. 2020. URL: https://github.com/srsLTE/srsLTE (visited on 02/14/2020).

[21] Ari Takanen, Jared D. DeMott, and Charles Miller. *Fuzzing for Software Security Testing and Quality Assurance*. English. Artech House information security and privacy series. Artech House, June 2008. ISBN: 9781596932159. (Visited on 01/31/2020).

[22] Spandan Veggalam, Sanjay Rawat, Istvan Haller, and Herbert Bos. "Ifuzzer: An evolutionary interpreter fuzzer using genetic programming." In: *Computer Security - 21st European Symposium on Research in Computer Security, ESORICS 2016, Proceedings* 9878 LNCS (2016), pp. 581–601. URL: https://download.vusec.net/papers/ifuzzer-esorics16.pdf.

8 Appendix

8.1 Glossary

All abbreviations and their full meaning is presented in the table below.

Abbreviation	Full meaning
16QAM	Quadrature Amplitude Modulation
ACC	Automatic Congestion Control
ACK	Acknowledgement
AM	Acknowledge Mode
BCCH	Broadcast Control Channel
BCH	Broadcast Channel
CCCH	Common Control Channel
DCCH	Dedicated Control Channel
DL-SCH	Downlink Shared Channel
DTCH	Dedicated Traffic Channel
DTX	Discontinous Transmission
eNB	E-UTRAN Node B
EPC	Evolved Packet Core
HARQ	Hybrid Automatic Repeat Request
LTE	Long-TermEvolution
MAC	Medium Access Control
MME	Mobility Management Entity
NAS	Non Access Stratum
PCCH	Paging Control Channel
PCH	Paging Channel
PDCP	Packet Data Convergence Protocol
PDU	Protocol Data Unit

Abbreviation	Full meaning
RAR	Random Access Response
RACH	Random Access Channel
RBS	Radio Base Station
Rf-antenna	Radio Frequency antenna
RLC	Radio Link Protocol
RNTI	Radio Network Temporary Identifier
ROP	Reporting Out-put Period
RRC	Radio Resource Control
SDR	Software-Defined radio
SDU	Service Data Unit
SUT	System under Test
UE	User Equipment
UL-SCH	Uplink Shared Channel