ASTRA-5G: Automated Over-the-Air Security Testing and Research Architecture for 5G SA Devices

Syed Khandker
New York University Abu Dhabi
Abu Dhabi, UAE
syed.khandker@nyu.edu

Michele Guerra
University of Molise
Pesche, Italy
michele.guerra@unimol.it

Evangelos Bitsikas
Northeastern University
Boston, USA
bitsikas.e@northeastern.edu

Roger Piqueras Jover*
Google
New York, USA
rogerpiqueras@google.com

Aanjhan Ranganathan
Northeastern University
Boston, USA
aanjhan@northeastern.edu

Christina Pöpper
New York University Abu Dhabi
Abu Dhabi, UAE
christina.poepper@nyu.edu

ABSTRACT

Despite the widespread deployment of 5G technologies, there exists a critical gap in security testing for 5G Standalone (SA) devices. Existing methods, largely manual and labor-intensive, are ill-equipped to fully uncover the state of security in the implementations of 5G SA protocols and standards on devices, severely limiting the ability to conduct comprehensive evaluations. To address this issue, in this work, we introduce a novel, open-source framework that automates the security testing process for 5G SA devices. By leveraging enhanced functionalities of 5G SA core and Radio Access Network (RAN) software, our framework offers a streamlined approach to generating, executing, and evaluating test cases, specifically focusing on the Non-Access Stratum layer. Our application of this framework across multiple 5G SA devices provides in-depth security insights, significantly improving testing efficiency and breadth.

CCS CONCEPTS

• **Security and privacy → Mobile and wireless security**.

KEYWORDS

5G, Security, Automated Testing, Open5GS, srsRAN

ACM Reference Format:

Syed Khandker, Michele Guerra, Evangelos Bitsikas, Roger Piqueras Jover, Aanjhan Ranganathan, and Christina Pöpper. 2024. ASTRA-5G: Automated Over-the-Air Security Testing and Research Architecture for 5G SA Devices. In *Proceedings of the 17th ACM Conference on Security and Privacy in Wireless and Mobile Networks (WiSec '24), May 27–30, 2024, Seoul, Republic of Korea.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3643833.3656141

1 INTRODUCTION

5G networks can be deployed using two main architectural approaches: Non-Standalone (NSA) and Standalone (SA). While both NSA and SA architectures facilitate 5G deployment, they differ significantly in their approach. NSA provides a quicker and more cost-effective entry point by leveraging existing 4G infrastructure, making it an ideal choice for early adopters. However, its reliance on older technology limits its potential. In contrast, SA represents the future-proof option. Operating independently, it unlocks the full spectrum of 5G capabilities, including advanced network slicing, ultra-low latency, massive machine-type communication, stronger authentication, and enhanced privacy mechanisms. This paves the way for groundbreaking applications and services, making SA the preferred choice for long-term investments and unlocking the true potential of the 5G revolution.

The specification outlines the maximum security levels achievable by a cellular network. However, research into previous generations (e.g., 3G, 4G) has revealed instances of less robust security implementations compared to those specified [19]. Various factors such as limited resources, technological constraints, implementation complexity, unclear specifications, or a lack of awareness regarding emerging threats could contribute to this deviation. Additionally, prioritizing other features during development may have led to compromises in security. Therefore, it is crucial to assess the level of implemented security, especially in 5G SA, as it represents the most recent advancement in cellular technology.

While extensive security research has been conducted on the 5G protocol [13, 18, 20], there remains a notable gap in attention towards verifying the security of implementations in 5G devices. Moreover, current security testing frameworks and fuzzing approaches are predominantly tailored for 4G/LTE networks [10, 17, 37]. While these frameworks may be applicable to 5G NSA, they lack the capability to detect security vulnerabilities in 5G SA environments. 5G SA introduces architectural changes (e.g., service-based architecture, network slicing) and new security mechanisms (e.g., stronger authentication, better privacy mechanism), which increases the complexity of network interactions. User Equipment (UE) needs to interact seamlessly with these network elements, and any vulnerabilities in their communication protocols or implementation could compromise the security of the entire network. Although some recent studies have begun addressing the security of 5G SA devices [6, 42], these efforts primarily rely on manual processes. Utilizing such tools also requires a deep understanding of the various protocols, devices, and networks, and involves labor-intensive tasks that consume considerable time to analyze test

*This author did not contribute any code for this project.

WiSec '24, May 27–30, 2024, Seoul, Republic of Korea

ACM ISBN 979-8-4007-0582-3/24/05.
https://doi.org/10.1145/3643833.3656141

results and make decisions. Essentially, there is an unexplored requirement for an automated testing framework tailored specifically to assess the implemented security in 5G SA UE, with the capability to autonomously generate test cases, conduct over-the-air execution, and comprehensively evaluate the resulting outcomes.

To address the aforementioned gap and work towards transparency and comparability of 5G UE security implementations, in this work, we design and develop an automated 5G SA device security testing framework. This framework can efficiently generate test cases based on user requirements, execute tests on a designated 5G smartphone, collect reports, and evaluate the results comprehensively. A significant challenge to achieving full automation is the complexity and scale of the system. The sheer number of possible message combinations, network configurations, and parameter settings that exist in 5G SA results in a state-space explosion, making it impractical and further complicating the task of identifying and mitigating security weaknesses. It is important to develop a framework that can efficiently navigate the vast array of potential states and interactions. We tackle this challenge by leveraging modified functionalities of open-source 5G SA core and Radio Access Network (RAN), and expert knowledge on specifications and network configuration.

Specifically, our paper makes the following contributions:

- We have developed the first automated over-the-air security testing framework tailored for 5G SA devices. The framework integrates open-source software (Open5GS, srsRAN, etc.) and streamlines the detection and mitigation of security vulnerabilities in 5G SA devices.
- Our framework introduces an advanced process for generating test cases that target the Non-Access Stratum (NAS) layer, enabling comprehensive testing considering both uplink and downlink traffic. This process is designed around a dynamic parameter selection mechanism, leveraging a combination of rule-based systems and algorithmic checks to enable testing across a wide range of scenarios, including security violations, protocol misimplementations, and undefined corner cases.
- For evaluating the test-case results, we design a rule-based automatic evaluation tool that simplifies the assessment of UE NAS security by systematically analyzing test outcomes. This innovation minimizes manual intervention and enhances accuracy in identifying security weaknesses. Additionally, we utilize publicly available Large Language Models (LLMs), which show promising potential for comprehensive evaluation.
- The framework demonstrates its effectiveness by detecting several security-related issues specific to 5G SA devices, such as the use of temporary Anti-Bidding-down Between Architectures (ABBA) values, incompetence in adhering to protocols when identity is requested, etc.

Responsible Disclosure and Open Sources. As part of responsible disclosure, we have notified the relevant UE manufacturers regarding our findings. To support further research and investigations on this topic, we make our code available on GitHub at https://github.com/s21sm/ASTRA-5G.

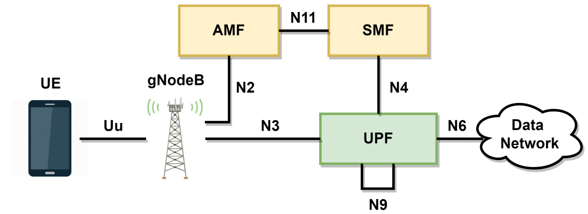

Figure 1: 5G SA Architecture

2 BACKGROUND AND RELATED WORK

2.1 5G Standalone Architecture

The SA architecture of 5G represents a significant advancement in cellular technology, providing a fully independent network infrastructure capable of delivering the full spectrum of 5G services and capabilities without relying on any legacy technology such as 4G. In 5G SA architecture, the core network functions have been redesigned to accommodate the unique requirements of 5G, enabling ultra-low latency, enhanced network slicing, and improved scalability compared to previous generations. Figure 1 shows the 5G SA architecture. In this section, we briefly give an overview of the various components of 5G SA architecture.

Access and Mobility Management Function (AMF): The AMF is a pivotal component in the 5G Core Network, tasked with managing access and mobility of UE. It oversees user registration, authentication, and session management, and supports mobility across different networks, ensuring continuous connectivity and efficient mobility management within the 5G infrastructure.

Session Management Function (SMF): The SMF plays a crucial role in session management within the 5G core network. It handles the establishment, modification, and release of user sessions, allocates network resources according to service requirements, and orchestrates the data flow within the network, ensuring that data packets are correctly routed and forwarded to their destinations.

User Plane Function (UPF): Serving as a key element in data forwarding and routing, the UPF connects the 5G network with external networks or services. It processes user data packets (User-Plane traffic), supporting packet inspection, charging, and application of traffic policies, which are essential for optimized network resource utilization and enhanced quality of service.

Next Generation NodeB (gNodeB): The gNodeB is the base station in the 5G RAN, providing wireless communication with UEs. It manages radio resources, implements advanced signal processing techniques, and connects to the 5G core network, enabling the delivery of diverse mobile services.

Data Network (DN): The Data Network represents external networks accessible by UEs, including the internet, corporate, and service networks. It is fundamental to the 5G architecture, enabling connectivity to a vast array of content and services.

User Equipment (UE): UEs include a wide range of devices such as smartphones, tablets, and IoT devices that communicate with the 5G network. They support various services enabled by 5G, while serving as critical endpoints in the network.

2.2 Related Work

Recent studies in 5G security testing for UE have introduced innovative methodologies [6, 43], offering strategies to modify the execution flows of 5G protocols. This enables practical examination of specific parameters in commercial off-the-shelf (COTS) UEs without heavy reliance on specifications, potentially uncovering implementation flaws. However, the predominantly manual nature of these methods limits their ability to conduct extensive testing across a broad spectrum of parameters and a vast number of tests.

The integration of natural language processing models and machine learning for detecting vulnerabilities in LTE and 5G protocols has been a subject of recent investigation [9, 20, 23]. These approaches involve scanning and parsing extensive collections of 3GPP specification documents to generate test cases, uncovering vulnerabilities or discrepancies within the protocols. Chen et al. [8] highlight the severe implications of specification errors, such as design flaws and presentation issues, with a particular focus on the widespread issue of inconsistent descriptions (misalignment) in security-critical content. In a different vein, Klischies et al. [27] have put forward a method for specification analysis without machine learning that focuses on modeling and synthesizing counterexamples, leading to undefined behaviors in LTE.

5GReasoner [18] offers a formal verification method for the 5G control plane, while LTE-Inspector [17] applies symbolic model checking and cryptographic protocol verification to uncover security flaws on LTE. ProChecker [22] utilizes semantic models as finite-state machines for testing, and the Non-compliance checker [19] adopts a property-agnostic, black-box approach for control-plane testing in COTS UEs. Alternatively, vulnerability detection through over-the-air dynamic testing within the LTE control plane has been demonstrated [12, 26], going beyond the capabilities of protocol verification. Additionally, DoLTEst [35] emphasizes negative testing with deterministic oracles based on specification analysis to reveal LTE implementation flaws. Rupprecht et al. [38] target the discovery of LTE security function flaws, such as in data encryption and network authentication.

Dedicated testing frameworks have also been developed to specifically target baseband vulnerabilities. BaseSAFE [29] and FIRMWIRE [16] stand out for their effectiveness in detecting security vulnerabilities through fuzzing. AutoFuzz [13] focuses on identifying vulnerabilities through malformed and out-of-order packet analysis, whereas Berserker [37] specializes in ASN.1-based fuzzing techniques. Kim et al. [25] offer a comparative analysis between baseband software and cellular specifications to systematically inspect how message structures are implemented in baseband software. Building on this, [24] presents BaseComp, an enhancement over BaseSpec that conducts a comparative analysis including integrity protection in cellular baseband software, and notably improving the level of automation.

Although there have been previous studies on over-the-air device security testing, they either focus on previous-generation networks (e.g., 4G LTE) or lack automation functionalities, posing challenges for large-scale security testing. Our framework provides automation for 5G security implementation testing for SA devices.

3 SECURITY TESTING AUTOMATION FRAMEWORK DESIGN

3.1 Overall Goal and Challenges

Our framework is designed with a single, clear objective: to enhance the security of 5G SA devices through an automated testing process. This process aims to identify and analyze both existing and emerging vulnerabilities, ensuring transparency and independence in evaluations. By focusing on automated over-the-air security evaluations, the framework provides researchers, developers, practitioners, and industry experts with detailed insights into the security of 5G SA devices, potentially enabling them to make informed decisions to improve device security. The design and development of such a framework face three main challenges as detailed below.

Test Case Generation. The first significant challenge arises from the need to generate test cases, a critical component of any security testing framework. The complexity here in the context of 5G cellular technology stems from the vast amount of protocol-related information scattered across numerous 3GPP documents, combined with the varied preferences for security testing among researchers. To tackle this, we have developed a test case generator that allows researchers to create customized test cases tailored to their specific needs, addressing the issues of information fragmentation and individual preference.

Test Execution. The second challenge involves executing the test cases on a 5G SA network to monitor the UE response. This process requires a network setup that supports 5G SA, including both Core and RAN that can facilitate the loading and execution of test cases. Although this can be achieved through a modified open-source 5G core and RAN, executing these test cases still requires a systematic procedure that includes deploying the core with the test case, activating the RAN with an RF front end, and guiding the UE to register with the network. If this process is manual, it must be repeated for each test case, making it labor-intensive and time-consuming. Our framework aims to alleviate these burdens by allowing for the sequential, automated running of multiple test cases, thus saving significant manual effort and time.

Result Evaluation. The final challenge is evaluating the UE response to determine the security test outcome. Traditionally, this step has been manual, involving the use of network packet analyzer tools like Wireshark, which is both time-consuming and labor-intensive. To overcome this, we propose a programming-based evaluation tool within our framework that automatically generates reports for each test case by analyzing the input (test case) and output (pcap file). This tool similar to [6] considers the state machine of the UE during both the test case generation and evaluation phases, facilitating an accurate assessment of whether the UE's behavior conforms to the standards.

In summary, our framework focuses on automating the entire pipeline of test case generation, test execution, and evaluation process. The current lack of automation in these areas hinders mass-scale testing, is time consuming, and requires a labor-intensive evaluation process prone to human errors. Specifically, our framework aims to automate not only the test generation and execution

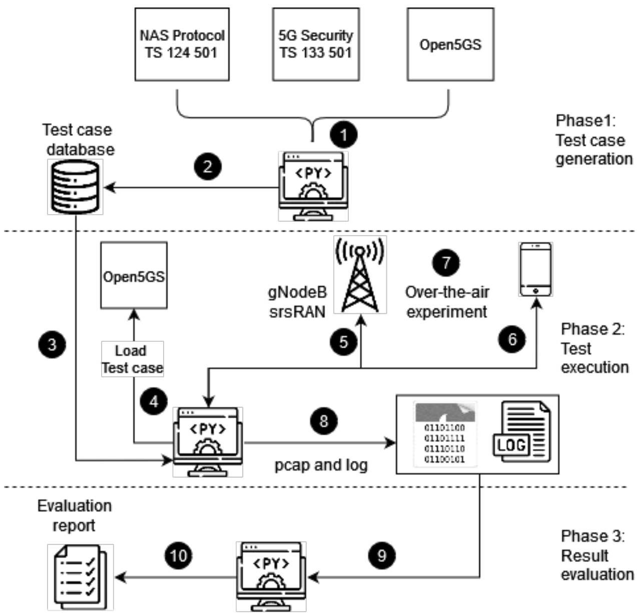

Figure 2: Automation process

processes but also the security evaluation phase. For example, we develop a rule-based security evaluation program. Besides, we explore the potential use of well-known LLMs for acquiring and analyzing results. By integrating our targeted automation solutions and LLMs, our framework takes a first step towards facilitating a more efficient, accurate, and scalable security assessment process for 5G SA devices.

3.2 High-level Framework Overview

Our framework, aimed at testing security-related issues in 5G SA devices, is structured into three main components: test case generation, test execution, and result evaluation, as depicted in Figure 2. This structure facilitates a systematic approach to identifying and addressing vulnerabilities in 5G SA devices.

Approach and Assumptions. To support our testing framework, we have made key selections and assumptions regarding the core and RAN software. After a comprehensive analysis, Open5GS [32] and srsRAN [39] were chosen for the core network component and gNodeB, respectively, due to their balance of simplicity, reliability, and robust support for 3GPP standards. These software choices are also well-regarded in the research community and have been leveraged in prior work [6, 21, 30].

Building on the execution methodology from [6], our framework adopts a similar strategy for modifying the NAS reception, handling, and machine state to facilitate NAS test cases. This method allows us to customize the message and parameter transmission, ensuring compatibility with the execution flows of established works. Our approach includes comprehensive security assessments of COTS UEs from various manufacturers. This necessitates access to the testing framework and a USRP for over-the-air testing, allowing for adjustments to UE and USIM settings as needed. These foundational assumptions set the stage for our security testing framework.

Test Case Generation. The initial phase involves an in-depth study of 3GPP documents to understand 5G device security (❶). We then explore Open5GS capabilities, focusing on supported uplink and downlink commands and parameters. This exploration leads to the development of a test case generator software, which selects downlink messages and their parameters, assigning values to create test cases (❷).

Test Execution. In this stage, our framework selects a test case from the database (❸), loads it into the core network (❹), and initiates the gNodeB (❺). The UE equipped with a programmed USIM, connected via USB (❻), is prepared for testing by toggling from airplane mode to normal mode using the pure-python-adb [41] library, enabling network registration (❼). Open5GS then sends a downlink NAS message based on the test case, to which the UE responds, completing the execution phase. Network packets are recorded using *tcpdump*, along with AMF, gNodeB, and ADB logs (❽), aiding in understanding the test's execution flow. Following the completion of a test case, the core and network services are torn down, and the UE is instructed to enter airplane mode in preparation for the next test case.

Result Evaluation. The final phase evaluates the UE's response using a program that compares the actions of the UE, based on pcap files, against the expected behavior defined by the test case (❾). The test case contains a set of custom instructions or information that is provided to the UE by the core using downlink NAS messages. These instructions can adhere to the protocol or intentionally violate the protocol to assess how the UE responds. Any deviation from the standard protocol is flagged, with the evaluation results compiled into a PDF report (❿). The framework utilizes both rule-based and LLM-based approaches to evaluate the UE response.

4 AUTOMATED TEST CASE GENERATION

We evaluate 5G device security using NAS messaging. Information regarding NAS messaging is dispersed across various 3GPP documents, such as the NAS protocol [1] and the 5G security architecture [2]. Any targeted security feature needs to be supported by the core network, which instructs the UE to act accordingly through the RAN. If the core network software implementation does not support a particular security feature, then it cannot be tested. Therefore, in addition to reviewing 3GPP documents, we also scrutinized the source code of Open5GS to gain a better understanding of the parameters for different test cases. In this section, we provide details on our automatic test case generation software.

NAS Messages and Security Parameters. The list of NAS messages and their respective parameters is defined in [1, in Section 8]. Each message includes mandatory and optional parameters, many of which offer multiple possible values. Sometimes, these values have security implications. For instance, the `Security header type` specifies the security mechanism applied to a message, directly influencing system security. UE security requirements, as outlined in [2], encompass algorithm implementation, subscriber privacy, and the secure handling of subscription credentials, etc.

Upon reviewing the protocols, we turned our attention to the implementation of Open5GS. Interestingly, Open5GS introduces

Table 1: Supported NAS messages in our framework

Uplink	Downlink
Registration Request	Identity Request
Identity Response	Authentication Request
Authentication Response	Security Mode Command
Security Mode Complete	Configuration Update Command
Registration Complete	Service Accept
UL NAS Transport	Service Reject
Service Request	GMM Status
Security Mode Reject	Deregistration Request
Authentication Failure	Deregistration Accept
Deregistration Request	Authentication Reject
	Registration Reject

additional parameters not listed in the 3GPP documents. For example, the `Security Header For Service Request Message` found in Open5GS is absent in the 3GPP specifications [1, in 9.3]. We cataloged all parameters for NAS messages supported by Open5GS, noting that while some parameters are protocol-defined e.g., 5GMM causes, others allow arbitrary values such as RAND and AUTN. We organized these into lists for supported downlink NAS messages, facilitating our testing approach. Table 1 summarizes the NAS messages analyzed in our experiments.

Execution Flow and Message Sequences. The uplink serves as a critical hooking point, directing the core to the intended execution point for the downlink message. For effectiveness, the chosen uplink must be a regular NAS message that occurs during normal attach flow. Through extensive analysis of numerous pcap files, we identified seven messages that regularly occur (the first seven uplink messages in Table 1). In our framework, other uplinks like `Security Mode Reject`, `Authentication Failure`, and `Deregistration Request` are also supported. However, since these messages lead to disconnection, there is no subsequent UE response for assessment.

Similarly, we determined that 11 downlink messages are appropriate for our testing, resulting in a total of 77 different possible sequences. These sequences vary; some are considered valid according to the protocol, while others are not. Investigating how a UE responds to both valid and invalid sequences presents an intriguing aspect of our study, potentially revealing insights into UE behavior under various conditions.

Design and Capabilities of the Implemented Tool. The test case generation software is built around a dynamic parameter selection mechanism, utilizing a combination of rule-based systems and algorithmic checks to cover a wide range of test scenarios. We have developed a Graphical User Interface (GUI) using the `Tkinter` Python library, which creates a user-friendly graphical interface. This interface facilitates interaction between the software's users and its backend logic. The GUI module is organized into distinct sections for selecting uplink and downlink messages, displaying associated parameters, and configuring options for test generation, as illustrated in Figure 5 (in the Appendix). This modular approach in the GUI design ensures dynamic updates of each component based on user interactions and preferences. For example, when a downlink message is selected, the parameters section is automatically populated with checkboxes for each available parameter of that message. Users can easily select or deselect any parameter, allowing for customized test case generation. If a mandatory parameter is not selected, Open5GS resorts to using the default value.

The test-case-generating module is encapsulated within an algorithm that takes into account user-selected parameters, their possible values, and the chosen mode of test generation (randomized or all-inclusive). This algorithm employs a mix of combinatorial logic and randomness to ensure a thorough exploration of the test space. It operates in dual mode: by default, it explores all possible subsets of the selected parameters through a combinatorial approach to maximize coverage, incorporating randomness to enhance the robustness of the generated test cases. Conversely, when the *use all selected params* option is activated, the algorithm shifts to a deterministic mode, using every selected parameter to create specific testing scenarios tailored to the user's needs. Notably, each parameter has several potential values listed in the codebase, with only one value selected randomly for each test case.

Based on the given *Number of tests* the program randomly generates that amount of test cases. To ensure reproducibility, a specific *seed* number can be utilized. This feature allows the software to consistently regenerate the exact test case or the entire suite of tests in the future. Some NAS messages, as specified, are security-protected, whereas others are not. The modified Open5GS, however, permits sending a security-protected message as either plain or unprotected. This functionality serves as an effective method for assessing UE security. Hence, in the test case generator, we include the option to *send as a plain message*, which inserts a security disable instruction into the test case. Consequently, instead of sending a security-protected message, Open5GS will transmit that downlink message as a plain one. Test cases are saved in `JSON` format. Figure 6 (in the Appendix) displays a sample test case, and Table 5 (in the Appendix) lists the supported downlink NAS messages and their parameters.

5 AUTOMATED TEST CASE EVALUATION

5.1 LLM-Based Evaluation

LLMs are advanced artificial intelligence systems designed to understand and generate human-like text or programming code. These models leverage deep learning techniques and are trained on vast amounts of diverse data, including 3GPP documents. Generally, these documents are very large in size (e.g., sometimes thousands of pages), and information regarding particular issue is sometimes scattered across different documents. Retrieving information about a specific issue from these documents can be cumbersome, and hectic, and may result in human errors. Researchers in the wireless security domain have started using LLMs to address these challenges [5, 20, 28]. However, their aims and approaches are quite different to ours. For example, [20] attempts to find vulnerabilities in cellular network protocols by synthesizing finite state machines using natural language processing. [5] utilize various variants of BERT [11] to identify the 3GPP specification categories with their corresponding working groups. [28] develops a conversational artificial intelligence tool for synthesizing information on wireless communication specifications, using ChatGPT, which provides better explanations for queries related to wireless communication. Our objective is to evaluate UE responses for different test-case scenarios. Here, we describe our approach.

5.1.1 ChatGPT. As a state-of-the-art language model developed by OpenAI [33], ChatGPT was trained with approximately 175 billion parameters. First, we checked whether ChatGPT has knowledge of 3GPP documents by asking random questions about the NAS protocol and requesting it to copy a paragraph from a specific page number. Although the answer was not perfect, it provided a satisfactory level of response. ChatGPT works based on prompts, which refer to the input or query provided by the user to the model. Initially, we generated a prompt using a Python program, describing the purpose of the test. The prompt includes crucial information such as UE state, downlink message parameters, UE's response, etc. To make the scenario clearer, we converted the content of the pcap file to a text-based format (pre-processing, see Figure 4 in the Appendix) and finally delivered the full prompt to the ChatGPT 4.0 model using the Python API. After processing, ChatGPT provided its evaluation, including the reasoning for the answer.

5.1.2 Google Bard. Google Bard [15] is an LLM offered by Google. It is based on the *PaLM 2* model, a more current updated model. It not only provides insights based on the latest advancements in the 5G domain but also has the capability to incorporate information directly from the web. Given the absence of public APIs, we used a third-party Python library [36] from GitHub to engage with Google Bard, simulating API interactions through session cookies. This method allowed us to query Bard and receive responses, effectively bypassing the lack of direct API access. In our implementation, we utilized specific *cookie* values (`Secure-1PSID`, `Secure-1PSIDTS`, `Secure-1PSIDCC`) to establish a session with Bard. The very same prompt format that was used for ChatGPT was given to Google Bard as well.

5.1.3 LLaMA-2. LLaMA-2 [31] is open-source and deployable on our hardware directly from Hugging Face, following the acquisition of a non-commercial license from Meta. We ran the LLaMA-2 language model on a local machine equipped with Python 3.11 to initiate our process by cloning the LLaMA-2 repository. After building the `llama.cpp` repo, we executed the `download.sh` script, specifying the download of the `llama-2-70b.chat` parameters. Our choice to use the chat-optimized model was driven by the aim of achieving results similar to those of conversational models like Bard and ChatGPT. This decision was particularly pertinent given our need for natural language explanations alongside test evaluations. We activated a Python virtual environment for LLaMA-2 to isolate project tools and prevent version control conflicts with other projects. We then converted both models to the f16 format and created quantized models, `ggml-model-q4-0.bin`, to reduce the size further. Adapting the Python script used previously for Bard and ChatGPT, we included LLaMA-2. The script was designed to send detailed prompts to LLaMA-2 and parse its responses.

5.2 Rule-Based Evaluation

We created a Python-based test case evaluation program. First, we generated approximately 26,000 test cases for 77 possible sequences (7 uplink × 11 downlink), the number of test cases depends on the parameters (listed in Table 5). We then executed those for two UEs, namely Huawei and Oneplus. The execution program saved all pcap files. Since some of our test cases try to violate the protocol, not all of them are executed (approximately 80% were executed). From the pcap files, initially, we manually checked a few hundred to retrieve UE responses against the test cases. After observing the pcap file structure and how the test case and UE response are related, we developed a Python script to retrieve UE responses from all pcap files. The unique UE responses are limited in number. For example, we have identified three possible UE responses to identity requests: Identity response, Deregistration request (UE originating), or no response at all. Thus, we developed a UE response database and grouped them according to the sequences. During the evaluation, this database helps to determine whether our program has experience with that UE response or not. If the response is found in the database, then it is confirmed that our program evaluation program has the evaluation logic for that. If not, then the evaluation program reports that as a new observation and instructs to check the result manually. In reality, almost all the time the UE response is found in the database since we listed all possible types of responses by running a large number of test cases on two different UEs. However, we have kept the database comparison option open to handle any unexpected situations, such as if a UE behaves very differently due to design or implementation differences.

We thoroughly examined all types of UE responses (i.e., UE's uplink messages against manipulated downlink messages with manipulated parameters) and tried to find out what factors affect them. Some downlink parameters are associated with security, while others are not. For example, during the registration process, if the system is congested, the AMF may send `Registration Reject` with timer T3346 value instructing UE to wait until the timer expiry before the next registration attempt. In this case, the downlink parameter T3346 is merely an instruction to the UE and is not related to security, so changing this timer value does not impact the UE response. However, in the `Security Mode Command`, the integrity algorithm and encryption algorithm, or in each downlink message, the security header type is directly associated with the UE response.

Our examination revealed which parameters impact the UE response, helping us develop the logic of evaluation. Besides downlink parameters, the current state of the UE also impacts the UE response. For example, if a UE does not complete registration, it cannot accept a `Deregistration Request` from the AMF. We thoroughly examined all these impacting scenarios and developed the evaluation program. We were particularly interested in uncovering cases where the UE could be susceptible to privacy violations (revealing sensitive information, such as IMSI), authentication-encryption-integrity-security violations, and denial of service.

The program requires a test case file and the corresponding pcap file to determine whether a UE passes or fails a test. Before each test is executed, the core, the gNodeB, and the UE are started from scratch. Thus, from the test case file's uplink and downlink commands, it is possible to determine the state of the UE. Then, downlink parameter values are extracted from the pcap file using *Tshark*. These parameter values can be extracted from the test case file; however, if a test case does not contain some mandatory parameter fields of a downlink message (i.e., not part of that test), Open5GS uses the default value. So, extracting the parameters from the pcap file is better. Next, it extracts the UE response.

The positive responses are more impactful than the non-positive responses. For example, in the case of the `Security Mode Command`

Figure 3: Experimental Setup

downlink message, a UE can respond with "Security mode rejected (unspecified)", "Security mode reject (UE security capabilities mismatch)", "Security mode complete", or not respond at all. Among these, only "Security mode complete" indicates that the UE accepted all the parameters given in the test case, making it a good candidate for checking UE security. However, for non-positive responses, it is still possible to check whether the UE handled the message integrity or security header correctly or considered the current connect state.

For each type of downlink message, the program checks whether the UE response complies with the protocol. The program also considers other security-related parameters, such as security header type, ABBA, integrity and ciphering algorithm, etc. After checking all the conditions, the evaluation program decides on whether a UE passes or fails the test, providing a brief explanation that we coded in the program. If a decision cannot be made (due to a corrupted pcap file, missing uplink or downlink commands, unforeseen UE response, or any other unexpected situation), it reports the decision as inconclusive and includes a remark to check manually. The program delivers a final report in PDF form, listing the test name, test content, UE response, test status, and remark.

6 EXPERIMENTS AND RESULTS

6.1 Experimental Setup

The test bed consists of a Lenovo ThinkPad E15 equipped with 38GB of RAM and an Intel Core i7-1255U processor, running Ubuntu 20.04, a USRP B210, and five 5G SA-supported UEs, as described in Table 3. The automated test case executor runs the Open5GS core and the srsRAN's gNodeB in the same device, with a few seconds delay to allow synchronization. To eliminate all sorts of interference, the tests were conducted inside a Faraday cage containing only the UEs and USRP antennas. Our custom-made Faraday cage has RF-protected USB and antenna connectors. The UE was connected to the laptop using a USB 2.0 connection, while the USRP used a USB 3.0 connection.

We used Sysmocom SJA2-type USIMs for the UEs. The USIMs come with default settings, which might not be valid for a 5G SA connection. To ensure that all settings are correct, we used a USIM programming tool named PySim [34] to update the USIM service table, especially enabling service 124 to support SUCI and all the necessary 5G features.

The network was configured with PLMN ID 00101, so the USIMs were programmed accordingly, i.e., MCC equal to 001, and MNC to 01. If the USIM and network use any value other than 00101, the UEs refuse to connect to the network, which is a common issue for all open-source software suites [7, 40]. It should be noted that for framework evaluation purposes, 00101 PLMN is sufficient. However, for accuracy of the security results, commercial settings are required. Only one of our test UEs (Honor) established a connection with both the USIM side and network side using a value other than 00101. To address 5G SA connection stability issues, it is advised to use a low value for both uplink and downlink Modulation and Coding Scheme (MCS). Our connection remained stable with any chosen value. Additionally, we utilized a GPS-disciplined oscillator (GPSDO) with the USRP which created a GPS lock for a more accurate clock reference. Figure 3 shows our experimental setup.

Table 2: Performance analysis of the test case generator

# Tests	Scenario	Execution Time (in seconds)	Function calls
1	Without Additional Options	0.002	957
	With Plain Message Option	0.002	1021
	With All Parameters Option	0.002	843
100	Without Additional Options	0.106	51789
	With Plain Message Option	0.125	91777
	With All Parameters Option	0.097	87149
1000	Without Additional Options	1.214	875733
	With Plain Message Option	1.070	920825
	With All Parameters Option	0.845	874004

Table 3: 5G SA devices and their specifications

Device	Chipset	OS	Model	Release
Honor X9a 5G	Snapdragon 695	Android 12	RMO-NX1	2023
Huawei P40 Pro 5G	Kirin 990 5G	Android 10	ELS-NX9	2020
Oppo Reno8 Z 5G	Snapdragon 695 5G	Android 13	CPH2457	2022
Realme 8 5G	Dimensity 700	Android 11	RMX3241	2021
OnePlus Nord 2 5G	Dimensity 1200 5G	Android 11	DN2101	2021

6.2 Performance Analysis of Automated Test Case Generation

We executed a performance analysis of the test case generator on the same device. We used the `cProfile` module, focusing on generating test cases for the downlink message `Security Mode Command`. This message was chosen due to its complexity, offering 20 selectable parameters. We observed distinct performance impacts when enabling options like *Send as a plain message* and *Use all selected params*, as described in Table 2. For instance, generating a single test case demonstrated baseline performance metrics. Enabling *Send as a plain message* slightly altered performance, indicating minimal overhead for this security feature. However, activating *Use all selected params* showcased the tool's capability to handle full parameter sets without significant performance degradation. Scaling the number of test cases to 100 and 1000 revealed the tool's scalability, with marginal performance differences between the configurations. This scalability is crucial for extensive testing sessions, demonstrating the tool's capability to handle large-scale testing demands without significant delays. Through this analysis, we affirm the tool's robustness and flexibility in generating a broad range of test cases.

6.3 LLM-Based Evaluation Results

LLMs provide evaluations of the test based on a given prompt. Using the prompt, through API, we queried LLMs whether the UE passed the security test or not. Sometimes, LLMs might not have a definite answer, so we instructed the LLMs that if they cannot provide a definite answer, they should report it as inconclusive. In any case, for all answers, we requested a brief explanation to support their response. To check the quality of the evaluation, we reviewed a few hundred responses from LLMs manually. Figure 4 shows an example of a prompt and ChatGPT answer.

Results. We found that LLMs provide decent answers for straightforward test cases. For example, for identity-revealing or null integrity test cases, all LLMs were able to identify the correct UE behavior. Consequently, we carried out more advanced experiments, in which we designed numerous malicious and complicated test cases that could lead to vulnerabilities, such as `Security Mode Command` with plain header and GSM support in replayed services. We observed that LLMs were unable to perform better when the test cases involve complicated parameter changes not mentioned in the specifications, and when multiple security parameters violations occur in the same test. Additionally, we observed that LLMs have the tendency to fabricate answers, which is not uncommon for probabilistic models, given the perplexed task which requires advanced logic. To cross-validate the results, we requested the LLMs to mention the section number of the protocol document from where answer derives from. In many cases, the section number and the text were not present in the actual document.

Accuracy. According to our evaluation of LLMs' performance, the answers provided by ChatGPT, Bard, and LLaMA-2 were correct approximately 59%, 57%, and 51% of the time, respectively. Additionally, there were instances where the answer was correct, but the explanation was inadequate or incorrect. A better training on protocol-related documents might help improve the results. However, foundation LLMs do not offer an option for training the model due to concerns about potential misuse, copyright violation, ethical considerations, etc. Instead, they provide a fine-tuning option (typically for private use without sharing option) where the model's output can be refined. Therefore, we thoroughly fine-tuned the ChatGPT model with a few hundred test case evaluations in our evaluation. In some cases though, we observed that, due to fine-tuning, the model can lose its 'creativity', meaning it provided the same textual output that was used during the fine-tuning process, with limited contribution from its learning process with the specifications. Additionally, when presented with unfamiliar test cases, it was unable to provide satisfactory responses.

Current limitations. Our attempt to train LLaMA-2 faced significant challenges, primarily due to the extensive time required for the training process and system limitations that led to crashes. Another limitation of LLMs is inconsistent responses; for the same prompt, the model may provide different responses at different times. To address this inconsistency, LLMs use an option called temperature, a hyperparameter ranging from 0 to 1 that controls the randomness of the model's output. During our experimentation, we noticed that if the temperature is set too low, there is less randomness in the answer, and vice versa. We typically opted for a temperature value of 0.5 to strike a balance. Due to all the mentioned limitations, we conclude that LLMs are currently performing poorly when it comes to complex test cases and unknown vulnerabilities.

The LLM-based evaluation is not the primary assessment module within our framework; rather, it functions as a connected component only. Our developed rule-based evaluation provides reliable and superior performance, as we describe in the next subsection.

6.4 Rule-Based Evaluation Results

Our rule-based evaluation operates by examining the uplink and downlink commands from the test case file. For each test case, the system starts fresh, with the uplink/hooking point indicating the current UE state. Then, the program checks the generated pcap file, identifying the uplink message first, followed by the downlink message, and then determines the immediate next uplink NAS message or UE response. The UE response is influenced by several factors such as UE state, uplink and downlink messages, downlink message security, and downlink parameters. For example, if the uplink command is `Registration Request` and the downlink command is `Identity Request`, when the downlink message is set as a plain message, the possible UE response could be `Identity Response` or no response at all depending on the downlink parameters (e.g., requested identity type in the downlink parameter such as SUCI, GUTI, S-TMSI, etc.) and the message sequence. For this particular scenario, the evaluation program does not expect any other types of UE response such as `Authentication Response` or `Security Mode Complete`. Consequently, the rule-based methodology knows the correct sequences and parameters at each stage. In cases where unexpected results are recorded, the program will detect that as an unusual test case and flag it for manual checking. Therefore, for each type of the mentioned 77 uplink and downlink message pairs, we implemented the state logic based on them, and the program makes decisions from the UE output. By automating this process, we can easily run test cases related to well-known vulnerabilities too, without the need of manual inspection. Generally, we conducted thousands of test cases for each UE, and we are summarizing the primary findings below.

UE Identities. For the downlink `Identity Request` the evaluation program found none of the tested UE revealed their IMEI or any identity other than the SUCI before the AKA was completed, which indicates that they passed the test. However, it was detected that our Huawei device provides SUCI identity for two occasions, when the identity octet is 000, meaning unused identity type or identity not available at the network side, and the identity octet is 001 meaning requested identity type SUCI (more details in [1, in Table 9.11.3.3.1]). The rest of the tested UEs responded with a SUCI identity only for the identity with octet 001. According to the

Table 4: Performance analysis of the framework

Downlink NAS message	Test case run per UE per uplink	Generation time per test case	Execution time per test case	Evaluation time per test case				Evaluation accuracy			
				ChatGPT	Bard	LLaMA-2	Rule-Based	ChatGPT	Bard	LLaMA-2	Rule-Based
Identity Request	200	0.002	12.50	10.75	7.56	210.85	0.29				
Authentication Request	400	0.002	13.03	10.38	7.62	217.81	0.70				
Security Mode Command	900	0.002	13.10	10.55	8.34	208.18	0.87				
Configuration Update Command	300	0.002	12.38	11.20	7.59	204.50	0.28				
Service Accept	400	0.002	12.50	10.08	8.64	209.02	0.16				
Service Reject	400	0.002	13.10	10.25	8.29	213.41	0.30	≈ 59%	≈ 57%	≈ 51%	100%
GMM Status	200	0.002	13.40	11.29	8.45	210.93	0.27				
Deregistration Accept	200	0.002	13.23	10.50	7.81	211.87	0.15				
Deregistration Request	400	0.002	12.49	10.23	8.06	205.95	0.38				
Authentication Reject	200	0.002	13.20	10.44	8.16	213.21	0.16				
Registration Reject	300	0.002	12.41	11.08	7.33	213.81	0.29				

Note: All units of time are in seconds

protocol, 000 is not a listed value and shall be interpreted as SUCI, if received by the UE. Therefore, it clearly shows that except for Huawei all other devices fail to adhere to the protocol.

Furthermore, the AMF can request the identity from a UE in plain messages until the security establishment. Therefore, technically, the downlink command `Identity Request` in plain messages, after the uplink NAS messages `Registration Request`, `Identity Response`, and `Authentication Response` should be considered valid. Our test case evaluator detected that except for the Realme phone, all UEs provided the SUCI identity when requested after the aforementioned three uplink commands. Realme only provided the SUCI identity when it was requested after `Registration Request` and did not respond in the other two cases. Although there are no severe security ramifications, the evaluation program detected different behavior among the UEs.

Anti-Bidding-down Between Architectures. ABBA is used in `Authentication Request` and `Security Mode Command`, standing for Anti-Bidding-down Between Architectures. This parameter enables the 5G system to enforce that a UE cannot access the network using older mechanisms that have had vulnerabilities associated with them. The 5G security architecture [2, in Annex A.7.1] defines this value as hexadecimal `0000`. However, in future releases, this value could be changed depending on security implementation. Currently, UEs need to ensure that the value is zero in order to attach to the 5G system [3]. The evaluation program detected that all UEs provided a positive `Authentication Response` even though the ABBA value was set to arbitrary values. Additionally, the NAS protocol [1, in section 9.11.3.10] states that if the UE receives an ABBA value different from 0000, the UE shall use that value. Since these values are not defined yet, the UE does not know which security is associated with this ABBA. Therefore, it can be concluded that currently, all UEs are accepting 'temporary' ABBA values, as implementations are still under development. Therefore, the contribution of this value to improving UE security at this moment is practical indeterminate and minimal.

Security Headers. The security header specifies the type of security applied to the NAS message. The specification [1, in 9.3] outlines five types of headers. These headers mainly indicate the type of security associated with the message. We generated test cases in which the downlink message contains the wrong security headers. For instance, the `Configuration Update Command` uses header type 2, but the test case selected a different header undermining security. The goal is to force the test UE to accept wrong headers, leading to potential security deactivation or implications in the already activated security context. In the experiments, we observed that if the `Security Mode Command` is sent after `Security Mode Complete` (repeating), OnePlus accepts the `Security Mode Command` for security header types 1 and 3. However, in practice, the `Security Mode Command` typically uses header type 3 in the normal sequence. The remaining UEs either responded with `Security Mode Reject (unspecified)` or did not respond at all. For all other downlink NAS messages, all the UEs responded only when the correct security header type for that message was used.

Out-Of-Sequence Messages. According to NAS specification, before security context establishment, some downlink NAS messages are to be processed by the UE even if those are without integrity protection [1, in 4.4.4.2]. Some of our test cases invoked such a scenario by sending a plain `Identity Request` message after `Registration Complete`. The evaluation program found that Huawei, Oppo, and Realme handle this situation by either not responding at all or by deregistering from the network. However, OnePlus and Honor exhibited different behavior. They did not provide the identity but initiated either a `PDU Session Establishment Request` or `Service Request`. Subsequently, the service request was rejected by the AMF with the cause stated as *UE identity cannot be derived by the network*. This demonstrates the varied approaches of UEs in handling such situations. Then we also checked the reverse scenario by sending a security-protected type message before AKA completion, for example, by sending a downlink command such as `Configuration Update Command` or `Deregistration Request` after the initial `Registration Request`. We tested all the UEs for this type of scenario. Our evaluation program did not observe any security concerns during these tests as all the UEs handled such situations by either not responding, getting deregistered, or, in some cases, providing a 5GMM cause of *Message not compatible with the protocol state*.

Other Parameters. For the tests containing incorrect or invalid RAND, AUTN, or ngKSI values, almost all devices behaved securely. They either did not respond at all, silently deregistering from the network, or provided a 5GMM cause, such as *ngKSI already in use* or *non-5G authentication unacceptable*. Similarly, for malicious `Security Mode Command` with NIA0 (null integrity protection), all devices either responded with a `Security Mode Reject` or silently discarded the message, indicating passing the test.

6.5 Framework Performance

Table 4 displays the overall performance of our framework. The number of test cases depends on the parameters (listed in Table 5). For each uplink, we generated the following numbers of downlink messages and ran the test cases. Approximately 26,000 test cases were run for each UE. Since the test cases sometimes violate the protocol, not all of them get executed (approximately 80% of them are executed). It took on average 0.002s to generate a test case. For the test case execution, each time the core and RAN needed to be restarted, and the UE needed to be toggled from airplane mode, which required an average of 13s for each test case. ChatGPT and Bard are backed by powerful computation resources providing evaluation within approximately 10s. However, LLaMA-2, running on our server, took approximately 4 minutes to deliver an evaluation. The rule-based evaluation shows the best timing performance by providing the evaluation in less than a second. Due to the generative nature of LLMs, there was a lack of consistency over time. There were instances where the answer was correct, but the explanation was poor or incorrect. Nevertheless, we believe that proper training may help improve the scenario. In contrast, the rule-based evaluation, which followed our given logic, resulted in almost all evaluations being correct upon cross-checking.

7 DISCUSSION

7.1 Challenges & Lessons Learned

Connectivity. During our experiments, we discovered that connecting a COTS 5G UE to a custom 5G network is not always straightforward. In our setup, five out of eight 5G SA UEs successfully connected to the network, while the remaining three failed to initiate a Random Access procedure. We utilized a GPSDO for improved clock accuracy, a Faraday cage to mitigate interference, and experimented with changing critical parameter values. However, some UEs were still reluctant to connect to our (open-sourced software) 5G network. Carrier policies may also restrict or limit access to the 5G network based on the PLMNs. We found that Android maintains a list of carriers [4], which is intended to be saved inside the operating system. Choosing any of them during USIM programming helps the connection to some extent. When the USIM was programmed with one of the listed PLMNs, it enabled the 5G network mode. This also helped the UE to fetch access point-related information from the OS. Unfortunately, even with commercial settings, the lack of security keys can limit the evaluation reaching up until the `Security Mode Command` during the registration process. Regarding UE connection to the laptop, though USB 3.0 is backward compatible with USB 2.0, some UEs did not show a workable connection if USB 3.0 was used. Additionally, some UEs showed unstable connection due to power-related issues, using a powered USB hub solved that problem. The Scrcpy [14] library can be used to see the UE display via USB connection when the UE is inside of a Faraday cage.

Framework. The test case generator can produce a large number of test cases; however, not all test cases exhibit the same level of quality and effectiveness. Therefore, it is up to user to decide which type of test case to generate, as a massive generation of test cases may result in redundancy. After many trials and errors, we determined the optimal time interval between two successive test case executions. This is crucial because, after each test case, the entire system needs to be torn down and deployed again. Open5GS has several services that utilize different ports, requiring us to terminate all services and close all ports. Under optimal settings, an average of four test cases can be executed per minute.

Test case evaluation is not a straightforward task; many parameters and conditions need to be checked. It is very common that for some test cases, the UE does not respond at all. In such a scenario, we consider the UE to have passed the over-the-air test, but internal firmware checks may also be needed for conclusive answers. We do not deploy such checks, when the UE silently ignores a downlink message, as access to the firmware incurs additional complexity and its out-of-scope in this work. If, for some reason, the evaluation program is unable to reach a concrete decision for any test result, it reports that as inconclusive, and some simple manual checks may be required.

7.2 Future Work

Our framework is designed to support 5G control-plane testing. However, currently, it supports only the NAS layer. In the future, we plan to extend to the RRC layer as well. Our framework is capable of conducting tests in various UE states (e.g., registered, deregistered, and CM connected-idle) that occur during the normal flow. Nonetheless, in the future, we will investigate changing the UE's state by accommodating more complicated aspects of security testing (i.e., multiply hooked downlink messages) to advance beyond the scope of the initial UE attachment. The performance evaluation of LLMs was not impressive. It clearly shows that out-of-the-box LLMs are not a good choice for this type of security testing evaluation. Foundation LLMs (such as ChatGPT and Google Bard) are intelligent and powerful but do not have a sharing option. Therefore, we plan to train a LLaMA-2 model for better performance.

8 CONCLUSION

In this paper, we present the first over-the-air *automated* 5G SA security testing framework for 5G SA UE. The framework consists of full automation packages where test cases for the NAS layer can be generated, executed, and evaluated in an automatic manner. The developed tool can help save a significant amount of time to conduct these labor-intensive tasks and reduce human error. We successfully applied our testing framework to five 5G SA UEs and reported identified flaws. The comparative test results clearly show that different UEs handle the security issue differently. We also provide our experience and lessons learned from the experiment, which could be useful for the research community.

ACKNOWLEDGEMENTS

This work was supported by the Center for Cyber Security at New York University Abu Dhabi (NYUAD), the National Science Foundation under grant number 2144914, and a gift from Google.

REFERENCES

[1] 3GPP. 2023. *5G; Non-Access-Stratum (NAS) protocol for 5G System (5GS); Stage 3.* 3rd Generation Partnership Project (3GPP). Version 17.9.0.

[2] 3GPP. 2023. *5G; Security architecture and procedures for 5G System.* 3rd Generation Partnership Project (3GPP). Version 17.8.0.

[3] 5G America. 2020. Security considrations for the 5G era. https://www.5gamericas.org/wp-content/uploads/2020/07/Security-Considerations-for-the-5G-Era-2020-WP-Lossless.pdf.

[4] Android. 2023. Android carrier list. https://android.googlesource.com/platform/packages/providers/TelephonyProvider/+/refs/heads/main/assets/sdk33_carrier_id/carrier_list.textpb.

[5] Lina Bariah, Hang Zou, Qiyang Zhao, Belkacem Mouhouche, Faouzi Bader, and Merouane Debbah. 2023. Understanding Telecom Language Through Large Language Models. arXiv:2306.07933

[6] Evangelos Bitsikas, Syed Khandker, Ahmad Salous, Aanjhan Ranganathan, Roger Piqueras Jover, and Christina Pöpper. 2023. UE Security Reloaded: Developing a 5G Standalone User-Side Security Testing Framework. In *Proceedings of the 16th ACM Conference on Security and Privacy in Wireless and Mobile Networks (WiSec '23).* ACM, New York, NY, USA, 121–132. https://doi.org/10.1145/3558482.3590194

[7] Open Cell. 2023. Commercial UEs tests. https://open-cells.com/index.php/2022/07/26/ues.

[8] Yi Chen, Di Tang, Yepeng Yao, Mingming Zha, Xiaofeng Wang, Xiaozhong Liu, Haixu Tang, and Dongfang Zhao. 2022. Seeing the Forest for the Trees: Understanding Security Hazards in the 3GPP Ecosystem through Intelligent Analysis on Change Requests. In *31st USENIX Security Symposium (USENIX Security 22).* USENIX Association, Boston, MA, 17–34.

[9] Yi Chen, Yepeng Yao, XiaoFeng Wang, Dandan Xu, Chang Yue, Xiaozhong Liu, Kai Chen, Haixu Tang, and Baoxu Liu. 2021. Bookworm Game: Automatic Discovery of LTE Vulnerabilities Through Documentation Analysis. In *42nd IEEE Symposium on Security and Privacy, SP 2021, San Francisco, CA, USA.* IEEE, 1197–1214. https://doi.org/10.1109/SP40001.2021.00104

[10] Merlin Chlosta, David Rupprecht, Thorsten Holz, and Christina Pöpper. 2019. LTE security disabled: misconfiguration in commercial networks. In *Proceedings of the 12th Conference on Security and Privacy in Wireless and Mobile Networks (WiSec '19).* ACM, New York, NY, USA, 261–266. https://doi.org/10.1145/3317549.3324927

[11] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. arXiv:1810.04805

[12] Kaiming Fang and Guanhua Yan. 2018. Emulation-Instrumented Fuzz Testing of 4G/LTE Android Mobile Devices Guided by Reinforcement Learning. In *Computer Security - 23rd European Symposium on Research in Computer Security, ESORICS 2018, Barcelona, Spain*, Vol. 11099. Springer, Berlin, Heidelberg, 20–40. https://doi.org/10.1007/978-3-319-98989-1_2

[13] Matheus E. Garbelini, Zewen Shang, Sudipta Chattopadhyay, Sumei Sun, and Ernest Kurniawan. 2022. Towards Automated Fuzzing of 4G/5G Protocol Implementations Over the Air. In *GLOBECOM 2022 - 2022 IEEE Global Communications Conference.* IEEE, 86–92. https://doi.org/10.1109/GLOBECOM48099.2022.10001673

[14] Genymobile. 2023. Scrcpy. https://github.com/Genymobile/scrcpy.

[15] Google. 2023. Google Bard. https://bard.google.com.

[16] Grant Hernandez, Marius Muench, Dominik Maier, Alyssa Milburn, Shinjo Park, Tobias Scharnowski, Tyler Tucker, Patrick Traynor, and Kevin R. B. Butler. 2022. FirmWire: Transparent Dynamic Analysis for Cellular Baseband Firmware. In *Proceedings of the 29th Annual Network and Distributed System Security Symposium, (NDSS'22), San Diego, California, USA.* The Internet Society. https://doi.org/10.14722/ndss.2022.23136

[17] Syed Rafiul Hussain, Omar Chowdhury, Shagufta Mehnaz, and Elisa Bertino. 2018. LTEInspector: A Systematic Approach for Adversarial Testing of 4G LTE. In *Proceedings of the 25th Annual Network and Distributed System Security Symposium, (NDSS'18), San Diego, California, USA.* The Internet Society. https://doi.org/10.14722/ndss.2018.23313

[18] Syed Rafiul Hussain, Mitziu Echeverria, Imtiaz Karim, Omar Chowdhury, and Elisa Bertino. 2019. 5GReasoner: A Property-Directed Security and Privacy Analysis Framework for 5G Cellular Network Protocol. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security (CCS '19).* ACM, NY, USA, 669–684. https://doi.org/10.1145/3319535.3354263

[19] Syed Rafiul Hussain, Imtiaz Karim, Abdullah Al Ishtiaq, Omar Chowdhury, and Elisa Bertino. 2021. Noncompliance as Deviant Behavior: An Automated Black-Box Noncompliance Checker for 4G LTE Cellular Devices. In *ACM SIGSAC Conference on Computer and Communications Security* (Virtual Event, Republic of Korea) *(CCS '21).* ACM, New York, NY, USA, 1082–1099. https://doi.org/10.1145/3460120.3485388

[20] Abdullah Al Ishtiaq, Sarkar Snigdha Sarathi Das, Syed Md. Mukit Rashid, Ali Ranjbar, Kai Tu, Tianwei Wu, Zhezheng Song, Weixuan Wang, Mujtahid Akon, Rui Zhang, and Syed Rafiul Hussain. 2023. Hermes: Unlocking Security Analysis of Cellular Network Protocols by Synthesizing Finite State Machines from Natural Language Specifications. arXiv:2310.04381

[21] Bedran Karakoc, Nils Fürste, David Rupprecht, and Katharina Kohls. 2023. Never Let Me Down Again: Bidding-Down Attacks and Mitigations in 5G and 4G. In *Proceedings of the 16th ACM Conference on Security and Privacy in Wireless and Mobile Networks (WiSec '23).* ACM, New York, NY, USA, 97–108. https://doi.org/10.1145/3558482.3581774

[22] Imtiaz Karim, Syed Rafiul Hussain, and Elisa Bertino. 2021. ProChecker: An Automated Security and Privacy Analysis Framework for 4G LTE Protocol Implementations. In *41st IEEE International Conference on Distributed Computing Systems, ICDCS 2021, Washington DC, USA.* IEEE, 773–785. https://doi.org/10.1109/ICDCS51616.2021.00079

[23] Imtiaz Karim, Kazi Samin Mubasshir, Mirza Masfiqur Rahman, and Elisa Bertino. 2023. SPEC5G: A Dataset for 5G Cellular Network Protocol Analysis. arXiv:2301.09201

[24] Eunsoo Kim, Min Woo Baek, CheolJun Park, Dongkwan Kim, Yongdae Kim, and Insu Yun. 2023. BASECOMP: A Comparative Analysis for Integrity Protection in Cellular Baseband Software. In *32nd USENIX Security Symposium (USENIX Security 23).* USENIX Association, Anaheim, CA, 3547–3563.

[25] Eunsoo Kim, Dongkwan Kim, CheolJun Park, Insu Yun, and Yongdae Kim. 2021. BaseSpec: Comparative Analysis of Baseband Software and Cellular Specifications for L3 Protocols. In *Proceedings of the 28th Annual Network and Distributed System Security Symposium, (NDSS'21), San Diego, California, USA.* https://doi.org/10.14722/ndss.2021.24365

[26] Hongil Kim, Jiho Lee, Eunkyu Lee, and Yongdae Kim. 2019. Touching the Untouchables: Dynamic Security Analysis of the LTE Control Plane. In *2019 IEEE Symposium on Security and Privacy, SP 2019, San Francisco, CA, USA.* IEEE, 1153–1168. https://doi.org/10.1109/SP.2019.00038

[27] Daniel Klischies, Moritz Schloegel, Tobias Scharnowski, Mikhail Bogodukhov, David Rupprecht, and Veelasha Moonsamy. 2023. Instructions Unclear: Undefined Behaviour in Cellular Network Specifications. In *32nd USENIX Security Symposium (USENIX Security 23).* USENIX Association, Anaheim, CA, 3475–3492.

[28] Manikanta Kotaru. 2023. Adapting Foundation Models for Information Synthesis of Wireless Communication Specifications. arXiv:2308.04033

[29] Dominik Maier, Lukas Seidel, and Shinjo Park. 2020. BaseSAFE: Baseband Sanitized Fuzzing through Emulation. In *Proceedings of the 13th ACM Conference on Security and Privacy in Wireless and Mobile Networks (WiSec '20).* ACM, New York, NY, USA, 122–132. https://doi.org/10.1145/3395351.3399360

[30] Lusani Mamushiane, Albert Lysko, Hlabishi Kobo, and Joyce Mwangama. 2023. Deploying a Stable 5G SA Testbed Using srsRAN and Open5GS: UE Integration and Troubleshooting Towards Network Slicing. In *2023 International Conference on Artificial Intelligence, Big Data, Computing and Data Communication Systems (icABCD).* IEEE, 1–10. https://doi.org/10.1109/icABCD59051.2023.10220512

[31] Meta. 2023. LLAMA 2. https://llama.meta.com.

[32] Open5gs. 2023. Open5gs. https://open5gs.org.

[33] OpenAI. 2023. OpenAI. https://openai.com.

[34] Osmocom. 2023. PySim. https://osmocom.org/projects/pysim/wiki.

[35] CheolJun Park, Sangwook Bae, BeomSeok Oh, Jiho Lee, Eunkyu Lee, Insu Yun, and Yongdae Kim. 2022. DoLTEst: In-depth Downlink Negative Testing Framework for LTE Devices. In *31th USENIX Security Symposium (USENIX Security 22).* USENIX Association, Boston, MA, 1325–1342.

[36] MinWoo Park. 2023. Bard API. https://github.com/dsdanielpark/Bard-API.

[37] Srinath Potnuru and Prajwol Kumar Nakarmi. 2021. Berserker: ASN.1-based Fuzzing of Radio Resource Control Protocol for 4G and 5G. In *17th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob).* IEEE, 295–300. https://doi.org/10.1109/WiMob52687.2021.9606317

[38] David Rupprecht, Kai Jansen, and Christina Pöpper. 2016. Putting LTE Security Functions to the Test: A Framework to Evaluate Implementation Correctness. In *USENIX Workshop on Offensive Technologies* (Austin, TX) *(WOOT'16).* USENIX Association, USA, 40–51.

[39] srsRAN. 2023. srsRAN. https://www.srsran.com.

[40] srsRAN. 2023. Tested COTS UEs. https://docs.srsran.com/projects/project/en/latest/knowledge_base/source/cots_ues/source/index.html.

[41] Swind. 2020. Pure Python ADB. https://github.com/Swind/pure-python-adb.

[42] Haohuang Wen, Phillip Porras, Vinod Yegneswaran, Ashish Gehani, and Zhiqiang Lin. 2024. 5G-Spector: An O-RAN Compliant Layer-3 Cellular Attack Detection Service. In *Proceedings of the 31st Annual Network and Distributed System Security Symposium (NDSS'24), San Diego, California, USA.* The Internet Society.

[43] Chuan Yu, Shuhui Chen, Ziling Wei, and Fei Wang. 2023. SecChecker: Inspecting the security implementation of 5G Commercial Off-The-Shelf (COTS) mobile devices. *Comput. Secur.* 132 (2023), 103361. https://doi.org/10.1016/j.cose.2023.103361

APPENDIX

Table 5: Downlink NAS messages with supported parameters

Downlink NAS message	Parameters
Identity request	identity type, security header type
Authentication request	abba, rand, autn, ngksi tsc, ngksi ksi, eap message, security header type
Security mode command	abba, ngksi tsc, ngksi ksi, eap message, imeisv request, security header type nas security integrity algorithm, nas security encryption algorithm selected nas eps security algorithm, replayed UE security capabilities nr ea replayed UE security capabilities nr ia, replayed UE security capabilities eutra ea replayed UE security capabilities eutra ia, replayed UE security capabilities gea additional security information derivation, replayed s1 UE security capabilities nr ea replayed s1 UE security capabilities nr ia, replayed s1 UE security capabilities eutra ea replayed s1 UE security capabilities eutra ia, additional security information retransmission
Deregistration request	T3346, gmm cause, rejected nssai, security header type, deregistration type tsc, deregistration type ksi deregistration type switch off, deregistration type access type, deregistration type reregistration required
Service reject	T3346, T3448, GMM cause, eap message, PDU session status, security header type
Service accept	T3448, eap message, PDU session status psi, PDU session reactivation result psi PDU session reactivation result error cause
Configuration update command	SMS indication type, security header type, network daylight saving time
Registration reject	T3346, T3502, eap, nssai, GMM cause, security header type
GMM status	GMM cause, security header type
Deregistration accept	security header type
Authentication reject	security header type

Prompt: This is a security test for a UE. The UE sent a registration request to the AMF. Then the AMF sent an identity request to the UE. The downlink message sent by the AMF was a plain 5GS NAS message, with the requested identity type = IMEI. Upon receiving the downlink message, the UE did not respond. At that time, AKA was not completed. The packet capture during the test is as follows:
0.0, UE to AMF, InitialUEMessage, Registration request, Registration request
0.0012581348419189453, AMF to UE, SACK (Ack=0, Arwnd=106496), DownlinkNASTransport, Identity request
Did the UE pass or fail the test? If a decision cannot be made, report it as inconclusive. Always include the reason for your decision.
ChatGPT answer: Based on the provided information, the UE behavior appears to be secure. The UE did not respond to the identity request from the AMF because the AKA (Authentication and Key Agreement) was not completed. This is a security measure to prevent unauthorized access or manipulation of the UE's identity. The UE is correctly waiting for the completion of the AKA before responding to the identity request, which is a good practice in maintaining the security of the UE. Therefore, the UE passes the test.

Figure 4: A brief example of a prompt with the associated answer

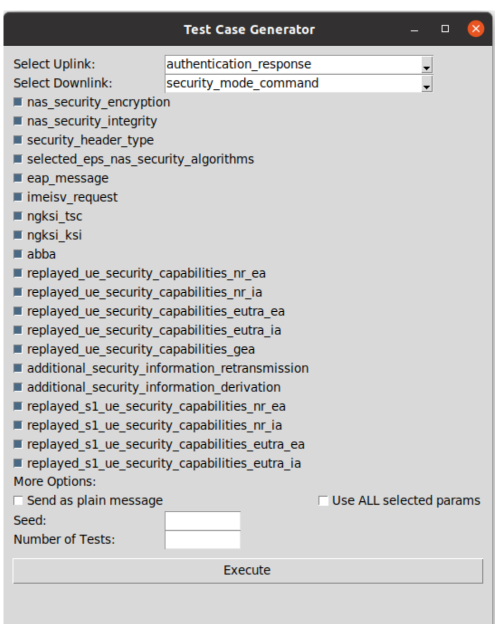

Figure 5: Our developed test case generator


```
[{
  "ue_ul_handle": "null",
  "dl_reply": "null",
  "command_mode": "null",
  "dl_params": "null"
},
{
  "ue_ul_handle": "authentication_response",
  "dl_reply": "security_mode_command",
  "command_mode": "send",
  "dl_params": {
    "ngksi_tsc": "1",
    "ngksi_ksi": "3",
    "abba": "0000",
    "nas_security_encryption": "OGS_NAS_SECURITY_ALGORITHMS_NEA3",
    "nas_security_integrity": "OGS_NAS_SECURITY_ALGORITHMS_NIA0"
  }
},
{
  "ue_ul_handle": "null",
  "dl_reply": "null",
  "command_mode": "null",
  "dl_params": "null"
}]
```


Figure 6: An example test case