OTABase: Enhancing Over-the-Air Testing to Detect Memory Crashes in Cellular Basebands

CheolJun Park*, Marc Egli†, BeomSeok Oh‡, Tuan Dinh Hoang‡, Suhwan Jeong‡, Martin Crettol†,
Insu Yun‡, Mathias Payer†, and Yongdae Kim‡
**Kyung Hee University, Korea, Email: cheoljunp@khu.ac.kr*
†*EPFL, Switzerland, Emails: {marc99.egli, martin.crettol}@gmail.com, mathias.payer@nebelwelt.net*
‡*KAIST, Korea, Emails: {beomseoko, tuan.hoangdinh, korea.1nteger, insuyun, yongdaek}@kaist.ac.kr*

Abstract—**Baseband processors (BPs) in cellular devices implement complex radio protocols, and memory corruption vulnerabilities in these implementations can lead to critical security breaches, including remote code execution. Traditional approaches to detecting such vulnerabilities rely on reverse engineering or emulation. However, these methods face significant scalability challenges due to proprietary firmware and architectural complexities. Over-the-air (OTA) testing offers broader applicability but poses challenges in managing UE state, detecting crashes, and ensuring protocol coverage.**

We present OTABase, an OTA testing framework that enables efficient detection of memory crashes in LTE basebands by leveraging protocol specifications. OTABase combines three key techniques: a network-side state control mechanism for efficient management of UE states and connections, a specification-guided test case generation targeting memory crashes in NAS and RRC protocols, and a two-phase crash detection oracle utilizing protocol-based liveness checks and manufacturer debug features. Evaluating OTABase on six commercial BPs from three major manufacturers, unearthed seven previously unpatched memory crashes. Among these, three were assigned CVEs, including one out-of-bounds write vulnerability that allows remote code execution. Additionally, we extend OTABase to 5G basebands for PoC, demonstrating its generalizability and practical utility.

1. Introduction

Cellular communication is deeply embedded in mobile devices and critical infrastructure, making its security paramount. At the core of this attack surface lies the BP, which manages radio signals and data packets for cellular services (e.g., SMS, voice calls, and internet). To operate securely, the BP depends on cellular control protocols for tasks like authentication, connection management, and security protections. Thus, vulnerabilities in these protocols can directly compromise end-user security.

To identify these flaws, researchers have proposed various approaches, including black-box testing [1], [2], [3], [4], [5], static analysis [6], [7], emulation [8], [9], natural language processing [10], [11], comparative analysis [12], [13] and formal analysis [14], [15]. Most of these approaches focus on two main categories of vulnerabilities. First, design flaws embedded in the standard itself—-such as exposure of sensitive data (*i.e.* identity, location) in plaintext [2], [16], [17], lack of spoofing defenses [2], [18], and weak authentication mechanisms enabling IP traffic manipulation [19], [20]. Second, implementation flaws, where the protocol behavior deviates from the specification. These can be further categorized into two types: logic bugs that violate the security policies outlined in the specification (*e.g.* incorrect validation of authentication messages), and memory bugs that result from unsafe memory operations.

Among the potential vulnerabilities in BPs, memory corruption bugs are particularly severe as they can lead to remote code execution (RCE) on smartphones. Since BPs are black-box, proprietary systems with limited debugging capabilities, detecting memory crashes serves as a practical approach to identify potential memory corruption vulnerabilities. This is especially relevant due to three key characteristics of current BP and mobile device architectures. First, BPs are typically developed using memory-unsafe languages (C/C++). Second, the shared memory design and inter-process communication mechanisms within mobile devices [21] allow vulnerabilities in the BPs to propagate across system boundaries. Third, the lack of robust memory protection mechanisms like ASLR [22], [21] simplifies exploitation. These combined factors enable attackers to escalate privileges within the mobile OS, as demonstrated repeatedly by baseband exploitation research [23], [24], [25], [26], [22]. For instance, Cama [27] successfully demonstrated a full-chain exploitation, achieving arbitrary file write within the Android file system.

Given these threats, detecting memory crashes to identify potential memory corruption vulnerabilities in BPs is crucial. While traditional approaches rely on static analysis [27], [28], [22], [6] or emulation-based fuzzing [8], [9], they face scalability limitations. These methods require reverse engineering – obtaining firmware is often difficult, analysis demands substantial expertise, and the process must be repeated when implementations undergo major changes. These limitations have constrained research to older devices, with Samsung studies mostly stopping at S10 series, limited MediaTek coverage, and Qualcomm analysis being challenging due to Hexagon architecture.

While OTA testing offers a more universal approach that can work across different baseband implementations without firmware access, practical challenges remain. OTA testing is inherently slow, further constrained by limited testing windows due to autonomous state transitions and connection timeouts – for example, Park et al. [4] reported 70% of testing time as idle due to such constraints. Also, lack of instrumentation due to BP's black-box nature make conventional code coverage-based fuzzing inapplicable, which is further complicated by the large protocol surface – RRC and NAS contain dozens of message types with thousands of their fields. A recent attempt at specification-based fuzzing [29] still faces limitations in protocol coverage and only supports open-source UEs [30], [31], while their detailed methodology remains undisclosed and not open-sourced. Moreover, BP's limited debugging capabilities and less observable user interface feedback make detecting memory bugs challenging in OTA testing [32].

In this paper, we present OTABase, an over-the-air testing framework for detecting memory crashes in LTE cellular basebands. Our approach focuses on the NAS and RRC protocols, which are essential for cellular control operations, and combines three key techniques to overcome the aforementioned challenges. First, to overcome the challenges of coordinating UE state and timeouts (C1), we implement a network-side state control mechanism guided by protocol specifications, enabling precise UE state management and stable testing conditions. This reduces downtime by avoiding autonomous transitions and long back-off periods. Second, to handle the large protocol surface (C2), we introduce a specification-guided test case generation method which creates standard-conformant NAS and RRC messages and systematically mutates security-sensitive fields to target memory crashes. This allows broad testing coverage without requiring implementation details or instrumentation. Third, to overcome limited crash observability (C3), we develop a two-phase crash detection oracle combining protocol-level liveness checks, ADB crash logs with manufacturer-provided debug features. This ensures reliable detection by identifying candidate crashes through connection monitoring with ADB logs and then confirming them using the manufacturer's kernel panic logs.

We evaluated OTABase against six COTS basebands, both new and old models from each three major manufacturers (Qualcomm, Samsung, and MediaTek), discovering seven previously unpatched memory crashes. Among them, three were marked as duplicates upon reporting, and three were assigned CVEs. Notably, one CVE involves an out-of-bounds write that enables RCE. We also validated OTABase by detecting a known crash in an old device, confirming its ability to find both new and known vulnerabilities. All findings were responsibly disclosed to the respective vendors.

In summary, our contributions are three-fold:

- We introduce OTABase, an over-the-air testing framework designed for efficient detection of memory crashes in COTS LTE basebands. We release OTABase as open-source for future research at https://github.com/OTABase/OTABase.
- We develop efficient test generation, state control techniques by leveraging protocol specification to overcome traditional OTA testing limitations. Also, we present a two-phase crash detection mechanism to provide reliable crash detection.
- We demonstrate OTABase's effectiveness by discovering seven previously unpatched memory crashes in BPs from major manufacturers, three of which were assigned CVEs.

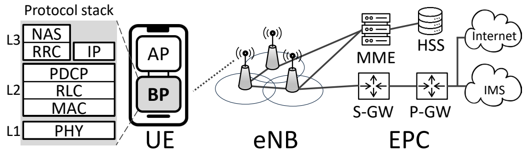

Figure 1: LTE network architecture

2. Background

2.1. LTE network architecture

LTE network comprises three components: User equipment, Evolved Node B, and Evolved Packet Core (Fig. 1).

User Equipment (UE) refers to the mobile device, such as a smartphone. UEs typically contain two main processors: an application processor (AP) running the operating system, and the baseband processor (BP), which handles cellular communications. The BP implements key protocols like RRC and NAS, which are essential for both communication and security. To access cellular services, the UE communicates with network entities over the wireless interface using the cellular protocol stack (§2.2).

Evolved Node B (eNB) acts as a base station, providing wireless connectivity to UEs. It handles both downlink (eNB to UE) and uplink (UE to eNB) transmissions using the Radio Resource Control (RRC) protocol, which manages radio connections. Additionally, it functions as a bridge, relaying control data between UEs and EPC.

Evolved Packet Core (EPC) functions as the backend of LTE network. Key components within the EPC include the Mobility Management Entity (MME), responsible for user authentication and session management via the Non-Access Stratum (NAS) protocol with UEs. Additionally, it manages user identifiers like the International Mobile Subscriber Identity (IMSI) and the International Mobile Equipment Identity (IMEI), along with their security keys (§2.3).

2.2. Protocol stack

As depicted in Fig. 1, the LTE protocol stack follows a layered structure. **Layer 3** consists of two aforementioned key protocols: NAS and RRC. These protocols are defined in the 3GPP standards [33], [34] and are designed to minimize the size to reduce overhead. **Layer 2** ensures reliable delivery through the Packet Data Convergence Protocol (PDCP) for encryption/integrity protection and the Radio Link Control (RLC) layer for segmentation and packet ordering. Important packets like RRC and NAS messages use Acknowledged Mode (AM), where the BP sends acknowledgement (ACK) for successfully received packets or request retransmission (NACK), ensuring reliable delivery.

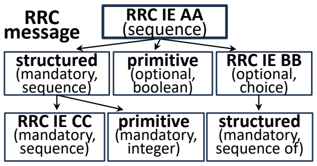
RRC message
RRC IE AA (sequence)
structured (mandatory, sequence)
primitive (optional, boolean)
RRC IE BB (optional, choice)
RRC IE CC (mandatory, sequence)
primitive (mandatory, integer)
structured (mandatory, sequence of)

Figure 2: Visualized structure of an RRC message

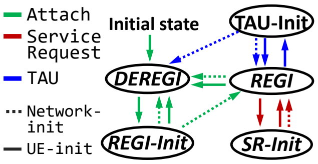

Figure 3: Overview of simplified EMM state transitions

For effective testing, it is essential to understand the structure of NAS and RRC protocols, as the BP implements functions specifically designed to handle them.

NAS messages follow a byte-aligned `Type-Length-Value` (TLV) structure. Each message consists of Information Elements (IEs), where each comprises 1–2 byte `Type` uniquely identifying the IE, along with its `Length` and `Value`, which define its contents. In particular, the first three bytes of a message uniquely define the *message type*, determining which IEs can be included, their order, and whether each IE is mandatory or optional. Mandatory IEs often omit their `Type` and `Length` fields, and include `Length` if the value has variable boundaries. In contrast, optional IEs explicitly include both `Tag` and `Value`, and often include `Length`.

RRC messages are defined in ASN.1 using Unaligned Packed Encoding Rules (UPER), resulting in compact bit-level representations [35]. The first five bits indicate the *message type*, followed by presence flags and `values` for individual *fields*, which may be mandatory or optional. These *fields* are either primitive types (`BOOLEAN`, `INTEGER`, `ENUMERATED`, `BIT STRING`, `OCTET STRING`) carrying values, or structured types (`SEQUENCE`, `SEQUENCE OF`, `CHOICE`) that again contain other *fields*. *Fields* can be primitives (*e.g.*`BOOLEAN`, `INTEGER`) that directly carry `values` or structured types (*e.g.*`SEQUENCE`, `CHOICE`), which recursively contain other *fields*. Structured types are often defined as reusable IEs with dedicated names, forming tree-like message structures across RRC messages (Fig. 2). While NAS uses IEs to carry values, this role is served by *fields* in RRC; we refer to both uniformly as "fields" unless otherwise noted.

Unlike NAS, RRC packets lack unique identifiers for *message types* and *fields*, and their lengths are often omitted. As a result, decoding depends on MAC-layer indicators and grammar definitions from the specification. For instance, RRC packets delivered over the downlink Dedicated Control CHannel (DCCH) typically carry critical control messages such as `Security Mode Command` for configuring security keys, or `DL Information Transfer` for transporting NAS messages.

2.3. LTE signaling procedures

This section provides details on `OTABase`'s testing state and control mechanism, examined further in this study.

UEs connect and maintain cellular service through three distinct signaling procedures–`Attach`, `Service Request`, and `Tracking Area Update (TAU)`–which ensure seamless registration, location updates, and service reconnection. Each procedure involves a sequence of RRC and NAS messages [33], [34], during which the UE transitions between specification-defined states outlined in clause 5.1.3 in [34]. Fig. 3 provides a concise overview of these states and their transitions. The specification defines five main EMM states: ***DEREGI***, ***REGI-Init***, ***REGI***, ***TAU-Init***, and ***SR-Init***. Appendix A provides detailed descriptions of each state, including behavior and security properties.

3. `OTABase` goals and challenges

Goals. The primary objective of `OTABase` is to efficiently detect memory crash vulnerabilities in COTS BPs through OTA testing. Prior studies have faced key challenges such as stateful behavior and connection timeouts, large protocol surfaces, and limited crash detection capabilities in black-box testing. Through specification-based approaches, we seek to achieve three key goals:

1. Establish efficient testing procedures that maximize the effectiveness of over-the-air interactions
2. Enable systematic testing of cellular protocol implementations without requiring implementation details
3. Develop an automated and reliable crash detection mechanism for over-the-air testing environments

Scope. Our work focuses specifically on memory crash vulnerabilities in the baseband that occur during the processing of signaling messages. We target NAS Mobility Management (EMM) and RRC DCCH messages, responsible for managing mobility, security, and radio resources. These are chosen for their critical role and their use of RLC AM, ensuring reliable message delivery. Note that we do not address design flaws in the specifications or other implementation bugs unrelated to memory corruption.

Challenges. To detect memory crashes in BP through OTA testing, we need to address the following challenges.

C1. Coordinating UE states and timeouts. The stateful behavior of the BP poses significant challenges for OTA testing. Depending on its internal state, the same message may be processed, ignored, or produce varying outcomes. This variability hinders systematic testing, as it requires precise control of the BP's state.

Maintaining the UE in a specific state is inherently difficult due to two main challenges: timer-driven state transitions and connection overhead. First, the BP autonomously changes its state based on internal timers. For instance, after sending an `Attach Request` (*i.e. REGI-Init*), the UE remains connected only for 15 seconds. Then, it disconnects and waits 10 seconds before attempting to reconnect. If several attempts, it enters a back-off period of 12 minutes [34]. Notably, our tests revealed that some BPs (*i.e.* MediaTek) extend this to 2 hours, resulting in 99.07% idle time (see evaluations in §B). Moreover, this behavior cannot be reset via standard methods such as toggling airplane mode; it requires SIM reinsertion or a reboot. Recent studies also reported that repeated Attach failures can lead to near-permanent disconnects, often requiring manual intervention [36]. Second, each connection and state transition incurs a delay of 0.3–1 second, making per-test-case repetition inefficient. Overall, `OTABase` requires precise control over

UE states and timers to ensure stable OTA connections and maximize testing efficiency.

C2. Large number of messages and optional fields to test. Testing BPs requires comprehensive coverage of a large protocol surface. The LTE specification defines over 100 message types with more than 1,000 fields in RRC and NAS protocols alone. COTS basebands must implement handlers for all these messages and fields to ensure specification conformance, making comprehensive testing challenging. The lack of coverage feedback in OTA testing further complicates assessing whether test cases adequately explore protocol implementations. Thus, `OTABase` needs to generate test cases that systematically explore protocol messages and fields to uncover memory bugs.

Prior methods. Prior works have explored mutation-based, specification-guided, and coverage-guided testing to address this challenge. However, they could only explore a limited subset of the messages and fields implemented in BP.

Mutation-based testing modifies traffic from live network operations [3], [37]. However, this approach leaves many protocol fields untested, as commercial traffic covers only a narrow subset of possible message types and fields. Worse, cellular protocols are highly grammar-sensitive, rejecting the vast majority of randomly mutated packets during parsing—long before they reach the target handlers.

To address these limitations, several studies proposed specification-guided testing [4], [38], [1], [39], [40]. These efforts leveraged the specification to craft prohibited or invalid messages, exposing logic bugs particularly in authentication-related fields. However, such approaches left memory bugs unaddressed despite improving protocol coverage. An attempt to target memory bugs [29] still faces limitations in coverage, supports only open-source UEs, while its detailed methodology remains undisclosed.

Coverage-guided fuzzing also suffers from limited code coverage. FirmWire [8], the state-of-the-art emulator-based fuzzer, achieved only 3.5% code coverage for the RRC due to its lack of network interaction support, restricting testing to early baseband states. Recent follow-up efforts [41], [42] improved emulator support but still exhibit limited coverage and only support MediaTek and Exynos BPs.

C3. Limited oracle for detecting a crash. Detecting memory corruption in BP is challenging due to the limited debugging capabilities. Unlike traditional software, BP operates in a closed, proprietary system with restricted memory access and debug interfaces [32]. This makes it difficult to inspect runtime memory state or utilize existing debuggers during testing. Moreover, memory corruption in BP is often less observable via the user interface (UI) of the UE. Crashes may remain silent, with no immediate UI indication; for instance, the device may still show full signal bars even after losing connectivity. Conversely, some UEs hide signal bars before completing signaling procedures (§2.3). In summary, `OTABase` requires a reliable and efficient method for detecting such silent crashes.

Prior methods. To assess existing approaches for detecting memory crashes in BP, we gratefully leveraged FirmWire's publicly released payloads [45], which enabled evaluation on COTS UEs in terms of usability, automation feasibility, and false-positive resilience. Tab. 1 summarizes our findings.

TABLE 1: Existing memory bug validation methods

Approach	Impact	Validation w/ 1-day	R	A
Visual feedback	Signal bar disappear [8]	✓	✗	✗
Cellular connection	Lose connectivity [43], [44]	✓	✗	✗
ADB log	”CP Crash” log [8]	✗	✓	✓
	Other logs [36]	✓	✓	✗
Bluetooth connection	Bluetooth dead [44]	✗	✗	✓
Manufacturer's debug mode	Kernel panic [43], MTKLogger [9]	✓	✓	✗
Proposed method	Liveness check fails (§4.3)	✓	✗	✓
	& Kernel panic [43], ADB log [36]	✓	✓	✗

R: ✓- Robust against false positives, ✗- prone to false positives
A: ✓- validated automatically ✗- validated manually

Our experiments revealed significant limitations in the practicality and reliability of commonly used validation methods, particularly those based on emulation or device logs. Emulation-based methods utilized address sanitizer [9] or custom fault detectors [8] to catch crashes during fuzzing. While these methods provide rich details on crash, they require firmware access and involve complex setup, making them impractical for testing diverse COTS devices.

Device log-based approaches extracted crash-related information from ADB logs [37], [8] or tools like ModemManager [46]. While some studies reported success using these methods, our experiments found that some are unreliable. In particular, when reproducing FirmWire's payload (§C), we confirmed that the payload triggered a crash and led to signal loss. However, the expected `CP Crash` log in ADB logcat, as reported in [8], did not appear, making this approach unsuitable for consistent crash detection across devices. Meanwhile, recent work highlighted that ADB logs upon crash exhibit vendor-specific differences, necessitating an initial manual analysis step [36].

Beyond these methods, prior work explored indirect or vendor-specific methods. For instance, Bluetooth disconnection was proposed as a crash indicator [44], but in our tests, the connection remained active after BP crashes. Other studies relied on manufacturer-specific modes, such as Samsung's debug panic trigger [43] or MediaTek's MTKLogger [9]. Both approaches have practical limitations: Samsung's mode requires manual intervention after each crash—-holding hardware buttons for 10 seconds—-while MTKLogger is proprietary and requires reverse engineering to interpret crash information.

3.1. Our approaches

To address the above challenges, we present the three contributions of `OTABase`.

A1. Network-side state control mechanism. To overcome the challenges of coordinating UE states and timeouts (**C1**), we propose precise state management through network-side control logic derived from 3GPP standards. Our design is based on a thorough analysis of the standards [34], [33], [47] and prior work on baseband state machines [14], [12]. This mechanism enables immediate transitions across key signaling procedures (§2.3), avoiding lengthy back-off timers and unintended UE-driven state changes that hinder testing.

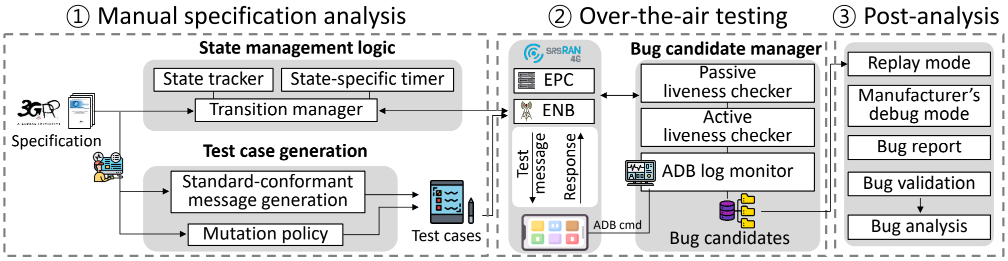

Figure 4: Overview of `OTABase`

It required extensive empirical validation, as the call flows for these state transitions are not explicitly documented in the specifications, and existing open-source protocol stacks lack the necessary support. While prior works also used specification-compliant messages (*i.e.* reject message) for state initialization [12], [13], we further extend this approach to achieve robust state control across standard-defined states and to avoid near-permanent disconnections requiring manual interventions [36]. This ensures consistent testing conditions and minimizes downtime by enabling reliable re-attachment, making sustained over-the-air testing practical.

A2. Specification-guided test case generation. To address the extensive testing surface (**C2**), we develop a systematic approach for generating and mutating test cases based on protocol specifications. Our method follows two key principles: First, we identify security-sensitive fields—such as length—that may trigger memory corruption, and generate valid, standard-conformant test messages that always include these fields. We also consider the theoretical limits of message and field lengths. Second, we apply grammar-aware mutations that selectively corrupt these fields while preserving the overall message structure. This is crucial, as a single bit-flip can even corrupt the entire packet, leading to early rejection at the parsing stage and inefficient testing. As a result, `OTABase` can test rarely exercised message handlers that are otherwise difficult to reach but still required by the specification. This represents a fundamentally different use of specification guidance from prior works.

A3. Two-phase crash detection oracle. To address the limited crash detection capabilities (**C3**), we develop a two-phase detection approach combining protocol-based liveness checking and vendor-specific debug features. The first phase identifies crash candidates by monitoring baseband liveness through cellular protocol messages and ADB logs. Specifically, `OTABase` performs both passive monitoring via RLC ACK messages and active probing using specific RRC and NAS messages. This insight stems from prior studies showing that baseband crashes consistently result in connection loss [43], [44]. While prior works such as DoLTEst also utilized protocol messages as oracles to detect logic bugs [4], their methods cannot be used for detecting the liveness of the baseband. We additionally monitor the ADB log to detect any signs for crash. Thereafter, we adopt a second validation phase utilizing manufacturers' debug features. These features trigger kernel panic upon detecting memory crashes, providing reliable confirmation without requiring firmware access or complex debugging setups.

4. `OTABase`

4.1. System overview

Fig. 4 illustrates the components of `OTABase` and its workflow. First, `OTABase` applies network-side state control logic derived from specification analysis (§4.2). Next, it generates test cases by creating standard-conformant messages and mutating security-sensitive fields (§4.4). We also design an oracle to detect memory bugs over the air (§4.3). Using these states and test cases, `OTABase` performs over-the-air testing on various UEs and collects bug candidates. Also, `OTABase` further reduces the downtime by using the airplane mode (§4.5). Finally, after testing, `OTABase` verifies the bug candidates using the manufacturer's debug mode, and further analyzes the suspected message field (§4.6).

Attack model. We consider several established attack models in cellular networks, including fake base stations (FBS)[14], [2], [48], [49], man-in-the-middle (MitM) attacks[1], [19], and signal injection attackers [18], [16] (Fig. 5). In addition, we adopt the rogue carrier model used in recent work [50], which assumes a stronger adversary that possesses the victim's cryptographic keys and can send authenticated messages.

We argue that memory vulnerabilities in the BP must be thoroughly evaluated, as attackers with key access can still exploit critical paths beyond the scope of cryptographic protection. Such vulnerabilities can enable unauthorized data access and persistent RCE backdoors. Recent real-world incidents such as Iran's SIAM [51] demonstrate that state-sponsored attacks are no longer hypothetical. Additionally, the Android security team has officially included adversarial network providers in its threat model [52], reinforcing the need to consider rogue carriers in `OTABase` 's design.

4.2. State control mechanism

`OTABase` 's state control mechanism maintains connectivity and systematically exercises key UE states (§2.3).

Timer-based state management. To prevent indefinite waiting, basebands enforce state-specific timers as defined in the specifications. `OTABase` conducts testing before these timers expire to avoid unintended state transitions and radio connection back-off, and uses targeted message sequences to re-enter each state. Specifically, `OTABase` manages the following NAS timers [34]: T3410 (15s) for *REGI-Init*, T3430 (15s) for *TAU-Init*, and T3417 (5s) for *SR-Init*.

Initialization. Upon initial UE connection, `OTABase` sends an `Attach Reject` with cause *Illegal UE* to clear any residual

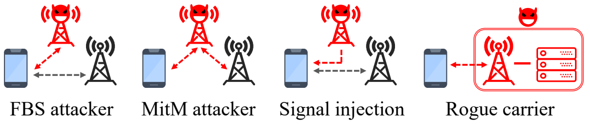

Figure 5: Attack model in cellular network

security or state context, forcing a transition to *DEREGI*. Once the UE reconnects, it enters *REGI-Init*. If the target state is not *REGI-Init*, `OTABase` completes the `Attach` procedure to reach *REGI*. Subsequent transitions to *TAU-Init* or *SR-Init* are achieved via specific message sequences described below.

State transition control. `OTABase` employs the following mechanisms for each testing state:

- ***REGI-Init***: Before the state timer expires, `OTABase` completes the `Attach` procedure to enter *REGI*, then sends a `NAS Detach Request` with detach type *re-attach required*, followed by an `RRC Connection Release` with cause *others*. This causes the UE to disconnect and immediately reconnect, returning to *REGI-Init* within approximately 0.7 seconds.
- ***REGI***: Although this state has no state timer, `OTABase` periodically resets it using a `Detach Request` to avoid state contamination. The procedure follows the *REGI-Init* cycle.
- ***TAU-Init***: `OTABase` first sends a `TAU Accept` to move the UE to *REGI*, then issues an `RRC Connection Release` with cause *load balancing TAU required*. This brings the UE back to *TAU-Init* in about 0.2 seconds.
- ***SR-Init***: `OTABase` uses `RRC connection reconfiguration` for *REGI* transition, followed by `RRC connection release` with cause *others* and `RRC paging` with UE's S-TMSI. This sequence returns UE to *SR-Init* within 1.7 seconds.

Batch testing. Even with full control over UE states, each connection and state transition introduces non-negligible delays, making it inefficient to repeat them for every test case. Thus, `OTABase` performs batch testing once the UE reaches a target state.

Handling unexpected transitions. Batch testing assumes the UE remains in the target state, but unexpected transitions can occur due to internal logic or test messages sent by `OTABase`. Implementation differences across devices can also lead UEs to ignore messages or transition to unintended states, due to specification ambiguities–as we observed when identical messages triggered inconsistent behavior. `OTABase` detects such cases through UE responses and recovers by first returning the UE to *REGI*, then re-executing the required state transition sequence.

4.3. Oracle

`OTABase` employs a two-phase crash detection oracle combining protocol-based liveness checks, ADB logs, and manufacturer-specific debug features. The rationale behind this design is that the BPs behave as a black box, requiring `OTABase` to leverage every observable crash indicator to maximize detection coverage.

However, the available signals differ by device family: while generic Android devices expose ADB logs and cellular liveness, Samsung devices additionally provide debug mode. Meanwhile, crash detection without false positives requires either debug mode or device-specific ADB crash log knowledge (Tab. 1), which can only be derived from real crashes. Thus, while final confirmation requires vendor feedback, `OTABase` can reliably detect crashes on Samsung devices via debug mode, or on Android devices with known ADB crash log signatures, as described below.

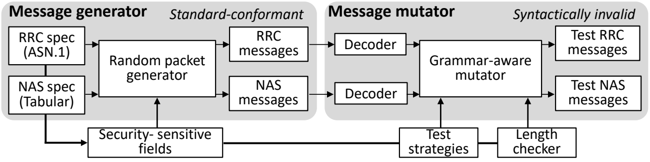

Figure 6: Process of test message generation

Phase 1: Protocol-based liveness and ADB log checking. This phase monitors BP's liveness using passive and active approaches, leveraging layer 2/3 cellular protocols, and in parallel, monitors ADB logs for potential crash evidence.

- **Passive liveness check:** `OTABase` inspects RLC ACKs for each test message, verifying matching sequence numbers. If the ACK is missing or mismatched, the test message is marked as a crash candidate.
- **Active liveness check:** To complement passive checking, `OTABase` periodically sends layer 3 probe messages (every 10 test messages): `RRC UE Capability Enquiry` and `NAS Identity Request` (IMSI type). These messages guarantee a response regardless of UE state, and missing replies are treated as potential crash indicators.

In **phase 1**, `OTABase` can autonomously collect crash candidates, as the modem driver reboots the BP after a crash. This reboot process typically takes within 10 seconds, enabling seamless automated testing. Active checks offer higher accuracy by catching crashes that don't immediately affect the lower layers. We observed such delayed effects during testing. However, frequent active probing degrades the testing speed, so `OTABase` uses it occasionally.

Collecting bug candidates. If a liveness check fails, `OTABase` replays recent packets (*i.e.* last 10 packets) to pinpoint the problematic one. In particular, it logs a message as a crash candidate only when both liveness checks consistently fail during replay, reducing false positives. When device-specific crash log is known, ADB logs serve as an additional signal to strengthen candidate collection.

Phase 2: Validation using manufacturer's debug mode. Each bug candidate is validated with the manufacturer's debug mode enabled. `OTABase` replays each candidate to check for kernel panics, filtering out false positives and confirming genuine memory crashes. The detailed operation is described in §D. For non-Samsung devices, validation relies only on ADB crash logs or vendor feedback.

4.4. Specification-guided test case generation

`OTABase` uses a grammar-aware approach combining valid message generation and targeted mutations (Fig. 6).

Standard-conformant message generation. `OTABase` leverages specification grammars to generate valid base messages, including optional fields that are rarely used in commercial networks. Given the infeasibility of testing all combinations (*e.g.* over 4,000 RRC fields), `OTABase` prioritizes security-sensitive fields that are most likely to trigger memory crashes. In particular, `OTABase` considers both mandatory and optional fields when applying the security-sensitive criteria explained below, ensuring that all such fields are explicitly included in the generated test cases. Also, nested structures defined in ASN.1 or TLV formats are recursively expanded so that inner elements can also be selected for mutation. The remaining fields are filled with compliant random values to ensure standard conformance.

Security-sensitive fields. `OTABase` defines the following categories of fields as security-sensitive to target:

- *Length-related fields:* (1) Single fields whose length defines content size (*e.g.*`MS network capability` in NAS, `codebookSubsetRestriction-r13` in RRC)), (2) List fields where length represents the total size of multiple single fields it contains (*e.g.*`Emergency number list` in NAS, `UE-CapabilityRAT-ContainerList` in RRC).
- *Range-constrained fields:* Fields with a valid range smaller than their allocated size, such as a 3-bit field that only defines values 0—5, leaving 6-–7 outside the standard-conformant range (*e.g.*`locationCE-ModeB-r15` in RRC).

Mutation Strategy. After generating base messages, `OTABase` applies grammar-aware mutations to introduce controlled invalidity. This preserves structural integrity up to the target field while breaking compliance beyond it.

- *Length-related fields:* `OTABase` independently mutates both length and content fields, using maximum, minimum, and random values both within and outside specification-defined ranges. Mismatches include cases like zero-length with non-empty content, or unchanged length with truncated content. For list fields, `OTABase` manipulates both total length and individual element fields independently, creating various mismatches between declared and actual sizes.
- *Range-constrained fields:* `OTABase` tests values outside the valid range, including edge cases and random invalid values.
- *Packet manipulation:* `OTABase` modifies entire packet payloads by adding or truncating random bits or bytes.

4.5. Handling unexpected radio disconnections

To sustain efficient testing, `OTABase` implements mechanisms to minimize UE disconnection time. While the state management logic (§4.2) typically ensures stable connectivity, disconnections may still occur due to:

- Certain field values in test messages can disrupt connections or corrupt RF settings–for example, `RRC Connection Release` may redirect the UE to invalid frequencies, or `RRC Connection Reconfiguration` may apply incompatible antenna parameters.
- External factors like poor signal or hardware faults.

Statically filtering such values is impractical due to the manual effort required to analyze thousands of fields.

To address these cases, `OTABase` adopts two mechanisms:

Temporary blacklisting. Upon detecting a connection drop via its oracle (§4.3), `OTABase` temporarily blacklists messages targeting the same field. If a message causes three disconnections, it is skipped for the next 30 attempts. This reduces repeated disruptions from unstable inputs.

Connection reset mechanism. If the UE fails to reconnect, `OTABase` toggles airplane mode via ADB to reset the radio stack. We initially attempted to recover using protocol messages (`RRC Connection Release`, `Paging`), which helped in partial disconnection scenarios. However, airplane mode toggling proved more effective overall.

4.6. Over-the-air testing workflow

Using the above components and logic, `OTABase` performs OTA testing and post-analysis as follows.

State preparation and testing setup. When a UE connects, `OTABase` transitions it to the user-specified testing state, confirming the transition by monitoring UE responses. It then initiates testing for a fixed duration based on the state's transition logic. For example, in *REGI-Init*, `OTABase` only establishes the RRC connection and sets a 13-second timer, without completing the `Attach` procedure.

Message testing process. `OTABase` sends RRC or NAS test messages to the UE. For states beyond *REGI-Init*, it adds appropriate security headers to avoid early rejection due to authentication failure. Each message is followed by a passive liveness check via uplink RLC ACK, allowing one message every 20–25 ms (per clause 11.2 of [33]). Active liveness checks are performed every ten messages, and testing resumes once a response is received.

Bug candidate collection. `OTABase` monitors liveness until the state timer expires. If a crash is suspected, it re-establishes the connection using state transition logic and continues until all test cases are processed.

Human intervention. While automated, `OTABase` occasionally requires human intervention due to SDR hardware failures. For instance, our USRP B210 encountered rare runtime errors during tests exceeding 10 hours, necessitating USB reconnection—a known issue with no reliable mitigation. To the best of our knowledge, there is currently no reliable mitigation for this issue.

Vulnerability analysis. After collecting crash candidates, `OTABase` conducts manual post-analysis using the manufacturer's debug mode. For crashes with kernel panics, we:

- Identify malformed fields using specification references [34], [33] and ASN.1 decoders [53], [54].
- Perform ablation testing by incrementally removing N bytes to isolate faulty segments.

Confirmed bugs are reported to the vendor. A detailed example is in §C.

5. Implementation

We implemented `OTABase` by integrating two key components: srsRAN [31], an open-source cellular stack, and pycrate [53], a Python library for cellular packet analysis. Our implementation consists of 5,116 lines of C++ extending srsRAN and 6,091 lines of Python using pycrate.

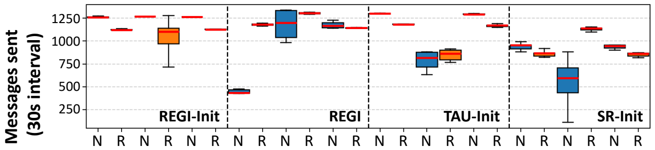

Figure 7: OTA testing speed per state (N: NAS, R: RRC)

For test case generation (§4.4), pycrate provides functionality for precise manipulation of RRC and NAS messages based on their grammar. It includes an ASN.1 compiler that converts RRC specification into Python structures, ensuring compliance. While pycrate supports NAS EMM messages, some fields (*e.g.*`CipherKeyData`) were found to have loosely constrained lengths inconsistent with the specification. To address this, we used BaseSpec [6] to extract correct values from the specification and applied them to pycrate. `OTABase` adopts release v17.4 of RRC and release v16.6 of NAS specifications. To support the state control mechanism (§4.2), we extended srsRAN to implement three procedures: `network-initiated Detach`, `Service Request`, and `Tracking Area Update`. We also referenced the ADB crash logs reported by LLFuzz [36] for ADB log-based detection.

Testing environment. We ran `OTABase` using a USRP B210 [55] as the radio interface on an Ubuntu 18.04 LTS equipped with an Intel Core i5-3570 CPU (4 cores) at 3.40 GHz and 16 GB DDR3 RAM. The UE under test was equipped with a programmable SIM card (sysmoISIM-SJA2 [56]) and placed inside a Faraday cage to prevent our test signals from interfering with commercial users.

6. Results

6.1. Generated test cases

`OTABase` identified 52 security-sensitive fields out of 62 IEs in NAS, 724 security-sensitive fields—395 from primitive types (37 out of 70 `OCTET STRING`, 12 out of 187 `BIT STRING`, and 346 out of 830 `INTEGER`) and 329 from structured types (329 out of 377 `SEQ OF`) in RRC. The remaining fields, both from the above types, other primitive types (851 `ENUM` and 165 `BOOLEAN`) and structured types (1,099 `SEQUENCE` and 471 `CHOICE`), do not contain length-related or range-constrained elements. Their bit flips either modify values within standard-conformant ranges or fully corrupt the packet structure. Additionally, there are 16 `NULL` fields that carry no data.

To systematically evaluate protocol implementations, we generate test messages that cover all possible positions where security-sensitive fields can appear. Since a security-sensitive field can be used in various locations within messages, we create standard-conformant messages for each possible case, resulting in 88 NAS messages and 10,007 RRC messages. Note that the large number of RRC messages results from its inherent recursive optional field structure and frequent field reuse across multiple locations.

After applying our mutation strategy, we obtain 51,185 RRC test cases and approximately 830 NAS test cases. We repeat this process 3 times for RRC and 50 times for NAS, ultimately generating 153,555 RRC test cases and 41,942 NAS test cases. For NAS, the number of test cases after mutation varies slightly due to additional randomness during packet mutation. See Appendix §G for test case details.

TABLE 2: Number of bugs discovered for each test device

BP vendor	Device model	Chipset model	Tested firmware version (YYMM)	NAS 0d —1d	RRC 0d —1d
Qc	Galaxy ZFlip4	SM8475	2311	0 — 0	0 — 0
Qc	Galaxy S8	MSM8998	2012	0 — 0	0 — 0
Exy	Galaxy S21	Exynos 2100	2307	3 — 0	0 — 0
Exy	Galaxy Note 8	Exynos 8895	2001	3 — 1	0 — 0
Mtk	Galaxy A32	Helio G80	2308	3 — 0	1 — 0
Mtk	Galaxy A31	Helio P65	2207	3 — 0	1 — 0

BP vendor: Qc-Qualcomm, Exy-Exynos, Mtk-MediaTek, **Tested firmware version**: For newer models, the firmware was updated to the latest version available at the time of testing, **0d — 1d**: Number of 0-day and 1-day bugs discovered.

6.2. Over-the-air throughput

We evaluated `OTABase`'s testing throughput by measuring the number of test messages sent over-the-air during a 10-minute period. The evaluation used RRC and NAS messages on three BPs (MediaTek, Exynos, Qualcomm) across all testing states. To focus on speed itself, we selected packets that do not cause crashes or disconnections. As in regular testing, `OTABase` performed both active and passive liveness checks, where these messages were not counted in the result. Fig. 7 presents the messages sent in 30-second intervals, which includes at least one state transition and any necessary recovery actions triggered by unexpected UE behavior.

The results show that all devices generally achieved 700–1250 test messages per 30 seconds across most states. The overall speed decreases in the *SR-Init* due to its shorter timer (§4.2), which forces more frequent state transitions. Notably, the Galaxy S21 with Exynos BP exhibits slower testing speeds, particularly due to extended state transition times. This performance degradation stems from its unstable state transition behaviors – the device often fails to properly respond to standard-conformant messages such as `RRC paging`, which leads to longer recovery times. Given our later findings of state transition misimplementations in the Exynos BP (§F), these issues likely contributed to the observed slowdown.

Importantly, these results reflect ideal conditions and do not represent actual testing throughput. In real tests, messages may trigger crashes or connection losses, causing delays of one to over ten seconds per incident.

In our experiments, the time to complete testing varied depending on whether crashes occurred in the target state. For NAS testing, test states with specific chipset combinations that had no crashes took an average of 25 to 55 minutes to complete. However, in states where crashes were discovered, the testing time increased to an average of approximately 150 to 270 minutes. Similarly, RRC took on average 6 to 10 hours in states without crashes, but increased to approximately 15 to 30 hours in states where crashes were found. We believe this significant increase in testing time is due to bugs found in relatively shallow locations (§6.3), causing them to be frequently triggered during our testing.

TABLE 3: Identified New Vulnerabilities in LTE Baseband

#	Category	Message	IE	Impact	Chipset	State	CVE	Duplicated
V1	NAS	EMM Information	Short name for network	RCE*, Remote DoS	Mtk	*REGI*	CVE-2024-20039	No
V2	NAS	Downlink Generic NAS Transport	Additional Information	Remote DoS	Mtk	*REGI*	-	No
V3	NAS	Downlink NAS Transport	NAS message container	Remote DoS	Mtk	*REGI*	-	No
V4	NAS	Attach Accept	Emergency number list	Remote DoS	Exy	*REGI*	-	Yes
V5	NAS	Detach Accept	-	Remote DoS	Exy	All	CVE-2023-37366	Yes
V6	NAS	Authentication Reject	-	Remote DoS	Exy	All	CVE-2023-37366	Yes
V7	RRC	DLInformationTransfer	DedicatedInfoNAS	Remote DoS	Mtk	All	CVE-2023-32890	No

Message: Message type associated with the vulnerability. **IE**: The name of IE associated with the vulnerability. If no specific IE is associated, it is represented as `-`. **Chipset**: Mtk-MediaTek, Exy-Exynos. **State**: The state in which the vulnerability is triggered. **CVE**: CVE identifier assigned by the manufacturer. **Duplicated**: Indicates whether the manufacturer classified our reports as duplicates. $*$: Independently verified.

6.3. Discovered bugs

We adopt `OTABase` to test six cellular devices from three major baseband manufacturers, Qualcomm, Exynos, and MediaTek, including both new and old models from each manufacturer. The details of the test devices and the number of bugs are presented in Tab. 2. Our findings include seven previously unpatched memory crash bugs: three in the NAS and one in the RRC of the MediaTek BPs, as well as three in the NAS of the Exynos BPs. Additionally, we identified a 1-day bug in an older version of the Exynos BPs, which aligns with the findings reported in [6]. Notably, several of the discovered bugs arise from deeply embedded states, triggered in the final stages of the `attach` procedure (*REGI*) by specific optional fields (**V1**, **V2**, and **V4**).

Tab. 3 summarizes the vulnerabilities, and the complete test payloads are provided in Appendix E, including details of V2-V3 and V5–V6. Below, we detail our findings.

V1: EMM Information. The `Short name for network` IE in `EMM Information` crashes the baseband when its length value exceeds 0xE7. This optional IE, designed to display an abbreviated network name to users, causes an out-of-bounds write according to MediaTek's patch report, enabling RCE without user interaction. This issue affects 75 chipsets, was classified as high severity, and was assigned a CVE.

V4: Attach Accept. A vulnerability in `Attach Accept` allows baseband crashes through malformed `Emergency Number List` IE. The crash occurs when the IE's length exceeds the range specified in [57]. While we confirmed this vulnerability in the latest Exynos firmware, Samsung indicated it duplicates a bug already undergoing patching. Interestingly, a memory bug was also found in the same IE, but in the MediaTek BPs, as reported by previous works [9], [58].

V7: RRC DLInformationTransfer. When the mandatory field `dedicatedInfoNAS` in `RRC DL Information Transfer` contains empty contents, it triggers a baseband crash. According to MediaTek, this malformed message triggers a null pointer dereference, leading to a temporary DoS. The issue affects 13 chipsets, and was assigned a CVE with high severity.

In addition to these memory corruption bugs, we also identified two misimplementations in Exynos BPs. Because these issues fall outside the primary scope of this work, their details are provided in Appendix §F.

6.4. Implications

Memory crashes identified in Tab. 3 can be exploited for both DoS attacks and RCE.

Denial-of-service. All the memory vulnerabilities we identified can lead to DoS attacks that can persistently disrupt cellular connectivity. When exploited, these vulnerabilities force BPs to crash and silently reboot, causing approximately 3 to 7 seconds of complete cellular service disruption. During this period, while the mobile OS continues to function, all cellular services become unavailable – including emergency calls, cellular menus, and basic connectivity, with the signal indicator disappearing entirely[1].

Notably, three of these vulnerabilities—**V5**, **V6**, and **V7**—arise before the authentication (*i.e. REGI-Init*). As UEs accept such messages without authentication, an adversary operating an FBS or performing signal injection [18] can reliably deliver the malformed payloads, leading to a DoS.

What makes these DoS attacks particularly concerning is their ease of automation. An attacker can use LTESniffer [59] to monitor nearby UEs attempting network connections, and then trigger crashes by using the pre-authentication payloads. This enables a self-sustaining attack cycle: the victim's BP crashes, attempts to reconnect, and is immediately attacked again, causing the device to crash on every reconnection attempt. Once deployed, such an attack can indefinitely deny service to all vulnerable devices in range, making it highly efficient and difficult to mitigate.

Remote code execution. Although our discovered bugs manifest primarily as service disruption, their underlying memory corruption may enable broader exploitation. Our discovered memory crashes also pose a risk of RCE, especially out-of-bounds write vulnerabilities (*i.e.* **V1**) that allow corruption of memory regions critical to code execution control. An external security researcher independently rediscovered **V1** and kindly shared with us that they had successfully developed a zero-click RCE exploit on a real device [41], confirming its practical exploitability. In the hands of a state-sponsored attacker (§4.1), such vulnerabilities can be weaponized for targeted attacks. Attackers can identify vulnerable devices by obtaining the UE model and software version via a `NAS Identity Request` asking for IMEISV [60] or through fingerprinting techniques [61], [62]. The attacker develops exploits tailored to devices with vulnerable BP by utilizing publicly available tools [8], [63], [50] or reverse engineering [64], [65], [27]. While building such exploits typically requires substantial expertise and resources [23], [24], [25], [26], [22], `OTABase` facilitates exploit development by identifying crash candidates with exploitation potential for prioritized investigation.

[1] The demo video is available at https://youtu.be/VgVlGe91doM

TABLE 4: Our test case coverage of problematic fields and comparison with prior LTE studies reporting memory bugs

Criteria	FW [8]	BSP [6]	BSF [9]	[43]	BB [41]	LR [42]
Total LTE NAS and RRC bugs found	4	2	1	3	6	1
Covered by our test cases	3	2	1	0	5	1
Our bugs not found by each work	7	7	6	7	5	7

FW: FirmWire, **BSP**: BaseSpec, **BSF**: BaseSAFE, **BB**: BaseBridge, **LR**: LORIS

6.5. Coverage of existing LTE works

We examined whether our test cases cover the memory bugs identified in previous works. Specifically, we analyzed publicly available details of NAS and RRC vulnerabilities documented in prior LTE research and assessed whether the problematic fields identified in these studies align with our security-sensitive fields (§6.1).

Tab. 4 summarizes the number of NAS and RRC bugs discovered in each study and how many of them are covered by our test cases. While OTABase contains test cases targeting many of these problematic fields, it does not cover vulnerabilities involving protocol messages outside our target scope, such as an RRC MCCH message reported by Firmwire [8], and several bugs identified by Klischies et al. [43] related to RRC BCCH broadcast messages and GSM SMS. As shown in the third row of Tab. 4, OTABase's test cases found several memory bugs that were missed by prior LTE works, highlighting its complementary coverage.

6.6. Speed comparison with prior works

We compare the testing speed of OTABase with existing baseband testing tools in terms of test case execution speed, as summarized in Tab. 5. While prior tools like LTEFuzz [8] and Berserker [29] did not report concrete speed metrics, DoLTEst [4] achieved a throughput of one message per two seconds, and Berserker required 20 to 125 seconds per message. For open-source tools where no speed was reported (*i.e.* 5Ghoul [37]), we reproduced their test executions and measured an approximate speed of three test cases per second. In contrast, OTABase reached a speed of around 41 messages per second under ideal conditions, enabled by our batch testing and protocol-aware state management. These measurements are based on stable testing runs without connection drops, to enable fair comparison across tools. While actual testing speeds are reduced by connection disruptions and crash recovery, OTABase achieves an order-of-magnitude speedup over prior tools, providing the improved throughput necessary for uncovering memory crashes in black-box OTA environments and enabling practical large-scale test execution.

6.7. PoC 5G measurement

TABLE 5: Comparison of tools and testing speed

Tool	Open-sourced?	Speed Reported?	Our Evaluation
LTEFuzz [3]	✗	✗	N/A
DoLTEst [4]	✓	1 msg per 2 seconds	–
Berserker [29]	✗	1 msg per 20–125 seconds	–
5Ghoul [37]	✓	✗	∼3 test cases every 1 second
OTABase	✓ (will be)	∼41 msgs per 1 second	–

✓: open-sourced tool; ✗: not open-sourced; OTABase will be open-sourced. Speeds are either reported in the paper or estimated.

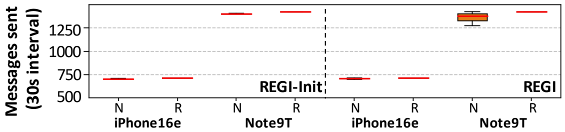

Figure 8: OTA testing speed per state in 5G

OTABase was originally developed for LTE, but its design elements—state control to enhance the speed of over-the-air testing, specification-guided test case generation, and bug oracle—are extensible to 5G basebands due to structural and protocol-level similarities between LTE and 5G. To demonstrate this, we implemented a proof-of-concept version of OTABase for 5G by modifying the srsRAN 5G stack [66], which required considerable porting effort. We integrated OTABase's core functionalities including automated state control mechanism (§4.2) and crash oracle (§4.3). Also, we reused the specification-guided NAS/RRC test case generation logic with minor adjustments for 5G message formats by extending Pycrate [53].

Using this prototype, we evaluated over-the-air throughput and bug-finding capability on a 5G baseband.

Throughput measurement. We evaluated this prototype on two COTS 5G UEs—Xiaomi Redmi Note 9T (MediaTek Dimensity 800U) and iPhone 16e (Apple C1)— that successfully connected to an open-source gNB. Following the same methodology used in our LTE measurement (§6.2), we measured the speed of our 5G prototype. Fig. 8 shows that the prototype maintains high execution speed under stable conditions, with an average of around 1,400 and 700 RRC/NAS messages sent every 30 seconds on the Note 9T and iPhone 16e, respectively. While the current PoC covers only the *REGI-Init* and *REGI* states-—due to the removal of TAU procedure in 5G and the exclusion of SR procedure from our evaluation—-it demonstrates the feasibility and scalability of our design in 5G settings.

5G NAS and RRC testing. We conducted an end-to-end experiment with this prototype on a Google Pixel 9 (5G) to evaluate its effectivess in discovering memory bugs. As summarized in Tab. 6, we executed over 20,000 NAS and RRC test cases in both the *REGI-Init* and *REGI* states.

As a result, we uncovered three previously unknown bugs: two in NAS and one in RRC.

- **5G-V8 (NAS)**: In Downlink NAS Transport, the mandatory variable-length payloadContainer, which can be configured as a multiple-container with nested payloadContainerEntry elements, triggered a crash when an internal length field exceeded the actual payload size. This confirmed by the vendor as a duplicate of a previously reported issue.
- **5G-V9 (NAS)**: In the Registration Accept message, the length of the optional field SOR transparent container must be at least 20 bytes. Supplying a shorter value caused a crash; the vendor assessed this bug as medium severity.
- **5G-V10 (RRC)**: In RRCReconfiguration, the optional secondaryCellGroup field (OCTET STRING) triggered a crash when set to zero length. Because this message is processed in *REGI-Init*, an FBS could reliably trigger repeated crashes in a 5G BP (§6.4). This case is under vendor review.

TABLE 6: Summary of 5G test results

Protocol	State	# of test case	Test time	Findings
NAS	*REGI-Init*	22633	28min 26sec	-
	REGI	22633	35min 50sec	5G-V8, 5G-V9
RRC	*REGI-Init*	21204	64min 31sec	5G-V10
	REGI	21204	24min 35sec	5G-V10

# of test case: Number of executed test cases in each state. **Findings**: Bugs newly identified during the tests.

TABLE 7: Impact of state control on UE state behavior

State	Occurrences		Improvement	Residence time (%)		Improvement
	Without	With		Without	With	
REGI-Init	15	324	21.6×	6.25	94.40	+88.2%p
REGI	5	294	58.8×	2.53	91.33	+88.8%p
TAU-Init	2	358	179×	0.83	98.65	+97.8%p
SR-Init	18	618	34.3×	2.50	78.99	+76.5%p

Occurrences: Number of times the UE entered the state in 60 minutes.
Residence time: Percentage of the session spent in the state.

6.8. Ablation study

To isolate the contributions of the main components of OTABase, we conducted two ablation studies.

State control mechanism. We quantify the effect of the state-control mechanism by comparing OTABase with and without it in a 60-minute connection. Without explicit state control, OTABase cannot directly drive the UE into target states and must instead wait for autonomous reconnections. This leads to long idle periods, reducing controllability and efficiency. In contrast, with state control enabled, OTABase deterministically re-enters the UE into target states on demand, eliminating unnecessary delays.

We conducted 60-minute measurements for each state on srsRAN, both with and with state control, while monitoring the UE's current state using XCAL logs and NAS/RRC messages. We measured the number of occurrences (how many times the UE entered a given state) and the residence time (the percentage of the session the UE remained in that state) for each target state. As shown in Tab. 7, state control substantially increased both the frequency of valid state entries and the time the UE remained in each state.

Specification-guided test case generation. We evaluate the impact of specification-guided test case generation by comparing OTABase with a baseline that mutates packets without specification guidance, which is based on traffic collected from live networks, following the methodology of prior work [37]. While OTABase systematically generates valid NAS/RRC packets to exercise both mandatory and optional security-sensitive fields for mutation, traffic logs from open-source (srsRAN) and commercial networks contain only a limited subset of fields.

To quantify this gap, we collected NAS/RRC traffic for 2.5-3.5 hours per network from two commercial and one open-source networks with XCAL [67], while inducing diverse events such as toggling internet connectivity. We then measured the number of security-sensitive fields included in the collected traces and compared it with those covered by our approach. As summarized in Tab. 8, specification guidance increased the coverage of security-sensitive fields, thereby enabling the broader testing of memory crashes.

TABLE 8: Comparison of security sensitive fields with and without specification guidance

Method	Entity	NAS	RRC				Time
			BIT	INT	OCT	SEQ	
Without	MNO_A	15	10	31	3	13	2h 36m
spec-guided	MNO_B	19	8	25	4	12	2h 56m
(Traffic logs)	srsRAN	10	1	2	2	5	3h 33m
Spec-guided	OTABase	52	12	346	37	329	N/A

NAS, RRC fields: Number of security-sensitive fields discovered in each category. **BIT**–BITSTRING, **INT**–INTEGER, **OCT**–OCTET STRING, **SEQ**–SEQUENCE OF. **Time**: Duration of traffic collection.

7. Discussions and limitations

Bug validation via Samsung debug mode & ADB logs. Our validation relies on Samsung's debug mode and device-specific ADB crash logs. OTABase can test various BP implementations, including Qualcomm, as long as they are integrated into Samsung devices. For non-Samsung devices, validation is feasible only when ADB crash log patterns are known; otherwise, vendor feedback is required. This reliance highlights that OTABase cannot validate crashes outside Samsung devices without such information, and the approach would not work if Samsung discontinues debug mode support. Nevertheless, some manufacturer dependency is inevitable in black-box OTA testing due to limited debugging capabilities and the absence of standardized crash behavior. While recent studies have adopted ADB logs as crash oracles [37], [36], these approaches also require prior knowledge of crash log or Samsung's debug mode. In this context, our results contribute to improving ADB log–based detection methods, particularly given the limited public knowledge on reproducing crash behaviors.

Exploitability is unknown. OTABase cannot determine the exploitability of detected memory crashes due to its black-box nature. While OTABase can detect crashes, understanding their root cause, impact beyond DoS, or whether they are exploitable requires source code access or detailed binary analysis. We were only able to assess the impact of our findings through vendor feedback during responsible disclosure, and exploitability of one bug was later confirmed independently by an external researcher. Future work could incorporate firmware reverse engineering [27] or debugging frameworks [8], [41] to better assess the vulnerabilities' severity and exploitability.

Explicit and implicit states. In this work, we focus on protocol states explicitly defined in the specification. As the UE is always in one of the EMM states when connected to the network, all discovered memory crashes in our evaluation were triggered in these states. This scope aligns with prior findings that basebands commonly implement functions for specification-defined procedures [68]. However, specifications contain numerous implicit states that arise from undefined or underspecified behaviors. Such implicit states may enable complex vulnerabilities that are not observable in single well-defined transitions, including memory bugs that manifest only after unusual state interactions or ambiguous signaling paths [43]. OTABase, by design, cannot capture these implicit-state bugs, leaving them as an open challenge.

Non-crash and multi-step vulnerabilities. All vulnerabilities identified by OTABase are memory crashes induced by single-message inputs, yet other classes of memory bugs remain outside this coverage. Because OTABase relies on observable crashes [32], it may miss other types of memory bugs, such as silent use-after-free, whose effects do not manifest immediately. In addition, bugs that require specific sequences of malformed messages to manifest cannot be detected by OTABase's current workflow. Exploring such non-crash and multi-step vulnerabilities is left for future work, and we believe that OTABase's state-control and oracle design can be extended to support these scenarios.

5G and other protocols. Beyond NAS and RRC, memory bugs in other cellular protocols may also be tested over the air. In LTE, four other protocols are delivered by the NAS EMM message: SMS, EPS session management (ESM), location service (LCS) [69] or positioning protocol (LPP). Since these protocols are defined in tabular or ASN.1 formats, our test case generation methodology may be applicable. Similarly, lower-layer protocols, other RRC messages (*e.g.* BCCH), other generations (*e.g.* GSM), or application-layer protocols (*e.g.* SIP) present further opportunities for discovering memory bugs. However, each extension would require protocol-specific support, such as managing stateful behaviors and designing dedicated oracles. For example, testing 5G BPs requires adapting to 5G-specific changes like the removal of the TAU procedure.

8. Related work

Identifying design bugs. Previous studies have reported various design bugs that can lead to denial-of-service [2], signal injection [18], identity exposure [16], [17], bidding down attacks [61], [70] and user data manipulation [19], [20]. To uncover these issues, researchers utilized multiple approaches. Formal verification are widely adopted, with a number of researches building formal models from the standards [71], [14], [15], [72], [43]. Additionally, natural language processing techniques are employed to analyze the specifications and automatically find potential vulnerabilities [10] or build formal models [11]. In contrast to those researches, OTABase's primary objective is to identify memory bugs in cellular basebands.

Finding logical bugs. Researchers proposed several approaches to identify logical vulnerabilities that stem from incorrect implementations of cellular protocols. One prominent direction is dynamic testing approach with non-standard-conformant messages [1], [3], [4], [38]. To reduce human efforts, some works automatically generated messages by leveraging natural language processing [10], [73], [74] or model-based techniques [5]. Alternative approaches include static analysis to uncover integrity protection bugs [7], automata learning for protocol state verification [75], [13] and noncompliance detection [12], as well as a combination of static and dynamic approaches [39]. Unlike those works, OTABase focuses on finding memory bugs instead of logical bugs.

Memory bugs. For static approaches, reverse engineering has proven effective in identifying memory bugs in baseband implementations, from early research on 2G/3G protocols, including SMS [76], cell broadcast [24], and other layer 3 messages [65], [21], [77], to extensive works on LTE implementations [27], [28], [25], [26], [22]. Recent automated techniques compare implementations against specifications to detect deviations [6] or undefined behaviors in standards [43]. Others applied coverage-guided fuzzing to RAN–Core interfaces [78] and NAS handlers in the core network [79], uncovering memory bugs in open-sources.

Dynamic analysis has also been explored through emulation and over-the-air fuzzing. Maier et al. [9] and Grassi and Chen [58] emulated MediaTek's RRC/NAS layers, while Hernandez et al. [8] emulated full Exynos and MediaTek firmware using AFL++ [80], discovering multiple memory bugs. Silvanovich et al. [81] extended this to SDP and SIP, revealing additional multiple memory bugs. However, these approaches can only be applied to devices with emulatable firmware, which limits their applicability to a specific range of devices. Recently, Lisowski et al. [50] introduced a SIM-side research tool, which could be integrated to emulation-based approaches that can improve state exploration.

Over-the-air fuzzing has also shown promise, particularly in detecting memory bugs at Layer 2 and 3 [82], [37], [44]. Potnuru et al. [29] applied specification-guided fuzzing to open-source LTE stacks, discovering bugs in the EPC. Beyond cellular, researchers have explored OTA fuzzing in Wi-Fi [83] and Bluetooth [84]. Similar to prior OTA efforts, OTABase uses over-the-air fuzzing—-but uniquely enables systematic memory bug detection in commercial LTE BPs.

9. Conclusion

While memory vulnerabilities in BPs pose critical security risks, their black-box nature has constrained research focused on detecting them. We present OTABase, a framework that uses the OTA interface to efficiently detect memory crashes in LTE BPs, combining network-side state control, specification-guided test generation, and two-phase crash detection. Testing six commercial BPs, OTABase revealed seven previously unpatched vulnerabilities–including three CVEs with one enabling RCE–demonstrating its effectiveness and showing that systematic memory testing is achievable through specifications without access to proprietary firmware. As BPs evolve, future work should expand OTABase beyond downlink LTE NAS and RRC to cover uplink traffic, 5G protocols, and non-terrestrial networks, which all introduce new challenges in protocol and state handling. Such expansion will be crucial for comprehensive testing of wireless infrastructure, especially in emerging domains like industrial control and robotics.

Acknowledgments

This work was supported by Institute of Information & Communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (IITP-2025-RS-2023-00266615, Convergence Security Core Talent Training Business Support Program, and No.IITP-2025-RS-2024-00437252, Development of Anti-sniffing Technology in Mobile Communication and AirGap Environments).

References

[1] D. Rupprecht, K. Jansen, and C. Pöpper, "Putting LTE security functions to the test: A framework to evaluate implementation correctness," in *USENIX Workshop on Offensive Technologies*, 2016.

[2] A. Shaik, R. Borgaonkar, N. Asokan, V. Niemi, and J.-P. Seifert, "Practical attacks against privacy and availability in 4G/LTE mobile communication systems," in *Network and Distributed System Security Symposium*, 2016.

[3] H. Kim, J. Lee, E. Lee, and Y. Kim, "Touching the untouchables: Dynamic security analysis of the LTE control plane," in *IEEE Symposium on Security and Privacy*, 2019.

[4] C. Park, S. Bae, B. Oh, J. Lee, E. Lee, I. Yun, and Y. Kim, "DoLTEst: In-depth downlink negative testing framework for LTE devices," in *USENIX Security Symposium*, 2022.

[5] S. M. M. Rashid, T. Wu, K. Tu, A. A. Ishtiaq, R. H. Tanvir, Y. Dong, O. Chowdhury, and S. R. Hussain, "State machine mutation-based testing framework for wireless communication protocols," in *ACM Conference on Computer and Communications Security*, 2024.

[6] E. Kim, D. Kim, C. Park, I. Yun, and Y. Kim, "BASESPEC: Comparative analysis of baseband software and cellular specifications for L3 protocols," in *Network and Distributed System Security Symposium*, 2021.

[7] E. Kim, M. W. Baek, C. Park, D. Kim, Y. Kim, and I. Yun, "BASECOMP: A comparative analysis for integrity protection in cellular baseband software," in *USENIX Security Symposium*, 2023.

[8] G. Hernandez, M. Muench, D. Maier, A. Milburn, S. Park, T. Scharnowski, T. Tucker, P. Traynor, and K. Butler, "FIRMWIRE: Transparent dynamic analysis for cellular baseband firmware," in *Network and Distributed System Security Symposium*, 2022.

[9] D. Maier, L. Seidel, and S. Park, "BaseSAFE: baseband sanitized fuzzing through emulation," in *ACM Conference on Security and Privacy in Wireless and Mobile Networks*, 2020.

[10] Y. Chen, Y. Yao, X. Wang, D. Xu, X. Liu, C. Yue, K. Chen, H. Tang, and B. Liu, "Bookworm game: Automatic discovery of LTE vulnerabilities," in *IEEE Symposium on Security and Privacy*, 2021.

[11] A. Al Ishtiaq, S. S. S. Das, S. M. M. Rashid, A. Ranjbar, K. Tu, T. Wu, Z. Song, W. Wang, M. Akon, R. Zhang, and S. R. Hussain, "Hermes: unlocking security analysis of cellular network protocols by synthesizing finite state machines from natural language specifications," in *USENIX Security Symposium*, 2024.

[12] S. R. Hussain, I. Karim, A. A. Ishtiaq, O. Chowdhury, and E. Bertino, "Noncompliance as deviant behavior: An automated black-box noncompliance checker for 4g lte cellular devices," in *ACM Conference on Computer and Communications Security*, 2021.

[13] K. Tu, A. Al Ishtiaq, S. M. M. Rashid, Y. Dong, W. Wang, T. Wu, and S. R. Hussain, "Logic Gone Astray: A Security Analysis Framework for the Control Plane Protocols of 5G Basebands," in *USENIX Security Symposium*, 2024.

[14] S. Hussain, O. Chowdhury, S. Mehnaz, and E. Bertino, "LTEInspector: A systematic approach for adversarial testing of 4G LTE," in *Network and Distributed System Security Symposium*, 2018.

[15] S. R. Hussain, M. Echeverria, I. Karim, O. Chowdhury, and E. Bertino, "5GReasoner: A property-directed security and privacy analysis framework for 5G cellular network protocol," in *ACM Conference on Computer and Communications Security*, 2019.

[16] S. Erni, M. Kotuliak, P. Leu, M. Roeschlin, and S. Capkun, "Adaptover: adaptive overshadowing attacks in cellular networks," in *International Conference on Mobile Computing and Networking*, 2022.

[17] A. Dabrowski, N. Pianta, T. Klepp, M. Mulazzani, and E. Weippl, "IMSI-catch me if you can: IMSI-catcher-catchers," in *ACM Conference on Computer and Communications Security*, 2014.

[18] H. Yang, S. Bae, M. Son, H. Kim, S. M. Kim, and Y. Kim, "Hiding in plain signal: Physical signal overshadowing attack on LTE," in *USENIX Security Symposium*, 2019.

[19] D. Rupprecht, K. Kohls, T. Holz, and C. Pöpper, "Breaking LTE on layer two," in *IEEE Symposium on Security and Privacy*, 2019.

[20] ——, "Imp4gt: impersonation attacks in 4G networks," in *Network and Distributed System Security Symposium*, 2020.

[21] R.-P. Weinmann, "Baseband attacks: Remote exploitation of memory corruptions in cellular protocol stacks." in *USENIX Workshop on Offensive Technologies*, 2012.

[22] D. Komaromy, "Basebanheimer: Now I Am Become Death, The Destroyer Of Chains," *OffensiveCon*, 2023.

[23] G. Miru, "Path of Least Resistance: Cellular Baseband to Application Processor Escalation on Mediatek Devices," *Comsecuris Blog*, 2017.

[24] N. Golde, "There's life in the old dog yet: Tearing new holes into intel/iphone cellular modems," *Comsecuris Blog*, 2018.

[25] M. Grassi, M. Liu, and T. Xie, "Exploitation of a modern smartphone baseband," *BlackHat USA*, 2018.

[26] M. Grassi and X. Chen, "Over the air baseband exploit: Gaining remote code execution on 5g smartphones," *BlackHat USA*, 2021.

[27] A. Cama, "A walk with shannon," *OPCDE*, 2018.

[28] ——, "ASN.1 and Done: A Journey of Exploiting ASN.1 Parsers in the Baseband." *OffensiveCon*, 2023.

[29] S. Potnuru and P. K. Nakarmi, "Berserker: ASN.1-based Fuzzing of Radio Resource Control Protocol for 4G and 5G," *arXiv preprint arXiv:2107.01912*, 2021. [Online]. Available: https://arxiv.org/abs/2107.01912

[30] "Openairinterface," "https://gitlab.eurecom.fr/oai/openairinterface5g".

[31] I. Gomez-Miguelez, A. Garcia-Saavedra, P. D. Sutton, P. Serrano, C. Cano, and D. J. Leith, "srsLTE: An Open-Source Platform for LTE Evolution and Experimentation," in *ACM International Workshop on Wireless Network Testbeds, Experimental Evaluation, and Characterization*, 2016.

[32] M. Muench, J. Stijohann, F. Kargl, A. Francillon, and D. Balzarotti, "What you corrupt is not what you crash: Challenges in fuzzing embedded devices." in *Network and Distributed System Security Symposium*, 2018.

[33] 3GPP. TS 36.331, v15.10.0, "LTE RRC Protocol specification," 2020.

[34] 3GPP. TS 24.301, v16.5.1, "Non-Access-Stratum (NAS) protocol for Evolved Packet System (EPS); Stage 3," 2020.

[35] International Telecommunication Union, "ASN.1 Encoding Rules: Specification of Packed Encoding Rules (PER)," ITU-T Recommendation X.691, February 2021, accessed: 2024-06-09. [Online]. Available: https://www.itu.int/rec/T-REC-X.691

[36] T. D. Hoang, T. Oh, C. Park, I. Yun, and Y. Kim, "Llfuzz: An over-the-air dynamic testing framework for cellular baseband lower layers," in *USENIX Security Symposium*, 2025.

[37] M. E. Garbelini, Z. Shang, S. Luo, and S. Chattopadhyay, "5GHOUL: Unleashing Chaos on 5G Edge Devices," SUTD, Tech. Rep., 2023.

[38] E. Bitsikas, S. Khandker, A. Salous, A. Ranganathan, R. Piqueras Jover, and C. Pöpper, "UE security reloaded: Developing a 5g standalone user-side security testing framework," in *ACM Conference on Security and Privacy in Wireless and Mobile Networks*, 2023.

[39] I. Karim, S. Hussain, and E. Bertino, "ProChecker: An automated security and privacy analysis framework for communication protocol," in *IEEE International Conference on Distributed Computing Systems*, 2021.

[40] M. Chlosta, D. Rupprecht, T. Holz, and C. Pöpper, "LTE security disabled: misconfiguration in commercial networks," in *ACM Conference on Security and Privacy in Wireless and Mobile Networks*, 2019.

[41] D. Klischies, D. Goos, D. Hirsch, A. Milburn, M. Muench, and V. Moonsamy, “Basebridge: Bridging the gap between emulation and over-the-air testing for cellular baseband firmware,” in *IEEE Symposium on Security and Privacy*, 2025.

[42] A. Ranjbar, T. Yang, K. Tu, S. Khalilollahi, and S. R. Hussain, “Stateful analysis and fuzzing of commercial baseband firmware,” in *IEEE Symposium on Security and Privacy*, 2025.

[43] D. Klischies, M. Schloegel, T. Scharnowski, M. Bogodukhov, D. Rupprecht, and V. Moonsamy, “Instructions Unclear: Undefined Behaviour in Cellular Network Specifications,” in *USENIX Security Symposium*, 2023.

[44] C. Mulliner, N. Golde, and J.-P. Seifert, “SMS of Death: From Analyzing to Attacking Mobile Phones on a Large Scale,” in *USENIX Security Symposium*, 2011.

[45] “Firmwire’s experimental data,” https://github.com/FirmWire/ndss22_experiments/.

[46] “Modemmanager, freedesktop.org,” https://www.freedesktop.org/wiki/Software/ModemManager/.

[47] 3GPP. TS 33.401, v16.3.0, “Security architecture,” 2020.

[48] H. Lin, “LTE REDIRECTION: Forcing targeted LTE cellphone into unsafe network,” in *Hack In The Box Security Conference (HITBSecConf)*, 2016.

[49] G. Lee, J. Lee, J. Lee, Y. Im, M. Hollingsworth, E. Wustrow, D. Grunwald, and S. Ha, “This is your president speaking: Spoofing alerts in 4G LTE networks,” in *ACM International Conference on Mobile Computing Systems (MobiSys)*, 2019.

[50] T. P. Lisowski, M. Chlosta, J. Wang, and M. Muench, “SIMurai: Slicing through the complexity of SIM card security research,” in *USENIX Security Symposium*, 2024.

[51] “Hacked documents: how Iran can track and control protesters’ phones,” https://theintercept.com/2022/10/28/iran-protests-phone-surveillance/.

[52] R. Mayrhofer, J. Vander Stoep, C. Brubaker, D. Hackborn, B. Bonné, G. S. Tuncay, R. P. Jover, and M. Specter, “The android platform security model (2023),” *arXiv preprint arXiv:1904.05572*, 2024.

[53] “Pycrate, a Python library containing ASN.1 compiler,” https://github.com/P1sec/pycrate.

[54] “Marben Products, ASN.1 Decoder for LTE Networks,” https://www.marben-products.com/decoder-asn1-lte/.

[55] “USRP B210,” https://www.ettus.com/UB210-KIT.

[56] “Sysmocom SIM/USIM/ISIM cards,” https://www.sysmocom.de/products/lab/sysmousim/index.html.

[57] 3GPP. TS 24.008, v16.3.0, “Mobile radio interface Layer 3 specification; Core network protocols; Stage 3,” 2019.

[58] M. Grassi and X. Chen, “Exploring the mediatek baseband,” *OffensiveCon*, 2020.

[59] T. D. Hoang, C. Park, M. Son, T. Oh, S. Bae, J. Ahn, B. Oh, and Y. Kim, “Ltesniffer: An open-source lte downlink/uplink eavesdropper,” in *ACM Conference on Security and Privacy in Wireless and Mobile Networks*, 2023.

[60] “GSMA IMEI Allocation and Approval Process,” https://imeidb.gsma.com/imei/resources/documents/TS.06%20v26.0.pdf.

[61] A. Shaik, R. Borgaonkar, S. Park, and J.-P. Seifert, “New vulnerabilities in 4G and 5G cellular access network protocols: exposing device capabilities,” in *ACM Conference on Security and Privacy in Wireless and Mobile Networks*, 2019.

[62] B. Oh, J. Ahn, S. Bae, M. Son, Y. Lee, and Y. Kim, “Preventing sim box fraud using device model fingerprinting.” in *Network and Distributed System Security Symposium*, 2023.

[63] “Tools for samsung shannon baseband,” https://github.com/grant-h/ShannonBaseband.

[64] D. Berard and V. Fargues, “How to design a baseband debugger,” in *Information and Communication Technology Security Symposium (SSTIC), Rennes, France*, 2020.

[65] N. Golde and D. Komaromy, “Breaking band: reverse engineering and exploiting the shannon baseband,” *REcon*, 2016.

[66] S. R. Systems, “srsRAN Project,” https://github.com/srsran/srsRAN_Project.

[67] “Accuver XCAL,” http://www.accuver.com/.

[68] E. Kim, “Comparative analysis of baseband software implementation and cellular specification for layer 3 protocols,” Ph.D. dissertation, KAIST, 2022.

[69] 3GPP. TS 44.071, “GSM/EDGE Location Services (LCS),” 2022.

[70] B. Karakoc, N. Fürste, D. Rupprecht, and K. Kohls, “Never let me down again: Bidding-down attacks and mitigations in 5g and 4g,” in *ACM Conference on Security and Privacy in Wireless and Mobile Networks*, 2023.

[71] D. Basin, J. Dreier, L. Hirschi, S. Radomirovic, R. Sasse, and V. Stettler, “A formal analysis of 5G authentication,” in *ACM Conference on Computer and Communications Security*, 2018.

[72] M. Akon, T. Yang, Y. Dong, and S. R. Hussain, “Formal analysis of access control mechanism of 5g core network,” in *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*, 2023, pp. 666–680.

[73] Y. Chen, D. Tang, Y. Yao, M. Zha, X. Wang, X. Liu, H. Tang, and D. Zhao, “Seeing the forest for the trees: Understanding security hazards in the 3GPP ecosystem through intelligent analysis on change requests,” in *USENIX Security Symposium*, 2022.

[74] Y. Chen, D. Tang, Y. Yao, M. Zha, X. Wang, X. Liu, H. Tang, and B. Liu, “Sherlock on Specs: Building LTE Conformance Tests through Automated Reasoning,” in *USENIX Security Symposium*, 2023.

[75] M. Chlosta, D. Rupprecht, and T. Holz, “On the challenges of automata reconstruction in lte networks,” in *ACM Conference on Security and Privacy in Wireless and Mobile Networks*, 2021.

[76] C. Mulliner and C. Miller, “Fuzzing the phone in your phone,” *Black Hat USA*, 2009.

[77] D. Komaromy and L. Szabo, “How to tame your unicorn: exploring and exploiting zero-click remote interfaces of modern Huawei smartphones,” *BlackHat USA*, 2021.

[78] N. Bennett, W. Zhu, B. Simon, R. Kennedy, W. Enck, P. Traynor, and K. R. Butler, “Ransacked: A domain-informed approach for fuzzing lte and 5g ran-core interfaces,” in *ACM Conference on Computer and Communications Security*, 2024.

[79] F. He, W. Yang, B. Cui, and J. Cui, “Intelligent fuzzing algorithm for 5g nas protocol based on predefined rules,” in *2022 International Conference on Computer Communications and Networks (ICCCN)*. IEEE, 2022.

[80] Marc Heuse, Heiko Eißfeld, Andrea Fioraldi, and Dominik Maier, “AFLplusplus (AFL++),” https://github.com/vanhauser-thc/AFLplusplus.

[81] N. Silvanovich, “How to hack shannon baseband (from a phone),” *OffensiveCon*, 2023.

[82] M. E. Garbelini, Z. Shang, S. Chattopadhyay, S. Sun, and E. Kurniawan, “Towards Automated Fuzzing of 4G/5G Protocol Implementations Over the Air,” in *IEEE Global Communications Conference*, 2022.

[83] H. Cao, L. Huang, S. Hu, S. Shi, and Y. Liu, “Owfuzz: Discovering Wi-Fi Flaws in Modern Devices through Over-The-Air Fuzzing,” in *ACM Conference on Security and Privacy in Wireless and Mobile Networks*, 2023.

[84] M. E. Garbelini, V. Bedi, S. Chattopadhyay, S. Sun, and E. Kurniawan, “BrakTooth: Causing havoc on bluetooth link manager via directed fuzzing,” in *USENIX Security Symposium*, 2022.

[85] “Objective Systems. ASN1VE: The ASN.1 Viewer/Editor,” https://obj-sys.com/products/asn1ve/index.php.

Appendix A. EMM States in LTE

This appendix provides detailed descriptions of the five main EMM (EPS Mobility Management) states defined in the LTE specification (3GPP TS 24.301 clause 5.1.3). Each state corresponds to a specific point in the UE's connectivity lifecycle and has distinct security characteristics.

EMM-Deregistered (*DEREGI*): UE is not connected to the network in this state. The UE remains in *DEREGI* when it is switched off or has not yet found a cell to connect. Also, UE transitions to this state from others when an unexpected failure occurs during protocol operations, or when it receives certain `Reject` message during the signaling procedure.

EMM-Registered-Initiated (*REGI-Init*): UE enters this state by initiating the `Attach` procedure with an `Attach Request`. During this state, unprotected messages are exchanged to complete the `Attach` procedure, including authentication and key establishment. UE transitions to *REGI* if successful, or to *DEREGI* if it fails.

EMM-Registered (*REGI*): UEs in this state completed the `Attach` procedure and registered to the network. All NAS and RRC messages are encrypted and integrity protected except for a few error-handling messages. UE transitions to *DEREGI* when it detaches from the network.

EMM-Tracking-Area-Update-Initiated (*TAU-Init*): UE moves to this state when it decides to initiate the `TAU` procedure by sending `TAU Request`. In this state, only NAS messages are protected since security keys for RRC need to be re-established. UE returns to *REGI* if the procedure succeeds, or moves to either *REGI* or *DEREGI* if it fails.

EMM-Service-Request-Initiated (*SR-Init*): Similar to *TAU-Init*, UE moves to this state when it decides to initiate `Service Request` procedure and only NAS messages are protected during this state. UE always returns to *REGI* state whether the procedure succeeds or fails.

Appendix B. Analysis on back-off behavior

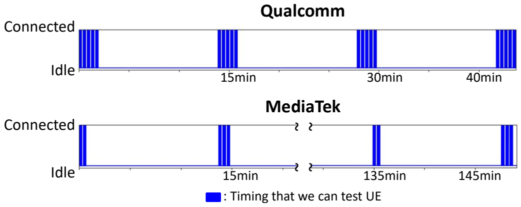

Figure 9: Different connection patterns in *REGI-Init*

TABLE 9: Back-Off measurements of different BPs

Chipset	Period (s)	Downtime (s)	Ratio (%)
Exynos (Galaxy S21)	∼1160	∼830	∼87.07
Qualcomm (Galaxy Z Flip 4)	835	760	91.02
MediaTek (Galaxy A32)	8065	7990	99.07
HiSilicon (Huawei P20)	∼835	∼760	91.02

We conducted experiments to analyze how back-off behavior impacts over-the-air testing across different baseband implementations during the signaling procedures (§2.3). In particular, we measured back-off behavior of different baseband implementations in *REGI-Init*, the most common target in cellular security research, to understand their available testing windows. Our experiments included devices with Samsung Exynos, Qualcomm, MediaTek, and HiSilicon basebands. When UE connects, we move it to *REGI-Init* and send `NAS Identity Request` messages at 0.02 second intervals. Note that UEs must respond with `Identity Response` messages in this state. Thus, the UE continues to respond as long as it remains connected, and stops responding when it switches to idle state. Through this method, we measured the duration of connected and idle periods for COTS UEs.

During our experiment, all devices initially remain connected in *REGI-Init* for approximately 15 seconds (as defined by T3410 in [34]) before entering the idle state. The devices then stay idle for 10 seconds before reconnecting to the network. However, after this first reconnection cycle, BPs exhibit different behavior patterns. Qualcomm and HiSilicon BPs repeat this reconnection cycle with 10-second idles three more times. At the fifth cycle, instead of 10 seconds, the UE backs off for 720 seconds. After exiting the back-off, they start again from their first reconnection cycle, as shown in Fig. 9. In contrast, the MediaTek BPs backs off for about 760 seconds during their second reconnection cycle. Moreover, at the fifth cycle, their back-off time increases to 7200 seconds. Thereafter, they also restart from their first reconnection cycle.

Tab. 9 summarizes our measurements across the different BP manufacturers. The period column represents the total time from the first reconnection cycle to the end of back-off status, while the downtime column shows the total time spent in idle. The last column presents the ratio of downtime to the total period.

Appendix C. Analyzing a 1-day bug

We analyzed a previously published one-day payload [45] to understand how malformed fields trigger crashes. Using an unpatched Galaxy S10 (Exynos BP) with vulnerable firmware, we reproduced the crash by transmitting the payload in the *REGI-Init* state and confirmed it with Samsung's debug mode. We then decoded the payload using standard RRC decoders [54], [85]. Note that reproducing such one-day bugs is generally difficult in practice: public disclosures (e.g., vendor patch reports and CVEs) rarely include full payloads, and while research papers and conference talks provide useful analysis [28], [27], [81], [25], [26], they seldom publish complete payloads. Moreover, anti-rollback protection typically prevents downgrading devices to vulnerable firmware versions, further limiting reproducibility.

Fortunately, we were able to obtain an unpatched Galaxy S10 running vulnerable firmware, which allowed us to reproduce one bug reported in FirmWire [45]. We modified srsRAN [31] to transmit the following payload over-the-air when the test device connects in *REGI-Init* state:

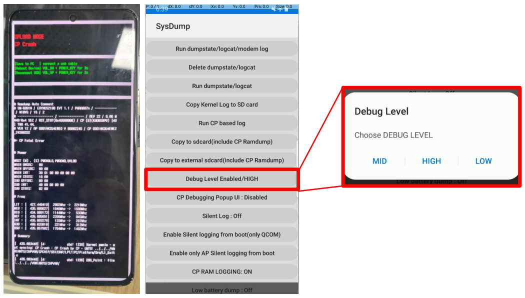
UPLOAD MODE
CP Crash
SysDump
Run dumpstate/logcat/modem log
Delete dumpstate/logcat
Run dumpstate/logcat
Copy Kernel Log to SD card
Run CP based log
Copy to sdcard(include CP Ramdump)
Copy to external sdcard(include CP Ramdump)
Debug Level Enabled/HIGH
CP Debugging Popup UI : Disabled
Silent Log : Off
Enable Silent logging from boot(only QCOM)
Enable only AP Silent logging from boot
CP RAM LOGGING: ON
Low battery dump : Off
Debug Level
Choose DEBUG LEVEL
MID
HIGH
LOW

Figure 10: Manufacturer's troubleshooting features

```
RRC_RRCConnectionReconfiguration_v890_IEs_PREFETCH_ABORT
2205800068079696969696969696969696969696169696969696967ff
7ff002000b4686835dbfff10064212f8b696e
```

We confirmed that our reproduction caused the crash by using Samsung's manufacturer debug mode. This further allow us to perform ablation studies; we found that the first seven bytes (up to `0x96`) were sufficient to trigger the crash. ASN.1 decoding (see Listing 1) further revealed that the problematic element was `lateNonCriticalExtension` within `RRCConnectionReconfiguration-v8m0-IEs`, where the `OCTET STRING` length value (`0x96`) exceeded the actual data size. This invalid length encoding caused the crash.

Listing 1: ASN.1 decoding result of FirmWire 1-day payload

```
<encoding> 22058000 680796 </encoding>
<DL–DCCH–Message> <message> <c1>
      <rrcConnectionReconfiguration>
         [...]
        <nonCriticalExtension>
          < lateNonCriticalExtension >
            <RRCConnectionReconfiguration–v8m0–IEs>
              <!–– Problematic field that triggered decoding error ––>
              < lateNonCriticalExtension / >
            </RRCConnectionReconfiguration–v8m0–IEs>
          </ lateNonCriticalExtension >
        </nonCriticalExtension >
      </rrcConnectionReconfiguration >
    </c1> </message> </DL–DCCH–Message>

<error>
  <description>Unexpected end of stream reached when decoding</
      description>
  <nature>fr.marben.asnsdk.japi . InconsistentStructureException </nature>
  <ErrorOffset>1</ErrorOffset>
  <ValuePath>DL–DCCH–Message.message.c1.rrcConnectionReconfiguration.
      criticalExtensions.c1.rrcConnectionReconfiguration–r8.
       nonCriticalExtension . lateNonCriticalExtension .
      RRCConnectionReconfiguration–v8m0–IEs.lateNonCriticalExtension</
      ValuePath>
</error>
```

Appendix D. Samsung's troubleshooting feature

Samsung devices provide a range of troubleshooting features accessible through their SysDump menu. As shown in Fig. 10, these features include options for setting the debug level. By default, the debug level is set to low, during which a memory crash causes the modem to silently crash and reboot. However, when set to mid or high, the device enters a kernel panic state, as illustrated on the left side.

Appendix E. Full payload used to discover our bugs

We provide a full payload of our test cases that were used to discover our bugs, with a brief explanation on them. In the payload, blue highlights the problematic IE, while red indicates their abnormal length values. Messages requiring authentication start with `0x27`, followed by a 5-byte field consisting of the valid message authentication code and sequence number. Messages that can be transmitted in plaintext start with `0x07`.

V1: "Short name for network" IE in EMM Information

```
27 MA CF IE LD SN 07 61 45 ff 82 d8 04 8d b0 be 75 c7 07 7a 8c 3a 84 e5
4e 65 e0 b0 38 e8 6a 71 8d f5 35 09 8e 9d 5d b5 98 a7 3b 32 ac 3d ef 72
7f 79 3c 25 64 1e d5 af 1a 03 e1 17 ce ac dd c7 12 62 ff 93 7d 7e 9c 16
71 ef 8b cc 81 ec f5 2b fd d7 7a cd 0c 2d 90 29 23 3d 0c 08 5a 87 ba 5e
c5 1b 01 fc 96 03 98 7b 69 a7 54 c5 7e 12 02 85 f0 2c 74 34 93 c5 93 f3
b9 fd 6c 73 44 7d 50 42 f5 b6 55 3a fc 78 d2 00 10 05 e7 c4 78 a7 2d c7
2c e1 3d 06 62 bf e8 0c ab 15 f7 7d 13 6a a0 03 b4 c0 56 a5 c2 42 d5 d9
36 6b e4 34 51 e6 3f c1 3b 83 92 7e 31 bc 62 3f ca ec 94 b6 09 99 76 51
ac 03 99 37 68 15 e0 ab 61 3c ff 27 56 1a cf c6 80 4f 1b 04 f8 23 02 d2
85 e0 95 b0 a6 e6 0d c1 d4 8a c0 a1 ed 87 cc b2 64 7d d2 a6 f9 65 16 ca
0a e0 e3 ae 5f ad 0e 9b d2 cc b8 fb b1 c6 00 82 59 0b 16 5e 03 55 9c ab
0d 49 89 01
```

`EMM Information` message containing the `short name for network` IE (`0x45`) with a length greater than or equal to 0xe7 crashes the MediaTek BPs. The the IE is indicated by two bytes (blue) `0x45`, and the length of the IE is specified by the following byte `0xff` (red). Although the specification does not restrict the length of this IE, the baseband crashes if the length is greater than or equal to `0xe7`.

V2: "Additional Information" IE in Downlink generic NAS transport

```
27 MA CF IE LD SN 07 68 02 00 02 a9 d0 65 00 d0
```

`Downlink generic NAS transport` message containing the optional IE, `additional information` (0x65) with a length value of 0x00 crashes the MediaTek basebands. The payload of the `additional information` IE consists of '6500d0'. Two bytes (blue), 0x65, indicate the IEI, and the following byte 0x00 (red) specifies the length of the field, followed by 'd0' as the contents. According to MediaTek, **V2** triggers a memory allocation with size 0; they classified this as low severity since it only results in temporary DoS after authentication (*i.e. REGI*).

V3: "NAS message container" IE in Downlink NAS transport

```
27 MA CF IE LD SN 07 62 00 09 10 51
```

`Downlink NAS transport` message containing the `NAS message container` IE with a length value of 0x00 crashes the MediaTek baseband. The IE is mandatory and thus contains only the length and value fields without IEI. The payload after '0x62' indicates the `Downlink NAS transport` IE, where the first two bytes (0x00) specify the length, followed by the contents. According to MediaTek, **V3** results in an invalid zero-length memory allocation, leading to a crash. This issue was classified as low severity.

V4: "Emergency number list" IE in Attach Accept

```
1) 27 MA CF IE LD SN 07 42 01 10 12 2c 78 09 74 f1 ca 00 91 96 05 c7 95
2e 27 02 97 01 14 00 03 00 00 00 34 0c ff 15 03 12 d7 95 02 1f 8a 02 00
e8 64 02 eb 00
2) 27 MA CF IE LD SN 07 42 02 56 17 28 11 31 93 f7 0f 41 29 19 84 74 8d
01 71 86 4a 65 26 74 66 71 ad ea 00 03 00 00 00 34 ff 04 0d 74 b7 7d 04
0c 8c 78 a7 04 0c bc e3 09 04 10 93 51 88 f0
```

NAS Attach Accept message containing the Emergency number list IE (0x34) with an abnormally large length value crashes the Exynos baseband. According to the specification [57], the length of this IE should be between 5 and 50 bytes. We identified two payload patterns that trigger this crash: [Payload 1] consists of '340cff', where the Emergency number list IE (0x34, blue) has a length of 0x0c, followed by an invalid first Emergency information length of 0xff (red). [Payload 2] contains five Emergency Number information elements, where the first four elements have valid 4-byte lengths, but the fifth element has an abnormal length value, 0xf0 (red).

V5: Detach Accept

```
07 46 01 0c 38 28 46 61 03 4b e8 05 63 16 34 89 2b 37 1d a6 a7 9f 43 0b
b4 67 3c 07 03 67 21 c6 f1 25 bc 32 6d d4 31 08 9a 32 df d8 ab ca c8 2d
77 64 13 51 29 01 98 83 65 6a 26 c7 d2 00 03 00 00 00 5e 01 de
```

NAS Detach Accept message containing additional data crashes the Exynos baseband. Per 3GPP specifications, this messages must be fixed-length with no additional IEs, having payloads of 0x0746. Thus any payload after the message type (first 2 bytes) violates the specification, and triggers a crash. Through ablation study, we confirmed that even a minimal payload of '074601' is sufficient to trigger the crash. Samsung marked **V5** as a duplicate of a known issue, and a CVE was later assigned by the Android security team. This issue affects 16 chipsets.

V6: Authentication Reject

```
07 54 01 0c 38 28 46 61 03 4b e8 05 63 16 34 89 2b 37 1d a6 a7 9f 43 0b
b4 67 3c 07 03 67 21 c6 f1 25 bc 32 6d d4 31 08 9a 32 df d8 ab ca c8 2d
77 64 13 51 29 01 98 83 65 6a 26 c7 d2 00 03 00 00 00 4a 09 72 05 54 93
42 19 75 18 68
```

NAS Authentication Reject message containing additional data crashes the baseband, similar to **V5**.

V7: "dedicatedInfoNAS" field in DLInformationTransfer

```
0a 00 00
```

RRC DLInformationTransfer message containing the dedicatedInfoNAS field with a length value of 0 crashes the MediaTek basebands. The dedicatedInfoNAS field carries a primitive type OCTET STRING with variable length. Thus, it consists a length field followed by contents.

Appendix F. Other findings

While developing OTABase, we discovered two misimplementations unrelated to memory in Exynos BPs.

First, we found an NAS security context handling issue. When a UE receives a Detach Request with the detach type *re-attach required*, it improperly deletes its NAS security context while still marking the following Attach Request as integrity protected (security header type 1) with an empty MAC (0x00000000). This behavior violates clause 7.2.5 of [47], which requires maintaining authentication data during re-attach. We demonstrated that this vulnerability enables unauthorized access to sensitive information. Specifically, after the UE entered this state, we sent NAS messages with invalid MACs: an Identity Request for IMEI and an EMM Information. Though the specification mandates discarding such messages, the UE accepted them, leaking a sensitive device identifier (IMEI) and allowing unauthorized modification of the device's time. Samsung classified this issue as low severity, and subsequently released a patch.

Second, we observed improper protocol state handling of Service Reject messages. According to [34], these messages should only be processed during the Service Request procedure (*i.e. SR-Init*), and ignored in other states like *REGI-Init* or *REGI*. However, Exynos BPs always disconnect upon receiving Service Reject regardless of state, then initiate a new Service Request. This behavior is unspecified and abnormal. Samsung reviewed this issue but did not classify it as a vulnerability requiring remediation.

Appendix G. Test case breakdown

TABLE 10: Number of test messages per NAS message type

Message name	Attach Accept	Detach Request	Detach Accept	TAU Accept	TAU Reject	Service Reject	Service Accept	GUTI Reallocation Command	Authentication Request
Number	12686	399	149	12657	940	1476	1258	2914	748
Message name	Authentication Reject	Identity Request	Security Mode Command	EMM Status	EMM Information	Downlink NAS Transport	CS Service Notification	Downlink Generic NAS Transport	
Number	149	250	2437	200	2275	600	1597	1207	

TABLE 11: Number of test messages per RRC message type

Message name	csfbParameters Response CDMA2000	dlInformation Transfer	handover FromEUTRA Preparation Request	mobility From EUTRA Command	rrcConnection Reconfiguration	security Mode Command
Number	138	750	138	726	108183	69
Message name	ueCapability Enquiry	counter Check	ueInformation Request-r9	logged Measurement Configuration-r10	rnReconfigu ration-r10	rrcConnection Resume-r13
Number	360	99	81	396	1797	40668
Message name	dlDedicated Message Segment-r16					
Number	150					

Tab. 10 and Tab. 11 present the number of test messages for each message type in NAS and RRC used by OTABase for testing. The large number of test messages for specific types, such as RRC Connection Reconfiguration and RRC Connection Resume, is primarily due to the extensive number of optional fields these messages can include.

RRC messages inherently possess a unique characteristic where optional fields can themselves include additional optional fields, creating a recursive structure. This recursive nature, combined with the large variety of optional fields present in these message types, allows packets to include a substantial number of security-sensitive fields, with the same field potentially appearing in multiple locations within a single packet. To ensure thorough testing of security-sensitive fields, all possible variations of such packets were generated and mutated, resulting in the high number of test messages observed.