Instructions Unclear: Undefined Behaviour in Cellular Network Specifications

Daniel Klischies
Ruhr University Bochum

Moritz Schloegel
CISPA Helmholtz Center for Information Security

Tobias Scharnowski
CISPA Helmholtz Center for Information Security

Mikhail Bogodukhov
Independent[*]

David Rupprecht
Radix Security

Veelasha Moonsamy
Ruhr University Bochum

Abstract

Cellular networks are a cornerstone of modern communication and indispensable to our daily lives. Their specifications span thousands of pages, written primarily in natural language. The ensuing complexity and lack of explicitness lead to underspecification, where only subsets of possible interactions are properly specified, while other behaviour is left undefined and open to interpretation by developers. In practice, this causes weird, unintended interactions in smartphone modems implementing the specification that, in the worst case, lead to security vulnerabilities.

In this work, we present the first generic approach for systematically discovering undefined behaviour in cellular specifications. Requiring solely a model of the behaviour defined in the specification, our technique extends this model to automatically reason about the presence of undefined behaviour. For each undefined behaviour, it automatically infers concrete examples as proof of existence. This not only allows improving the specification but also enables us to test smartphone modems. This way, we can verify whether an instance of undefined behaviour leads to a security vulnerability within modem firmware. With our approach, we identify 58 cases of undefined behaviour in LTE's Public Warning System, SMS, and Radio Resource Control specifications. Five of these cases resulted in previously unknown vulnerabilities that allow adversaries to read modem memory contents and perform remote Denial of Service attacks (in one case just via a single SMS) against commonly used smartphone modems. So far, two CVEs of high and one CVE of critical severity have been assigned.

1 Introduction

Cellular networks play an essential role in global communications. The foundations enabling this globally interoperable communication network are the *specifications* issued by the 3rd Generation Partnership Project (3GPP), an international specification consortium. The specifications define the communication protocol stack, from radio frequencies, mobility management, authentication and encryption to a variety of applications, such as text messages, emergency alerts, or web browsing. Due to the abundance of applications and mobility requirements, the individual layers of a cellular network are highly complex and cannot be compared to protocols with a single purpose, such as TCP/IP. As a result, the specifications reflect this complexity.

Moreover, as the specifications are written primarily in natural language, the documentation does not specify every possible scenario a device could be presented with. For instance, although the specifications define several message fragmentation and reassembly procedures, the expected behaviour in case of changing message metadata is left for interpretation by developers implementing the specifications. Subsequently, this gives rise to *undefined behaviour*, whereby the responsibility of deciding how to handle such situations is delegated to the manufacturers of user and network equipment. In reality, when faced with nonconforming messages, phone modems may simply display error messages, crash, disconnect, allow an attacker to dump memory, send sensitive information without encryption, or behave in any other way. Without a specified behaviour, there is no definitive decision on how to react, leading to diverse and vulnerable implementations rather than standardised and sane procedures.

The importance of cellular protocol specifications and the impact of potential perils have been extensively researched in the past [12, 21, 24–26, 37]. Most of the existing work, and the specification's own test suite, focus on testing *positive behaviour*, i. e., verifying whether modem firmware adheres to the specified behaviour or constraints [12, 21, 24–26]. Only recently, Park et al. [37] proposed to employ *negative testing* to find vulnerabilities in phone modem implementations. To do so, they identify Protocol Data Units (PDUs) not expected by User Equipments (UEs) during regular operation, either because it is explicitly forbidden in the specification or implicitly ignored, i. e., the specification does not state what should

[*]This work was completed while Mikhail Bogodukhov was at Ruhr University Bochum, Germany.

or should not happen. Despite their approach not being able to handle message sequences, fragmentation, or timers, they successfully find numerous bugs in implementations, many of them caused by a lack of specified behaviour. This further motivates the need for finding undefined behaviour, not only to test phone modems but also to fix the root cause in the specification.

In this paper, we present the first approach towards systematically identifying undefined behaviour in cellular network protocol specifications. Our method allows specification organisations to discover and eliminate undefined behaviour in the specifications during their creation or modification, avoiding any room for misinterpretation on the developer side. To do so, we require a human expert to solely model the *defined* behaviour of the specification in Temporal Logic of Actions (TLA$^+$) [29]. Our approach then extends this model via simple syntactic rules to detect undefined behaviour, making it easy to use and avoiding the requirement of an intricate understanding of undefined behaviour in the specification. At the same time, our approach can synthesize concrete PDU sequences that trigger undefined behaviour. This way, we can test how phone modems implementing the specification react to such sequences. We evaluate our approach by applying it to the Public Warning System (PWS), Short Message Service (SMS), and Radio Resource Control (RRC) functionalities in the LTE specification. The broad range of features and complexity covered by them showcases that our approach is applicable in practice and can model challenging parts, such as timers or message sequences leading to undefined behaviour. Overall, we identify 58 instances of undefined behaviour in the modelled parts of the specification. When testing five commercial smartphones from vendors Samsung, Oppo, Huawei, and OnePlus, we identified four representative message sequences that lead to modem crashes and Denial of Service (DoS) attacks and one sequence leading to an out-of-bounds memory read. After responsibly disclosing all vulnerabilities, three CVEs have been assigned.

In summary, our contributions are as follows:

- We present the first generic approach to systematically identify undefined behaviour in cellular network specifications.
- Our technique infers the undefined behaviour from a model of defined behaviour, reducing the burden for a human expert and enabling easy integration into the specification process.
- We automatically synthesize a representative "counter example" that proves the existence of undefined behaviour and allows us to test phone modems, leading to five vulnerabilities and three CVEs.

To foster future research, we publish our models and srsRAN forks at https://zenodo.org/record/8013704.

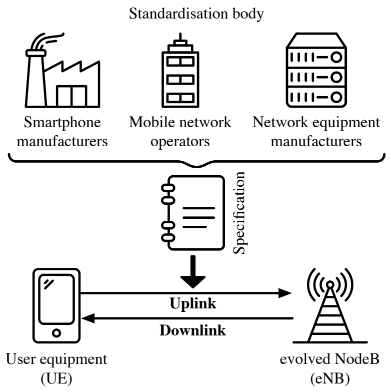

Figure 1: Overview of the cellular network ecosystem.

2 Background

A cellular network operated by a Mobile Network Operator (MNO) consists of two primary components: the base station, also called eNodeB (eNB), and the underlying backbone network, referred to as the Evolved Packet Core (EPC). The customers of the MNO then use UEs, devices containing a cellular modem such as a phone, to wirelessly communicate with the individual eNBs. This connection enables a UE to call or send messages to other UEs, but also allows the MNO to authenticate subscribers, enforce data transfer quotas, or broadcast emergency messages.

To implement this heterogeneous set of functionalities, UEs and eNBs share a common syntax and semantics in their communication. As illustrated in Figure 1, this common understanding is codified in the cellular specification. It is designed and written by the 3GPP, an umbrella organization that consists of regulatory institutions, MNOs, UE and Network Equipment manufacturers as well as multiple non-profit interest groups. They express the semantics of the cellular specifications in natural language. For newer generations (see below), the syntax of individual PDUs is described using Abstract Syntax Notation One (ASN.1), a formal language for data type declarations. Older generations usually use an encoding defined in a custom, per-feature manner.

2.1 Standardisation Process

The standardisation process of a specification is divided in several ways. First, there are multiple protocol generations, such as 3G, LTE, and 5G. Within each generation, there are multiple backwards-compatible releases. The purpose of a new protocol generation is then to introduce improvements to performance, scalability, or functionality aspects that are not backwards compatible. For each protocol generation there exist many *technical specifications (TS)* documents, each specifying a concrete part of the protocol.

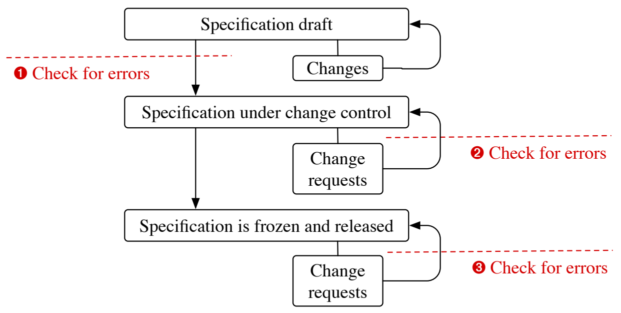

Figure 2: 3GPP procedure for developing new technical specifications [8]. Possible points in time where to check for errors and undefined behaviour are numbered and highlighted in red.

The creation of a new TS is a three-phased process, as illustrated in Figure 2 and specified in [8]. This process lasts for months or even years, involves all 3GPP members, and, interestingly, follows no unified way of describing the contents, adding to the specifications' complexity. As an example, some documents are written from the perspective of the eNB implementation [6, Rel. 15.10, pp. 918], while others are written from the perspective of the UE implementation. First, an initial version of the specification is drafted and frequently updated. Secondly, once the TS draft is mature, the specification enters change control and refinement changes are less frequent. Lastly, the specification is frozen and released. Only occasional changes are made in the form of new releases, and device vendors create implementations according to these released specifications.

There are critical points in time during this process where the specification should be checked for errors, as depicted in Figure 2. These concern ❶ the transition from a draft into change control, as correcting errors under change control is a significant additional effort and a specification under change control is assumed to be dependable, and ❷ when implementing a change request under change control or to the released specification. Lastly, ❸, any change request for a *release* should be checked as well. Currently, error checking is mostly manual work that focuses on the correct implementation of security algorithms and internal coherence [5]. However, the presence of *undefined behaviour* is not thoroughly investigated.

2.2 Undefined Behaviour

Intuitively, *undefined behaviour* can be regarded as the absence of a specification or definition on how a component is expected to handle the reception of a certain PDU in a specific state. The specification contains the syntax that is used for the communication between UE and eNB, and the defined behaviour used to decode PDUs based on this syntax is rather simple: If a message adheres to the syntax, then it is to be processed. Otherwise, it is ignored. The behaviour w. r. t. the syntax is therefore always defined.

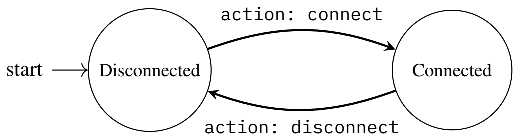

Figure 3: Simple automaton defining an artificial UE connection procedure.

The more complex process is determining the (un-)defined behaviour of the *semantic* part of the specification. This requires taking into account two factors: the current state of the phone (UE) and all PDUs that adhere to the defined syntax. The specification defines a behaviour that a UE should show when it receives a certain PDU in a certain state. Undefined behaviour is caused by the remaining pairs of states and PDUs for which no expected behaviour is given in the specification.

Definition 1: Undefined behaviour

An undefined behaviour is a combination of ***UE state*** *and* ***a set of syntactically valid downlink PDUs****, for which the cellular network specification does not define an expected reaction by the UE.*

To illustrate this, consider the simple automaton in Figure 3, which will serve as our running example. Assume that the syntax allows one PDU field called `action` that can contain either `connect` or `disconnect`. Further, assume that sending the respective message allows moving between the *Disconnected* and *Connected* state as shown. This simple example contains undefined behaviour, as it is not clear what should happen if the device receives `action: connect` while connected or `action: disconnect` if it is not connected. While one might intuitively suggest ignoring such messages, this already requires awareness of the undefined behaviour and a decision on how to proceed, e. g., forcing a reconnect of the device. Even if the UE ignores such messages, the eNB may make different assumptions, leading to desynchronization.

Without a defined behaviour, implementations may assume these scenarios never to occur or react unpredictably, for example, causing the device to crash or leak data. This is because developers have to speculate on how to correctly implement the intent of the specification, possibly resulting in wrong assumptions or mismatching expectations between different implementations. Our hypothesis is that at least some occurrences of undefined behaviour result in security vulnerabilities, making it worthwhile to prevent any misunderstanding in the first place by ensuring that every behaviour is well-defined within the specification.

3 Method

Our main goal is to find undefined behaviour in cellular network specifications, which could potentially introduce security-critical bugs into UE implementations. To this end, we propose a method comprising of multiple steps. First, a human analyst must model the *defined* behaviour of the specification, using Temporal Logic of Actions (TLA$^+$) by Lamport [29]. This task is significantly more intuitive than to consider all the potentially missing behaviours. Our method then derives all instances of undefined behaviour from this model through the extension of the model with an explicit *undefined* state. By systematically identifying all transitions leading to undefined behaviour, we can create a model that is amenable to model checking. Using a model checker, we can then synthesize counter examples, each of which proves the existence of undefined behaviour. As their number can be high, we further propose a method to find a small number of representative counter examples. The complete process is visualised in Figure 4. While these examples can help improve the specification, we also show how *counter examples* can be used to test existing UEs for security vulnerabilities.

3.1 Modelling Challenges

So far, we have illustrated undefined behaviour using a simplified example. When analysing the cellular specification issued by 3GPP, it becomes clear that identifying undefined behaviour is a complex task. In particular, we identify three challenges that we need to take into account when modelling cellular network specifications.

C1: Relations of PDU and state fields. PDUs contain a multitude of fields, and these fields are commonly related to each other or the state of the modelled system, i. e., the UE. For example, one field might define the length of another field. To represent defined behaviour, our approach must account for such relations.

C2: PDU Sequences. Communication between the UE and eNB typically involves multiple PDUs sent in a sequence that all relate to each other. For instance, during the connection establishment procedure, multiple handshakes and information exchanges happen. Another example is reassembly routines. A popular case is SMS, which must be fragmented by the sender and reassembled by the receiver if the SMS' content exceeds 160 characters. Therefore, any generalisable approach to model defined and detect undefined behaviour must be capable of dealing with PDU sequences.

C3: Timers. Many parts of the specification contain timers. They are commonly used to implement timeouts for certain procedures, where their expiry triggers a state change, for instance to reverse any changes made by the procedure that ran into a timeout. Accounting for such timeouts is required to preserve the correctness of a model formalising the specification.

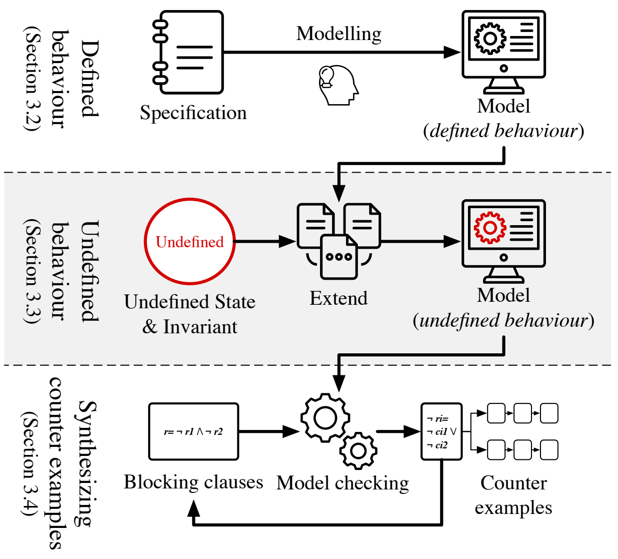
Defined behaviour (Section 3.2)
Specification
Modelling
Model (*defined behaviour*)
Undefined behaviour (Section 3.3)
Undefined
Undefined State & Invariant
Extend
Model (*undefined behaviour*)
Synthesizing counter examples (Section 3.4)
r= ¬ r1 ∧ ¬ r2
Blocking clauses
Model checking
¬ ri= ¬ ci1 ∨ ¬ ci2
Counter examples

Figure 4: Our proposed process for modelling and synthesis of counter examples leading to undefined behaviour.

3.2 Modelling Specifications

With these modelling challenges in mind, we discuss how a human expert can model the 3GPP technical specifications. In particular, we require that the UE must be capable of storing an internal state and that this state must be modifiable (C1) when receiving (multiple) messages (C2) or as time passes (C3). The state space of our model depends on the modelled functionality and thus on the set of technical specifications covered by the model. In the following, we outline how to model the 3GPP technical specifications based on our simple running example from Figure 3.

Initial model. The state space of our running example contains a single variable which we call $s \in \{\text{Disconnected}, \text{Connected}\}$. To define our system's initial state, we introduce an initial condition:

$$\text{Init} \triangleq s = \text{“Disconnected”}$$

Then, we need to define a next state relation, such that we can modify the internal state, for instance upon receiving a message. We formalise the syntax of messages $m \in M$ as follows. Here, $[key : vals]$ denotes a dictionary-like structure with a key containing a value arbitrarily chosen from $vals$. For our running example:

$$M = [action : \{\text{“connect”}, \text{“disconnect”}\}]$$

We can now model the preconditions for the two possible state changes shown in Figure 3 as follows:

$$\text{CanConn}(m) \triangleq m.\text{action} = \text{“connect”} \land s = \text{“Disconnected”}$$
$$\text{CanDiscon}(m) \triangleq m.\text{action} = \text{“disconnect”} \land s = \text{“Connected”}$$

This effectively solves challenge **C1**, as we can relate arbitrary PDU fields and state variables using any operation that is available in first-order logic. We are not bound to simple comparisons, we can use predicates for counting list lengths, and we can chain conditions in a functionally complete way.

Updating the state. We now require a way to express an update to our state s to implement a state change. Importantly, s is part of the internal state of the model and not an externally supplied value like the incoming messages m. We denote such updates using an apostrophe, e. g., $s' = \text{foo}$ assumes that in the next temporal iteration the state s equals foo. To handle such a temporal property, we follow TLA$^+$ and use the $\Box[\text{Next}]_s$ construct, which states that each step into the future, satisfies the relation *Next*. Intuitively, s' is renamed to be the new s with this step. Consequently, our system can be formalised as

$$\begin{aligned}
\text{DoConn}(m) &\triangleq s' = \text{“Connected”} \\
\text{DoDiscon}(m) &\triangleq s' = \text{“Disconnected”} \\
\text{Next} &\triangleq \exists m \in M : \\
&\qquad (\text{CanConn}(m) \land \text{DoConn}(m)) \\
&\qquad \lor (\text{CanDiscon}(m) \land \text{DoDiscon}(m)) \\
\text{Spec} &\triangleq \mathit{Init} \land \Box[\text{Next}]_s
\end{aligned}$$

In particular, this formalisation allows us to handle sequences of arbitrary length. Effectively, the $\Box$ operator will cause the $\exists$ quantifier to repeatedly draw a new PDU m and then update the internal state according to the rules set within the quantifier. This solves challenge **C2**, and *Spec* now effectively represents the model introduced in Figure 3.

Modeling timers. To address **C3** and introduce support for timers, we draw from another approach presented by Lamport [30]. They model time to be an element in $\mathbb{R}$, which approximates a continuous dynamical system and allows to model arbitrary time distances. However, it comes at the price of computability, as $\mathbb{R}$ is non-denumerable and the model could not be exhaustively computed, even if the time frame in which the model operates has an upper bound. We observe that every timer defined in a 3GPP technical specification is defined in milliseconds or a unit that is a multiple of a millisecond. Thus, we can discretize our time system into milliseconds and use integers as our basic time unit. We couple this with an upper bound for system time, such that the amount of different states is limited and computability is preserved.

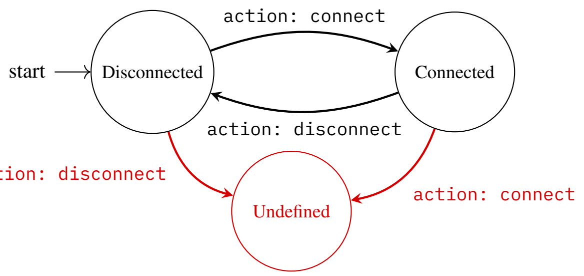

Figure 5: Extension of our automaton with an explicit Undefined state capturing undefined state transitions (in red).

Formally, we add a new state variable $n \in \mathbb{Z}_0$ that represents the current system time in milliseconds. We also add a new “Pseudo”-PDU that advances the time, i. e., $T = [\text{advance} : \mathbb{Z}]$, and replace the *Next* state relation as follows:

$$\begin{aligned}
\text{Next} \triangleq &\exists m \in M \cup T : \\
&\quad (\text{CanConn}(m) \land \text{DoConn}(m)) \\
&\quad \lor (\text{CanDiscon}(m) \land \text{DoDiscon}(m)) \\
&\quad \lor (\text{“advance”} \in m \land n + m.\text{advance} < \mathit{LIMIT} \\
&\qquad \Rightarrow n' = n + m.\text{advance})
\end{aligned}$$

This introduces the notion of time into our system. Any timer is now another state variable t_i that is set to the system time at which the timer expires. Actions taken upon expiry are then added to the $(\text{“advance”} \in m \land n' = n + m.\text{advance})$ fragment. If TiExpiredAction is the relation to be fulfilled once a timer i expires at time t_i, this can be formalised as:

$$\begin{aligned}
&\text{“advance”} \in m \land n + m.\text{advance} < \mathit{LIMIT} \\
&\Rightarrow (n' = n + m.\text{advance} \land n + m.\text{advance} > t_i \Rightarrow \text{TiExpiredAction})
\end{aligned}$$

This strategy allows us to model behaviour defined in the technical specifications, including the relationship between PDU and state fields (**C1**), message sequences (**C2**), and timers (**C3**).

3.3 Extracting undefined behaviour

Given the capability to model the technical specification, we can now focus on how to extend this model to identify *undefined* behaviour. Our goal is that we can evaluate our model for unspecified behaviour using a model checker such as `tlc` [42]. Thus, we need to turn the problem of finding state sequences leading to undefined behaviour into a model checking problem. One classic example of such a model checking problem is to verify if there exists a path into a dangerous state, such as a crash state. Therefore, we introduce a new state into the model, which we call the undefined state, and add those transitions that correspond to PDUs without a defined behaviour. Figure 5 shows how the additional state and transitions (in dark red) extend our simple example.

To avoid further manual efforts, we need to automatically deduce all *undefined* transitions that lead to this undefined state. Our insight is that PDUs which do not satisfy any precondition lead to undefined behaviour, allowing us to exhaustively identify all transitions to the undefined state. Translating this into TLA$^+$ corresponds to adding the conjunction of all negated preconditions as a precondition of the newly added undefined state. By construction, the precondition of the undefined state thus has the form

$$\neg \underbrace{r_1}_{CanConn(m)} \wedge \neg \underbrace{r_2}_{CanDiscon(m)} \wedge \neg \underbrace{r_3}_{\text{“advance”} \in m} \ldots \wedge \neg r_n .$$

Notably, this operation is purely a syntactical one for the person writing the model. It does not require that preconditions are manually interpreted. The presence of the new *Undefined* state then allows us to add an invariant, with which we can then perform classical model checking. Formally, this corresponds to the following and final modification of our next relation for our simple example in Figure 5.

$$\begin{aligned}
\text{Next} \triangleq \exists m \in M \cup T :& \\
&(\text{CanConn}(m) \wedge \text{DoConn}(m)) \\
&\vee(\text{CanDiscon}(m) \wedge \text{DoDiscon}(m)) \\
&\vee(\text{“advance”} \in m \wedge n + m.\text{advance} < LIMIT \\
&\quad \Rightarrow n' = n + m.\text{advance}) \\
&\vee(\neg\text{CanConn}(m) \wedge \neg\text{CanDiscon}(m) \\
&\quad \wedge \text{“advance”} \notin m \wedge s' = \text{“Undefined”}) \\
\text{Invariant} \triangleq s' \neq \text{“Undefined”}&
\end{aligned}$$

Finally, we leverage the temporal $\Box$(Invariant) operator to express that we want to ensure that the *Invariant* is never violated and arrive at the theorem Spec $\Rightarrow \Box$(Invariant) with Spec $\triangleq Init \wedge \Box[\text{Next}]_s$, which we can check using `tlc`. More information on the semantics of the $\Box$ operator can be found in the TLA$^+$ book by Lamport [31, pp. 89].

3.4 Synthesizing Counter Examples

When verifying if the previous theorem holds, the model checker can either find no violation, indicating the absence of any undefined behaviour, or find an instance of undefined behaviour. In this case, it will by design synthesize a *counter example* in form of a concrete assignment, i. e., determine a series of state transitions with corresponding messages m.

Definition 2: Counter example

A counter example is an automatically synthesized invariant violation of the model. It consists of a sequence of PDUs that transition the UE from the idle into the undefined state. It is therefore a representative and concrete, testable sample of an undefined behaviour.

Without further consideration, this will potentially lead to the generation of a large number of counter examples. However, in many of these scenarios, the counter examples will have a common root cause for undefined behaviour. For example, consider the precondition $r_i \triangleq s = \text{foo} \wedge m.x < 2$ for PDUs $m \in [x : 1..200]$. This precondition is false and, thus, leads to undefined behaviour from state “foo” for any $m.x \geq 2$. The model checker would potentially generate 199 counter examples for the same root cause of undefined behaviour, leaving it to a human to realise that they share one root cause. Knowledge about this root cause is, however, required to amend the specification and specify a behaviour for the state “foo” and the $m.x \geq 2$ case. To assist in this process and avoid the generation of unnecessary counter examples, we propose the following technique to add clauses to the *Next* state relation, which block PDUs causing already discovered instances of undefined behaviour.

We find the smallest part of a precondition (r_i) that we have to enforce on generated PDUs, such that the current counter example PDU sequence is no longer generated, and that does not cause PDUs that currently lead to defined behaviour to be no longer generated.

$\neg r_i$ will then be part of the precondition of the undefined state, and will have the following structure:

$$\neg \underbrace{r_i}_{s=\text{foo} \wedge m.x<2} = \neg \underbrace{c_{i_1}}_{s=\text{foo}} \vee \neg \underbrace{c_{i_2}}_{m.x<2}$$

We then check each sub-clause $\neg c_{i_j}$ that is satisfied by the undefined behaviour-generating PDU for whether it can be split up further. This is not the case for our example. If not, then we check if discarding any PDU that does not satisfy c_{i_j} would lead to any valid PDU being discarded. We cannot enforce $c_{i_1} = \text{foo}$, as that would mean that the model cannot ever leave the “foo” state. If so, we move on to the next sub-clause c_{i_2}. We can also not enforce $m.x < 2$ as that would block the generation of messages with $m.x \geq 2$ for all states, including states that are not “foo” and might have a defined behaviour for such cases. If we can enforce the truthiness of a sub-clause, it is the defining one for this group of undefined behaviour, and it will block the generation of PDUs causing the same undefined behaviour when added as a disjunction into the $\exists$ quantifier. If no clause is found, then we back-track on our downwards recursion and reassemble clauses again, i. e., instead of checking the individual c_{i_j}, we would check the r_i by applying the same criteria. For our example, that means that $\neg(s = \text{foo} \wedge m.x < 2)$ characterises one group of invalid input by blocking the generation of any further samples.

This approach follows a similar idea as blocking clauses used to solve the All-SAT problem [36]. However, in the generalised All-SAT case, blocking clauses have no relation to the underlying semantics. As our goal is to group semantically similar cases of undefined behaviour, we operate on formula fragments and take into account the order in which the individual functionality parts were modelled. Not normalising the formula ensures that sub-clauses which belong to the same semantic concepts stay in close proximity to each other within the formula, as they were likely introduced at the same time.

Inferring semantics via proximity—a property caused by the manual modelling—thus distinguishes our approach from the generic All-SAT approach.

The resulting grouping is based on the specification, not the implementation. Therefore, messages which exhibit the same undefined behaviour w. r. t. the specification end up in the same group.

3.5 Testing Implementations

To test UEs implementing the specification, we need to convert the counter example synthesized for each group into one or more messages. Due to the faithful modelling of PDU fields, variables in our model can be mapped to concrete fields. A counter example generated by the model checker always assigns concrete values to these variables, allowing us to simply use these assignments for their respective fields and to generate concrete messages we can test UEs. This way, we can not only improve the specification but test existing implementations for security vulnerabilities caused by undefined behaviour.

We depict the steps to evaluate UE implementations in Figure 6. It consists of two parts: First, concrete counter examples are replayed. To do so, the UE under test is connected to an existing LTE eNB implementation using a Software Defined Radio (SDR). We observe the upper layers of the eNB to determine when the UE awaits our input before starting to replay the counter example. As the layered architecture of eNBs ends with an encoding step, we can inject each PDU in between upper layers and the encoding step.

The second part of this procedure is then to observe and evaluate UE behaviour. While the modem firmware itself operates as a black box, we utilize two of its external interfaces to obtain information: The interface between modem and Android using log monitoring and the radio frequency (RF) communication by inspecting up- and downlink traffic captures obtained at the eNB. While monitoring of UE crashes using these channels is fully automated, detecting more complex vulnerabilities, like information leaks or dangerous state transitions, requires a human in the loop. Fully automating this is not possible in general, since – due to the nature of undefined behaviour – there is no defined, benign behaviour that we could verify against. Instead, human judgement is required to identify vulnerabilities.

4 Case Studies

With our approach, we implement several 3GPP technical specifications in TLA$^+$. This demonstrates that technical specifications can be formally modelled in isolation, such that undefined behaviour can be discovered without modelling the entire specification of a protocol generation.

The criteria for the selection of the modelled specifications are twofold: First, the technical specifications should relate to

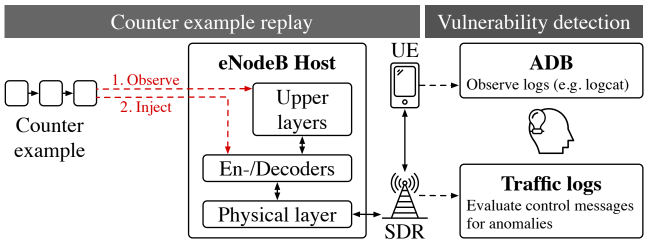

Figure 6: Procedure to determine if a UE firmware implementation is vulnerable to an undefined behaviour. Counter examples are generated according to Figure 4.

Table 1: Summary of undefined behaviours found in PWS, SMS, and RRC. We report model size, states discovered by `tlc`, time in CPU core hours required to compute the state space, number of undefined behaviours (UBs) found, and the average number of PDUs needed to reach the undefined state.

Model	Lines	States	CPU hrs.	#UBs	Avg. #PDUs
PWS	283	13,959,856	85.6	8	1.6
SMS	415	8,676,939	180,000	22	1.3
RRC	731	955	3.3	28	2.7

different functionalities that result in different requirements w. r. t. the modelling method. Secondly, the modelled functionalities should exhibit an attack surface via message injection that does not require that the victim UE is attached to the network of a compromised MNO, demonstrating the security impact of the exploitation of undefined behaviour on UEs.

We select three different functionalities, which are detailed below. Our resulting models are summarised in Table 1.

4.1 Public Warning System (PWS)

Cellular networks support the distribution of emergency messages, such as warnings of extreme weather conditions. For scalability reasons, cell broadcasting is implemented by multicasting (one-to-many) instead of sending messages individually to each subscriber. In LTE, cell broadcasting is implemented via a specific System Information Block (SIB). The base station sends various kinds of SIB types to broadcast meta information such as cell identity and scheduling information. In an emergency, the base station uses SIB type 12 to deliver the warning message. The message itself, which is shown on the UEs display after reception, is transmitted in a GSM-era encoding for legacy reasons. Each SIB12 also contains a message identifier that is used for deduplication and ensures that the same message is not displayed twice.

PWS supports message fragmentation on two layers: SIB12s can be fragmented on the LTE layer, and the text can be further fragmented into pages on the "inner" GSM layer, again for legacy reasons. The reception procedure requires that the individual segments are reassembled on the LTE level before being forwarded to the GSM message handling procedure [6, pp. 66].

A PWS message is considered to be reassembled if all segments from segment number 0 to the segment where type is set to `lastSegment` have been received. Any LTE message may contain one or multiple pages, and page reassembly happens after LTE segment reassembly, such that the GSM pages are ordered by the segment number of the LTE segment they were transmitted in.

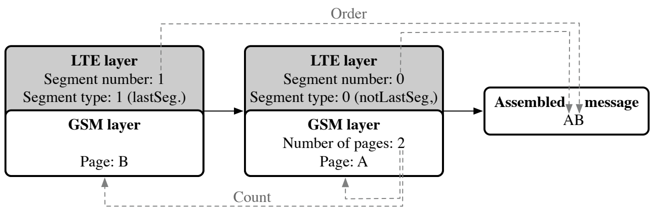

Figure 7: Message reassembly of a valid PWS message sequence. The segment number determines the order of the message parts A and B.

Once the message content is passed to the GSM layer, the individual pages of the PDUs are reassembled. The reassembly process is illustrated in Figure 7, depicting a correct and valid reassembly sequence.

This use case illustrates two of the previously introduced challenges. There is an interdependence between the fields of a PDU, even across layers, for the segment number, type, the number of pages, and the amount of actually transmitted pages. In parallel, the model must also account for the internal state (previously received PDUs) to determine if the newly received PDU completes a fragmented PWS message or must be buffered and thus added to the state (**C1**). In addition, to even be able to handle reassembly scenarios of multiple PDUs, as well as to model the deduplication, the model must be capable of handling sequences of separate PDUs containing PWS fragments (**C2**).

Attack surface. As has been shown before [11, 32], SIBs are not integrity protected against injection into an Over-the-Air (OTA) data stream. An attacker thus only requires a SDR in RF range to the victim to send a malicious PWS message to the UE, as described by Lee et al. [32].

TLA$^+$ Model. Our model of the PWS is based on TS 36.331 [6], release 14. In particular, we implement the procedure described in section 5.2.2.19, along with the reassembly procedure described in section 9.4.3.2.4 of TS 23.041 [3]. As the specification does not make assumptions about the maximum size of an SIB, we arbitrarily allow two `warningMesageSegments` per PDU. While our model supports all fields introduced in the specification, we decrease the value range for field contents to boundary values or otherwise interesting cases, as is common practise [37]. We discuss resulting potential limitations in Section 6.

The state of the model is the `buffer`, which is a list of previously received PDUs awaiting reassembly, a list of message identifiers to model the deduplication, along with some helper variables that store the current sequence and previous sequences of PDUs, that we use to extract the test case during counter-example generation.

Results. Model checking the PWS model using `tlc` yielded eight different undefined behaviours. To find these, the model checker explored 13,959,856 distinct states in 5:21 hours on 16 logical CPU cores. Four of the eight undefined behaviours require more than one PDU, with an average length of 1.6 PDUs per behaviour and a maximum length of 3 PDUs for the longest sequence required to reach undefined behaviour. From the eight different undefined behaviours, four lead to a security vulnerability of high or critical severity on at least one UE, which we describe in detail in Section 5.3, two were caused by behaviours requiring a PDU sequence.

Human effort. Modelling the PWS functionality took one person, starting without prior knowledge about the PWS specification, two weeks to understand the specification and model the behaviour. Extracting test cases and adjusting the blocking clause based on the automatically generated model checker output takes 2-3 minutes per behaviour, such that eight behaviours are covered by around 20 minutes of additional manual work after modelling.

4.2 SMS in LTE

SMS over SGs (SG-SMS) reuses GSM PDUs and packs them into LTE network packets. This allows modem firmware developers to derive the implementation of SG-SMS from existing GSM SMS implementation. Contrary to PWS PDUs, the GSM SMS PDUs are not described in ASN.1, highlighting that our approach is also capable of modelling such non-standardised specification techniques. The GSM PDU has numerous fields, of which the most important are the sender's and receiver's phone numbers, the sending timestamp, an optional user data header (used to reassemble SMS that have been cut off at the 160-character limit), and the SMS text itself, the encoding of which is also included in the PDU. SMS, therefore, is not only an important communication medium but also has many interdependent fields (**C1**) and requires sequences to model message reassembly (**C2**).

Attack surface. One way to inject malicious SMS PDUs is to use a modified UE to send the modified PDU to a cellular network, which then forwards it to the victim. This requires that the network does not validate the PDU, which is not an uncommon configuration [34]. Alternatively, an attacker could also try to inject the data stream into the connection between the eNB and the UE. That requires that the commercial network is misconfigured to allow disabling or downgrading the encryption and integrity protection mechanisms,

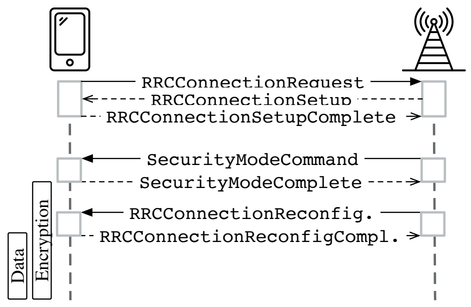

Figure 8: Typical RRC connection procedure. "Encryption" and "Data" denote the points in time when encryption and IP connectivity are (optionally) available.

which is the case for some networks, as has been shown by Chlosta et al. [15].

TLA⁺ model. is focused on SMS-DELIVER PDUs. Therefore, we implement the SMS-DELIVER procedure described in TS 24.301 sections 5.6.3.3 and 9.9.3.22 [7] for protocol identifiers equalling zero, carrying the CP-/RP-Data PDUs specified in TS 24.011, section 8.2 [2]. We also adhere to TS 23.040, sections 3.2.1-3.2.5 and 3.2.10 [4]. We have implemented support for the PDU fields described in 9.2.2.1 and 9.2.3.10 of the same TS. In addition, we support the user data headers described in 9.2.3.23-24, as they are required for the reassembly of segmented SMS. The state of the model is the buffer of unassembled messages, along with the helper variables already introduced in the PWS model.

Results. Model checking the SMS model using `tlc` yielded 22 different undefined behaviours. The model checker explored 8,676,939 distinct states in 75 days on 100 logical CPU cores to find these. The time difference to the PWS model is caused by the larger number of fields in SMS PDUs, which results in a larger number of different PDUs that need to be checked per state. Six of the 22 undefined behaviours require two instead of one PDU to be reached.

Human effort. Similarly to the PWS case, modelling the SMS functionality took one person, starting without prior knowledge about the SMS specification, 2 weeks to understand the specification and model the behaviour. Extracting test cases and adjusting the blocking clause based on the automatically generated model checker output takes 2-3 minutes per behaviour, such that 22 behaviours are covered by around an hour of additional manual work after modelling.

4.3 Radio Resource Control (RRC)

RRC is a network layer used for communication between the UE and eNB and is responsible for the configuration of encryption and integrity protection, as well as mobility management. A typical handshake between a UE and eNB is illustrated in Figure 8. It is particularly important to notice that encryption and data transfer capabilities are not available from the beginning, and the use of encryption is not mandatory once it is available. RRC can also be used by the eNB to request a UE to connect to another eNB, for instance because that eNB is in closer proximity to the UE. This handover uses an internal timer, such that if the handover is not successful within an amount of time defined within the reconfiguration request, the UE reconnects to the previous eNB.

Some RRC PDUs are only specified to be sent encrypted or integrity-protected by the eNB, while the specification does not define the expected behaviour of a UE receiving an unencrypted PDU. Furthermore, there exist PDUs such as `RRCReconfigurationRequest` that require encryption or integrity protection depending on the presence of some optional fields within the PDU, such that the UE must first evaluate the PDU before deciding if it must be rejected due to missing integrity protection. This highlights the interdependence of fields within this functionality (**C1**). In addition, the handshake mechanism shown in Figure 8 requires a message sequence to complete and also enables many potentially undefined sequences, such as a connection setup PDU after the connection is already established (**C2**), while also demonstrating that our approach is capable of modelling the diverse set of used PDU types. Lastly, the handover timer illustrates the importance of being able to model time (**C3**).

Attack surface. Since the RRC layer is responsible for enabling encryption and integrity protection, it cannot rely on lower layers for these tasks. Injection of RRC messages via an SDR is thus easily achievable, and the only safeguard is a careful specification of the protocol itself. Therefore, it is an integral building block of the security architecture of LTE and is interesting to any attacker.

TLA⁺ model. Our TLA⁺ model is based on TS 36.331 [6], Chapters 5.3 and 5.5, as well as Appendix A6. We provide a detailed list in Appendix C. In addition to the actual PDU, we extend the input space of our model to contain a metadata structure that specifies the integrity and encryption algorithm that is used to transmit this message. This gives the model checking algorithm the opportunity to test if undefined behaviour arises when transmitting a message using a mode of encryption or integrity protection that is unexpected. The state of our model contains the configured signal and data channels, the last configured encryption and integrity protection algorithms, whether there has been a `SecurityModeCommand`, the RRC state as defined in the specification (either `IDLE` or `CONNECTED`), the model time, and the expiry time of the aforementioned handover timer. Consequently, we also implement a "Pseudo" PDU to advance the model's time state, as described in Section 3.2, as well as an upper limit for the model time.

Results. Using `tlc` to check the RRC model yielded 28 different undefined behaviours. The model checker explored 955 distinct states in 2 minutes on 100 logical CPU cores to find these. The large difference in states and, consequently, CPU time is caused by the very different nature of the RRC layer. PWS and SMS perform message reassembly, and consequently, each combination and order of received, unassembled PDUs is a separate state. In contrast, the number of different bearers, security algorithms and connection state combinations that lead to the overall RRC state is very small and dominated by the time variable. If we exclude the model time from the state, the number of different states decreases to 117.

In addition to the 28 undefined behaviours, the model checker also found two purely theoretical undefined behaviours. In these cases, either RRCConnectionRelease or SecurityModeCommand were to be sent while the UE was in an IDLE state. However, this is impossible to do in practice, because at that point the dedicated channel used to transmit these messages is not yet established. The generation of these cases is a result of our RRC model being viewed in isolation, i. e., without modelling the lower layers providing the different channels used to transmit RRC PDUs.

26 of the 28 undefined behaviours require more than one PDU to be reached, with an average of 2.71 PDUs per behaviour and a maximum of 5 PDUs. The greater sequence length compared to the PWS and SMS models is caused by most undefined behaviours resulting from a conflicting state configuration rather than an internal conflict in a single PDU.

Human effort. Since the RRC layer is more complex than PWS and SMS, understanding its specification and modelling the specification required more time, which is also reflected in the larger number of lines of this model. In total, it took a single person without prior in-depth knowledge of RRC roughly 3.5 weeks to understand and model this layer. Analogously to the previous cases, modifying the blocking clauses and extracting the undefined behaviour from `tlc` output took required another 1.5 hours in total.

5 Evaluation

We first compare our approach against DoLTEst before analysing the root causes of undefined behaviour in the specification. Finally, we investigate how our synthesized messages can be used to test five commercial phone modems from popular smartphone vendors.

5.1 Comparison to DoLTEst

Despite DoLTEst's [37] different goal of testing phone modems instead of improving the specification, their use of *negative testing* draws both from defined but *prohibited* behaviour as well as undefined behaviour to create test cases. Thus, we can compare against the part of their evaluation investigating the RRC layer in LTE, which relies on undefined behaviour. To do so we match the syntactic "Guidelines" used by DoLTEst to our counter examples and regard cases where a syntactic guideline matched a PDU in one of our counter example sequences as found by both approaches.

Table 2: Comparison of our approach to DoLTEst. "Guideline" refers to Table 5 in [37]. If DoLTEst did not find the behaviour, the reason is given instead. UNM = Unmodelled by DoLTEst, INI = Incorrect inital state, SEQ = Requires a sequence, TIM = Requires Timer.

Message Type	**Undefined behaviour**	**Our approach**	**DoLTEst**	**Guideline**
RRCConnectionSetup	`AFTER_SECURITY_ENABLED`	✓	✗	(UNM)
	`DRB_WITHOUT_SECURITY`	✓	✗	(INI)
	`SRB2_WITHOUT_SECURITY`	✓	✗	(INI)
RRCConnectionReconfiguration	`INVALID_DRB_CONFIG`	✓	✓	1
	`INVALID_SRB_CONFIG`	✓	✓	2
	`UNPROTECTED_MEAS_OBJ_ADD`	✓	✓	3
	`UNPROTECTED_REPORT_ADD`	✓	✓	3
	`UNPROTECTED_QUANT_CONF`	✓	✓	3
	`UNPROTECTED_SPEED_STATE_PARS`	✓	✓	3
	`UNPROTECTED_SECURITY_CONFIG`	✓	✓	4
	`MOBILITY_CONTROL_NO_DRB`	✓	✗	(SEQ)
	`MOBILITY_CONTROL_NO_SRB2`	✓	✗	(SEQ)
	`NO_CIPHERING_DESPITE_SECURE`	✓	✗	(INI)
	`NO_INTEGRITY_DESPITE_SECURE`	✓	✗	(INI)
	`WHILE_T304_RUNNING`	✓	✗	(TIM)
RRCConnectionReestablishment	`BEFORE_SECURITY`	✓	✗	(INI)
RRCConnectionReject	`AFTER_SECURITY`	✓	✗	(UNM)
RRCConnectionRelease	`NO_CIPHERING_DESPITE_SECURE`	✓	✗	(UNM)
	`NO_INTEGRITY_DESPITE_SECURE`	✓	✓	5
SecurityModeCommand	`CIPHER_WITHOUT_INTEGRITY_CONF`	✓	✓	6
	`AFTER_SECURITY_ENABLED`	✓	✗	(SEQ)
UECapabilityEnquiry	`NO_CIPHERING_DESPITE_SECURE`	✓	✗	(UNM)
	`NO_INTEGRITY_DESPITE_SECURE`	✓	✓	7
CounterCheckMessage	`NOT_CIPHERED`	✓	✗	(UNM)
	`NOT_INTEGRITY_PROTECTED`	✓	✓	8
UEInformationRequestMessage	`BEFORE_SECURITY`	✓	✗	(INI)
	`NO_CIPHERING_DESPITE_SECURE`	✓	✗	(UNM)
	`NO_INTEGRITY_DESPITE_SECURE`	✓	✓	9
DLInformationTransfer	Not undefined behaviour	✗	✓	10

The result of this evaluation is shown in Table 2. Overall, our approach finds 28 undefined behaviours, while DoLTEst reports 13. For the `DLInformationTransfer` PDU, our approach does not report an undefined behaviour, while DoLTEst does. Manually investigating the specification (Page 918 of [6] Release 15.10), we find it does not mention that integrity protection is mandatory, thereby contradicting the rule set of DoLTEst. Consequently, we consider this a false positive by DoLTEst.

We now investigate the reasons why our approach found additional undefined behaviour.

Capabilities. First, our approach generates PDU sequences that start from the UE being in an IDLE state and thus sets up the UE state. DoLTEst generates a single message, and thus requires a single UE state from which the undefined behaviour needs to be reachable *(INI)*.

Secondly, the authors of DoLTEst require that a connection to the eNB is established and start their tests from this point. Our approach allows establishing the state in a sequence of PDUs, such that undefined behaviour can be caused by the reception of multiple, dependent PDUs *(SEQ)*.

DoLTEst does not support timers as their state is static and cannot be updated to simulate the passage of time *(TIM)*.

Unexpected Message Types. We also found that two cases, related to RRCConnectionReject and RRCConnectionSetup were not found by DoLTEst because the corresponding Guidelines are missing *(UNM)*. We suspect that this is because these PDUs are not normally used after a phone is connected, as required by the DoLTEst authors, and they thus did not include them by accident. This illustrates that an approach where only *defined* behaviour needs to be manually modelled and *undefined* behaviour is synthesized automatically, is advantageous in finding *undefined* behaviour in unexpected places because there is no need to speculate about which PDUs could cause undefined behaviour while modelling.

In summary, this demonstrates that our approach can not only find undefined behaviour in previously unstudied functionality but is also able to surpass the state of the art on testing cellular network protocols.

5.2 Root causes of undefined behaviour

We analyse the root causes that lead to undefined behaviour in the specification and find three reasons.

Ad-hoc definitions in old specifications. Over time, the specification procedure has improved, and new techniques and best practices have been established that promote secure implementations by relying on standardised and easy-to-implement approaches. However, these improvements are not back-propagated, as existing specifications and implementations cannot be modified in a way that would break the compatibility of existing network equipment. Legacy solutions, such as non-standard protocol parsers thus remain in use, even though specifications of newer generations of cellular networking protocols solely rely on standardised and well-defined ASN.1 parsing procedures. Issues in the GSM specification are related to 5 out of 8 undefined behaviours discovered by the PWS model and are involved in all 22 undefined behaviours in SMS.

Interface between previous and current generation specifications. Our results show that deep integration of different protocol generations and underspecification are prone to causing security vulnerabilities. If information elements in newer generation PDUs need to be taken into account when processing information elements carried by older generation PDUs, this appears to increase the chance of undefined behaviour. In the PWS use case, 3 out of 8 undefined behaviours expose cases where all PDUs are valid if they are considered in isolation on either the LTE or the GSM layer, but the information on both layers, i. e., regarding the completeness of a reassembly procedure, are contradicting.

Specifications written from a single perspective. Finally, undefined behaviour is commonly caused by specifications that are written from the perspective of a single network component. For instance, the RRC specification [6], Release 15.10, on p. 918 describes the security requirements of RRC messages as follows: "The following list provides information which messages can be sent (unprotected) prior to security activation and which messages can be sent unprotected after security activation." However, it does not explain how a violation is handled on the receiver side, which leads to widely different interpretations by developers as shown in Appendix D.

5.3 Security Vulnerabilities

We evaluate our approach using over-the-air testing of five Commercial Off-The-Shelf (COTS) UEs.

Setup. As the global market for cellular modems is split between Qualcomm, Samsung LSI, Mediatek, and HiSilicon [40], we diversify the UEs to test on devices of each of these manufacturers. Additionally, we test two Mediatek modems to see how much the results on two modems manufactured by the same company differ. Notably, both Mediatek modems are recent models, albeit the MT6768 only supports 4G, while the MT6853V supports 5G as well.

We employ the method described in Section 3.5, using `srsRAN` [18], an open-source implementation of the LTE network, which we operate inside a shielding box using an Ettus SDR X300 acting as the eNB. The code that implements the first of the TLA$^+$-generated messages shown in Appendix A is presented in Appendix B. For each test run, we then send the selected message sequence to the UE and monitor the UE via Android Debug Bridge for modem crashes and loss of IP connectivity, as well as received SMS and PWS messages. We also obtain a network log from `srsRAN`. The phone state is reset between test runs by toggling airplane mode (RRC) or rebooting the phone (SMS, PWS) and advancing the system time (PWS) as a measure to counteract deduplication. We automatically obtain the log files mentioned in Section 3.5 and any unusual behaviour, such as dropped IP connections and messages with unexpected contents, is later manually investigated.

Results. We found five undefined behaviours that—following responsible disclosure—led to confirmed security vulnerabilities in COTS UEs with assigned CVEs. Table 3 provides an overview. Beyond these cases, undefined behaviours often lead to different consequences across smartphone modems. A full overview for all undefined behaviours can be found in Appendix D. In the following, we discuss the undefined behaviours leading to vulnerabilities.

5.3.1 PWS: Denial of Service against Mediatek modems (CVE-2022-26446)

We observed three different undefined behaviours leading to crashes of Mediatek modems, which Mediatek summarised under CVE-2022-26446 in response to our report. In cases (1) and (2), the crashes are caused by a sequence of test messages, while (3) is caused by a single message inducing undefined behaviour.

Table 3: Selected test cases leading to security-critical behaviour with a CVE assigned. Cell colours indicate the different behaviours shown by the various phone modems when receiving the message sequence causing the undefined behaviour.

	Undefined behaviour	Samsung A41 Mediatek MT6768	Oppo A73 5G Mediatek MT6853V	Huawei P40 Lite 5G HiSilicon Kirin 820 5G	Samsung S20 5G Samsung Exynos 990	OnePlus 8 Qualcomm SM8250
PWS	`SIB_COMPLETE_BUT_MORE_PAGES`	🟦	🟦	🟦	🟨 CVE-2022-39881	🟪
PWS	`LAST_MESSAGE_BUT_FURTHER_MESSAGES`	🟥 CVE-2022-26446 (1)	🟥 CVE-2022-26446 (1)	🟦	🟦	🟦
PWS	`MESSAGE_BEYOND_LAST_SEGMENT`	🟥 CVE-2022-26446 (2)	🟥 CVE-2022-26446 (2)	🟦	🟦	🟦
PWS	`EMPTY_WARNING_MESSAGE_SEGMENT`	🟥 CVE-2022-26446 (3)	🟥 CVE-2022-26446 (3)	🟦	🟦	🟦
SMS	`GSM7BIT_INCORRECT_USER_DATA_LENGTH`	⬜	🟧 CVE-2022-32591	🟦	▨	🟧

🟥 Modem crash - 🟨 Test message and benign messages shown, indications of overflow - 🟧 Parts of both test messages shown, benign message shown completely
🟦 Only benign messages shown - ▨ Part of one test message shown, benign message shown completely - 🟪 One test message shown, benign message modified
⬜ Only a part of one test message shown

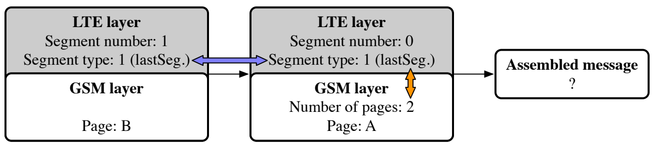

Figure 9: Message reassembly sequence with two undefined behaviours: Multiple segments signal to be the final segment (blue, CVE-2022-26446 (1)) and a segment signals completion on the LTE layer while indicating more pages than transmitted on the GSM layer (orange, CVE-2022-39881).

To verify if all counter examples cause a crash for the same reason, or if these have different root causes, we performed a static analysis of the firmware binary extracted from a Samsung A41. Our analysis thus showed that, while the firmware ultimately crashed because it attempts to read uninitialised array or buffer entries, the root cause are three separate cases of missing input validation: (1) For sequences that lower the segment number by sending multiple segments that have the last segment flag, as depicted in Figure 9, and (2) for sequences which contain segments with higher segment numbers than initially indicated, along with a case (3) of not accounting for unforeseen, empty, input. Our synthesizing procedure has successfully captured this difference by creating separate counter examples for each of these categories, helping us to pinpoint the specification deficiency and assisting the vendor in mitigating these issues.

We also noticed that the same crash occurs in both Mediatek-based phones and developed a tool to parse the log files generated by the modem firmware. This analysis showed that the execution traces inferred from the log files are identical, and we thus assume that both firmware contain the same code segment for PWS handling.

Impact: The CVE has received a CVS Score of 7.5 (high) by NIST. Depending on the debug level configured in the developer menu, the Samsung A41 phone either completely crashes with a kernel panic or loses all cellular connectivity. The Oppo phone does not have a developer setting and always loses cellular connectivity. If the modem crashes and a loss of cellular connectivity occurs, the Android modem driver restarts the modem, and connectivity can be re-established. Consequently, this allows an attacker to perform a DoS attack either on cellular connectivity or preventing users from using their phones. Suppose an attacker continuously broadcasts the corrupted PWS messages. In that case, the DoS persists across reboots as the modem will immediately crash again after it has restarted. Cellular modems are also used without Android drivers, e. g., for industrial applications. If the drivers used in such applications do not actively restart the modem, then the DoS attack becomes persistent even if an attacker only broadcasts the corrupted message once.

5.3.2 PWS: Out of bounds read on Samsung modems (CVE-2022-39881)

We found an out of bounds memory read vulnerability in Samsung modems that is caused by sending a message with the `lastSegment` flag set but that is missing pages on the GSM layer, as illustrated in Figure 9 via the orange arrow. We observed that the Samsung Exynos-based phone showed a fragment of the test message leading to undefined behaviour. This should not happen as the corrupted message in this counter example is incomplete; while the message appears to be completely received on the LTE layer, it is lacking pages on the GSM layer. We suspect that the message is "completed" incorrectly by reading unrelated data from memory. Upon further investigation, we discovered that by manually increasing different length denominators in the model-generated, malicious message, we could read out-of-bounds memory contents.

Impact: The CVE has received a CVS Score of 9.1 (critical) by NIST. As the exact location of the memory read cannot be known because it appears to be dynamically allocated heap memory, exploitation requires techniques like heap grooming to lead to reliable results. An attacker could then use it to read arbitrary memory contents. In particular, given a reliable heap grooming routine, one might be able to access Packed Data Convergence Protocol (PDCP) session encryption keys. To exploit this in practice, an attacker would require physical access to the phone to read the data from the display or Android debug logs, or needs to find an additional vulnerability that enables an over-the-air back-channel.

5.3.3 SMS: Denial of Service against Mediatek modems (CVE-2022-32591)

Another interesting undefined behaviour is `GSM7BIT_INCORRECT_USER_DATA_LENGTH`, synthesized from our SMS model, which lead to a so-called “SMS of death” attack against Mediatek modems, similar to the famous Ping of Death attack against Microsoft Windows.

The undefined behaviour results from a length denominator for the message body being smaller than the body length of the SMS, as the specification does not provide a rule on truncation. In the counter example, the difference between the signalled and the actual body length is one. The Mediatek MT6853 and the Qualcomm modems both react to this in the same way, i. e., by truncating the affected two-letter test messages after one letter. A further investigation showed that this test group appears to reliably crash the Mediatek modem if one chooses a counter example for this undefined behaviour with an even smaller “user data length” value, while the Qualcomm modem ignores the message. This behaviour demonstrates that implementations may behave differently in the presence of undefined behaviour, even if it seems to be implemented in a similar way at first sight. The issue has been confirmed and was assigned CVE-2022-32591 by Mediatek. Diagnosis by Mediatek revealed that this bug does not directly crash the modem but crashes the Mediatek-supplied Radio Interface Layer (RIL) in Android, which then causes a modem restart and corrupts debug information.

Interestingly, the Mediatek MT6768 reacts differently to the `GSM7BIT_INCORRECT_USER_DATA_LENGTH` counter example and only shows the first part of the first malicious message. We thus speculate that the modem firmware or RIL developers might have modified the SMS functionality between the 4G and 5G modem generations. The MT6768 also does not crash for smaller user data length values.

Impact: The CVE has received a CVS Score of 7.5 (high) by NIST. The test message that crashed the Mediatek 5G baseband (CVE-2022-32591) allows an attacker to perform a DoS-attack by sending a malicious SMS to the victim’s UE. Depending on the MNO’s network configuration, this could be potentially done by sending the victim an SMS using a modified attacker UE. In such a case, this is a low-cost attack that does not require RF proximity or precise timing.

Since the RIL crash disables cellular connectivity, such an attack is suitable to prevent a victim from communicating using their phone, including preventing them from calling emergency services. Embedded devices affected by this attack will also not be able to communicate anymore if they use the Android RIL, which is especially problematic for safety-critical functionalities, such as automatic emergency calls by cars after an accident or in applications affecting critical infrastructure, such as monitoring wind turbines or solar power plants. In summary, undefined behaviour can have real and devastating consequences on UEs. Our approach can help verify whether smartphone modems react in security-critical ways when facing situations that have not been specified.

6 Discussion

A number of points warrant further discussion.

Human error. Translating the natural-language cellular network protocol specification into a formal model grounded in temporal logic is a completely manual step in our process. This has the drawback that human error in the process of modelling the specification, such as misinterpretation of the specification document, might lead to an incorrect model that does not correspond to it.

Automating the process of model creation would eradicate the potential for human error. However, automating the process of translating natural language description into a model grounded in logic via natural language processing is an unsolved research question; this is a limitation that related works [12, 37] similar to ours also suffer from. We stress that our approach attempts to minimize human error by focusing the task on modelling *defined* behaviour and not requiring the human to think about behaviours that are *undefined*.

Sampling. As the state space of logic models expands exponentially relative to the number of state variables, our models would become too large to be computed on limited resources. To counteract this state explosion we sample specific values such as message identifiers and length specifiers rather than using their entire permitted value range, as is common in unit testing and state of the art [37]. For identifiers, this does not limit the practical capabilities of our model, as we allow more identifiers than any of our counter examples require. For the length delimiters, it could become a practical problem, especially if models become larger and dependencies between individual length delimiters become more unclear. In these cases, such issues could be overcome by additional computing power. The `tlc` allows for parallel execution on multiple systems, and therefore larger state spaces become less of a problem given enough hardware resources.

Model boundaries. In line with previous work [12,37], we only model subsets of the specification. Modelling specific parts reduces the manual effort needed and consumes fewer computation resources. Given the specification’s complexity, a model of the entire LTE stack would likely be too large to handle, even when using massively distributed computing resources. However, we argue that given the real-world attacks that we found modelling subsets of the specification, it shows how modelling subsets can improve the specification and find undefined behaviour.

Relation between undefined behaviour and vulnerabilities. We stress that not every undefined behaviour results in an implementation vulnerability. This is due to the nature of undefined behaviour, which represents a lack of a definition, such that developers may make *any* assumption, including both insecure and secure ones. As a consequence, a vulnerability may manifest in some, but not all, implementations simply due to the varying interpretations by different developers. Crucially, at the specification-level, we cannot predict with certainty whether a vulnerability will result from undefined behaviour. This is not a limitation regarding our primary target user group, which is specification bodies and people involved in the creation of specifications. However, if our approach is used by developers and pentesters to find implementation vulnerabilities caused by undefined behaviour, they require an auxiliary second step to test implementation behaviour. Our setup described in Section 3.5 is one way of handling this issue. Another approach is to manually and statically analyse behaviour of a given implementation regarding a PDU that induces undefined behaviour.

Lastly, as undefined behaviour is not the only reason for security vulnerabilities, eliminating all undefined behaviours and having a well-defined specification does not guarantee that implementations will be secure. Additionally, incorrect implementation of defined behaviour, insecure specifications, and side channels are other common causes for vulnerabilities, all of which can be addressed by techniques designed to mitigate such issues.

Industry's perspective on undefined behaviour. We have discussed our findings within the GSMA CVD programme. In their response, CVD panel members pointed out that the specifications' primary goal is to ensure interoperability. Undefined behaviour is not generally considered a security issue,if it can be reasonably expected that an implementor understands how each undefined behaviour can be implemented in a secure way. Furthermore, the feedback of some GSMA CVD panel members mentioned that there is a trade off between writing specifications in a well-defined manner to promote implementation security and intentionally permitting undefined behaviour to enable innovation, differentiation, and subsequent competition between implementations. For instance, undefined behaviour in theory could enable implementers to use the unspecified areas for optimisations (akin to compilers) and additional features or differentiate themselves in regard to implementation quality.

As the features we have investigated leave no room for additional features or significant optimisations, we, therefore, believe that this argument does not apply for every undefined behaviour. Instead, we argue that in many cases, the specifications should be as strict as possible to promote secure implementations and protect end users from implementation vulnerabilities. However, this opens up a new research question for future works, namely, identifying such potentially beneficial undefined behaviour, that might enable optimisations or additional features.

7 Related Work

Beyond DoLTEst by Park et al. [37], which is closest to our work, there is a large number of works on modelling cellular communications using formal verification to validate *defined* behaviour [1, 10, 16, 21, 23, 41]. These approaches rely on a manually created formal model and validate the model against manually created invariants, representing the specification's intention. Similarly, there exists a multitude of attacks based on specification-based vulnerabilities in various LTE and 5G features [38, 39]. All of these works focus on finding security issues in the defined behaviour of the specification and do not address undefined behaviour.

Some works focus on leveraging the specification to find security issues in the implementation. They automatically extract finite state machines from the implementation, using either source code instrumentation [24] or analysing network packet captures to recover the internal state [14]. While these approaches remove the need to manually interpret the specification, they can only handle scenarios where the model checking is performed on defined behaviour.

Another set of approaches targeting UE implementations focuses on validating the positive behaviour defined by the specification. Kim et al. [25] presented a concept that compares the PDU parser implementations in UE firmware against the PDU specifications. Chen et al. [12] validated UE behaviour against sections of the specification, which are specifically highlighted using keywords. Kim et al. [26] took a fuzzy approach to generate test messages from pre-recorded real-world UE behaviour. Other works validate specific aspects of eNB and UE implementations in more detail to detect information leaks [9, 13, 20, 22, 27, 28] or perform normally prohibited actions [33].

In contrast to the presented approaches, focusing on defined behaviour or targeting UE implementations, our approach is the first to enable a systematic evaluation of cellular specifications regarding undefined behaviour. Furthermore, our approach uniquely provides formal descriptions of undefined behaviour, which allows us to systematically eradicate these issues from the specification.

Recently, there have also been fuzzing approaches targeting modem firmware, which send messages to real [17] or emulated modems [19, 35] in rapid succession, and monitor them for crashes or other indicators of malfunction. The systematic difference between fuzzers and our approach lies in the fact that fuzzers are designed to identify implementation issues, but do not exhaustively reason about a given feature. They also do not apply during the specification stage, as they require a finished implementation, instead of a specification. As such, we see fuzzing as orthogonal to our proposed approach.

8 Conclusion

We demonstrate the importance of checking cellular network specifications for undefined behaviour. Our proposed model-based approach was able to find 58 cases of undefined behaviour in the LTE specifications for PWS, SMS and RRC. In doing so, we brought forth the dangers of reusing features from previous generations of cellular network specifications, and in particular highlighted issues arising at the interface between the protocol generations.

Acknowledgments

We would like to thank our anonymous reviewers for their valuable comments and suggestions. We also thank Marc Leinweber, Merlin Chlosta, Lukas Bernhard and Thorsten Holz for their feedback. This work was funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany's Excellence Strategy - EXC 2092 CASA - 390781972.

References

[1] 3GPP. Formal Analysis of the 3G Authentication Protocol. TR 33.902, 3rd Generation Partnership Project (3GPP), 10 2001.

[2] 3GPP. Point-to-Point (PP) Short Message Service (SMS) support on mobile radio interface. TS 24.011, 3rd Generation Partnership Project (3GPP), 02 2010.

[3] 3GPP. Technical realization of Cell Broadcast Service (CBS). TS 23.041, 3rd Generation Partnership Project (3GPP), 12 2010.

[4] 3GPP. Technical realization of the Short Message Service (SMS). TS 23.040, 3rd Generation Partnership Project (3GPP), 09 2010.

[5] 3GPP. Evolved Universal Terrestrial Radio Access (E-UTRA) and Evolved Packet Core (EPC); User Equipment (UE) conformance specification; Part 1: Protocol conformance specification. TS 36.523-1, 3rd Generation Partnership Project (3GPP), 06 2011.

[6] 3GPP. Evolved Universal Terrestrial Radio Access (E-UTRA); Radio Resource Control (RRC); Protocol specification. TS 36.331, 3rd Generation Partnership Project (3GPP), 06 2011.

[7] 3GPP. Non-Access-Stratum (NAS) protocol for Evolved Packet System (EPS); Stage 3. TS 24.301, 3rd Generation Partnership Project (3GPP), 06 2011.

[8] 3GPP. Technical Specification Group working methods. TR 21.900, 3rd Generation Partnership Project (3GPP), 06 2011.

[9] Sangwook Bae, Mincheol Son, Dongkwan Kim, CheolJun Park, Jiho Lee, Sooel Son, and Yongdae Kim. Watching the Watchers: Practical Video Identification Attack in LTE Networks. In *USENIX Security Symposium (SSYM)*, 2022.

[10] David Basin, Jannik Dreier, Lucca Hirschi, Saša Radomirovic, Ralf Sasse, and Vincent Stettler. A Formal Analysis of 5G Authentication. In *Conference on Computer and Communications Security (CCS)*, pages 1383–1396, 2018.

[11] Evangelos Bitsikas and Christina Pöpper. You have been warned: Abusing 5G's Warning and Emergency Systems. In *ACM Annual Computer Security Applications Conference (ACSAC)*, 2022.

[12] Yi Chen, Yepeng Yao, XiaoFeng Wang, Dandan Xu, Chang Yue, Xiaozhong Liu, Kai Chen, Haixu Tang, and Baoxu Liu. Bookworm Game: Automatic Discovery of LTE Vulnerabilities Through Documentation Analysis. In *IEEE Symposium on Security and Privacy (SP)*, 2021.

[13] Zishuai Cheng, Mihai Ordean, Flavio D Garcia, Baojiang Cui, and Dominik Rys. Watching your Call: Breaking VoLTE Privacy in LTE/5G Networks. In *Privacy Enhancing Technologies Symposium (PETS)*, 2023.

[14] Merlin Chlosta, David Rupprecht, and Thorsten Holz. On the Challenges of Automata Reconstruction in LTE Networks. In *Conference on Security & Privacy in Wireless and Mobile Networks (WiSec)*, 2021.

[15] Merlin Chlosta, David Rupprecht, Thorsten Holz, and Christina Pöpper. LTE Security Disabled: Misconfiguration in Commercial Networks. In *Conference on Security & Privacy in Wireless and Mobile Networks (WiSec)*, 2019.

[16] Cas Cremers and Martin Dehnel-Wild. Component-based Formal Analysis of 5G-AKA: Channel Assumptions and Session Confusion. In *Symposium on Network and Distributed System Security (NDSS)*, 2019.

[17] Matheus E Garbelini, Zewen Shang, Sudipta Chattopadhyay, Sumei Sun, and Ernest Kurniawan. Towards Automated Fuzzing of 4G/5G Protocol Implementations Over the Air. In *IEEE Global Communications Conference*, 2022.

[18] Ismael Gomez-Miguelez, Andres Garcia-Saavedra, Paul D Sutton, Pablo Serrano, Cristina Cano, and Doug J Leith. srsLTE: an open-source platform for LTE evolution and experimentation. In *ACM International Workshop on Wireless Network Testbeds, Experimental Evaluation and Characterization (WiNTECH)*, 2016.

[19] Grant Hernandez, Marius Muench, Dominik Maier, Alyssa Milburn, Shinjo Park, Tobias Scharnowski, Tyler Tucker, Patrick Traynor, and Kevin RB Butler. FIRMWIRE: Transparent Dynamic Analysis for Cellular Baseband Firmware. In *USENIX Security Symposium (SSYM)*, 2022.

[20] Byeongdo Hong, Sangwook Bae, and Yongdae Kim. GUTI Reallocation Demystified: Cellular Location Tracking with Changing Temporary Identifier. In *Symposium on Network and Distributed System Security (NDSS)*, 2018.

[21] Syed Hussain, Omar Chowdhury, Shagufta Mehnaz, and Elisa Bertino. LTEInspector: A Systematic Approach for Adversarial Testing of 4G LTE. In *Symposium on Network and Distributed System Security (NDSS)*, 2018.

[22] Syed Rafiul Hussain, Mitziu Echeverria, Omar Chowdhury, Ninghui Li, and Elisa Bertino. Privacy Attacks to the 4G and 5G Cellular Paging Protocols using Side Channel Information. *Symposium on Network and Distributed System Security (NDSS)*, 2019.

[23] Syed Rafiul Hussain, Mitziu Echeverria, Imtiaz Karim, Omar Chowdhury, and Elisa Bertino. 5GReasoner: A Property-Directed Security and Privacy Analysis Framework for 5G Cellular Network Protocol. In *Conference on Computer and Communications Security (CCS)*, 2019.

[24] Imtiaz Karim, Syed Rafiul Hussain, and Elisa Bertino. ProChecker: An Automated Security and Privacy Analysis Framework for 4G LTE Protocol Implementations. In *Conference on Distributed Computing Systems (ICDCS)*, 2021.

[25] Eunsoo Kim, Dongkwan Kim, CheolJun Park, Insu Yun, and Yongdae Kim. Basespec: Comparative analysis of baseband software and cellular specifications for L3 protocols. In *Symposium on Network and Distributed System Security (NDSS)*, 2021.

[26] Hongil Kim, Jiho Lee, Eunkyu Lee, and Yongdae Kim. Touching the Untouchables: Dynamic Security Analysis of the LTE Control Plane. In *IEEE Symposium on Security and Privacy (SP)*, 2019.

[27] Martin Kotuliak, Simon Erni, Patrick Leu, Marc Roeschlin, and Srdjan Čapkun. LTrack: Stealthy Tracking of Mobile Phones in LTE. In *USENIX Security Symposium (SSYM)*, 2022.

[28] Nitya Lakshmanan, Nishant Budhdev, Min Suk Kang, Mun Choon Chan, and Jun Han. A Stealthy Location Identification Attack Exploiting Carrier Aggregation in Cellular Networks. In *USENIX Security Symposium (SSYM)*, 2021.

[29] Leslie Lamport. Introduction to TLA. Technical report, Microsoft, 1994.

[30] Leslie Lamport. Specifying concurrent systems with TLA+. *Calculational System Design*, pages 183–247, 1999.

[31] Leslie Lamport. *Specifying Systems: the TLA+ Language and Tools for Hardware and Software Engineers*. Addison-Wesley, 2002.

[32] Gyuhong Lee, Jihoon Lee, Jinsung Lee, Youngbin Im, Max Hollingsworth, Eric Wustrow, Dirk Grunwald, and Sangtae Ha. This is your president speaking: Spoofing alerts in 4G LTE networks. In *Conference on Mobile Systems, Applications, and Services*, 2019.

[33] Chi-Yu Li, Guan-Hua Tu, Chunyi Peng, Zengwen Yuan, Yuanjie Li, Songwu Lu, and Xinbing Wang. Insecurity of Voice Solution VoLTE in LTE Mobile Networks. In *Conference on Computer and Communications Security (CCS)*, pages 316–327, 2015.

[34] Adaptive Mobile Security Limited. Simjacker. https://f.hubspotusercontent10.net/hubfs/8487362/Reports/AdaptiveMobile_Security_Simjacker_Technical_Paper_v1.01.pdf, 10 2019. [Online; accessed June 9, 2023].

[35] Dominik Maier, Lukas Seidel, and Shinjo Park. Basesafe: Baseband SAnitized Fuzzing through Emulation. In *Conference on Security & Privacy in Wireless and Mobile Networks (WiSec)*, 2020.

[36] Ken L McMillan. Applying SAT methods in Unbounded Symbolic Model Checking. In *International Conference on Computer Aided Verification*, 2002.

[37] CheolJun Park, Sangwook Bae, BeomSeok Oh, Jiho Lee, Eunkyu Lee, Insu Yun, and Yongdae Kim. DoLTEst: In-depth Downlink Negative Testing Framework for LTE Devices. In *USENIX Security Symposium (SSYM)*, 2022.

[38] David Rupprecht, Katharina Kohls, Thorsten Holz, and Christina Pöpper. Breaking LTE on Layer Two. In *IEEE Symposium on Security and Privacy (SP)*, 2019.

[39] Altaf Shaik, Ravishankar Borgaonkar, Shinjo Park, and Jean-Pierre Seifert. New Vulnerabilities in 4G and 5G Cellular Access Network Protocols: Exposing Device Capabilities. In *Conference on Security and Privacy in Wireless and Mobile Networks*, 2019.

[40] Parv Sharma and Dale Gai. MediaTek Captures Record 43% Share of Smartphone AP/SoC Shipments in Q2 2021. https://www.counterpointresearch.com/mediatek-captures-record-43-share-smartphone-apsoc-shipments-q2-2021/. [Online; accessed June 9, 2023].

[41] Jiaqi Yin, Huibiao Zhu, Yuan Fei, and Qiwen Xu. Formal Modelling and Verification of the RTPS Behavior Module. In *International Symposium on Theoretical Aspects of Software Engineering (TASE)*, pages 127–134. IEEE, 2021.

[42] Yuan Yu, Panagiotis Manolios, and Leslie Lamport. Model Checking TLA$^+$ Specifications. In *Correct Hardware Design and Verification Methods, 10th IFIP WG 10.5 Advanced Research Working Conference, CHARME*, 1999.

A Exemplary TLA$^+$ output

An exemplary test case generated via TLA$^+$, belonging to the test case group `LAST_MESSAGE_BUT_FURTHER_MESSAGES`. The `<<elem1, elem2, ... >>` notation refers to a sequence of elements, while `[key |-> val, key2 |-> val2]` refers to a structure like object. In this particular case, this denotes a sequence of SIB12 messages to be send sequentially.

Note that both messages contain the last message flag (warningMessageSegmentTypeR9 is 1). Since the message segment that arrives first claims to be the second message of the reassembly sequence, the UE waits for the first message in the reassembly sequence. When the first message of the reassembly sequence, which is the second message to be send, arrives, that message also claims to be the last message. This is a contradiction to the previously received message and a case that is not covered by the specification.

```
<< [ warningMessageSegmentTypeR9 |-> 1,
     messageIdentifierR9 |-> 4370,
     serialNumberR9 |-> 12289,
     warningMessageSegmentNumberR9 |-> 2,
     warningMessageSegmentR9 |-> <<
        [messageInformationLength |-> 1]
     >> ],
   [ warningMessageSegmentTypeR9 |-> 1,
     messageIdentifierR9 |-> 4370,
     serialNumberR9 |-> 12289,
     warningMessageSegmentNumberR9 |-> 1,
     warningMessageSegmentR9 |-> <<
        [messageInformationLength |-> 1]
     >> ]
>>
```

B Implementation in srsRAN

This code fragment implemenents the first of the two messages shown in appendix Section A in `srsRAN` . Integrating it into the existing SIB scheduling procedure of `srsRAN` causes automatic encoding of the message and transfer to the UE upon the next scheduled SIB window. As the test cases generated via TLA$^+$ are non-symbolic, the translation from ASN1 to `srsRAN` is trivial and does, with the exception of replacing the dummy text, not require substituting symbolic for concrete values.

```
sib12.sib12_v920()
  .warning_msg_segment_type_r9 =
asn1::rrc::sib_type12_r9_s
  ::warning_msg_segment_type_r9_opts
  ::last_segment;

sib12.sib12_v920().msg_id_r9 =
  sib12.sib12_v920().msg_id_r9
  .from_number(4370);

sib12.sib12_v920().serial_num_r9 =
  sib12.sib12_v920().serial_num_r9
  .from_number(12289);

sib12.sib12_v920()
  .warning_msg_segment_num_r9 = 2;

sib12.sib12_v920().warning_msg_segment_r9 =
  sib12_2.sib12_v920().warning_msg_segment_r9
  .from_string(
    "01<Dummytext␣of␣1␣septet>01"
  );

sib_list.push_back(sib12);
```

C RRC Model Details

For our RRC model, we implement the following downlink RRC PDUs: RRCConnectionSetup, RRCConnectionReconfiguration, SecurityModeCommand, UECapabilityEnquiryMessage, CounterCheckMessage, UEInformationRequest, DLInformationTransfer, RRCConnectionReestablishment, RRCConnectionReject, RRCConnectionRelease.

D Non-security-critical Undefined Behaviour

Beyond undefined behaviours leading to security-critical reactions from the UEs, others might be discarded silently or lead to the display of unintended messages. To analyse whether the response of the different smartphones differs, we report all undefined behaviours that led to CVEs in Table 3. Table 4 lists all undefined behaviours for PWS. Visibly, a large number of test cases lead to incoherent behaviour. Table 5 reports the results for SMS, where a number of test cases cause the exact same behaviour across all smartphones, while others vary significantly. Lastly, Table 6 contains undefined behaviours for RRC. In this case, no messages (such as the warning or SMS) can be displayed. Instead, we monitor whether the procedure completes or the connection is reestablished in some way. As the data shows, behaviour varies between the different smartphones.

Table 4: Evaluation of test cases generated using the PWS model. Cell colours indicate behaviour shown by the phone. “Seque.” denotes test cases using message sequences.

Undefined behaviour	Seque.	Samsung A41	Oppo A73 5G	Huawei P40 Lite 5G	Samsung S20 5G	OnePlus 8
		Mediatek MT6768	Mediatek MT6853V	HiSilicon Kirin 820 5G	Samsung Exynos 990	Qualcomm SM8250
`SEGMENT_0_LACKS_NUM_PAGES`		▨	▨	🟦	🟦	🟦
`SEGMENT_NON_0_CONTAINS_NUM_PAGES`		▨	▨	🟦	🟦	🟦
`SIB_COMPLETE_BUT_MORE_PAGES`		🟦	🟦	🟦	🟨 CVE-2022-39881	◈
`LAST_MESSAGE_BUT_FURTHER_MESSAGES`	●	🟥 CVE-2022-26446 (1)	🟥 CVE-2022-26446 (1)	🟦	🟦	🟦
`MESSAGE_BEYOND_LAST_SEGMENT`	●	🟥 CVE-2022-26446 (2)	🟥 CVE-2022-26446 (2)	🟦	🟦	🟦
`TOO_MANY_PAGES`	●	▨	▨	🟦	🟦	🟦
`NUMBER_OF_PAGES_TOO_SMALL`	●	▨	▨	🟦	🟪	🟪
`EMPTY_WARNING_MESSAGE_SEGMENT`		🟥 CVE-2022-26446 (3)	🟥 CVE-2022-26446 (3)	🟦	🟦	🟦

🟦 Only benign messages shown - 🟥 Modem crash - 🟨 Test message and benign messages shown, indications of overflow
◈ One test message shown, benign message modified - 🟪 Some test messages and some benign messages shown - ▨ Incoherent behaviour

Table 5: Evaluation of test cases generated using the SMS model. Cell colours indicate behaviour shown by the phone. “Seque.” denotes test cases using message sequences.

Undefined behaviour	Seque.	Samsung A41	Oppo A73 5G	Huawei P40 Lite 5G	Samsung S20 5G	OnePlus 8
		Mediatek MT6768	Mediatek MT6853V	HiSilicon Kirin 820 5G	Samsung Exynos 990	Qualcomm SM8250
`ORIGINATOR_ADDRESS_TOO_SHORT`		🟨	🟩	🟦	🟦	🟦
`INCORRECT_RP_USER_DATA_LENGTH`		🟦	🟦	🟦	🟫	🟦
`NO_UDH_BODY_LEN_UNEQ_USER_DATA_LEN`		🟦	🟦	🟦	🟦	🟦
`DESTINATION_ADDRESS_NOT_EMPTY`		🟨	🟩	🟨	🟨	🟦
`NO_UDH_DESPITE_SIGNALLED`		🟦	🟦	🟦	🟦	🟦
`UDH_DESPITE_NOT_SIGNALLED`		🟦	🟦	🟦	🟦	🟦
`UCS2_INCORRECT_USER_DATA_LENGTH`		🟦	🟦	🟦	🟦	🟦
`8BIT_INCORRECT_USER_DATA_LENGTH`		🟦	🟦	🟦	🟦	🟦
`GSM7BIT_INCORRECT_USER_DATA_LENGTH`		◈	🟧 CVE-2022-32591	🟦	🟪	🟧
`IE0_WITH_NUM_MESSAGES_ZERO`		🟨	🟩	🟨	🟨	🟩
`INCORRECT_UDH_LENGTH`		🟨	🟩	🟨	🟨	🟩
`IE0_SEQ_NUM_ZERO`		🟨	🟩	🟨	🟨	🟩
`IE0_SEQ_NUM_GREATER_NUM_MESSAGES`		🟨	🟩	🟨	🟨	🟩
`TIMESTAMP_WITH_INVALID_DIGITS`		🟨	🟩	🟨	🟨	🟦
`TIMESTAMP_WITH_MONTH_ZERO`		🟨	🟩	🟨	🟨	🟦
`UCS2_ODD_BODY_LENGTH`		🟨	♦	🟦	🟦	🟦
`CONFLICT_MESSAGE_EXISTS`	●	🟦	🟦	🟦	🟦	🟦
`CONFLICT_NUM_MESSAGES_CHANGED`	●	🟦	🟦	🟦	🟦	🟦
`INCORRECT_MESSAGE_REFERENCE`	●	■	■	■	■	■
`CONFLICT_MORE_MESSAGES_TO_SEND`	●	■	■	■	■	■
`CONFLICT_REPLY_PATH`	●	■	■	■	■	■
`CONFLICT_ALPHABET`	●	🟨	🟦	🟦	🟦	🟦

🟦 Only benign message shown 🟩 Both test messages and benign message shown 🟨 One test message and benign message shown
■ Reassembled test messages and benign message shown 🟫 Both test messages shown with incorrect character decoding, benign message shown
◈ Only a part of one test message shown 🟧 Parts of both test messages shown, benign message shown completely
🟪 Part of one test message shown, benign message shown completely
♦ Both test messages shown with partially incorrect character decoding, benign message shown correctly

Table 6: Evaluation of test cases generated using the RRC model. Cell colours indicate behaviour shown by the phone. “Seque.” denotes test cases using message sequences. Details on how we evaluate behaviour can be found in Section 3.5.

Undefined behaviour	Seque.	Samsung A41	Oppo A73 5G	Huawei P40 Lite 5G	Samsung S20 5G	OnePlus 8
		Mediatek MT6768	Mediatek MT6853V	HiSilicon Kirin 820 5G	Samsung Exynos 990	Qualcomm SM8250
`CounterCheckMessage_NOT_CIPHERED`	●	🟦	🟦	🟥	🟥	🟥
`CounterCheckMessage_NOT_INTEGRITY_PROTECTED`	●	🟨	🟨	🟨	🟦	🟨
`RRCConnectionReconfigurationMessage_INVALID_DRB_CONFIG`	●	🟦	🟦	🟦	🟦	🟦
`RRCConnectionReconfigurationMessage_INVALID_SRB_CONFIG`	●	🟦	🟦	🟦	🟦	🟦
`RRCConnectionReconfigurationMessage_MOBILITY_CONTROL_NO_DRB`	●	🟨	🟫	▨	▨	▨
`RRCConnectionReconfigurationMessage_MOBILITY_CONTROL_NO_SRB2`	●	🟨	🟦	🟦	🟨	🟨
`RRCConnectionReconfigurationMessage_NO_CIPHERING_DESPITE_SECURE`	●	🟪	🟪	🟪	🟪	🟪
`RRCConnectionReconfigurationMessage_NO_INTEGRITY_DESPITE_SECURE`	●	🟪	🟪	🟪	🟪	🟪
`RRCConnectionReconfigurationMessage_UNPROTECTED_MEAS_OBJ_ADD`	●	🟥	🟥	🟥	🟥	🟥
`RRCConnectionReconfigurationMessage_UNPROTECTED_QUANT_CONF`	●	🟥	🟥	🟥	🟥	🟥
`RRCConnectionReconfigurationMessage_UNPROTECTED_REPORT_ADD`	●	🟥	🟥	🟥	🟥	🟥
`RRCConnectionReconfigurationMessage_UNPROTECTED_SECURITY_CONFIG`	●	🟥	🟥	🟦	🟥	🟦
`RRCConnectionReconfigurationMessage_UNPROTECTED_SPEED_STATE_PARS`	●	🟥	🟥	🟥	🟥	🟥
`RRCConnectionReconfigurationMessage_WHILE_T304_RUNNING`	●	🟨	🟨	🟦	🟨	🟨
`RRCConnectionReestablishment_BEFORE_SECURITY`	●	▨	▨	▨	▨	▨
`RRCConnectionReject_AFTER_SECURITY`	●	▨	▨	▨	▨	▨
`RRCConnectionRelease_NO_CIPHERING_DESPITE_SECURE`	●	🟥	🟦	🟥	🟥	🟥
`RRCConnectionRelease_NO_INTEGRITY_DESPITE_SECURE`	●	🟥	🟨	🟨	🟨	🟨
`RRCConnectionSetupMessage_AFTER_SECURITY_ENABLED`	●	🟦	🟦	🟦	🟦	🟦
`RRCConnectionSetupMessage_DRB_WITHOUT_SECURITY`		🟥	🟥	🟥	🟫	🟦
`RRCConnectionSetupMessage_SRB2_WITHOUT_SECURITY`		🟦	🟦	🟦	🟫	🟫
`SecurityModeCommandMessage_AFTER_SECURITY_ENABLED`	●	🟪	🟪	🟪	🟪	🟪
`SecurityModeCommandMessage_CIPHER_WITHOUT_INTEGRITY_CONF`	●	🟪	🟪	🟪	🟪	🟪
`UECapabilityEnquiryMessage_NO_CIPHERING_DESPITE_SECURE`	●	🟥	🟥	🟥	🟥	🟥
`UECapabilityEnquiryMessage_NO_INTEGRITY_DESPITE_SECURE`	●	🟨	🟨	🟨	🟨	🟨
`UEInformationRequestMessage_BEFORE_SECURITY`	●	▨	▨	▨	▨	▨
`UEInformationRequestMessage_NO_CIPHERING_DESPITE_SECURE`	●	🟥	🟥	🟥	🟥	🟥
`UEInformationRequestMessage_NO_INTEGRITY_DESPITE_SECURE`	●	🟨	🟨	🟨	🟨	🟨

🟥 UE signals completion - 🟦 UE sends a new connection request - 🟨 UE reestablishes a connection
🟪 SecurityModeCommand fails ▨ PDU(s) ignored 🟫 Incoherent behaviour