Instructions Unclear: Undefined Behaviour in Cellular Network Specifications

Daniel Klischies, *Ruhr University Bochum;* Moritz Schloegel and Tobias Scharnowski, *CISPA Helmholtz Center for Information Security;* Mikhail Bogodukhov, *Independent;* David Rupprecht, *Radix Security;* Veelasha Moonsamy, *Ruhr University Bochum*

https://www.usenix.org/conference/usenixsecurity23/presentation/klischies

This artifact appendix is included in the Artifact Appendices to the Proceedings of the 32nd USENIX Security Symposium and appends to the paper of the same name that appears in the Proceedings of the 32nd USENIX Security Symposium.

August 9–11, 2023 • Anaheim, CA, USA

978-1-939133-37-3

Open access to the Artifact Appendices to the Proceedings of the 32nd USENIX Security Symposium is sponsored by USENIX.

USENIX'23 Artifact Appendix: Instructions Unclear: Undefined Behaviour in Cellular Network Specifications

Daniel Klischies*, Moritz Schloegel†, Tobias Scharnowski†
Mikhail Bogodukhov‡, David Rupprecht§, Veelasha Moonsamy*

*Ruhr University Bochum, †CISPA Helmholtz Center for Information Security
‡Independent, §Radix Security

A Artifact Appendix

A.1 Abstract

The artifacts for *Instructions Unclear: Undefined Behaviour in Cellular Network Specifications* consist of two main parts: The **TLA⁺ models** used to discover undefined behaviours and the **modified `srsRAN` implementations** used to test smartphone implementations of undefined behaviour. For each of the three LTE features, PWS, SMS, and RRC that we evaluate against, there is a separate TLA⁺ model and `srsRAN` version.

This document describes how to use the models to derive concrete examples of undefined behaviour in LTE specifications, and then employ our `srsRAN` forks to replay these concrete examples against a commercial UE. This way, one can verify the presence of undefined behaviour in multiple LTE feature specifications, as well as determine their real world impact.

A.2 Description & Requirements

A.2.1 Security, privacy, and ethical concerns

Evaluating the TLA⁺ models for undefined behaviour does not entail any direct risks.

Replaying the counter examples using `srsRAN` and an SDR against real UEs, however, imposes multiple risks:

1. You will create a (small) LTE cell; doing so without an RF shielding box might violate local laws.

2. Since you will replay network packets (PDUs) that you cannot target to a specific phone, any phone in the vicinity of your SDR will receive these PDUs and might be temporarily or permanently affected by that PDU. As our experiments have shown, this may cause a Denial of Service attack, information leakage, or potentially even more severe problems. This is why it is absolutely essential to use an RF shielding box, even if your local laws would permit running the experiment without it.

3. You might brick any phones that you test. We had at least one non-reproducible case where a phone temporarily refused any connection until we reset its NVRAM.

4. If misconfigured, you might brick the SDR that you are using. To mitigate this, we recommend that you read and obey its instruction manual carefully.

A.2.2 How to access

The artifact is available for download at https://zenodo.org/record/8013704.

A.2.3 Hardware dependencies

TLA⁺ models. The TLA⁺ models are CPU intensive. To reproduce the CPU time in Table 1 of our paper, a dual Intel Xeon Gold 6230R system, totaling 52 cores with 128 GB RAM is required. Using a different setup is possible, but might result in a different core-hour measurement. All other results will remain the same, no matter which CPU is used.

Replaying counter examples via `srsRAN`. To be able to replay concrete examples of undefined behaviour, the following components are required:

- Computer with Gigabit LAN
- Ettus Research USRP X300 with SFP+ to Gigabit RJ45 module
- 2x Ettus Research VERT900 antennas
- GPSDO for USRP X Series (PCB-Mounted GPS-Disciplined OCXO)
- Programmable SIM-card (e.g. Sysmocom sysmoISIM-SJA2)
- RF shielding box (faraday cage)
- Smartphone(s): We evaluate against a Samsung A41, Samsung S20 5G (European edition), Oppo A73 5G, Huawei P40 Lite 5G and OnePlus 8. To reproduce our exact results, all of these phones must be updated to firmware patch level *April 2022*

for the SMS and PWS tests and *January 2023* to reproduce the RRC results.

A.2.4 Software dependencies

TLA+ models. To run the TLA+ models, a `tlc` is required. We recommend to use https://github.com/pmer/tla-bin. This requires Linux or a BSD derivative with at least `Java 11` and `curl` installed.

Replaying counter examples via srsRAN. Installing srsRAN 4G requires multiple external libraries, drivers, and firmware for the USRP. As the requirements of our forks mostly match the requirements of the upstream version, we recommend following the official setup guide at https://docs.srsran.com/projects/4g/en/latest/general/source/1_installation.html#installation-from-source.

SMS: For the SMS fork, you will also have to install `libosmocore` (we tested with version `1.6.0`), `Python 3.8` and the `pyzmq` package (version `22.3.0`).

Virtualisation: We recommend *against* using virtual machines to run srsRAN, as the overhead induced by the virtualisation interferes with the delicate timing requirements of the connection between computer and USRP.

A.2.5 Benchmarks

The data set (concrete counter examples that represent undefined behaviour) used for the UE evaluation is derived via the TLA+ models. If you do not want to rerun the TLA+ model checking procedure, you can use the counter examples that we prepopulated our srsRAN forks with.

A.3 Set-up

A.3.1 Installation

TLA+ models. Assuming that `tlc` has bin installed via `tla-bin` as described in A.2.4, no more setup steps are required.

Replaying counter examples via srsRAN. Assuming that all dependencies of `srsRAN` have been installed as described in A.2.4, the next step is compiling one of our `srsRAN` forks. To do so, `cd` into the directory of the fork, and then run the following commands:

```
mkdir build
cd build
cmake ../
make
sudo make install
srsran_install_configs.sh service
```

The next step is to configure srsRAN correctly. To do so, please follow the instructions at https://docs.srsran.com/projects/4g/en/latest/app_notes/source/cots_ue/source/index.html. Since srsRAN has undergone multiple name changes, files might be named slightly differently in our forks (e. g., `srsran_install_configs.sh` instead of `srsran_4g_install_configs.sh`).

A.3.2 Basic Test

TLA+ models. Open a shell in the `models/rrc` directory of the artifact and run `tlc -deadlock rlc`. After 5-20 minutes (depending on your system), this results in a message saying `Model checking completed. No error has been found.`. The number of threads can be controlled using the `-workers n` parameter.

You can perform the same test for the SMS and PWS models. They require that you also supply `-maxSetSize 10000000` and take significantly longer (e. g., the SMS model will arrive at the same output after 75 days on 100 threads).

Replaying counter examples via srsRAN. Change into the directory of one of the three `srsRAN` clones. Then compile and install according to Section . You can now run the clone as follows:

1. (Only applies to SMS testing: `cd zmq_server && python3 zmq_server.py`).
2. Launch srsepc:
   ```
   sudo srsepc \
   --config /usr/local/share/srsran/epc.conf
   ```
3. Launch srsenb:
   ```
   sudo \
   UHD_IMAGES_DIR=/usr/share/uhd/images/ \
   srsenb /usr/local/share/srsran/enb.conf
   ```
4. On the phone under test: Ensure that your APN is set to `srsapn`
5. Open an adb connection to the phone and run `ping 8.8.8.8`. You should see that 8.8.8.8 is reachable.

If you receive any error regarding missing UHD firmware, double check with the srsRAN and Ettus documentation that the supplied path matches the location of the firmware binaries. This is not specific to our modifications.

Similarly, if srsRAN does not work or the internet connection of you phone does not work, we recommend following the srsRAN documentation – unless you are running a test case (more on that later), our srsRAN forks operate exactly the same as the upstream versions, such that all their troubleshooting guides apply.

A.4 Evaluation workflow

A.4.1 Major Claims

For your convenience, we summarize major claims our paper makes:

(C1): Modelling via TLA$^+$ of LTE specification parts is computationally feasible. This is demonstrated by experiment E1, described in Sections 4.1 to 4.3 in the paper, with the results shown in Table 1 in the paper.

(C2): Using our approach, one can find more undefined behaviours than using the state-of-the-art approach "DoLTEst". This is discussed in Table 2 and Section 5.1 of our paper and demonstrated using Experiment E2.

(C3): Our approach can find undefined behaviours that lead to real world vulnerabilities. We list these in Table 2 of our paper and describe the vulnerabilities in sections 5.3.1-5.3.3. We reproduce these results in E3.

A.4.2 Experiments

(E1): [< 1 human-hour + up to 180,000 compute-hours]:

Execution: To measure the number of *States* and *CPU hrs.* according to Table 1 of our paper, run `tlc` with the same parameters described previously in A.3.2.

To measure the number of *Undefined Behaviours* and number of *PDUs* shown in Table 1, you have to read the `.cfg` file of the model (e. g., `models/rrc/rrc.cfg`) and modify the corresponding `.tla` file. The `.cfg` file contains a `CONSTANTS` section listing each test case. These test cases also correspond to the test cases listed in Tables 4-6 in the Appendix of our paper. To verify that each of these test cases is an undefined behaviour, you change the `ConstTestCase` variable in the `.tla` file to the test case you want to run. The line in the file is marked by a comment saying `Modify this value to choose the behaviour that you want to generate a counter example for`. An example of a correct assignment in the RRC model is `ConstTestCase == RRCConnectionReject_AFTER_SECURITY`.

Results: for the first part, the model checking will eventually terminate. Note that this takes 180,000 core hours for the SMS model, so we recommend parallelising using the `-workers` flag. If you do not supply a number of workers, TLC might choose a (suboptimal) number below the number of available CPU threads. The command line output should look like the following (for the RRC model).

```
3019102 states generated,
955 distinct states found,
0 states left on queue.
[...]
Finished in 07min 00s at ([...])
```

The number of distinct states (here, 955) and the "Finished in" time multiplied by the number of threads should match the corresponding columns in Table 1. Note that the timing will vary a bit for the shorter test cases (RRC and PWS), as it is dominated by startup time. The measurement for the SMS test case is much more stable as it is dominated by the actual model checking time.

When evaluating a test case (i. e., after modifying `ConstTestCase`), the model checking will terminate early. You should see the following command line output (this example corresponds to `ConstTestCase == RRCConnectionReject_AFTER_SECURITY`):

```
Error: Invariant Invariant is violated.
Error: The behavior up to this point is:
[...]
State 4: <Next line 603,
col 9 to line 713,
col 43 of module rrc>
[...]
/\ currentSequence =
  << << "RRCConnectionSetupMessage",
      [ rrcTransactionIdentifier |-> 0,
        [...]
      ] >>,
    << "SecurityModeCommandMessage",
      [ rrcTransactionIdentifier |-> 0,
        [...]
      ] >>,
    << "RRCConnectionReject",
      [ criticalExtensions |->
          [...]
    >> >>
[...]
```

This illustrates that to trigger this undefined behaviour, a counter example has been generated that contains 3 separate PDUs and their value assignments: RRCConnectionSetupMessage and SecurityModeCommandMessage to setup the state, followed by a RRCConnectionReject that ultimately triggers the transition into an undefined state. By counting the number of different undefined behaviours (assignments of `ConstTestCase`) and the number of PDUs (entries in `currentSequence` of the final state before model checking procedure terminates), you can reproduce the *#UBs* and calculate *Avg. PDUs* columns of Table 1 in our paper.

(E2): [10 human-hours + 5 compute-hours]:

Execution: Repeat the procedure of setting `ConstTestCase` for each undefined behaviour that we found, to generate a counter example for each of these. Then, compare these to the RRC section of Table 5 in the DoLTEst paper[1]. To match our and their test cases, you must compare their message setup[2] to the sequences generated by our approach.

[1] https://www.usenix.org/system/files/sec22-park-cheoljun.pdf

[2] https://github.com/SysSec-KAIST/DoLTEst/blob/e2251bfa8cd74f49b23369619722255ed895ef5e/srsepc/src/mme/fzmanager_epc.cc#L560

Results: For each test case, you should get a different `currentSequence`. In the cases where the *Guideline* column in Table 2 of our paper contains a number, you will find a test case in the DoLTEst code that corresponds to the final message of the generated sequence. For cases where our table does not show a number, the DoLTEst authors do not provide a corresponding test case.

(E3): [3 human-hours + 200 compute-hours]:

Execution: To reproduce our CVEs, you will need to use the PWS and SMS srsRAN forks. To verify that our test cases match what was generated by our TLA$^+$ models, open the following files: `srsran/pws/srsenb/src/stack/rrc/rrc.cc` (PWS) and `srsran/sms/srsepc/src/mme/sms.cc` (SMS) and compare the values assigned to the structs to the TLA$^+$-generated counter examples. As the counter example generation is not deterministic, you might end up with different assignments (counter examples), but the same undefined behaviours.

To run the tests, turn on the phone and the SDR, and start `srsenb` and `srsepc` as described previously in Section A.3.2. For the PWS samples, you have to schedule SIB12 in the `sib.conf`, by including `12` in `si_mapping_info` and adding a `sib12` entry to the same file. For the SMS test case, `zmq_server.py` must be started. To choose a test case, you enter the test cases index into the enb window (PWS) or enter it in the `zmq_server.py` shell (SMS) and press enter. For sequences longer than one PDU, you have to repeat this as many times as there are PDUs in the sequence. The indexes can be found in the code, the ZMQ command line output, or by using the row indices of our result tables 4 and 5. Remember to restart enb, epc and the phone between each test case. Also remember to increase the system time of the phone by a week for each PWS test, as described in the paper (Section 3.5).

Results: You should see the behaviours described in Sections 5.3.1, 5.3.2 (PWS), and 5.3.3 (SMS) of our paper. To trigger a modem crash you might have to make the adjustments described in the paper. To determine the modem indeed crashed we recommend setting the phone into debug mode by dialling `*#9900#` and setting "Debug Level" to "HIGH" in the resulting hidden menu. Note that this setting only exists on Samsung phones (A41 and S20 in our case).

For the OOB read on Samsung S20 phones, please note that you might have to perform multiple attempts, as it depends on the current heap memory contents. We recommend indicating 5-7 pages (first two digits of the first `warning_msg_segment_r9`), as this demonstrates the attack while working relatively reliably.

We have reported the issues to Mediatek and Samsung, and the vulnerabilities might have been patched in firmware versions released after April 2022.

A.5 Version

Based on the LaTeX template for Artifact Evaluation V20220926. Submission, reviewing and badging methodology followed for the evaluation of this artifact can be found at https://secartifacts.github.io/usenixsec2023/.