USENIX Security '26 Artifact Appendix: <Semantics Over Syntax: Uncovering Pre-Authentication 5G Baseband Vulnerabilities>

Qiqing Huang

A Artifact Appendix

A.1 Abstract

This artifact provides an end-to-end implementation of CONSET (Constraint-Guided Semantic Testing), a framework for systematically extracting specification-level constraints from 3GPP standards and generating targeted test cases to uncover semantic inconsistencies in 5G UE implementations.

The artifact consists of three main components: **(A1)** a simulation testbed providing a pre-configured Docker environment and remote access for executing experiments; **(A2)** a constraint-driven toolchain comprising 26 executable scripts that extract semantic constraints from 3GPP specifications, normalize them into DSL rules using LLM-based reasoning, and generate syntactically valid test cases that deliberately violate these constraints; and **(A3)** a collection of test cases and proof-of-concept exploits that reproduce crashes in the OpenAirInterface (OAI) UE within the simulation environment.

The artifact includes pre-processed 3GPP specification documents and enables evaluators to: (i) re-run the complete constraint extraction pipeline from specifications to DSL rules, (ii) generate test cases based on extracted constraints, and (iii) reproduce the 29 unique crash sites reported in the paper on the OAI UE. Note that portions of the constraint extraction pipeline that rely on Mathpix and OpenAI APIs are marked as optional and can be skipped without affecting the core evaluation workflow. All proof-of-concept exploits are limited to the simulation environment; over-the-air exploits are deliberately withheld to prevent misuse.

A.2 Description & Requirements

A.2.1 Security, privacy, and ethical concerns

Execution Environment: The artifact is designed to run within an isolated Docker container, which minimizes risks to the evaluator's host system. Local execution of the A1/A3 Docker-based testbed requires elevated Docker privileges (e.g., `-privileged`, `-ipc=host`, `-network=host`) as documented in the Set-up section; however, all experiments are confined to an isolated simulation environment and do not interact with real networks or devices.

API Usage (Optional): Portions of the constraint extraction pipeline optionally use the OpenAI API and Mathpix API. If evaluators choose to use their own API keys, they should be aware that: (i) specification text excerpts from 3GPP documents will be transmitted to these external services for processing, and (ii) API usage will incur costs based on the providers' pricing models. Evaluators may skip these optional steps and use pre-computed intermediate results provided in the artifact.

Exploit Code: The artifact includes proof-of-concept exploit code that triggers crashes in the OpenAirInterface UE. This code is strictly limited to the simulation environment and cannot be used for over-the-air attacks. Evaluators should execute these exploits only within the provided testbed and should not attempt to adapt or deploy them against production networks or devices.

Network Isolation: All experiments run in a self-contained 5G SA simulation environment. No real cellular networks or commercial devices are involved during artifact evaluation.

Data Collection: The artifact does not collect, transmit, or store any personally identifiable information from evaluators. All experimental outputs remain local to the execution environment.

A.2.2 How to access

The artifact is permanently archived on Zenodo at: https://zenodo.org/doi/10.5281/zenodo.17984911

The Zenodo archive serves as the stable, versioned record of the artifact for long-term availability. It contains all components described in the Open Science section of the paper:

- **A1. Simulation testbed.** This component includes both the source-level modifications to the OAI gNB (based on the 2025.w5 branch) and a pre-built Docker image that encapsulates the complete simulation environment with all required dependencies pre-installed.
- **A2. Constraint-driven toolchain.** This component provides the full analysis and test generation pipeline, including specification preprocessing scripts, field dependency extraction, DSL-based mutation logic, and ASN.1 decoding/encoding utilities.
- **A3. Test cases and proof-of-concept exploits.** This

component contains the generated test cases and corresponding proof-of-concept exploit code that operate through the OAI gNB interception path in the simulation environment.

For Phase-2 functionality and reproducibility assessment, we additionally provide a public GitHub repository as a convenient working copy of the artifact: https://github.com/qiqingh/contest_AE_final

The GitHub repository mirrors the content of the Zenodo archive and provides updated documentation and execution instructions to facilitate artifact evaluation. It does not replace the archived Zenodo version.

For the A1 simulation testbed, although the original modified source code is provided for completeness, we ***strongly recommend*** that evaluators use the pre-configured remote environment. This ensures a consistent and stable evaluation environment.

Optional remote access. For reviewer convenience, we additionally provide optional access to a pre-configured remote environment in which the Docker-based A1 and A3 setup is already deployed and ready to run. This option allows evaluators to validate the simulation-based experiments without any local installation effort. Remote access is provided purely as a convenience mechanism and is not required for artifact evaluation. Access details are described in Experiment E2 below.

A.2.3 Hardware dependencies

None. The artifact does not require any specialized 5G wireless hardware. All evaluations can be completed entirely in a simulation environment. In particular, the artifact does not require SDR devices (e.g., USRP), commercial smartphones, SIM cards, antennas, or any over-the-air radio infrastructure.

The artifact runs on a standard x86-64 machine (e.g., Intel or AMD CPUs) and does not require GPU acceleration. A machine with 32 GB of RAM and 80 GB of available disk space is recommended. The full T3 inter-IE pipeline requires more RAM due to peak memory usage during multi-process serialization; evaluators with less RAM may use the lightweight verification alternatively. The artifact has been verified on the following configuration:

- CPU: 12th Gen Intel Core i7-1260P (12 cores)
- RAM: 32 GB
- OS: Ubuntu 22.04 LTS

For reviewer convenience, we additionally provide direct SSH access to a pre-configured remote environment where the Docker-based simulation environment is already deployed and ready to run. Access details are provided in Section A.3.1.

Although the artifact may trigger crashes or assertion failures of the simulated UE or gNB, such behavior is strictly confined to an isolated simulation environment and does not affect real devices or production networks.

A.2.4 Software dependencies

The artifact has been tested on Ubuntu 22.04 LTS and is expected to run on a standard Linux environment. No other operating systems are officially supported for artifact evaluation.

Docker is optional and is provided as a convenience to validate the A1 simulation testbed and A3 test cases. All required system libraries and runtime dependencies for these components are pre-installed in the provided Docker image, and reviewers are not required to compile or install additional third-party software when using this option. Alternatively, reviewers may access a pre-configured remote desktop environment that hosts the same Docker-based setup.

The A2 constraint-driven toolchain runs natively on Ubuntu 22.04 using Python 3.10.12. All Python dependencies are open-source and can be installed via `pip3`. For transparency and reproducibility, we provide an `installed_packages.txt` file generated via `pip3 freeze`, which enumerates the exact Python package versions used in our experiments.

The artifact does not rely on any proprietary or licensed software, vendor-specific SDKs, firmware images, or restricted datasets. Large language models are not invoked during Phase-2 artifact evaluation; all LLM-assisted processing has been performed offline prior to evaluation.

A.2.5 Benchmarks

This work does not rely on traditional machine-learning benchmarks or pre-collected datasets. Instead, the experiments are driven by protocol specifications, reference protocol messages, and target UE implementations.

Specifically, the artifact relies on the following inputs:

- **3GPP protocol specifications.** The constraint-driven analysis (A2) uses publicly available 3GPP technical specifications (e.g., TS 38.331 for RRC) as normative input to extract field-level and cross-field semantic constraints. Relevant specification excerpts used during evaluation are documented and referenced in the artifact.
- **Reference RRC messages.** The test generation pipeline operates on syntactically valid RRC messages that serve as seeds for decode–modify–re-encode mutation. These reference messages are either generated by the simulation environment or included in the artifact as encoded inputs.
- **UE implementations as workloads.** The evaluated workloads consist of the OpenAirInterface UE running in a simulation environment. The UE implementation

serves as the target system under test rather than a benchmark dataset.

All inputs required for the experiments are either publicly available (3GPP specifications) or included in the artifact and do not require any external datasets.

A.3 Set-up

This section describes the installation steps required to prepare the environment for evaluating the CONSET artifact. The artifact follows a two-stage workflow: **E1** runs on the host environment (A2), and **E2** runs inside a Docker-based simulation environment (A1/A3).

A.3.1 Installation

Operating system. The artifact is evaluated on Ubuntu 22.04 LTS (x86-64). No proprietary software is required.

E1 — Python environment (A2). The constraint-driven toolchain (A2) requires Python 3.10 on Ubuntu 22.04. Evaluators may start from a clean Docker container as the host environment:

```
docker run -it --name ae_e1_test ubuntu:22.04 bash
```

Then install the required system packages:

```
apt update && apt install -y \
  git python3 python3-venv python3-pip python3-dev \
  gcc g++ build-essential make cmake pkg-config \
  libcairo2-dev libsystemd-dev gettext \
  libdbus-glib-1-dev libgirepository1.0-dev \
  libdbus-1-dev libcups2-dev
```

Clone the repository and set up a Python virtual environment:

```
git clone https://github.com/qiqingh/contest_AE_final
cd contest_AE_final/A2_constraintdriven_toolchain
python3 -m venv .venv
source .venv/bin/activate
pip3 install -r installed_packages.txt
```

This installs all non-proprietary packages required by the analysis, constraint extraction, and ASN.1 decoding/encoding pipeline.

E2 — Simulation environment (A1/A3). For the simulation-based testbed and proof-of-concept exploits (A1/A3), evaluators may set up the environment locally by pulling the pre-built Docker image:

```
sudo docker pull kqing0515/oai_testing:v3
sudo docker run -it \
  --name oai25_testing \
  --cpus="8" \
  --privileged \
  --ipc=host \
  --network=host \
  --mount type=tmpfs,destination=/dev/shm \
  --mount type=tmpfs,destination=/dev/mqueue \
  kqing0515/oai_testing:v3
```

A.4 Evaluation Workflow

This section describes the evaluation workflow used by the Artifact Evaluation Committee (AEC) to assess the functionality and reproducibility of the CONSET artifact. The workflow focuses on validating the key claims of the paper that are directly supported by the released artifacts, without requiring over-the-air (OTA) experiments on commercial devices.

A.4.1 Major Claims

(C1): *CONSET enables systematic generation of syntactically valid but semantically inconsistent 5G RRC messages by extracting specification-level constraints and cross-field dependencies. This claim is supported by the constraint extraction and DSL-based test generation pipeline described in Sections §4 and §5 of the paper.*

(C2): *Constraint-guided semantic testing with CONSET uncovers real implementation flaws in 5G UE implementations that are difficult to expose using syntax-level or schema-valid mutation alone. This claim is supported by the simulation-based evaluation results reported in Sections §6.2–§6.4 of the paper.*

A.4.2 Experiments

(E1): *Constraint extraction and semantic test generation [30 human-minutes + ∼2.5 compute-hours for full T3 inter-IE (or <10 minutes using the lightweight alternative). Total estimated time: ∼3 hours for full run, or <30 minutes using the lightweight alternative].* This experiment validates Claim (C1) by reproducing the constraint-driven analysis and test generation pipeline of CONSET.

Preparation: Set up the A2 Python environment on the host as described in the Set-up section. Note that **T1** (3GPP PDF preprocessing, requires Mathpix API key) and **T4** (LLM-based DSL synthesis, requires OpenAI API key) are **optional**. Pre-generated intermediate outputs for all stages are included in the repository and can be used to skip these steps.

Execution: All commands run from `contest_AE_final/A2_constraintdriven_toolchain/`. The recommended path is T2 → T3 → T5 → T6 (no API keys required):

T2 — IE Collection:

```
cd toolchain2_IE_collection/intra-IE/code
python3 00_extract_IE_id.py
python3 01_filter_IE_with_ASN.py
python3 02_greedy_set_cover_intra.py && cd -

cd toolchain2_IE_collection/inter-IE/code
python3 00_extract_IE_id.py
python3 01_filter_IE_with_ASN.py
python3 02_greedy_set_cover_inter.py && cd -
```

T3 — Context Extraction (intra-IE):

```
cd toolchain3_field_pair_context_extraction/intra-IE/code
python3 00_intra-IE_context_extract.py --all-pairs --no-self
cd -
```

T3 — Context Extraction (inter-IE, ∼2.5 hours): Verified on 12-core i7-1260P, 32 GB RAM, Ubuntu 22.04. Observed runtimes: 2h19m, 2h28m, 2h23m, 2h29m across four runs.

```
cd toolchain3_field_pair_context_extraction/inter-IE/code
python3 00_generate_aggressive_inter_ie_config.py
python3 01_inter_ie_enhanced_extractor.py && cd -
```

T5 — Test Case Generation:

```
cd toolchain5_dsl_to_testcase/code
python3 unified_test_generator.py && cd -
```

T6 — OTA Payload Generation:

```
cd toolchain6_generate_OTA_testcase/intra-IE/code
python3 ./run_T6.py && cd -

cd toolchain6_generate_OTA_testcase/inter-IE/code
python3 ./run_T6.py && cd -
```

If the full T3 inter-IE run is not feasible (e.g., insufficient RAM), a lightweight alternative is provided that uses the same extractor code with a reduced cluster configuration. Estimated runtime: <10 minutes:

```
cd toolchain3_field_pair_context_extraction/inter-IE/code
python3 01_inter_ie_enhanced_extractor_lite_ae.py
cd -
```

Results: The expected outputs are as follows:

- **DSL rules:** Search for `"has_valid_rule": true` in:
  `toolchain4_field_pair_LLM_query/intra-IE/outputs/intra-IE_DSL_results_gpt4o`
  `toolchain4_field_pair_LLM_query/inter-IE/output/inter_ie_dsl_rules_gpt4o`
- **Generated payloads:**
  `toolchain6_generate_OTA_testcase/intra-IE/output/06_payloads`
  `toolchain6_generate_OTA_testcase/inter-IE/output/06_payloads`

Each payload file contains **deterministic offset-based modifications** derived from the extracted semantic constraints.

(E2): *Simulation-based vulnerability reproduction on OAI UE [30 human-minutes].* This experiment validates Claim (C2) by reproducing crash-triggering behaviors on the open-source OAI UE in a controlled simulation environment.

Preparation: Set up the simulation environment locally as described in the Set-up section. Then enter the Docker container:

```
docker exec -it oai25_testing bash
cd /home/5ghoul-5g-nr-attacks
```

Execution: Step 1 — Baseline run (no exploit):

```
sudo bin/5g_fuzzer \
  --EnableSimulator=true \
  --EnableMutation=false \
  --GlobalTimeout=false
```

Expected: repeated `[!] UE process stopped` / `[!] UE process started` cycles without frequent crashes.

Step 2 — Exploit run:

```
sudo ./bin/5g_fuzzer \
  --exploit=mac_sch_6e99a9e9 \
  --MCC=001 --MNC=01 \
  --GlobalTimeout=false \
  --EnableMutation=false
```

Expected: frequent `[!] UE process crashed` output.

E1→E2 end-to-end validation: To validate that E1 outputs can be directly used in E2, copy any payload generated by E1 (T6) into the Docker container, compile, and run. The following uses `mac_sch_f108_f117` as an example (verified to reliably trigger crashes). If using the provided SSH environment, the repository is located at `/home/qiqingh/Desktop/contest_AE_final`; run the following from that directory:

```
docker cp A2_constraintdriven_toolchain/\
toolchain6_generate_OTA_testcase/inter-IE/\
output/06_payloads/mac_sch_f108_f117.cpp \
oai25_testing:/home/5ghoul-5g-nr-attacks/\
modules/exploits/5gnr_gnb/
```

Then enter the container, compile, and run:

```
docker exec -it oai25_testing bash
cd /home/5ghoul-5g-nr-attacks
sudo bin/5g_fuzzer --list-exploits
sudo ./bin/5g_fuzzer \
  --exploit=mac_sch_f108_f117 \
  --MCC=001 --MNC=01 \
  --GlobalTimeout=false \
  --EnableMutation=false
```

The following E1-generated payloads have been verified to reliably trigger UE crashes, together with their corresponding pre-compiled E2 names available under `A3_test_case_exploits/compiled_payloads/`:

E1 payload name	E2 exploit name
mac_sch_f108_f117	mac_sch_e67cf618
mac_sch_f133_f359	mac_sch_aca1c1fc
mac_sch_f452_f453	mac_sch_90ea1170
mac_sch_f690_f692	mac_sch_1a93ba9a
mac_sch_f694_f697	mac_sch_57890405
mac_sch_f823_f830	mac_sch_a243cd52
mac_sch_f825_f830	mac_sch_91c06e74
mac_sch_f831_f832	mac_sch_a86ecf35
mac_sch_f927_f933	mac_sch_b3a4a80b

Automated E1–E2 End-to-End Validation. To validate that E1-generated payloads can be directly replayed in E2, we provide an au-

tomated script `send_exploit_test.py` under `A3_test_case_exploits/auto-run/`. The script runs the 9 verified crash-triggering E1 payloads sequentially against the OAI UE in the Docker container and saves the logs for each exploit run.
To run the script in the provided SSH environment:

```
cd contest_AE_final/A3_test_case_exploits/auto-run
sudo python3 send_exploit_test.py
```

Note: The script is pre-configured for the provided SSH environment. If running in a different environment, update the following two variables at the top of `run_test()` before executing:

- `container_id`: the Docker container ID or name of the simulation environment (e.g., `oai25_testing`)
- `directory_path`: the path to the directory containing the pre-compiled `.so` payload files

Note on benign errors: The following errors may appear during execution and can be safely ignored:

- **`credentials.json` not found:** An email alert feature inherited from the upstream 5Ghoul framework fails due to missing Gmail credentials. This does not affect exploit execution. See: https://github.com/asset-group/5ghoul-5g-nr-attacks/issues/29
- **`idemptables` errors:** Legacy `iptables` setup scripts from the 5Ghoul environment. These do not affect simulation or exploit behavior.
- **Compiler warnings/errors during exploit compilation:** Running `sudo bin/5g_fuzzer -list-exploits` may output compiler warnings or errors. These do not necessarily indicate a failed build. To verify successful compilation, check whether the corresponding `.so` and `.o` files exist at `/home/5ghoul-5g-nr-attacks/modules/exploits/5gnr_gnb/`.

Results: Frequent `[!] UE process crashed` messages confirm that the exploit payload successfully triggers the vulnerability in the simulated OAI UE, consistent with the crash sites reported in Section §6.3 of the paper.

A.5 Notes on Reusability

The CONSET artifact is designed with modularity and low coupling between components, which facilitates reuse beyond the specific evaluation scenarios presented in this paper.

In particular, the constraint-driven toolchain (A2) is largely decoupled from the underlying execution environment and UE implementation. Its specification preprocessing, field-pair extraction, and DSL-based mutation logic operate on decoded protocol structures and can be reused for other RRC messages or extended to additional 3GPP control-plane protocols with minimal structural changes.

Similarly, the simulation-based test execution pipeline (A1 and A3) separates message construction from delivery mechanisms, allowing alternative backends or execution environments to be integrated without modifying the core analysis logic.

While the current artifact focuses on pre-authentication 5G RRC testing in a simulation environment, the modular design enables researchers and practitioners to adapt individual components of CONSET for related protocol testing or specification-driven analysis tasks.

A.6 Post-Submission Updates

The following details have been updated following further communication with the vendors after the camera-ready submission.

- **Table 1 (CVE assignments):** CVE-2025-20703 corresponds to the csi-MeasConfig entry and CVE-2025-20666 corresponds to the ServingCellConfig entry.
- **Table 1 (Affected IE):** The Affected IE for the third MediaTek row has been updated to ServingCellConfig.
- **Section 6.2:** The number of tested commercial smartphones has been updated to eight.

A.7 Version

Based on the LaTeX template for Artifact Evaluation V20231005. Submission, reviewing and badging methodology followed for the evaluation of this artifact can be found at https://secartifacts.github.io/usenixsec2026/.