USENIX

THE ADVANCED COMPUTING
SYSTEMS ASSOCIATION

CoreCrisis: Threat-Guided and Context-Aware Iterative Learning and Fuzzing of 5G Core Networks

Yilu Dong, Tianchang Yang, Abdullah Al Ishtiaq, Syed Md Mukit Rashid, Ali Ranjbar, Kai Tu, Tianwei Wu, Md Sultan Mahmud, and Syed Rafiul Hussain, *The Pennsylvania State University*

https://www.usenix.org/conference/usenixsecurity25/presentation/dong-yilu

This artifact appendix is included in the Artifact Appendices to the Proceedings of the 34th USENIX Security Symposium and appends to the paper of the same name that appears in the Proceedings of the 34th USENIX Security Symposium.

August 13–15, 2025 • Seattle, WA, USA

978-1-939133-52-6

Open access to the Artifact Appendices to the Proceedings of the 34th USENIX Security Symposium is sponsored by USENIX.

USENIX Security '25 Artifact Appendix: CoreCrisis: Threat-Guided and Context-Aware Iterative Learning and Fuzzing of 5G Core Networks

Yilu Dong, Tianchang Yang, Abdullah Al Ishtiaq, Syed Md Mukit Rashid, Ali Ranjbar,
Kai Tu, Tianwei Wu, Md Sultan Mahmud, Syed Rafiul Hussain
The Pennsylvania State University
{yiludong, tzy5088, abdullah.ishtiaq, szr5848, aranjbar, kjt5562, tvw5452, mqm7099, hussain1}@psu.edu

A Artifact Appendix

A.1 Abstract

We present CoreCrisis, a context-aware black-box testing framework for 5G Core Network (5GC) implementations. Considering the stateful nature of network protocols, CoreCrisis adopts a state-aware strategy to navigate the implementation effectively. Unlike previous security analysis efforts of cellular networks which rely on manually-crafted, static testing symbols and are limited to identifying only logical errors, CoreCrisis implements a dynamic two-step approach. Initially, CoreCrisis builds an initial finite state machine (FSM) representation of the 5GC's implementation using only benign symbols (expected inputs) with its novel divide-and-conquer and property-driven equivalence-checking learning. During testing, it utilizes the learned FSM to target under-explored states and introduces attacking symbols (mutated inputs). Based on the responses observed from the Core Network, CoreCrisis continuously refines the FSM to better guide its exploration. Evaluating CoreCrisis on four 5GC systems, including both commercial and open-source implementations, we identified 7 categories of deviations from the technical specifications and 13 crashing vulnerabilities. These logical and crashing vulnerabilities could lead to denial-of-service (DoS), authentication bypass, and billing fraud.

A.2 Description & Requirements

A.2.1 Security, privacy, and ethical concerns

None.

A.2.2 How to access

Our artifact is available at Zenodo: https://zenodo.org/records/15043603. We also hosted on GitHub: https://github.com/SyNSec-den/CoreCrisis.

A.2.3 Hardware dependencies

Our artifact does not have specific hardware requirements. However, if the processor is not fast enough or the system does not have enough memory, the artifact may run into issues. The experiment **E1** requires the system to have at least 32GB of memory.

A.2.4 Software dependencies

We tested the artifact on Ubuntu >= 20.04, other Linux distributions may also work.

To run the experiments, docker, JDK11, GCC, and python3 must be installed in the system.

Detailed instructions to install these dependencies are provided inside the artifact.

A.2.5 Benchmarks

None.

A.3 Set-up

A.3.1 Installation

First, the required files should be downloaded from Zenodo or GitHub. Then, follow the readme file in the main folder and each subfolder to build and run the program.

A.3.2 Basic Test

A basic functional test can be conducted by running the CoreFuzzer. By following the readme files, the program should successfully run in a docker container with terminal outputs.

A.4 Evaluation workflow

A.4.1 Major Claims

(C1): The property-driven equivalence checking reduced the number of queries needed to learn the state machine, described in Section 6.3.1 in the paper. We show this in (E1).

(C2): CORECRISIS has been used to identify new bugs in 5G core network implementations, as shown in Table 2 in the paper. This can be proven by (E2).

(C3): We compared CORECRISIS against other fuzzers, as shown in Section 6.2 in the paper. This can be proven by (E3).

A.4.2 Experiments

(E1): [Property-driven equivalence checking] [1 human-hour + 15 compute-hour]:
How to: The difference can be seen by enabling and disabling the Property-driven equivalence checking in state machine learning.
Preparation: Follow the instructions in Corelearner folder to run the program one time.
Execution: We provided the equivalence checking queries under the folder "CEStore". The evaluator should run the state machine learning with the in lines in "input" and check "learner.log" for the number of queries. Then the evaluator should repeat the steps with an empty input file and and check learner.log again.
Results: By checking the logs and the learned state machine, the number of queries should be significantly reduced if Property-driven equivalence checking is enabled.

(E2): [CoreFuzzer] [1 human-hour + 24 compute-hour]:
How to: The evaluator should follow the instructions and run the CoreFuzzer.
Preparation: Follow the instructions and build the docker containers for CoreFuzzer.
Execution: Enter the docker container and run the file "run_hourly.py". The database files used for logging the results should appear in the "logs" folder.
Results: The database file should indicate some crashes from the core network implementation.

(E3): [Comparison with other fuzzers] [1 human-hour + 24 compute-hour]:
How to: The evaluator should follow the instructions and run the CoreFuzzer along with other fuzzers provided for 24 hours to compare the coverage numbers.
Preparation: Follow the instructions and build the docker containers for each fuzzers.
Execution: Enter the docker containers and run the file "run_hourly.py". The coverage information should appear in the "logs" folder.
Results: Our implementation should outperform all the compared implementations.

A.5 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/usenixsec2025/.