

USENIX

THE ADVANCED COMPUTING SYSTEMS ASSOCIATION

A Formal Security Analysis of CAN XL

Zhaozhou Tang, *Georgia Institute of Technology;* Khaled Serag, *Qatar Computing Research Institute;* Z. Berkay Celik, *Purdue University;* Vijay Ganesh, Saman Zonouz, and Raheem Beyah, *Georgia Institute of Technology*

https://www.usenix.org/conference/usenixsecurity26/presentation/tang-zhaozhou

This artifact appendix is included in the Artifact Appendices to the Proceedings of the 35th USENIX Security Symposium and appends to the paper of the same name that appears in the Proceedings of the 35th USENIX Security Symposium.

August 12–14, 2026 • Baltimore, MD, USA

ISBN 978-1-939133-58-8

Open access to the Artifact Appendices to the Proceedings of the 35th USENIX Security Symposium is sponsored by



جامعة الملك عبدالله للعلوم والتقنية

King Abdullah University of Science and Technology



USENIX Security '26 Artifact Appendix: A Formal Security Analysis of CAN XL

Zhaozhou Tang
Georgia Institute of Technology

Khaled Serag
Qatar Computing Research Institute

Z. Berkay Celik
Purdue University

Vijay Ganesh
Georgia Institute of Technology

Saman Zonouz
Georgia Institute of Technology

Raheem Beyah
Georgia Institute of Technology

A Artifact Appendix

A.1 Abstract

We conduct a formal security analysis of CAN XL, the latest generation of the Controller Area Network (CAN) protocol. We model the protocol using the nuXmv model checker and discover several vulnerabilities in the standard. To demonstrate their exploitability, we evaluate several attacks comprising the discovered vulnerabilities on a testbed. In this artifact, we provide nuXmv models of CAN XL, scripts to run the analysis, and implementations of the evaluated attacks. Due to licensing restrictions of the proprietary software required for the attack experiments (explained in A.2.4), the reproducibility evaluation workflow in this document focuses on reproducing the formal analysis results and validating the proposed standard revisions.

A.2 Description & Requirements

A.2.1 Security, privacy, and ethical concerns

Our artifact does not perform destructive steps. Executing them does not pose security or privacy concerns to the user.

A.2.2 How to access

The artifact can be downloaded from `https://zenodo.org/records/20262062`.

A.2.3 Hardware dependencies

CAN XL Formal Analysis. The analysis can be performed on a standard PC. We used a PC with 32 GB of RAM. The compute times in A.4.2 are based on results using this PC.
Attack Experiments. Replicating them requires a Vector VN1641 CAN interface, an STM32 NUCLEO-H755ZI-Q development board, and a CAN transceiver (we used SN65HVD230). Due to difficulties explained in A.2.4, we are unable to provide access to the experiment platform.

A.2.4 Software dependencies

CAN XL Formal Analysis. We performed the analysis on a Windows 11 PC using Python 3.10.11 and nuXmv 2.1.0. The analysis can also be reproduced on a macOS or Linux machine, both of which are supported by nuXmv. On a macOS machine, libgmp needs to be installed to use nuXmv.
Attack Experiments. Running them requires a Windows PC with Vector CANoe, VN1641 driver, and STM32CubeIDE. We used CANoe 18.4.26 (SP4), VN1641 driver 24.10.10, and STM32CubeIDE 1.12.1. CANoe requires a paid and machine-bound license. Our license allows us to use CANoe only on one author's personal PC. Therefore, it is difficult to provide remote access to the experiment platform.

A.2.5 Benchmarks

None.

A.3 Set-up

A.3.1 Installation

After downloading the artifact archive from Zenodo, extract its contents. Install Python 3.10 or higher by following the instructions at `https://www.python.org/downloads/`. No additional Python packages are needed. Install nuXmv 2.1.0 by going to `https://nuxmv.fbk.eu/previous-version.html` and clicking on the download link for the appropriate OS. Extract the downloaded file. The nuXmv binary is located in the `bin` directory within the extracted folder. For a macOS machine, install libgmp with Homebrew using the command `brew install gmp`.

A.3.2 Basic Test

Navigate to any one of the following three directories: `models/error_signalling_enabled`, `models/error_signalling_disabled`, or `models/error_signalling_disabled_revision`, and run:

```
python run.py --nuxmv-dir
    "absolute_path_to_nuXmv_binary" --test
```

This analyzes the model of the arbitration phase and finds one possible ending state of the protocol when it starts executing from the idle state. If the setup is correct, a message similar to the following should be displayed:

```
Executing arbitration_phase.smv from
    (n1_state=idle, n2_state=idle) can reach
    (n1_state=bus_reintegration,
    n2_state=bus_reintegration).
```

A.4 Evaluation workflow

The workflow below evaluates the reproducibility of the artifact's formal analysis component, including vulnerability discovery and standard revisions, which covers the paper's most important contributions. The attack experiments are not included due to difficulties in providing access to the experiment platform, as explained in A.2.4. Nevertheless, the artifact repository includes the attack implementation and documentation for interested readers.

A.4.1 Major Claims

(C1): We formally model the CAN XL protocol and verify the properties described in Sec. 4.1 of the paper. We discover that CAN XL inherits all CAN CC vulnerabilities and introduces seven new standard-level vulnerabilities. The vulnerabilities are summarized in Tbl. 2 and described in detail in Sec. 5 of the paper. These results can be reproduced with experiments E1 and E2.

(C2): We propose standard revisions to CAN XL's error signalling disabled configuration and successfully verify $\mathcal{P}_2$ and $\mathcal{P}_3$ for the revised protocol. Therefore, the standard revisions fix *V2* without introducing new problems. The revisions are detailed in Sec. 8 of the paper and can be validated with experiment E3.

A.4.2 Experiments

(E1): [Analyzing error signalling disabled configuration] [5 human-minutes + 5 compute-minutes]:
How to: Navigate to the `models/error_signalling_disabled` directory. It contains the nuXmv models of CAN XL's error signalling disabled configuration, a `run.py` script to run the analysis, and counterexample traces we generated for each vulnerability (*V1*, *V2*, *V5*) in the *traces* directory. Run the command:

```
python run.py --nuxmv-dir
    "absolute_path_to_nuXmv_binary"
```

This command first computes reachable states of the protocol, then verifies the relevant properties ($\mathcal{P}_1$-$\mathcal{P}_6$), and generates counterexamples for each error signalling disabled vulnerability (*V1*, *V2*, *V5*).
Results: The script displays that verification for all properties fails except for $\mathcal{P}_2$. A new directory *counterexamples* is created. For each provided counterexample in the `trace` directory, there is a file with the same name and similar content in the *counterexamples* directory.

(E2): [Analyzing error signalling enabled configuration] [5 human-minutes + 20 compute-minutes]:
How to: Navigate to the `models/error_signalling_enabled` directory. It contains the nuXmv models of CAN XL's error signalling enabled configuration, a `run.py` script to run the analysis, and counterexample traces we generated for each vulnerability (*V4-V7*, *Va-Vc*) in the *traces* directory. Run the command:

```
python run.py --nuxmv-dir
    "absolute_path_to_nuXmv_binary"
```

This command first computes reachable states of the protocol, then verifies the relevant properties ($\mathcal{P}_1$-$\mathcal{P}_9$), and generates counterexamples for each error signalling enabled vulnerability (*V4-V7*, *Va-Vc*).
Results: The script displays that verification for all properties fails except for $\mathcal{P}_7$. A new directory *counterexamples* is created. For each provided counterexample in the `trace` directory, there is a file with the same name and similar content in the *counterexamples* directory.

(E3): [Analyzing the standard revisions] [5 human-minutes + 5 compute-minutes]:
How to: Navigate to the `models/error_signalling_disabled_revision` directory. It contains the nuXmv models of CAN XL's error signalling disabled configuration with our proposed standard revisions and a `run.py` script to run the analysis. It does not have a *traces* directory as the verification is successful and no counterexamples are generated. Run the command:

```
python run.py --nuxmv-dir
    "absolute_path_to_nuXmv_binary"
```

This command first computes reachable states of the protocol, and then verifies $\mathcal{P}_2$ and $\mathcal{P}_3$.
Results: The script displays that verification of both $\mathcal{P}_2$ and $\mathcal{P}_3$ succeeds. No counterexamples are generated and no *counterexamples* directory is created.

A.5 Notes on Reusability

Our artifact can be reused to analyze other aspects of CAN XL. New properties can be added to the nuXmv models in the `template` directory for each error signalling configuration. The property names need to be added to the `properties` field of the `model` variable at the beginning of `run.py`. The analysis can then be performed by invoking `run.py` using the same commands documented in A.4.2.

A.6 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/.