USENIX Security ’25 Artifact Appendix: From Constraints to Cracks: Constraint Semantic Inconsistencies as Vulnerability Beacons for Embedded Systems

Jiaxu Zhao[1,2,3,4], Yuekang Li[5], Yanyan Zou[1,2,3,4]✉, Yang Xiao[1,2,3,4], Naijia Jiang[1,2,3,4]
Yeting Li[1,2,3,4], Nanyu Zhong[1,2,3,4], Bingwei Peng[1,2,3,4], Kunpeng Jian[1,2,3,4], Wei Huo[1,2,3,4]✉
[1]Institute of Information Engineering, Chinese Academy of Sciences, China
[2]School of Cyber Security, University of Chinese Academy of Sciences, China
[3]Key Laboratory of Network Assessment Technology, Chinese Academy of Sciences, China
[4]Beijing Key Laboratory of Network Security and Protection Technology, China
[5]University of New South Wales, Australia

A Artifact Appendix

A.1 Abstract

This artifact presents Nüwa, a novel static analysis technique that leverages constraint semantic inconsistencies to detect vulnerabilities in embedded systems. Nüwa achieves scalable and precise vulnerability discovery by addressing the challenges of identifying constraint semantics across diverse implementations and accurately extracting them. We implemented Nüwa and evaluated it using known vulnerability datasets, including 31 vulnerabilities from 13 vendors, and compared its performance to five state-of-the-art (SOTA) tools.

A.2 Description & Requirements

A.2.1 Security, privacy, and ethical concerns

Nüwa is designed for discovering vulnerabilities in embedded firmware. It does not pose any direct risk to the evaluator’s system security, data privacy, or other ethical concerns during its execution. Nüwa operates on firmware images in an offline manner and does not require network access or interaction with live systems. Moreover, Nüwa does not disable any system security mechanisms nor perform any destructive operations by default.

As Nüwa is open-sourced, it is technically possible for users to discover real-world vulnerabilities. However, Nüwa is released strictly for academic research purposes. We strongly encourage that any vulnerabilities discovered using the tool be reported in accordance with responsible disclosure practices, as outlined in Section 4.5 of our paper, to help protect the broader Internet ecosystem.

✉Corresponding Authors.

A.2.2 How to access

The source code necessary to evaluate Nüwa, along with the dataset used in our paper and the Python virtual environment required to run the tool, is available in our archived Zenodo repository at: https://doi.org/10.5281/zenodo.15605329. This DOI provides a stable and citable reference to the evaluated version of the artifact, as required for the Artifacts Available badge.

A.2.3 Hardware dependencies

Our evaluation of Nüwa was conducted on a host machine with a 28-core Intel Xeon processor and 256 GB of RAM running Ubuntu 22.04. However, Nüwa does not require any specific hardware features and can run on any Linux-based operating system. No specialized hardware (e.g., GPU, FPGA, or hardware counters) is needed. Higher-end configurations may improve performance, but they are not mandatory for functionality. We recommend at least 8 CPU cores and 32 GB of RAM for smooth execution and reasonable analysis time.

Nüwa can also be executed within a virtual machine environment and does not require root privileges for installation or execution.

A.2.4 Software dependencies

To evaluate Nüwa, the most critical requirement is the installation of IDA Pro (version 9.0 or 9.1) with full decompiler plugin support. Since IDA Pro is a commercial reverse engineering software, we are unable to provide a pre-configured environment containing it. Evaluators must obtain a valid license and install the software independently.

In addition, NÜWA was evaluated using Python version 3.10.12, and we strongly recommend using the same version to ensure compatibility and reproducibility.

A.2.5 Benchmarks

We used two datasets to evaluate NÜWA. The first is a known-vulnerability dataset, included in the vuln_dataset folder, which consists of firmware samples containing vulnerabilities with publicly available ground truth. The second is a firmware dataset used for discovering previously unknown vulnerabilities, included in the firmware_dataset folder. Due to ongoing coordination with vendors and responsible disclosure processes, only a subset of the firmware images in this dataset can be made publicly available at this time.

A.3 Set-up

A.3.1 Installation

As described in Section A.2.4, to download and install dependencies as well as the main artifact, you only need to install IDA Pro (version 9.0 or 9.1) and Python (version 3.10.12 is recommended).

Then, follow the instructions provided in our Zenodo repository to configure the Python virtual environment.

Or you can use NÜWA in the Docker provided in our Zenodo repository, which only you need to install IDA Pro and configure IDALib.

A.3.2 Basic Test

- Modify 104 lines of code in main.py. Change *for todo_item in binary_todo* to "*for todo_item in binary_todo1*"

- Enter the known vulnerability folder defined in *binary_todo1*

- Run main.py in the python virtual environment.

- In the known vulnerability folder defined by *binary_todo1*, there are the outputs after firmware unpackaging, the front-end source identification output (front folder), the front-end constraint analysis results (frontend.log), and the back-end constraint analysis results (backend.log).

- In the back-end analysis results, the function summary analysis results, slicing results, constraint extraction results, and semantic inconsistencies results between backend explicit and desired constraints are listed in detail.

A.4 Evaluation workflow

A.4.1 Major Claims

(C1): NÜWA can detect 28 known vulnerabilities and 12 additional known vulnerabilities. This is proven by the experiment E1.

(C2): The constraint extraction and constraint semantic inconsistencies results are proven by the experiment E2.

A.4.2 Experiments

(E1): *[Known Vulnerabilities Detection] [5-10 compute-hour + 80GB disk]*
Execution: Run main.py in the python virtual environment.
Results: The analysis result is in each vulnerability folder. The vulnerability detection result need to be confirmed manually.

(E2): *[Constraint Analysis] [12 human-hour]*
How to: Manually analyse the result file: frontend.log and backend.log. Among them, the semantic inconsistency between the back-end explicit constraint and the desired constraint has been automatically analysed in backend.log, while the semantic inconsistency between the front-end and back-end explicit constraints requires manual analysis of the display constraints in frontend.log and backend.log.

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/.