USENIX Security '24 Artifact Appendix: A Friend's Eye is A Good Mirror: Synthesizing MCU Peripheral Models from Peripheral Drivers

Chongqing Lei[†], Zhen Ling[†,*], Yue Zhang[‡], Yan Yang[†], Junzhou Luo[†], Xinwen Fu[§]
[†] Southeast University, Email: {leicq, zhenling, yanyang, jluo}@seu.edu.cn
[‡] Drexel University, Email: zyueinfosec@gmail.com
[§] University of Massachusetts Lowell, Email: xinwen_fu@uml.edu

A Artifact Appendix

A.1 Abstract

In this artifact, we provide the source code of PERRY , as well as the required materials to replicate our experiments described in the paper. This document describes how to reproduce all results described in §5.2 and §6 of our paper.

A.2 Description & Requirements

A.2.1 Security, privacy, and ethical concerns

Most of the experiments are conducted within a Docker container and will not harm the computer. However, a reproducer has to execute several commands on the host machine during the experiment. Specifically, the producer has to enable or disable Bluetooth device on the host machine in **E4**, and prepares the machine for fuzzing on the host machine in **E5**. These commands are widely used and will not harm the computer.

A.2.2 How to access

The code, as well as the experiment materials are available on GitHub: https://github.com/VoodooChild99/perry/tree/sec24-ae-accepted.

A.2.3 Hardware dependencies

- **Processor**: We recommend using a machine with two Intel Xeon E5-2620 v2 CPUs (12 cores, 24 threads) to reproduce the experiment. However, comparable hardware may also suffice.
- **Memory**: At least 64GB of RAM.
- **Storage**: At least 256GB.
- **Bluetooth**: The machine must be equipped with a Bluetooth device to replicate **E4**.

*Corresponding author: Prof. Zhen Ling of Southeast University, China.

A.2.4 Software dependencies

Ubuntu 20.04, git and Docker.

A.2.5 Benchmarks

We utilize P2IM unit tests for consistency experiments, and P2IM real-world firmware samples for universality experiments. Additionally, we collected various drivers and custom firmware samples for the experiments described in §5.2 and §6.2. These materials are all available and are referenced in the "Building With Docker" section of the repository's `README` file.

A.3 Set-up

A.3.1 Installation

After cloning or downloading the repository, please refer to the "Building With Docker" section of the `README` file to build the Docker image. In case you cannot build the image, we also provide a pre-built Docker image, please refer to the "Using the Pre-Built Docker Image" section of the `README` file for detailed instructions.

A.3.2 Basic Test

If the Docker image is built or downloaded successfully, you can run `./run_docker.sh` in the repository, which will spawn a shell within the container on success. Note that the container is still running after exiting from the shell, and you can always spawn a new shell by running `./run_docker.sh` later.

A.4 Evaluation workflow

A.4.1 Major Claims

(C1): PERRY is efficient in synthesizing hardware models (§5.2 "Efficiency" and Figure 3).

(C2): PERRY can generate hardware models consistent with actual hardware behaviors (§5.2 "Consistency" and Table 3).
(C3): Hardware models generated by PERRY can be used to emulate various firmware (§5.2 "Universality" and Table 5).
(C4): Hardware models generated by PERRY can be easily fixed or extended to support missing hardware functionalities (§5.2 "Scalability" and Tables 3, 5).
(C5): PERRY can be used to find specification violation bugs (§6.1).
(C6): PERRY can be used to reproduce firmware vulnerabilities (§6.2 "CS-II").
(C7): PERRY can be used to fuzz RTOS (§6.2 "CS-III").

A.4.2 Experiments

Please refer to the "Replicating Our Experiments" section of the `README` file in the repository for detailed steps to reproduce our experiments.

We design five experiments (**E1-E5**) to confirm **C1-C4** and **C6-C7**. We provide details about the specification violation bugs we found under `perry-experiments/spec-violation-bugs.md` to confirm **C5**.

(E1): *[Efficiency] [5 human-minutes + 100 compute-hours]:* PERRY 's model synthesis efficiency is evaluated by the consumed time. You may find the results in `perry-experiments/01-efficiency/result.csv`. **C1** is confirmed if the acquired results are comparable with Figure 3.
(E2): *[Consistency] [10 human-minutes + 30 compute-minutes]:* PERRY 's consistency is evaluated by the passing rate on P2IM unit tests. PERRY should be able to pass 49 unit tests initially, and all unit tests after fixing models, which is consistent with Table 3 and confirms **C2**. We prepend the comment `// PERRY PATCH` before every fix to help identify them, you can crosscheck these fixes with Table 3 to confirm **C4**.
(E3): *[Universality] [20 human-minutes + 20 compute-minutes]:* PERRY 's universality is evaluated by executing various firmware samples. PERRY should be able to execute all of them without crashes and hangs as shown in Table 5, which confirms **C3**. We prepend the comment `// PERRY PATCH` before every fix to help identify them, you can crosscheck these fixes with Table 5 to confirm **C4**.
(E4): *[CVE Reproducing] [1 human-hour + 1 compute-hours]:* We use the STM32F407 hardware model generated by PERRY to emulate `Zephyr`, and reproduce two CVEs. **C6** can be confirmed if the two CVEs can be successfully reproduced.
(E5): *[LiteOS Fuzzing] [5 human-minutes + 66 compute-hours]:* The script will try to utilize all CPU cores and fuzz each target for 6 hours. If your machine has more than 11 cores, the experiment only takes around 6 compute-hours. **C7** is confirmed if the results are comparable with Table 6.

A.5 Notes on Reusability

PERRY is designed to generate MCU hardware models from driver code. To support a new MCU, a user needs to compile the driver code with the provided Clang plugin and compiler wrapper, and write a configuration file for the MCU. We have included such information in the "Supporting New MCUs" section of the `README` file in the repository.

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/usenixsec2024/.