RapidPatch: Firmware Hotpatching for Real-Time Embedded Devices

Yi He and Zhenhua Zou, *Tsinghua University and BNRist;* Kun Sun, *George Mason University;* Zhuotao Liu and Ke Xu, *Tsinghua University and BNRist;* Qian Wang, *Wuhan University;* Chao Shen, *Xi'an Jiaotong University;* Zhi Wang, *Florida State University;* Qi Li, *Tsinghua University and BNRist*

https://www.usenix.org/conference/usenixsecurity22/presentation/he-yi

This artifact appendix is included in the Artifact Appendices to the Proceedings of the 31st USENIX Security Symposium and appends to the paper of the same name that appears in the Proceedings of the 31st USENIX Security Symposium.

August 10–12, 2022 • Boston, MA, USA

978-1-939133-31-1

ARTIFACT EVALUATED usenix ASSOCIATION AVAILABLE

ARTIFACT EVALUATED usenix ASSOCIATION FUNCTIONAL

A Artifact Appendix

A.1 Abstract

This artifact contains the source code of RapidPatch *and the stuff for running it. Since* RapidPatch *is designed for hotpatching embedded devices, to evaluate the basic functions, you need to have a Cortex-M3/M4 based arm development board. If you do not have these devices, we also provide a simple version that can run on qemu, and can demonstrate the functionable of RapidPatch by running the hotpatching process using fixed patch points (only one of the three hotpatching strategies supported by our tool). To fully evaluate and reproduce the results, you need to have at least one of these STM32F407/STM32L475/STM32F429/NRF52840 developing boards. Note that you can use any of the MacOS/Windows/Linux Platform to develop or evaluate it, we provide Docker and PlatformIO-based VSCode cross-platform building environments.*

A.2 Artifact check-list (meta-information)

- **Binary: Pre-build RapidPatch firmware for different devices (you can also build from scratch).**
- **Hardware: Qemu and real devices, such as, STM32F429/NRF52840/STM32L475 and ESP32 developing boards.**
- **How much time is needed to prepare workflow (approximately)?: 3h**
- **Publicly available?: Yes**
- **Code licenses (if publicly available)?: GPL v3.0**
- **Archived (provide DOI or stable reference)?: Yes**

A.3 Description

A.3.1 How to access

All the documents and source code are available on github. *https://github.com/IoTAccessControl/RapidPatch/tree/ae-v1.0.*
(Commit: 591f82e5cf4f91cfa440bb376cb4975ce78ce871)

A.3.2 Hardware dependencies

RapidPatch *relies on the Debug Monitor Handler of Cortex-M3+ MCU to dynamically trigger patches without modifying the Flash ROM. The recommended devices are NRF52840, STM32F429, or STM32L475. You can also port* RapidPatch *to other devices with Cortex-M3/M4 MCUs via PlatformIO.*
Note that for devices other than Cortex-M3+, you can only use compiling time patch points placement.

A.3.3 Software dependencies

To compile the source code from scratch, you need to install the following software.

- Docker (manually)
- gcc-arm-none-eabi (installed by Docker)
- qemu-system-arm (installed by Docker)
- VSCode and the PlatformIO plugin (manually)
- Keil (optional)

If you do not have any required hardware and just want to quickly try it, we provide Docker scripts with a push-button to run the core functionalities of RapidPatch *on any platform that supports Docker. In this case, you do not need to install any aforementioned software.*

A.4 Installation

To run on Docker, you can use our docker images or build from the Dockerfile. The detail steps is shown in docker-qemu.md document.

To try RapidPatch on real devices, you can build and flash these projects with the Keil project or Platform-IO projects or just use the pre-build firmware.

A.5 Experiment workflow

You can follow the HOWTO.md document to test the functions of RapidPatch*. There detailed steps of deploying a patch is as follows.*

1. Integrate the RapidPatch *Runtime to the firmware of your devices.*

2. Write a patch based on the origin C source code patch.

3. Generate the eBPF bytecode via the RapidPatch *Toolchain's patch generator.*

```
python3 main.py gen test_cve1.c \
    test_cve1.bin
```

4. Verify the eBPF bytecode via the RapidPatch *Toolchain's patch verifier. Note that, for the filter patch, the verifier can automatically insert the SFI instructions for loops.*

```
python3 main.py verify test_cve1.bin
```

5. Deploy the patch to real devices with our Usart tool or directly paste the patches' bytecode to your firmware code.

```
python3 main.py monitor COM15
> install test_cve1.json
```

6. Test the patch functions with the Usart commend line interface.

A.6 Evaluation and expected results

After setting up the firmware, you can use a serial port tool (e.g., CoolTerm) to connect to the devices and trigger commends to conduct the evaluation. To preform the micro-evaluation, you need to use the Usart shell commend (e.g., run exp_idx) to execute the corresponding experiments.

The results of micro-benchmark is output to the Usart shell message and contains the execution time and CPU cycles.

```
Event 0 -> cycle: 38 time(us): 0.475000
```

To evaluate the macro-benchmark, you can use the the pre-built Zephyr Apps and the test tools to measure the performances. The results are written to local files.