ACM CCS'23 Artifact Appendix: SHERLOC: Secure and Holistic Control-Flow Violation Detection on Embedded Systems

Xi Tan
CactiLab, University at Buffalo

Ziming Zhao
CactiLab, University at Buffalo

1 Artifact Appendix

1.1 Abstract

This artifact includes the source code and documentations of the ACM CCS'23 SHERLOC paper. The artifact demonstrates SHERLOC's effectiveness of control-flow violation detection on embedded systems considering asynchronous interrupts and context switches using an interrupt- and scheduling-aware violation detection algorithm. The GitHub repository contains data and scripts for setting up the environment and evaluating the SHERLOC prototype. The experiments run on the MPS2+ FPGA prototyping board. To facilitate reproduction, the repository also includes pre-compiled benchmark applications and firmware.

1.2 Description & Requirements

1.2.1 Security, privacy, and ethical concerns

There are no ethical concerns.

1.2.2 How to access

Source code and documentations can be accessed at https://github.com/CactiLab/Sherloc-Cortex-M-CFVD.

1.2.3 Hardware dependencies

The experiments run on the MPS2+ FPGA prototyping board configured with the AN505 FPGA image. The board must be connected to a computer using a UART connector, debugger connector, and power supply (Figure 1).

1.2.4 Software dependencies

The experiments were evaluated on Windows 11 operating system. Python3, Putty, and Jupyter Notebook are recommended. The pre-built benchmarks are placed at Example/out/eval/O3/elf_ns and Example/out/eval/oz/elf_ns. A pre-built SHERLOC runtime is placed at host_tools/evaluation/elf_s.

1.2.5 Benchmarks

The repository includes five benchmarks:

(1) Non-interrupt bare-metal projects built with the BEEBS benchmark suite. The projects are at Example/Sherloc_S_NS/Sherloc_ns, and the BEEBS source code is in the Sherloc_runtime/evaluation folder;

(2) Interrupt-aware bare-metal project: Blinky. The project and its source code are available at Example/Sherloc_Blinky_S_NS/Sherloc_Blinky_ns;

(3) Interrupt- and scheduling-aware benchmark: FreeRTOS. The project is located at Example/Sherloc_FreeRTOS_MPU_S_NS/FreeRTOS_MPU_ns, and the FreeRTOS source code is at Sherloc_runtime/freertos;

(4) A customized trigger-based project based on FreeRTOS is available at Trigger_S_NS/FreeRTOS_MPU_ns;

(5) Customized vulnerable projects are located at Vulfoo_S_NS/FreeRTOS_MPU_ns (buffer overflow) and Vulfoo_Task_S_NS/FreeRTOS_MPU_ns (malicious task rescheduling).

1.3 Set-up

1.3.1 Installation

To get started, please follow the steps below:

1. Clone the project.

```
$ git clone --branch v1.0-ccs2023
↪  https://github.com/CactiLab/Sherloc-Cortex-M-CFVD.git
```

2. Install additional dependencies.

```
$ cd ./host_tools/evaluation/
$ pip install -r requirements.txt
```

3. Connect and backup the board.

   - Connect the board to the computer (Figure 1).
   - Locate the file system of the board on your system. Usually, the drive name is `V2M_MPS2`.
   - Assign the `E` letter to the `V2M_MPS2` drive.
   - Backup the content of `V2M_MPS2` drive.

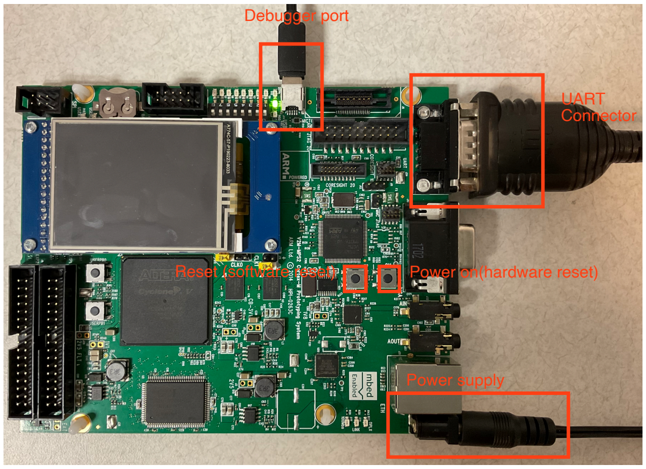

Figure 1: Board connection.

4. Configure the FPGA image and loading files for MPS2+ board, referring to "Using the Cortex-M33 IoT Kit Image on MPS2+" step 7 - 11.

    4.1. Download Cortex-M33 IoT Kit FPGA image from ARM website.

    * Unzip the download file, you will find *Cortex-M33_IoT_Kit_2_0/boards/Recovery* folder.
    * Copy the whole *Recovery* folder to the `V2M_MPS2` drive. Replace *Recovery/MB/HBI0263C/AN505/image.txt* with *Sherloc-Cortex-M-CFVD/Sherloc_runtime/image.txt*.

    4.2. If you cannot download the FPGA image from ARM website, you can download it from: Google Drive Link.

    * Unzip the downloaded file named *Cortex-M33_IoT_kit.zip*.
    * Copy the contents of *Cortex-M33_IoT_kit* directory to the `V2M_MPS2` drive.

5. Connect to the board's serial port using Putty. Set the baud rate to 115200. You can identify the serial port by checking the system's *device manager*.

1.3.2 Basic test

Run:

```
$ cd ./host_tools/evaluation/ae
$ python basic-test.py
```

Press the `Reset` button on the board. The output should resemble the following. If no such output appears, please check the `eval.log` file in the `ae` folder.

```
...
NONE: 11720703
NONE: 11720703
NONE: 11720703
NONE: 11720703
NONE: 11720703
F
```

1.4 Evaluation workflow

The experiments evaluate SHERLOC's effectivenss of validating the control-flow of various systems, including non-interrupt bare-metal systems, interrupt-aware bare-metal systems, interrupt- and scheduling-aware RTOS, and trigger-based RTOS. The test scripts are available in the folder host_tools/evaluation/ae.

1.4.1 Major claims

(C1) SHERLOC can provide interrupt- and scheduling-aware control-flow violation detection (CFVD) of embedded systems. The experiments (E1), (E2), and (E3) described in Section 4.4 of the paper demonstrate the effectiveness.

(C2) SHERLOC can provide trigger-based CFVD. The experiment (E4) described in Section 4.5 of the paper demonstrates the effectiveness.

(C3) SHERLOC can detect control-flow hijacking, such as buffer overflow attacks and malicious task rescheduling in RTOS. The experiments (E5) and (E6) described in section 5.3.2 of the paper demonstrate the effectiveness.

1.4.2 Experiments

(E1): *[Non-interrupt CFVD] [2 human-minutes + 1 compute-minute]:*
Execution: Run: $ python c01-non-interrupt.py. After that, power on the board and then reset it. The power on operation ensures that the board reloads the system.
Results: The Putty window will show like:

```
EVAL: 26476886, enter_exit: 11785059,
 ↪ sherloc_detection: 14691827
EVAL: 26493499, enter_exit: 11785133,
 ↪ sherloc_detection: 14708366
EVAL: 26493303, enter_exit: 11785135,
 ↪ sherloc_detection: 14708168
EVAL: 26493466, enter_exit: 11785133,
 ↪ sherloc_detection: 14708333
EVAL: 26476662, enter_exit: 11785062,
 ↪ sherloc_detection: 14691600
F
```

(E2): *[Interrupt-aware CFVD] [2 human-minutes + 1 compute-minute]:*
Execution: Run: $ python c01-interrupt-aware.py. After that, power on the board and then reset it.
Results: The Putty window will show like:

```
EVAL: 173604084, enter_exit: 77863659,
↪ sherloc_detection: 95740425&
```

(E3): *[Interrupt- and scheduling-aware CFVD] [2 human-minutes + 1 compute-minute]:*
Execution: Run: $ python c01-rtos.py. After that, power on the board and then reset it.
Results: The Putty window will show like:

```
EVAL: 10504233, enter_exit: 873838,
↪ sherloc_detection: 9630395&
```

(E4): *[Trigger-based CFVD] [2 human-minutes + 1 compute-minute]:*
Execution: Run: $ python c02-rtos-trigger.py. After that, power on the board and then reset it.
Results: The Putty window will show like:

```
Trigger: 3977354, enter_exit: 3972225
&Trigger: 4007474, enter_exit: 3997500
&Trigger: 4016280, enter_exit: 3997503
&Trigger: 3990400, enter_exit: 3972228
&Trigger: 4012910, enter_exit: 3997497
```

(E5): *[Buffer-overflow detection] [2 human-minutes + 1 compute-minute]:*
Execution: Run: $ python c03-vulfoo.py. After that, power on the board and then reset it.
Results: The Putty window will show like:

```
You ha[0]!!! illegal indirect call: 0x1f542020
Check stack. top: 1.
0x00201e5c
0x002025c8
Check task list. num: 0.
```

(E6): *[Malicious task rescheduling detection] [2 human-minutes + 1 compute-minute]:*
Execution: Run: $ python c03-vulfoo-task.py. After that, power on the board and then reset it.
Results: The Putty window will show like:

```
[0] Wrong IRQ exit: 0x00201f00, 0xfffffffbc,
↪ 0xffffffbd, 0x0020264c
Check stack. top: 0.
0x00202588
Check task list. num: 6.
0x00201e3c
0x0020160a
0x002026a8
0x0020264a
0x00202676
0x002026a8
```

1.5 Notes on Reusability

The claims focus on the functionality of SHERLOC, not its performance. So we narrowed down the test cases. To replicate the performance evaluation results from Figures 8 and 9 in our paper, we recommend to connect the board using two USB2TTL adapters, two logic analyzer clippers, two male and female jumper wires. Due to the page limit, please view our connection setup on GitHub for more information.

After finishing the board connection, run the host_tools/evaluation/eval_run.py to automatically evaluate all benchmarks. Since we provide pre-built benchmarks, this script will skip the project building and metadata generation steps. If you wish to rebuild them from scratch, ensure you have the licensed Keil IDE installed and uncomment all *xx_prepare_run()* and *xx_emu_run()* functions within the *xx_eval_run_all()* function in *eval_run.py* (e.g., beebs_eval_run_all()).

We also provide raw evaluation results at Example/out/eval/o3/eval_log and Example/out/eval/oz/eval_log folders. To check them, run

```
$ cd ./host_tools/evaluation
$ python result.py > result.log
```

The `result.log` file displays final performances and individual sub-step contributions. Numbers may vary from the paper due to compiler version differences.

```
processing beebs eval results...
bubblesort O3
result_dict:  {'SHERLOC': 14772763, 'enter_exit':
↪ 64430, 'read': 12362, 'ins_identification':
↪ 11151736, 'forward': 38861, 'ss': 3505374}
eval_overhead:  {'SHERLOC': 126.04, 'enter_exit':
↪ 0.55, 'read': 0.11, 'ins_identification': 95.15,
↪ 'forward': 0.33, 'ss': 29.91}
...
processing blinky eval results...
...
```

To keep the pre-built benchmarks, but re-generate evaluation results, simply delete *Example/out/eval/[o3/oz]/eval_log* folder. To build benchmarks from scratch, please delete the whole *Example/out/eval* folder.

1.6 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/acmccs2023/`.