ARTIFACT EVALUATED usenix ASSOCIATION AVAILABLE

ARTIFACT EVALUATED usenix ASSOCIATION FUNCTIONAL

Artifact Appendix
RIPencapsulation: Defeating IP Encapsulation on TI MSP devices

Prakhar Sah
Virginia Tech
sprakhar@vt.edu

Matthew Hicks
Virginia Tech
mdhicks2@vt.edu

A Artifact Appendix

A.1 Abstract

This artifact appendix describes how to use the toolchain and reproduce functionality results for RIPencapsulation. The artifact contains source code for RIPencapsulation and several benchmarks and is publicly available as a GitHub repository. This artifact does not produce performance data for RIPencapsulation; it is intended to be used to demonstrate basic functionality—exfiltration of IP Encapsulated firmware—on real hardware.

A.2 Description & Requirements

A.2.1 Security, privacy, and ethical concerns

This artifact does not exploit any security breaches on evaluators' machines. The malicious code is run on the MSP430 target evaluation board and is not destructive to evaluators' environments. The only code running on the evaluators' machines are the debugger, post-processing and automation scripts.

A.2.2 How to access

RIPencapsulation is publicly available through its git repository at `https://github.com/FoRTE-Research/RIPencapsulation`. The repository contains RIPencapsulation benchmark source code, automation/post-processing scripts, as well as some prebuilt binaries to facilitate testing.

A.2.3 Hardware dependencies

RIPencapsulation was developed and tested on the MSP430FR5994 (end-to-end attack) and MSP432P401R (proof-of-concept) Launchpads.

A.2.4 Software dependencies

- msp430-gcc is the Texas Instruments (TI) open source toolchain for MSP430 microcontrollers. It is necessary to generate the benchmark binaries.
- mspdebug is a free, open source debugger for MSP430 MCUs. It is used to flash benchmark binaries on the MSP430FR5994 launchpad.
- CCStudio is TI's integrated development environment for debugging TI's microcontrollers and processors. It is used for Debbuger Server Scripting (to erase IP encapsulated memory on MSP430FR5994) and for testing the benchmarks on the MSP432P401R launchpad.

A.2.5 Benchmarks

All data required for the evaluation is part of the artifact. We provide Texas Instruments and custom-developed benchmarks. Some pre-compiled binaries for the MSP430FR5994 are included.

A.3 Set-up

A.3.1 Installation

RIPencapsulation: Clone the repository from `https://github.com/FoRTE-Research/RIPencapsulation` and follow the instructions provided in the README.md.

msp430-gcc: We do not provide msp430-gcc as part of the artifact evaluation repository; install it by following the instructions at `https://www.ti.com/tool/MSP430-GCC-OPENSOURCE`.

mspdebug: A detailed description of the mspdebug installation process is available in README.md of the github repository.

CCStudio: Install it by following the instructions at `https://www.ti.com/tool/CCSTUDIO`.

A.3.2 Basic Test

To check whether mspdebug is running and the target device is connected correctly, use the command: `mspdebug tilib`. Another important thing to verify is that the `dss.sh` script is running correctly. To test this, run the following commands: `cd ~/path/to/your/project/dir/eval/javascript/` `dss.sh remove_ipe.js` This will trigger some warnings which you can ignore, but if the log says that `dss.sh ~/..` fails because `dss.sh` could not be found, then you need to add the `dss.sh` script to the path using the instructions given on the GitHub repository.

A.4 Evaluation workflow

A.4.1 Major Claims

(C1): The evaluation goal of the artifact is to exfiltrate the IPE memory and demonstrate the end-to-end attack on multiple security-critical libraries compiled at different optimization levels.

RIPencapsulation is an exploit that illustrates the vulnerabilities present in the current implementation of IP Encapsulation (IPE) security feature offered on Texas Instruments MSP microcontrollers. It consists of three phases—1) CPU register state dumps, 2) reverse-engineering, and 3)firmware exfiltration—explained in more detail in the paper. The `/eval/python/RIPencapsulation.py` script automates the three phases of the attack by doing the following:

- flash the benchmark (IPE protected) along with the malicious code on the MSP430FR5994
- collect the CPU register state dump on the workstation
- reverse-engineer the register state dump and guess the IPE instructions
- perform more tests on the indirect load guesses to find an IOP gadget
- reflash the device with the new malicious code
- collect the exfiltrated IPE memory dump

Besides demonstrating the three-phase attack on the MSP430FR5994, the artifact also contains additional reverse-engineering scripts, old evaluation results as well proof-of-concept code for the MSP430P401R, in the interest of facilitating future research on the topic.

A.4.2 Experiments

To verify the functionality of the end-to-end attack in exfiltrating the protected IPE firmware on the MSP430FR5994 launchpad, we recommend following this example workflow. We have also provided some pre-built binaries for the benchmarks, compiled at the -O0 optimization level. If you wish to use these binaries directly, ignore step 1.

1. In `msp430/Makefile`, uncomment the `BENCHMARK` and `OPTFLAG` you wish to evaluate. Build the binary using the following commands on the terminal:
```
> cd ~/path/to/your/project/dir/msp430/
> make clean
> make
```
2. In `eval/python/RIPencapsulation.py`, uncomment the `benchmark` you wish to evaluate.
3. On the terminal, change the working directory to `eval/python/` by running the command:
```
> cd ~/path/to/your/project/dir/eval/python/
```
4. Connect the MSP430FR5994 launchpad (via the USB debug cable) to your workstation and run the python script `RIPencapsulation.py` by entering the command on the terminal:
```
> python3 RIPencapsulation.py
```
5. Give the script some time to complete execution. If the script fails in the middle (due to serial communication error), press `CTRL+C` to break out of the script and then rerun it. In some cases, it might be required to hard reset the device (by unplugging the launchpad and plugging it back in) before a rerun.
6. If the script finishes execution displaying "Unsuccessful: Could not find any indirect load instructions" this means that the number of register dumps taken was insufficient for finding any indirect load instructions. This can be solved by changing the `reg_dump_size` and `reg_dump_sleep_time` variables to bigger values. The current values are set to work for the pre-built binaries. Note that the bigger the values, the more time it will take to exfiltrate the IPE memory.
7. If the script finishes execution displaying "Exfiltrating IPE firmware..." give it some time to exfiltrate the IPE memory to an xlsx file ($\sim$120sec). The exfiltrated dumps can be found in the `eval/dumps/` directory. The exfiltrated IPE firmware can be found in the file `eval/dumps/exfiltrated_firmware.xlsx`, where the first column is the IPE address location and the second column is the memory value at that location.

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/usenixsec2024/.