In-Kernel Control-Flow Integrity on Commodity OSes using ARM Pointer Authentication

Sungbae Yoo, Jinbum Park, Seolheui Kim, and Yeji Kim, *Samsung Research;*
Taesoo Kim, *Samsung Research and Georgia Institute of Technology*

https://www.usenix.org/conference/usenixsecurity22/presentation/yoo

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

Open access to the Artifact Appendices to the Proceedings of the 31st USENIX Security Symposium is sponsored by USENIX.

A Artifact Appendix

A.1 Abstract

Our artifact provides code and binaries and scripts to reproduce experimental results in the paper. As for benchmarks, we demonstrated experimental results on two real machines (Mac mini based on M1 chip, and rpi3), but our artifact has been prepared to run only on Mac mini, since rpi3 lacks a support of ARM Pointer Authentication (PA). Other results than benchmarks can be reproducible on host PC machines.

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

- **Algorithm:** compiler instrumentation, pointer analysis
- **Program:** Linux kernel, LLVM plugin, GCC plugin, python scripts (all sources and binaries included)
- **Compilation:** LLVM 9.0, Modified GCC 7.3 (binaries and sources included)
- **Transformations:** Security check insertion implemented as a GCC plugin.
- **Binary:** Pre-built root file system, kernel images with various configurations.
- **Run-time environment:** Ubuntu 18.04 or 20.04. Host environment, not virtual environment, is recommended.
- **Hardware:** Mac mini with M1 chip
- **Security, privacy, and ethical concerns:** Some of shared codes are subject to intellectual property. Please do not redistribute them.
- **Output:** static analysis results of context analyzer and static validator, benchmark results on Mac mini.
- **How much disk space required (approximately)?:** The artifact repository takes up around 5GB.
- **How much time is needed to prepare workflow (approximately)?:** It takes about 1-2 hours to prepare.
- **How much time is needed to complete experiments (approximately)?:** It takes about 1 hour to complete.
- **Publicly available (explicitly provide evolving version reference)?:** Some codes, which have no issue of intellectual property, will be available at https://github.com/SamsungLabs/PALinux/ soon.

A.3 Description

A.3.1 How to access

You can access all materials for the artifact evaluation through a repository in the Bitbucket: https://bitbucket.org/jinb-park/pal-ae/. Note that this is a private repo because we cannot open the source code to the public yet due to an issue of intellectual property. Reviewers can access this repo by using the SSH key posted on "Artifact access" section in the hotcrp submission site.

A.3.2 Hardware dependencies

It is required to have a physical access to a mac mini built on M1 chip for reproducing benchmarks. Also, it requires a USB-to-C cable and a HDMI cable for connection between the mac mini and host PC.

A.3.3 Software dependencies

We have confirmed this artifact on Ubuntu 18.04/20.04 host machines, not virtual guest machines. We also tried our artifact on virtual guests but found that some of experiments can go wrong. So we recommend running this on host machines if possible.

A.4 Installation

First of all, copy the SSH key content into a file (e.g., pal_rsa), and then type the following command:

```
$ chmod 600 pal_rsa
$ GIT_SSH_COMMAND='ssh -i pal_rsa -o
    IdentitiesOnly=yes' git clone git@bitbucket.
    org:jinb-park/pal-ae.git
```

Next, follow the guide, README.md, in the repository.

A.5 Experiment workflow

Our experiment workflow is twofold.
First, for functional evaluation, run the context analyzer to get a CFI precision report and a guide for dynamic contexts; then build linux kernel along with the guide; lastly, run the static validator on the built kernel binary to find out insecure uses of PA (Pointer Authentication) instructions. See "full-workflow/README.md" to get to know all instructions needed for it.
Second, for reproducing key results, we put appropriate prebuilt files as well as a README file that contains required instructions in each directory in the repository. (analyzer/, precision/, static-validator/, benchmarks/)

A.6 Evaluation and expected results

Key results that are reproducible are:

- **Context analyzer (Table 6):** We present the prebuilt llvm bitcode file for a whole kernel binary and the source code of context analyzer in the form of LLVM plugin. See "analyzer/README.md" for detail.
- **CFI precision (Table 2 and Table 3):** It shows how CFI precision gets better as dynamic contexts are used, and reproduces Table 2 and Table 3 in our paper. See "precision/README.md" for detail.
- **Static validator (Section 4.5 - Results.):** We give an in-depth analysis for violations that our validator found (Section 4.5 Results). Also, we present two prebuilt kernel binaries (iOS and PAL) and validator's code written in python, allowing to run our validator. See "static-validator/README.md" for detail.

- **Benchmarks on Mac mini (Table C1 and Section 6.3 - Performance Overhead)):** We present kernel images built with or without PAL, and a root filesystem that contains macro- and micro-benchmarks, and a helper script to run them on Mac mini. See "benchmarks/README.md" for detail.

A.7 Experiment customization

Each README.md in the bitbucket repo explains on how to customize experiments in detail.

A.8 Notes

The results of benchmarks can fluctuate. Even when we used the same mac mini, we saw that its results can vary around 2 times slower or faster at maximum, especially for benchmarks that take a relatively short time.

A.9 Version

Based on the LaTeX template for Artifact Evaluation V20220119.