USENIX Security '26 Artifact Appendix: XCFI: Comprehensive Control-Flow Integrity for Arm TrustZone-M

Yunju Gu
Pusan National University
ventus07@pusan.ac.kr

Jaeyeol Park
Pusan National University
woduf0628@pusan.ac.kr

Donghyun Kwon *
Pusan National University
kwondh@pusan.ac.kr

A Artifact Appendix

A.1 Abstract

This artifact provides the prototype of XCFI, a comprehensive control-flow integrity (CFI) mechanism for Arm TrustZone-M (TZ-M). Built on the Armv8.1-M PACBTI extension, XCFI uses a 32-bit control-flow identifier (CID) to enforce fine-grained CFI across the secure and non-secure states, covering indirect branches, returns, exceptions, and cross-state control-flow events. The artifact includes our modified LLVM toolchain, the Zephyr/TF-M software stack, the benchmarks, the ported ret2ns attack, and the scripts to build, flash, and run them. The experiments run on an Arm MPS3 FPGA board (AN555 image, Armv8.1-M Cortex-M85). The artifact is available on GitHub and archived on Zenodo.

A.2 Description & Requirements

XCFI is built with our modified LLVM toolchain and integrated with Zephyr and Trusted Firmware-M (TF-M); both secure and non-secure code is compiled with the modified toolchain and run on an Arm MPS3 FPGA board (AN555 image, Cortex-M85). At this board is not generally available, we provided remote access to a pre-configured one.

A.2.1 Security, privacy, and ethical concerns

XCFI is a defensive CFI mechanism and poses no risk to the evaluator's machine; no security mechanism is disabled. The artifact includes a port of the public `ret2ns` proof-of-concept attack, which runs only against the protected firmware on the board, not the evaluator's host.

A.2.2 How to access

The artifact is available at https://github.com/cslab-pnu/XCFI and https://doi.org/10.5281/zenodo.20195744.

*Corresponding Author.

A.2.3 Hardware dependencies

XCFI requires an Arm MPS3 FPGA board with the AN555 image, providing an Armv8.1-M Cortex-M85 core with PACBTI, TZ-M, and the MPU enabled; measurements use the DWT cycle counter.

A.2.4 Software dependencies

The host runs Ubuntu 22.04. The artifact builds with our modified LLVM 21 and integrates Zephyr v4.3 and TF-M v2.2. On a clean Ubuntu host, `install_llvm.sh` installs the modified LLVM toolchain and `zephyr_setup.sh` installs the Zephyr/TF-M environment and all remaining dependencies.

A.2.5 Benchmarks

BEEBS (20 longest-running benchmarks), the Zephyr `tfm_integration` examples (`psa_crypto`, `tfm_ipc`, `tfm_secure_partition`), and the ported ret2ns attack.

A.3 Set-up

A.3.1 Installation

On a clean Ubuntu host, set up from scratch:

```
git clone https://github.com/cslab-pnu/XCFI.git
cd XCFI
./install_llvm.sh
./zephyr_setup.sh
cd XCFI-zephyr
```

A.3.2 Basic Test

From `XCFI-zephyr/`, build and flash the `tfm_ipc` example, which exercises the full toolchain (modified LLVM, Zephyr, and TF-M across the secure/non-secure boundary):

```
./build_with_tfm.sh \
    samples/tfm_integration/tfm_ipc xcfi
./flash_with_tfm.sh
```

Reboot the board from the configuration console (`picocom -b 115200 /dev/ttyUSB0`, then type `reboot`). On the application console (`picocom -b 115200 /dev/ttyUSB1`) the boot chain (BL2 → TF-M secure → Zephyr non-secure) completes and the output ends with:

```
*** XCFI enabled ***
TF-M IPC on mps3
...
```

The `*** XCFI enabled ***` line and the completed boot chain confirm that all required components are functioning.

A.4 Evaluation workflow

All experiments use the same build/flash workflow. A build is selected with a mode: `base` (unprotected baseline) or `xcfi` (full XCFI); the experiments below use `xcfi`. `build.sh`/`flash.sh` build and flash a single Zephyr image (no TF-M), while `build_with_tfm.sh`/`flash_with_tfm.sh` build and flash the full TF-M image set (BL2 + secure + non-secure). The board exposes two serial ports: `/dev/ttyUSB0` is the configuration console (used to reboot the board after each flash) and `/dev/ttyUSB1` carries the application output. For each experiment, open a `picocom -b 115200` session on each port, run the build/flash commands, type `reboot` on `ttyUSB0`, and observe `ttyUSB1`.

A.4.1 Major Claims

(C1): XCFI provides fine-grained intra-state CFI (forward-edge, backward-edge, and exceptions); non-secure benchmarks run correctly with XCFI enabled, demonstrating that the instrumentation preserves legitimate execution. Shown by (E1), corresponding Sections 4.2 and 6.3 of the paper.

(C2): XCFI extends CFI to cross-state control flow between the secure and non-secure states; a full TF-M workload involving security state transitions and exceptions runs correctly with XCFI enabled. Shown by (E2), corresponding to Sections 4.3 and 6.6 of the paper.

(C3): XCFI detects and blocks real cross-state control-flow hijacking attacks (`ret2ns`). Shown by (E3), corresponding to Section 7.1 of the paper.

A.4.2 Experiments

(E1): [1 human-minute + 2–3 compute-minutes per benchmark]:
How to: build and run a non-secure BEEBS benchmark with XCFI enabled; it runs to completion, showing intra-state instrumentation preserves correct execution.
Preparation: from `XCFI-zephyr/`, with both serial consoles open.
Execution:

```
./build.sh benchmark/beebs/bubblesort xcfi
./flash.sh
```

then type `reboot` on `ttyUSB0`. To run a different BEEBS benchmark, change the project argument.
Results: `ttyUSB1` shows the boot banner, the `*** XCFI enabled ***` line, and the benchmark's cycle count.

(E2): [1 human-minute + 2–3 compute-minutes per benchmark]:
How to: build and run a full TF-M workload (BL2 + secure + non-secure) with XCFI enabled; it runs across security state transitions and exceptions, showing cross-state instrumentation preserves legitimate cross-state control flow.
Preparation: as in E1.
Execution:

```
./build_with_tfm.sh \
    samples/tfm_integration/tfm_ipc xcfi
./flash_with_tfm.sh
```

then type `reboot` on `ttyUSB0`. The `psa_crypto` and `tfm_secure_partition` examples are selected by changing the project argument.
Results: on `ttyUSB1` the boot chain (BL2 → TF-M secure → Zephyr non-secure) completes, `*** XCFI enabled ***` appears, and the PSA services return their results, and the workload reports its cycle count (full output in the repository README), confirming XCFI does not interfere with legitimate cross-domain control flow.

(E3): [1 human-minute + 2–3 compute-minutes]:
How to: run the ported `ret2ns` attack against the secure boundary; XCFI must detect and stop it.
Preparation: as in E1.
Execution:

```
./build_with_tfm.sh \
    samples/tfm_integration/ret2ns xcfi ATTACK2
./flash_with_tfm.sh
```

then type `reboot` on `ttyUSB0`. `ATTACK1` selects the backward-edge (BXNS) attack; `ATTACK2` selects the forward-edge (BLXNS) attack. See the README for details.
Results: on `ttyUSB1` ends with `[XCFI] Control-flow violation detected in Secure world!` showing the attack is detected and stopped before control is transferred. (In `base` mode the attack instead succeeds and `attacker controlled` is printed.)

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/usenixsec2026/.