

USENIX

THE ADVANCED COMPUTING SYSTEMS ASSOCIATION

Trust Nothing: RTOS Security without Run-Time Software TCB

Eric Ackermann and Sven Bugiel, *CISPA Helmholtz Center for Information Security*

https://www.usenix.org/conference/usenixsecurity26/presentation/ackermann

This artifact appendix is included in the Artifact Appendices to the Proceedings of the 35th USENIX Security Symposium and appends to the paper of the same name that appears in the Proceedings of the 35th USENIX Security Symposium.

August 12–14, 2026 • Baltimore, MD, USA

ISBN 978-1-939133-58-8

Open access to the Artifact Appendices to the Proceedings of the 35th USENIX Security Symposium is sponsored by



جامعة الملك عبدالله للعلوم والتقنية

King Abdullah University of Science and Technology



USENIX Security '26 Artifact Appendix: Trust Nothing: RTOS Security without Run-Time Software TCB

Eric Ackermann, Sven Bugiel
CISPA Helmholtz Center for Information Security

A Artifact Appendix

A.1 Abstract

Embedded devices face an ever-expanding threat landscape: vulnerabilities in application software, operating system kernels, and peripherals threaten the embedded device integrity. Existing computer-architectural defenses fully consider at most two of these threat vectors in their security model.

Our paper aims at addressing this gap using a novel capability architecture. To this end, we combine a token capability approach suitable for building an untrusted operating system with protection against malicious devices without requiring hardware changes to peripherals.

First, we develop and evaluate Bredi, a full FPGA implementation of our capability architecture around legacy hardware components. Further, we present a soft real-time operating system Skadi based on Zephyr that has no run-time software TCB. To this end, we disaggregate Zephyr's subsystems into small, mutually isolated components. All subsystems that exist at run time, including scheduler, allocator and DMA drivers, and all peripherals are fully untrusted. We believe that our work offers a foundation for more rigorous security-by-design in tomorrow's security-critical embedded devices.

We evaluate the overhead of our design. While overheads are significant, we manage to demonstrate that our system can be used in practical scenarios, and room for improvement continues to exist.

Our artifact comprises Skadi, Bredi, instructions for configuring and running the systems as well as raw data from our benchmark executions.

A.2 Description & Requirements

Requirements for building the FPGA SoC. Bredi targets the *Digilent Genesys2* FPGA board. Building the SoC for this board requires *Xilinx Vivado version 2024.2* with a license appropriate for the xc7k325tffg900-2 part (evaluation license should be sufficient). Furthermore, we utilize Xilinx' proprietary AXI ethernet subsystem. To this end, building the SoC requires the appropriate licenses (EF-DI-TEMAC-PROJ or EF-DI-TEMAC-SITE, evaluation license suffices). Provided scripts assume an *Ubuntu 22.04 host system* with root access. Furthermore, for accessing area reports and troubleshooting, we require GUI access to Vivado (e.g., via X forwarding).

It is possible to evaluate the *functionality* of our artifact (sans the DMA) on the Digilent Arty A7-100t FPGA board. This board requires *only free (built-in) Vivado licenses*.

Requirements for building Skadi RTOS. We provide scripts for building Skadi from source. This requires a workstation with Ubuntu 22.04, root access and sufficient free disk space (>32 GB).

Requirements for running the Artifact. All of our evaluations were conducted on a *Digilent Genesys2* FPGA board. Furthermore, we require an SD card, (optionally, for the Arty A7) a suitable USB JTAG debugger (we use a *Digilent HS2*), jumper cables, micro USB cables and an Ethernet connection to a workstation for network performance testing. We used a Raspberry Pi 4 for network performance testing.

Included benchmarks.

The artifact comprises the following benchmarks relevant for our performance claims:

- FPGA chip area can be reported by Vivado in GUI mode (table 2 in the paper).
- ASIC chip area can be reported using our provided scripts (table 3 in the paper). As the run time for the entire ASIC pipeline exceeds the AEC timeframe, and ASIC area is not a major claim of our work, we have elected to *not* reproduce it here. If you want to reproduce these results, follow the instructions under *7.1 - Chip Area—ASIC* in our artifact README and plan with 1-2 weeks of run time.
- Code size is reported using *GNU size* (baseline) and via the console (Skadi) for selected samples (table 4 in the paper).
- Load time is reported via the console for selected samples (table 5 in the paper).
- We provide a custom microbenchmark *sim_capability_test* (table 6, 7 in the paper).
- We use an existing microbenchmark from Zephyr *syscall_perf* (table 7 in the paper).

- We ship the *coremark* and the *stream* benchmarks in a ported version for measuring compute performance (table 8 in the paper).
- For network benchmarks, we use the *ping* and *iperf* utilities with a ported *zperf* sample, a custom shell script and sample for measuring RX/TX latency and Eclipse Mosquitto for measuring MQTT-over-TLS performance (table 9 in the paper).
- Finally, for IRQ performance, we use a custom sample in two modes (table 10 in the paper).

A.2.1 Security, privacy, and ethical concerns

Skadi and Bredi are purely defensive in nature (see ethical discussion in our paper). We do not anticipate any security or privacy risks from using the artifact.

A.2.2 How to access

The permanent reference for the version of the artifact used for our original benchmarks is https://doi.org/10.5281/zenodo.20259754. The version of the artifact that was evaluated during the AE process is https://doi.org/10.5281/zenodo.21486644. An active development version can be found at https://github.com/cispa/Northcape. *Please note:* The updated versions have additional security and performance improvements beyond the original artifact, so there will be some deviation in the results. We aim for the same *order of magnitute* as the results reported in the paper.

For reproducing the results found in the paper, please use the original version of the artifact.

A.2.3 Hardware dependencies

For artifact evaluation, we provided remote access to a workstation suitable for building and running the artifact, including a fully configured Genesys2 FPGA board. For reproducing our results independently, a Digilent Genesys 2 FPGA board and the accessories mentioned above are needed. For building the hardware and software, we recommend a workstation with at least 32 GiB RAM and 200 GB of available disk space.

A.2.4 Software dependencies

Our scripts assume Ubuntu 22.04 with docker installed. In order to install Xilinx Vivado 2024.2, please follow the official instructions to register with Xilinx and download version 2024.2 of the software: https://www.xilinx.com/support/download/index.html/content/xilinx/en/downloadNav/vivado-design-tools/2024-2.html. Make sure to install *Vitis* (including Vivado) and ensure you install board files for at least *Kintex 7*.

A.2.5 Benchmarks

All benchmarks are provided in source code form with the artifact.

A.3 Set-up

A.3.1 Installation

Please make sure that Vivado version 2024.2 is installed first. Our scripts assume it is in */opt/Xilinx/Vivado/2024.2/*.

Then, from the top directory, run our scripts for compiling the FPGA bitstream (hardware):

```
cd scripts/Vivado
# Bredi SoC
./create_project.sh
./create_bitstream.sh
# Baseline for SoC without Bredi
./create_project_non_bredi.sh
./create_bitstream_non_bredi.sh
```

FPGA synthesis is non-interactive and should take around 2-3 hours.

In parallel to or after compiling the hardware, you can compile the *Linux* and *Zephyr* baselines:

```
# Linux
sudo apt-get update &&
sudo apt-get install -y \
        autoconf \
        automake \
        autotools-dev \
        curl \
        libmpc-dev \
        libmpfr-dev \
        libgmp-dev \
        libusb-1.0-0-dev \
        gawk \
        build-essential \
        bison \
        flex \
        texinfo \
        gperf \
        libtool \
        patchutils \
        bc \
        zlib1g-dev \
        device-tree-compiler \
        pkg-config \
        libexpat-dev \
        wget \
        cpio \
        python3 \
        python3-pip \
        unzip \
        rsync \
        git \
        picocom \
        psmisc
```

```
cd software/include/northcape-linux-stack
# Linux setup
make images
# flash SD card.
# ALL DATA ON SDDEVICE ARE LOST
sudo make flash-sdcard SDDEVICE=/dev/sdX
chmod 777 scripts/ssh*
# Zephyr / Skadi setup
cd ../northcape-zephyr-stack/
sudo docker build -t skadi_build:latest . \
-f scripts/skadi/Dockerfile
cd ../
# adapt USB device: you can get bus
# and function number XXX and YYY
# from lsusb
# look for ID 0403:6010, FT2232
sudo docker run -v .:/workspace \
--device /dev/bus/usb/XXX/YYY \
-it skadi_build:latest bash
# within the container
cd northcape-zephyr-stack
./scripts/skadi/install-dependencies.sh
```

Software compilation involves building a GNU toolchain from scratch with a custom configuration. In combination, Linux and Zephyr can compile for 5 or more hours. Also, Linux and Zephyr/Skadi setup can be run in parallel.

The provided scripts configure a Docker container for compiling Zephyr/Skadi. *Note:* You can use the same scripts to configure a persistent build environment. However, this will modify your home and temp directory.

A.3.2 Basic Test

We suggest two distinct basic tests: Booting Linux into a root shell and loading Skadi's hello world sample.

Booting Linux. Insert the SD card into the board. From the top directory, run the following commands to launch Vivado interactively and run the hardware server:

```
cd scripts/Vivado
# must have GUI environment
# e.g., SSH X forwarding
./open_project.sh
# second shell, in scripts/Vivado
./launch_hw_server.sh
# third shell
# adapt to Genesys' FTDI UART
# add your user to dialout
# or run as root
picocom -b 115200 /dev/serial/by-id/*
```

Start the TFTP server in a separate shell:

```
cd software/include/
cd northcape-linux-stack/scripts
./start-tftp.sh
```

When Vivado has fully loaded the project, expand *Open Hardware Manager* in the left-most *Flow Navigator* pane. Click *Open Target* and *Open New Target*. Select *Remote server* and configure its location as `localhost:3121`. The dialog should list an *xck7* device, corresponding to the Genesys 2. Accept defaults and continue through the dialog.

You can now click *Program Device* and the *xck7* device in the *Flow Navigator*. The bit stream generated in the setup should be pre-filled in the dialog, and the probes file is optional. Hit *Program* to program the device.

After programming the FPGA, you should receive output from the boot ROM, OpenSBI and u-boot stages without further action. If everything works correctly, the board should boot into a root shell.

Running Skadi's hello-world sample.

Build and load the hello-world sample:

```
cd software/include
# mount a new container if needed
sudo docker run -v .:/workspace \
--device /dev/bus/usb/XXX/YYY \
-it skadi_build:latest bash
cd northcape-zephyr-stack
# run this in every new Docker container
# already run in the first container
./scripts/skadi/setup-docker.sh
source .venv/bin/activate
# build the sample
west build -p always -b \
    cv64a6_genesysII \
    samples/hello_world
west debug --skip-rebuild
```

When GDB has completed loading the sample, hit *c* and enter to continue. You should see Skadi relocating its subsystems one by one before printing a final hello world message.

Please note previous versions of the artifact were prone to errors in the debug module, causing loading Skadi/Zephyr to fail. We provide a fix of this problem in the final version of the artifact.

Loading the hello-world sample should take less than one minute. You should see GDB load the initial sections of the loader ELF relatively quickly, followed by a longer transfer of the read-only data section. If you receive an error from GDB or it does not start loading any sections, repeat *west debug*. If the second attempt fails as well, kill all instances of *openocd*, program the FPGA again via Vivado and try again.

A.4 Evaluation workflow

A.4.1 Major Claims

(C1): Bredi's area overhead matches the values given in table 2 of the paper, with small variations. This is proven by experiment (E1).

(C2): Bredi's code size overhead matches the values given in table 4 of the paper, with small variations. This is proven by experiment (E2).

(C3): Bredi's load time overhead matches the values given in table 5 of the paper, with small variations. This is proven by experiment (E3).

(C4): Bredi's operation microbenchmarks are fast, consuming less than 1000 cycles from a software and less than 100 cycles from a hardware perspective each, matching the values in table 6 and 7. This is proven by experiment (E4).

(C5): Bredi's compute performance is indistinguishable from the Zephyr baseline and faster than the Linux baseline, matching table 8. This is proven by experiment (E5).

(C6): Bredi's network performance overhead over Zephyr is less than factor three for the ping, TCP performance and TX duration benchmarks, matching table 9 from the paper. This is proven by experiment (E6).

(C7): Bredi's network performance overhead over Zephyr is approximately 4 for the RX duration benchmark, matching table 9 from the paper. This is proven by experiment (E6).

(C8): Bredi's MQTT performance overhead over Zephyr is less than 10%, matching table 9. This is proven by experiment (E6).

(C9): Bredi's network performance overhead over Linux is less than 2 for the ping, RX and TX duration benchmarks, matching table 9 from the paper. This is proven by experiment (E6).

(C10): Bredi's IRQ performance overhead over Zephyr is less than 3 for independent interrupts. This is proven by (E7).

(C11): Bredi's IRQ performance overhead over Zephyr is less than 10 for monotonic interrupts. This is proven by (E7).

A.4.2 Experiments

(E1): [Chip Area] [5 human-minutes]: Determine the FPGA Area overhead of Bredi.
Preparation: Synthesize both versions of the SoC as explained in the Set-up steps.
Execution: Follow the steps under *7.1 - Chip Area—Northcape FPGA SoC* in *paper_artifact/README.md*.
Results: The results comprise the chip area reported by Vivado for each component.

(E2): [Code Size] [30 human-minutes]: Determine the code size of Skadi compared to its baseline, Zephyr.
Preparation: Synthesize and program the FPGA configuration. Setup the software toolchain.
Execution: Follow the steps under *7.2 - Software: Code Size—zephyr Code Size - Genesys2* and *Skadi Code Size - Genesys2* in *paper_artifact/README.md*.
Results: *paper_artifact/README.md* explains how to collect results.

(E3): [Load Times] [1 human-hour]: Determine the load time of Skadi compared to its baseline, Zephyr.
Preparation: Synthesize and program the FPGA configuration. Setup the software toolchain.
Execution: Follow the steps under *7.3: Load times—Load times zephyr - Genesys 2* and *Load times Skadi - Genesys 2* in *paper_artifact/README.md*. *Note:* We loaded each sample 5 times to determine a standard deviation, which ended up being negligible. You can reduce the number of repetitions in order to save time if you are confident in the significance of our result.
Results: The samples will print the load time onto the UART console.

(E4): [Micro Benchmarks] [20 human-minutes]: Determine the performance of Northcape/Bredi operations.
Preparation: Synthesize and program the FPGA configuration. Setup the software toolchain.
Execution: Follow the steps under *7.4: Microbenchmarks—Microbenchmarks - Genesys 2* in *paper_artifact/README.md*.
Results: The samples will report minimum, maximum, mean and standard deviation for each operation. CPU cycles are given in lines like this: *create microbenchmarks (Samples) discarded/min/max/avg/stddev ns: 0/622/635/629.273684/4.387643*, where *create* is the name of the operation. Operations module cycles are given in lines like this: *create microbenchmarks (ops only) (Samples) discarded/min/max/avg/stddev ns: 0/0/0/0.000000/0.000000*. They are only significant in the second sample built by the documented steps (the sample that includes *ops_cycles.conf*).

(E5): [Compute Benchmarks] [10 human-minutes + 1 compute-hour]: Determine the performance of Skadi and Bredi for compute workloads.
Preparation: Synthesize and program the FPGA configuration. Setup the software toolchain.
Execution: Follow the steps under *7.5 System: Compute Performance—zephyr Compute Performance - Genesys2*, *Skadi Compute Performance - Genesys2* and *Linux compute Performance - Both* in *paper_artifact/README.md*. *Note:* Reprogram the FPGA in order to allow it to boot into Linux.
Results: The stream benchmark will print a table of results. Coremark will print an individual score. For both benchmarks, our main paper reports the average score only.

(E6): [Network Benchmarks] [2 human-hours]: Determine the performance of Skadi and Bredi for network workloads.
Preparation: Synthesize and program the FPGA configuration. Setup the software toolchain.
Execution: Follow the steps under *7.6 System: Network Performance—zephyr Network Performance - Genesys2*, *Skadi Network Performance - Genesys2* and *Linux Network Performance - Both* in *paper_artifact/README.md*. *Note:* Reprogram the FPGA

in order to allow it to boot into Linux.

Results: The ping and iperf utilities print their results to stdout. The RX/TX duration and MQTT-TLS benchmarks print summaries similar to the micro benchmarks.

(E7): [IRQ Benchmarks] [20 human-minutes]: Determine the IRQ latency for Skadi and Zephyr.

Preparation: Synthesize and program the FPGA configuration. Setup the software toolchain.

Execution: Follow the steps under *7.7 System: Real-Time Capability—zephyr Real-Time Capability - Genesys2* and *Skadi Real-Time Capability - Genesys2* in *paper_artifact/README.md*. *Note*: Our original submission contained additional measurements, comprising a Linux equivalent of the benchmarks and a scheduler microbenchmark (latency_sample). These benchmarks were cut from the paper and need not be repeated.

Results: The IRQ benchmarks print summaries similar to the micro benchmarks.

A.5 Notes on Reusability

We intend Skadi and Bredi to serve as a platform for future computer-architectural research. To this end, we provide additional documentation for toolchain and physical setup and wiring of the boards in *README.md*. As mentioned previously, a slightly restricted version of the SoC can be evaluated on the cheaper Arty A7 board without requiring licenses for Vivado or AMD/Xilinx IP. All experiments mentioned above can be attempted on the Arty A7 as well.

We discuss porting software to Skadi in our artifact and provide code samples. To this end, please refer to software/include/northcape-zephyr-stack/doc/hardware/porting.

You can use Zephyr's menuconfig to review supported configuraton options for Skadi, depending on your application needs.

Finally, the Bredi-specific hardware components exist in an individual directory with a README and an UVM-based testbench. Please see hardware/include/northcape. Certain parameters of Bredi components can also be tuned in the block design of the Vivado GUI, e.g., cache algorithms and sizes.

A.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/usenixsec2026/.