USENIX Security '26 Artifact Appendix: PICASSO: Scaling CHERI Use-After-Free Protection to Millions of Allocations using Colored Capabilities

Merve Gülmez*, Ruben Sturm†‡, Hossam ElAtali§, Håkan Englund*,
Jonathan Woodruff‖, N. Asokan§,¶, Thomas Nyman#
*Ericsson Security Research, †DistriNet, KU Leuven, §University of Waterloo,
‖University of Cambridge, ¶KTH Royal Institute of Technology, #Ericsson Product Security
{merve.gulmez,hakan.englund,thomas.nyman}@ericsson.com, ruben.sturm@kuleuven.be
hossam.elatali@uwaterloo.ca, jonathan.woodruff@cl.cam.ac.uk, asokan@acm.org

A Artifact Appendix

This paper introduces *colored capabilities* that add a controlled form of indirection to CHERI's capability model. This enables *provenance tracking* of capabilities to their respective allocations via a hardware-managed *provenance-validity table*, allowing bulk retraction of dangling pointers without needing to quarantine freed memory. Colored capabilities significantly reduce the frequency of capability revocation sweeps while improving security. Colored capabilities are realized in PICASSO, an extension of the CHERI-RISC-V architecture on a speculative out-of-order FPGA softcore (CHERI-Toooba). Colored-capability support is integrated into the CheriBSD OS and CHERI-enabled Clang/LLVM toolchain. PICASSO successfully detects use-after-free and double-free defects in NIST Juliet test cases without false positives and real-world CVEs in BZip2 (CVE-2016-3189), libiberty (CVE-2016-4487), nasm (CVE-2017-10686, CVE-2019-8343), libzip (CVE-2019-17582), NGINX (CVE-2020-24346), LibreDWG (CVE-2022-35164), Lua (CVE-2019-6706), and three other use-after-free issues reported in mjs (issues 73 and 78) and yasm (issue 91). The efficiency of PICASSO is demonstrated using the SPEC CPU benchmark and real-world SQLite, PostgreSQL, and gRPC benchmarks.

A.1 Abstract

This artifact provides the complete implementation of PICASSO on the speculative out-of-order CHERI-Toooba FPGA softcore, along with the Colored capability-enabled CheriBSD operating system, Clang/LLVM toolchain, QEMU implementation, runtime setup, and detailed instructions necessary to reproduce the experimental results.

The artifact supports three evaluation environments:

- *A QEMU-based emulation environment* with support for colored-capability-enabled CheriBSD. This environment can be used to reproduce the security evaluation using the NIST Juliet test suite, real-world CVEs, and the other real-world use-after-free issues.
- *A bare-metal hardware simulation environment* based on the Bluespec simulator. This environment can be used to reproduce performance evaluation using the MiBench suite and Coremark.
- *An FPGA-based evaluation environment* with support for colored-capability-enabled CheriBSD. This environment can be used to reproduce the security evaluation and the performance evaluation using the CheriBSD-based benchmarks (SPEC, SQLite, PostgreSQL, and gRPC).

Note: The Artifact Evaluation Committee evaluated the artifact in the QEMU emulation and bare-metal hardware simulation environments. Further details on the reproduced experiments are provided in § A.4.2.

A.2 Description & Requirements

A.2.1 Security, privacy, and ethical concerns

Theoretically methods such as PICASSO can be misused for malicious vulnerability discovery. However, executing PICASSO on the field-programmable gate array (FPGA) platform or QEMU-based emulation environment supported by our artifact is orders of magnitude slower than alternative software-based approaches for vulnerability detection, such as sanitizer-instrumented binaries. Consequently, the PICASSO artifact does not provide additional capabilities beyond such existing tools. We therefore do not expect security, privacy, or ethical concerns associated with the PICASSO artifact.

[3] R.S. contributed to the research while at Ericsson Security Research.

A.2.2 How to access

Our artifact can be accessed at: https://github.com/coloredcapabilities/colored-artifact and https://doi.org/10.5281/zenodo.20353117.

A.2.3 Hardware dependencies

The QEMU-based emulation environment, the bare-metal hardware simulation environment, and the build environment for software artifacts targeting each evaluation environment require a **x86_64 Linux machine** with Docker installed. The build for the docker images requires roughly 120 GB of disk space.

The FPGA-based evaluation environment requires a **Xilinx/AMD VCU118 (UltraScale+ Virtex) evaluation board**, and a host machine with JTAG/Vivado Hardware Manager.

A.2.4 Software dependencies

The artifact provides a Dockerfile for the QEMU-based emulation environment, the bare-metal hardware simulation environment, and the build environment for software artifacts. The Dockerfile has been tested on **Ubuntu Server 24.04.4 LTS** using **Docker version 29.1.3** obtained from https://ubuntu.com/server

Xilinx/AMD Vivado Design Suite 2019.1 and license is required to build/flash bitstreams for the FPGA-based evaluation environment. Pre-built bitstreams are also provided, requiring only a means of programming the board, e.g. JTAG/Vivado Hardware Manager from **Vivado Lab 2019.1**. The Vivado Design Suite and Lab 2019.1 installers can be obtained from https://www.amd.com/en/support/downloads/adaptive-socs-and-fpgas/development-tools/2019-1.html

Reproducing the performance evaluation using SPEC CPU requires the **SPEC 2006 CPU ISO image** and license from https://www.spec.org/cpu2006/

A.2.5 Benchmarks

We use the NIST Juliet test suite, eight known CVEs and three other use-after-free issues for the security evaluation of PICASSO and Mibench, Coremark, SPEC SPEC CPU 2006, SQLite, PostgreSQL, and gRPC benchmarks to evaluate the performance.

A.3 Set-up

Before installing the PICASSO artifact for the first time, please perform a clean install of **Ubuntu Server 24.04.4 LTS**, install **Docker**, and add $USER to the docker group:

```
sudo apt install docker.io
sudo usermod -aG docker $USER
```

Logout, then login again for the group change to take effect.

On the host connected to the VCU118 evaluation board, download and install **Vivado Design Suite 2019.1**. A license key for the tool should have been included with your VCU118 board. See the Vivado Design Suite User Guide for installation instructions: https://docs.amd.com/v/u/2019.1-English/ug973-vivado-release-notes-install-license

If using different machines for building the artifact and FPGA host, only the build machine needs a license in order to build new bitstreams. In this case you can install **Vivado Lab 2019.1** (obtained from the same URL as Vivado Design Suite) on the FPGA host machine because it does not require a license and is solely used to program the FPGA.

A.3.1 Installation

1. Clone the artifact repository:

```
git clone
 ↪  https://github.com/coloredcapabilities/colored-artifact.git
```

2. Build the PICASSO container image:

```
cd colored-artifact
## 112 GiB, and takes 7 hours, it builds the whole
 ↪  stack, qemu, bluespec simulator, llvm, cheribsd,
 ↪  gdb
docker build --network=host -f Dockerfile -t picasso .
```

A.3.2 Basic Test

```
# Start the container
docker run -i -t  --name picasso-run picasso

# Boot CheriBSD
cd ~/cheri/cheribuild
./cheribuild.py
run-riscv64-purecap --skip-update
 ↪  --run-riscv64-purecap/ssh-forwarding-port 10222
```

The basic test should result in a CheriBSD login prompt. Entering `root` as the username (no password is needed) should result in a funtional shell in the CheriBSD system.

The artifact directory contains a `FPGA.md` file with instructions for flashing the FPGA with CheriBSD and obtaining a shell on the FPGA guest system.

A.4 Evaluation workflow

A.4.1 Major Claims

(C1): PICASSO detect all use-after-free and double-free violations in the heap-based temporal-memory-safety CWEs of the NIST Juliet test suite (CWE-415/CWE-416) and in eight real-world use-after-free/double-free CVEs and three other use-after-free issues. This is demonstrated by experiment (E1).

(C2): The cycle-count overhead of PICASSO on the MiBench suite and Coremark running on the Bluespec simulator is small. This is demonstrated in experiment (E2), corresponding to Table 3 and Table 5 in Appendix C in the paper.

(C3): PICASSO incurs only a small (≈5% geometric mean) execution-time overhead over baseline CHERI-Toooba on SPEC CPU2006, while requiring substantially less frequent capability revocation sweeps than prior sweeping-based approaches. This is proven by experiment (E3), corresponding to §7.2 and Figure 10 in the paper.

(C4): For long-running workloads (SQLite, PostgreSQL, gRPC), PICASSO provides lower and more consistent latency than prior CHERI temporal-safety mechanisms that rely on periodic memory-sweeping revocation. This is proven by experiment (E4, E5, E6), corresponding to §7.2 and Figures 11, 12, and 13 in the paper.

A.4.2 Experiments

Note: E1, E2, and E4 were evaluated and successfully reproduced by the Artifact Evaluation Committee. E3, E5, and E6 require specific hardware and software licenses. Due to these hardware and license dependencies, E3, E5, and E6 could not be reproduced during the artifact evaluation.

(E1): *Security evaluation (Juliet & real-world CVEs)* [1 human-hour + 1–2 compute-hours]: demonstrates that PICASSO detects all heap UAF/double-free violations in Juliet CWE-415/CWE-416, eight real-world CVEs, and three other use-after-free bugs. Can be run on QEMU or on the FPGA.

Preparation: The Juliet test suite, CVE and bug PoCs are pre-built in the container image.

```
### Start the container
docker run -i -t  --name picasso-run picasso

## Boot CheriBSD in Terminal 1
cd cheribuild
./cheribuild.py run-riscv64-purecap --skip-update
 ↪  --run-riscv64-purecap/ssh-forwarding-port 10222

## Open Terminal 2 and run:
## Transfer Juliet binaries
docker exec -it picasso-run bash
~/cheri/utils_script/transfer_juliet.sh
 ↪  root@127.0.0.1 -p 10222

## Run in Terminal 2:
## Transfer CVE and bug tests
~/cheri/validation/transfer.sh root@127.0.0.1 -p
 ↪  10222
```

Execution: To run Juliet test suite, CVE and bug PoCs:

```
# Run in Terminal 1:
cd /root/juliet-bin
sh juliet-run.sh 415 2s < /tmp/in.txt
sh juliet-run.sh 416 2s < /tmp/in.txt

## Run in Terminal 2:
## Run the CVE and bug tests
cd /home/ubuntu/cheri
validation/run_all_remote.sh root@127.0.0.1 -p
 ↪  10222
## Collect juliet result
utils_script/collect_juliet_results.sh
 ↪  root@127.0.0.1 -p 10222
```

Results: For CWE-415 (Double Free), the program exits with code 255 (`exit(-1)`). For CWE-416 (Use After Free), the program receives signal 34 (SIGPROT, an in-address space CHERI capability exception), resulting in exit code 162. For each of the CVEs, and issue reports `DETECTED (signal 34)` for UAF cases or exit code 255 for double-free cases; an exit code of 0 (`VULNERABLE`) would indicate the bug was not caught.

(E2): *PICASSO run-time overhead (MiBench and Coremark)* [15 human-minutes + 1 compute-hours for MiBench – 8 compute-minutes for Coremark]:

Preparation: The Bluespec simulator, Mibench, and Coremark binaries are pre-built in the container image.

```
### Start the container (if not already running)
docker run -i -t  --name picasso-run picasso
```

Execution: Inside the Docker images

```
## Run in shell inside container:
cd /home/ubuntu/bench
./run_mibench.sh

## Run in shell inside container:
cd /home/ubuntu/bench/coremark
./run_coremark_for_sim.sh
```

Results: MiBench incurred an average overhead of 1.32% in simulation. For CoreMark, PICASSO purecap incurred overheads of 3.12% and 6.14%, depending on the configuration. The simulated CoreMark overheads differ slightly from the FPGA results reported in the paper (§7.2).

(E3): *PICASSO run-time overhead (SPEC CPU2006)* [1 human-hour + 42 compute-hours]: Due to licensing restrictions, we do not provide the SPEC CPU2006 source code. Evaluators must have their own valid SPEC CPU2006 license to reproduce these results.

Preparation: To build SPEC for CheriBSD:

```
### Start the container in Terminal 1
### (if not already running)
docker run -i -t --name picasso-run picasso

## Transfer SPEC CPU2006 ISO from Terminal 2:
docker cp /path/to/cpu2006-1.2.iso
 ↪  picasso-run:/home/ubuntu/
```

```
## Run in Terminal 1
cd cheribuild
./cheribuild.py spec2006-riscv64-purecap
↪ --spec2006/iso-path /home/ubuntu/cpu2006-1.2.iso
```

The built SPEC folder is placed at:

```
~/cheri/build/spec2006-riscv64-purecap-build
```

Copy the built SPEC folder to the **host** system:

```
## Run in Terminal 2:
mkdir -p ~/cheri/build
docker cp
↪ picasso-run:~/cheri/build/spec2006-riscv64-purecap-build
↪ ~/cheri/build/
```

Once copied, built SPEC folder on the **host** is placed at:

```
~/cheri/build/spec2006-riscv64-purecap-build
```

Now prepare the SPEC benchmark folder from a shell on the **host**:

```
# Remove unnecessary files to reduce transfer size
$~/cheri/utils_script/spec/spec_folder_reduce_folder.sh

# Instrument benchmark run scripts with timing/stat counters
python3 ~/cheri/utils_script/spec/automate.py
```

Copy the SPEC folder and `$ARTIFACT_DIR/utils_script/spec/run_all_spec_fpga.sh` to the FPGA (see `FPGA.md` included in artifact for copying instructions).

Execution: Run inside **FPGA guest**:

```
cd /bench/SPEC/CINT2006
sh /path/to/run_all_spec_fpga.sh
↪ /bench/SPEC/CINT2006
```

Results: The results for each benchmark are output to:

```
<benchmark>/<benchmark>_OUTPUT/
```

After collecting results for all each benchmark, place the resulting `_OUTPUT` directories under `utils_script/spec/colored_paper/` directory:

```
utils_script/spec/colored\_paper/<benchmark>\_OUTPUT/
```

We include the output directories for our experiments using all configurations (`baseline`, `colored_paper`, `cornucupia`) in the artifact. Replace the `colored_paper` output directories to obtain the overhead tables for the reproduced results. Once the reproduced output directories are in place, run the analysis script to generate the overhead tables and figures:

```
python3
↪ $ARTIFACT_DIR/utils_script/spec/analyze_spec_overhead.py
```

This produces:

- Per-benchmark cycle, memory, tagcache, and DRAM traffic overhead tables
- Geometric mean summaries across all benchmarks
- Overhead figures saved as PDF and PNG alongside the script

Comparing the resulting tables with the reference data in the paper should yield comparable results.

(E4): *PICASSO memory overhead and reduce revocation sweep (SQLite) for QEMU* [5 human minutes – 30 compute-minutes for QEMU, 1 hours for SQLite]:
Preparation: SQLite binaries are pre-built in the container image.

```
### Start the container (if not already running)
docker run -i -t --name picasso-run picasso

### Run PICASSO QEMU
./cheribuild.py run-riscv64-purecap --skip-update
↪ --run-riscv64-purecap/ssh-forwarding-port 10222

### Run Baseline QEMU in the third terminal
~/cheri/utils_script/run_baseline_qemu.sh
```

Execution: Inside the container

```
## Run in shell inside container:
cd ~/cheri/utils_script/sqlite
## For PICASSO
./run_speedtest1.sh root@127.0.0.1 -p 10222

## For Baseline
./run_speedtest1.sh root@127.0.0.1 -p 10223

## Check the results, after the speedtest1 results,
↪ look at the mrs[process_id] revoke counter
cat ./speedtest1_logs/speedtest1_$date.log
```

Results: Compared to the baseline, the results show improved performance with zero PICASSO revoke events (vs. 272 in the baseline) and a significantly lower maximum resident set size of 13,268 (vs. 30,708), indicating reduced memory usage and better resource efficiency.

(E5): *PICASSO memory overhead and reduce revocation sweep (PostgreSQL) for FPGA* [30 human minutes – 3 hours for FPGA]:
Preparation: PostgreSQL binaries are not pre-built in the container image. Please compile from scratch.

```
### Start the container (if not already running)
docker run -i -t --name picasso-run picasso
### Compile the postgres
cd $CHERI_ROOT
git clone
↪ https://github.com/CTSRD-CHERI/postgres.git
cd postgres
git apply $ARTIFACT_DIR/patches/postgress.diff
cd $CHERIBUILD
./cheribuild.py postgres-riscv64-purecap -d
```

Execution: Copy scripts to FPGA

```
## Inside the FPGA
## creates the database and runs pgbench - 60
↪ minutes
sh ./postgres-bench-stats.sh

## Inside the Host
```

```
cd ~/cheri/utils_script
sh postgress/pgbench-client.sh

postgress/plot_combined_cdf.py
```

Results: This produces the Figure 13 in the paper. For p50 latency, PICASSO incurs a 3% overhead, while Cornucopia incurs a 7% overhead. The raw output can be found at `/cheri/utils_script/postgress`

(E6): *PICASSO memory overhead and reduce revocation sweep gRPS for FPGA* [30 human minutes – 4 hours for FPGA]:
Preparation: Build inside the container

```
### Start the container (if not already running)
docker run -i -t  --name picasso-run picasso
### Compile the qrps
cd ~/cheri/cheribuild

./cheribuild.py grpc-native -d
./cheribuild.py grpc-riscv64-purecap -d
```

Execution: Run the in the host machine

```
~/cheri/utils_script/grpc/grpc-client-bytes.sh

## get the plot
~/cheri/utils_script/grpc/analyze_qps_boxplot.py
```

Results: PICASSO degrades transactions per second (TPS) by 6.2%, compared to 23.1% for Cornucopia. Figure 13 plots individual transaction latencies, Cornucopia exhibits periodic latency spikes occurring approximately every 20 transactions (60 allocations per transaction).

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/.