USENIX Security '26 Artifact Appendix: DMGUARD: Safeguarding Kernels from Physical-Page Use-After-Free Vulnerabilities

Juhee Kim*, Jaeyoung Chung*, Dae R. Jeong†, Byoungyoung Lee†
Seoul National University
{kimjuhi96,jjy600901,dae.r.jeong,byoungyoung}@snu.ac.kr

A Artifact Appendix

A.1 Abstract

This artifact provides the implementation and evaluation environment for DMGUARD, a runtime solution that detects physical-page use-after-free vulnerabilities. The artifact includes the following components:

- Linux kernel v6.6 and Mali GPU Driver r52p0 patches implementing DMGUARD
- Five real CVE PoCs and five synthetic PoCs
- Automated scripts for security and performance evaluation
- A QEMU ARM64 evaluation environment based on Docker

Note that some tests requiring specific hardware or kernel versions are excluded from this artifact. Since the performance evaluation runs on QEMU emulation, absolute numbers may differ from those reported in the paper.

A.2 Description & Requirements

A.2.1 Security, privacy, and ethical concerns

This artifact contains PoC code for five CVEs: CVE-2024-1065, CVE-2022-36449, CVE-2025-0072, CVE-2022-33917, and CVE-2024-0671. All of these vulnerabilities have already been publicly disclosed and patched. All code runs only within a QEMU virtual environment, posing no impact on real systems and no security risk to the host system.

A.2.2 How to access

The artifact is available from two sources:

- GitHub: https://github.com/compsec-snu/dmguard
- Zenodo DOI: https://doi.org/10.5281/zenodo.17970502

*These authors contributed equally to this work.
†Corresponding authors

A.2.3 Hardware dependencies

The artifact requires an x86_64 host machine. The minimum requirements are 8GB RAM and 100GB free disk space. We recommend 16GB RAM and 8+ CPU cores for faster kernel builds. No special hardware is required as the artifact uses QEMU-based emulation.

A.2.4 Software dependencies

The artifact requires Linux with Docker Engine and Docker Compose installed. All other dependencies are included in the Docker image and require no additional installation. These include LLVM-19/Clang for kernel builds, QEMU system ARM64, aarch64-linux-gnu cross-compiler, debootstrap, expect, and the source code for Linux kernel and Mali GPU driver.

A.2.5 Benchmarks

The CVE PoC code was collected from public GitHub repositories and Project Zero issue tracker. For performance evaluation, we use LMBench microbenchmark, which is publicly available online. All related code and installation scripts are included in this artifact, allowing evaluators to reproduce results without additional downloads.

A.3 Set-up

A.3.1 Installation

The installation consists of four steps.

Step 1: Register binfmt (run once on host).

```
docker run --rm --privileged \
  multiarch/qemu-user-static --reset -p yes
```

Step 2: Clone repository and build Docker image.

```
git clone \
  https://github.com/compsec-snu/dmguard
cd dmguard
docker compose build    # ~10 minutes
```

Step 3: Start and enter container.

```
docker compose up -d
docker compose exec dmguard-ae bash
```

Step 4: Build kernels and VM image (120 compute-minutes).

```
# Inside container
./ae/scripts/setup-all.sh
```

A.3.2 Basic Test

Run the basic test script to verify that the environment is correctly set up (5 compute-minutes):

```
# Inside container
./ae/scripts/basic-test.sh
```

If VM boot and SCP to VM succeed, a pass message will be displayed. Passing this test indicates that security and performance evaluations can run without issues.

A.4 Evaluation workflow

A.4.1 Major Claims

(C1): DMGUARD detects physical-page use-after-free vulnerabilities across all Page and Map Context combinations, including both Free-Before-Unmap and Map-After-Free types. `CONFIG_PAGE_TABLE_CHECK` (i.e., `ptcheck`), in contrast, only detects Free-Before-Unmap with CPU-only contexts and remains vulnerable to race condition bypasses. This is proven by experiment (E1) described in §6.2, with results shown in Table 2.

(C2): DMGUARD exhibits acceptable runtime overhead, though relatively higher than `ptcheck`. This is proven by experiment (E2) described in §6.3, with results shown in Tables 4, 7, and 9.

A.4.2 Experiments

(E1): Security Evaluation (20 compute-minutes): This experiment compares the detection capabilities of `ptcheck` and DMGUARD for GPU driver vulnerabilities and synthetic vulnerabilities reproducible in QEMU.
Preparation: Ensure that the basic test in A.3.2 passes.
Execution: Run the security evaluation script: `./ae/scripts/run-security-eval.sh`
Results: Results can be found in `results/security/table2_security.txt`. Expected results are documented in the repository's `README.md` under (E1). C1 is confirmed if DMGUARD detects all 10 vulnerabilities and `ptcheck` detects only 2 vulnerabilities (CVE-2024-1065 and CVE-2022-36449). Note that this artifact reproduces 10 of the 15 vulnerabilities in the paper's Table 2. The remaining 5 CVEs are excluded because CVE-2023-6241 and CVE-2023-33107 require real Mali/Adreno GPU hardware unavailable in the QEMU emulation environment, and CVE-2024-0582, CVE-2024-47674, and CVE-2024-53096 affect kernel versions other than v6.6.0 used in this artifact.

(E2): Performance Evaluation (120 compute-minutes): This experiment measures the overhead of DMGUARD and `ptcheck` compared to baseline, reproducing Table 4 (LMBench), Table 7 (Boot Time), and Table 9 (Memory Overhead).
Preparation: Ensure that the basic test in A.3.2 passes.
Execution: Run the performance evaluation script: `./ae/scripts/run-performance-eval.sh`
Results: Results can be found in three files: `results/performance/table4_lmbench.txt`, `results/performance/table7_boot_time.txt`, and `results/performance/table9_memory.txt`. Expected results are documented in the repository's `README.md` under (E2). C2 is confirmed if DMGUARD overhead is observed to be approximately 3x that of `ptcheck`. Note that since this artifact runs in a QEMU emulation environment, absolute numbers will be higher than those measured on actual Pixel 8 hardware as reported in the paper. Nevertheless, the overhead remains acceptable and similar trends are maintained. Additionally, due to the instability of the QEMU environment, we use median instead of average, unlike in the paper.

A.5 Notes on Reusability

The DMGUARD kernel patch code is written based on Linux kernel v6.6. Manual modifications are required to apply it to other kernel versions or drivers.

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/.