

USENIX

THE ADVANCED COMPUTING SYSTEMS ASSOCIATION

Melting the Flesh of PHP's Memory Hardening

Yifan Wu, *University of California, Riverside;* Xiaochuan Yu, *University of California, San Diego;* Zhiyun Qian, *University of California, Riverside*

https://www.usenix.org/conference/usenixsecurity26/presentation/wu-yifan

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: Melting the Flesh of PHP's Memory Hardening

Yifan Wu
UC Riverside

Xiaochuan Yu
UC San Diego

Zhiyun Qian
UC Riverside

A Artifact Appendix

A.1 Abstract

This artifact reproduces the end-to-end exploits and targeted mitigations evaluated in our paper. It contains a uniform PHP-8.4.0 test environment with all four prototyped heap hardening protections enabled, five end-to-end exploit cases (three CVEs and two CTF challenges) that achieve remote command execution under this environment, and two prototype mitigations (Patch-hashtable and Patch-refcnt) that block these exploits. Each case is packaged as a one-click Docker setup and a Python exploit script, and each mitigation is provided as a one-click run script that rebuilds PHP with the patch applied and re-runs the exploit to confirm it is mitigated.

A.2 Description & Requirements

A.2.1 Security, privacy, and ethical concerns

All exploits target a PHP interpreter running inside a self-contained Docker container, bound to localhost ports on the evaluator's host. No destructive operations are performed on the host. The exploit scripts only interact with the containerized target over HTTP and read back the output of `uname -a` and `date` as proof of remote command execution. The vulnerabilities exercised are existing public CVEs and CTF challenges, and no new vulnerabilities are introduced.

A.2.2 How to access

the archived copy of the artifact is available on Zenodo at https://zenodo.org/records/20401797. The more user-friendly Git repository is available at https://github.com/GhostFrankWu/PHP-security-research.

A.2.3 Hardware dependencies

A x86_64 machine with Docker support is sufficient for reproducing the exploit-success and mitigation claims. At least 8 GB of RAM and 20 GB of free disk space is recommended.

A.2.4 Software dependencies

Ubuntu 22.04 (or any Linux distribution with Docker support), Docker 24.0 or newer, Python 3.10+ with the `requests` and `tqdm` packages. All PHP toolchain and library dependencies are installed inside the provided Docker images.

A.2.5 Benchmarks

The overhead measurements for the two mitigations use Zend bench, Symfony Demo 2.2.3, and WordPress 6.2 as benchmark workloads. The benchmark drivers and target applications are included in the artifact and are invoked automatically by the provided scripts.

A.3 Set-up

A.3.1 Installation

Unpack the artifact archive and `cd` into the repository root. From the repository root, build the five exploit target images:

```
./build.sh  # Will build five Docker images as:
# ae-cve-2024-2961
# ae-cve-2022-31626
# ae-cve-2019-6977
# ae-ctf-case-a
# ae-ctf-case-b
```

Each build applies the two prototyped PHP defenses (`heap-isolation.patch` and `metadata-ro.patch`) on top of PHP-8.4.0 before compilation, so all builds reflect the same hardened configuration used in the paper. The mitigation evaluation scripts under `Patch-hashtable/` and `Patch-refcnt/` build their own images on demand.

A.3.2 Basic Test

Pick any one of the five built images, start it bound to its expected local port, and confirm the target PHP application responds:

```
docker run --rm -p 8082:80 ae-cve-2024-2961
# in another terminal:
curl -s http://127.0.0.1:8082/ | head
```

A successful basic test produces an HTTP response from the containerized PHP application without any visible error. The container can then be stopped before proceeding to the full evaluation workflow.

A.4 Evaluation workflow

A.4.1 Major Claims

(C1): All five exploits (CVE-2024-2961, CVE-2022-31626, CVE-2019-6977, CTF-Case-A, CTF-Case-B) achieve remote command execution against a PHP-8.4.0 interpreter with all four prototyped heap hardening protections enabled, demonstrated by the remote output of `uname -a` and `date`. This supports the central evaluation claim of the paper, summarized in Table 3 (column "Recon").

(C2): For the three CVE cases (CVE-2024-2961, CVE-2022-31626, CVE-2019-6977), the success rate and HTTP-request counts of the rebuilt exploits match the stability numbers reported in Table 4 of the paper.

(C3): Applying Patch-hashtable prevents the exploit from achieving command execution and from crashing the target. This patch covers the three CVE cases, consistent with the coverage discussed in §8.2.

(C4): Applying Patch-refcnt prevents the provided exploits in all five cases from achieving command execution and from crashing the target. The CTF-Case-A exploit path not covered by our prototypes has been independently addressed by two upstream patches (https://github.com/php/php-src/pull/16765 and https://github.com/php/php-src/pull/19287), as noted in §8.2.

A.4.2 Experiments

(E1): *End-to-end exploitation under heap hardening* [5 human-minutes + 20 compute-minutes + 8 GB disk]: each of the five exploits is run against its corresponding hardened target container and is expected to print the remote `uname` and `date` output. Supports (C1).
Preparation: Build the five target images as described in *Installation*.
Execution: For each case, start the target container bound to its expected port and run the exploit script from the repository root:

```
docker run --rm -p 8082:80 ae-cve-2024-2961
export URL=http://127.0.0.1:8082/
python3 CVE-2024-2961/exploit.py

docker run --rm -p 8083:80 ae-cve-2022-31626
export URL=http://127.0.0.1:8083/
python3 CVE-2022-31626/exploit.py

docker run --rm -p 8080:80 ae-cve-2019-6977
export URL=http://127.0.0.1:8080/
python3 CVE-2019-6977/exploit.py

docker run --rm -p 8084:80 ae-ctf-case-a
export URL=http://127.0.0.1:8084/index.php
python3 CTF-Case-A/exploit.py

docker run --rm -p 8081:80 ae-ctf-case-b
export URL=http://127.0.0.1:8081/dataform
python3 CTF-Case-B/exploit.py
```

Results: Each exploit prints the remote `uname` and `date` output on success. Successful runs of all five cases confirm (C1).

(E2): *Stability test on three CVE cases* [5 human-minutes + 2 compute-hours]: re-run each of the three CVE exploits 100 times and record the success rate and the number of HTTP requests sent. Supports (C2).
Preparation: Use the three CVE target images built in (E1).
Execution: For each of the three CVE cases, run its exploit script in a loop of 100 iterations against a freshly started container; record success/failure and the number of HTTP requests reported by the script.
Results: The observed success rate and the mean and worst-case request counts are expected to match Table 4 of the paper within normal variance. The CVE-2024-2961 and CVE-2022-31626 scripts first scan the default positive probe window and then a small negative fallback window. This covers hosts where the correct libc candidate sits a few pages before the default probe start. This confirms (C2).

(E3): *Patch-hashtable mitigation evaluation* [5 human-minutes + 10 compute-minutes]: apply Patch-hashtable on top of the hardened PHP build and re-run each of the three CVE exploits. Supports (C3).
Preparation: None beyond the artifact checkout; the script rebuilds PHP with the patch applied.
Execution: From the repository root:

```
export CLEANUP=1
Patch-hashtable/CVE-2024-2961/run.sh
Patch-hashtable/CVE-2022-31626/run.sh
Patch-hashtable/CVE-2019-6977/run.sh
```

Results: For each CVE, the exploit no longer produces the `uname` or `date` output and the PHP worker process does not crash. This confirms (C3).

(E4): *Patch-refcnt mitigation evaluation* [5 human-minutes + 40 compute-minutes]: apply Patch-refcnt on top of the hardened PHP build and re-run all five exploits. Supports (C4).
Preparation: None beyond the artifact checkout.
Execution: From the repository root:

```
export CLEANUP=1
Patch-refcnt/CVE-2024-2961/run.sh
Patch-refcnt/CVE-2022-31626/run.sh
Patch-refcnt/CVE-2019-6977/run.sh
```

`Patch-refcnt/CTF-Case-A/run.sh`
`Patch-refcnt/CTF-Case-B/run.sh`
Results: For each of the five cases, the provided exploit no longer produces the `uname` or `date` output and the PHP worker process does not crash. This confirms (C4).

A.5 Notes on Reusability

The artifact is structured so that each of the five exploit cases and each of the two mitigation prototypes is self-contained, i.e., a separate Dockerfile, a separate exploit script, and a separate mitigation run script. Adding a new exploit case requires only a new directory with a Dockerfile and a Python exploit script following the `URL` convention. Evaluating an additional mitigation on the same five cases requires only a new patch directory mirroring the layout of `Patch-hashtable/` or `Patch-refcnt/`, since the run scripts are parameterized by patch path and target case. The same harness can be reused to test other prototype defenses on top of the hardened PHP-8.4.0 baseline.

Beyond verifying that the exploits are mitigated, the artifact also supports measuring the runtime overhead of the two mitigations against Zend bench, Symfony Demo 2.2.3, and WordPress 6.2 using the scripts provided under `Patch-hashtable/` and `Patch-refcnt/`. The overhead measurements reported in the paper were collected on an Intel Core Ultra 9 285HX (24 cores) with 64 GB DDR5-4400 RAM and a 2 TB NVMe SSD, so reproducing the runtime overhead numbers benefits from a similar configuration. The reported full-run runtime overhead is approximately 0.17% for Patch-hashtable and approximately 8.86% for Patch-refcnt, with absolute timings dependent on hardware.

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/.