SHADOWBOUND: Efficient Heap Memory Protection Through Advanced Metadata Management and Customized Compiler Optimization

Zheng Yu, Ganxiang Yang, and Xinyu Xing, *Northwestern University*

https://www.usenix.org/conference/usenixsecurity24/presentation/yu-zheng

This artifact appendix is included in the Artifact Appendices to the Proceedings of the 33rd USENIX Security Symposium and appends to the paper of the same name that appears in the Proceedings of the 33rd USENIX Security Symposium.

August 14–16, 2024 • Philadelphia, PA, USA

978-1-939133-44-1

Open access to the Artifact Appendices to the Proceedings of the 33rd USENIX Security Symposium is sponsored by USENIX.

USENIX Security '24 Artifact Appendix: SHADOWBOUND: Efficient Heap Memory Protection Through Advanced Metadata Management and Customized Compiler Optimization

Zheng Yu
Northwestern University

Ganxiang Yang
Northwestern University

Xinyu Xing
Northwestern University

A Artifact Appendix

A.1 Abstract

In this paper, we present SHADOWBOUND, a unique heap memory protection design. At its core, SHADOWBOUND is an efficient out-of-bounds defense that can work with various use-after-free defenses (e.g., MarkUs, FFMalloc, PUMM) without compatibility constraints. We harness a shadow memory-based metadata management mechanism to store heap chunk boundaries and apply customized compiler optimizations tailored for boundary checking. This artifact is seeking the **Artifacts Available** badge, the **Artifacts Functional** badge, and the **Results Reproduced** badge. To facilitate the artifact evaluation, we have provided multiple Docker environments. These Docker environments are designed to provide main evidence to support the claims of SHADOWBOUND.

A.2 Description & Requirements

A.2.1 Security, privacy, and ethical concerns

All experiments are conducted in Docker, ensuring they will not harm the host computer.

A.2.2 How to access

https://github.com/cla7aye15I4nd/shadowbound/tree/1.0.0

A.2.3 Hardware dependencies

- Processor: We recommend using a 12th Gen Intel i7-12700 CPU with a clock speed of 4.9 GHz to achieve results similar to our experiments. However, comparable hardware may also suffice.
- Memory: A minimum of 32GB RAM.
- Storage: At least 1TB of SSD storage.

A.2.4 Software dependencies

- Docker & Compose
- Ubuntu 22.04

A.2.5 Benchmarks

It is preferable to use the SPEC CPU2017 benchmarks (a license is required). With these benchmarks, you can manually use SHADOWBOUND for compilation. If you do not have access to these benchmarks, you can use our Docker instead. Our Docker provides the SPEC binaries compiled with SHADOWBOUND, allowing you to reproduce the evaluation results.

A.3 Set-up

You should first clone our Github repository to directory named `shadowbound` under your home directory.

A.3.1 Installation

You should use the following command to build the base image of SHADOWBOUND, all evaluation is based on the image.

```
$ cd shadowbound
$ docker compose up --build shadowbound
```

A.3.2 Basic Test

Our basic test includes using SHADOWBOUND to compile nginx and run it. You can achieve this by following the steps below:

1. Navigate to the shadowbound directory.
2. Build and run the Docker container for the nginx evaluation using the provided command.

```
$ cd shadowbound
$ docker compose up --build nginx-eval
```

After running the command, you can access the result at `artifact/nginx/results/shadowbound.txt`, the file content should be like this:

```
Running 1m test @ http://localhost:80/index.html
8 threads and 100 connections
 Thread Stats   Avg      Stdev     Max   +/- Stdev
 Latency       816.19us 418.18us  50.10ms  99.18%
 Req/Sec       14.96k   717.36    30.03k   92.40%
 Latency Distribution
  50%  786.00us
  75%  805.00us
  90%    0.85ms
  99%    1.15ms
7149133 requests in 1.00m, 5.68GB read
Requests/sec: 118955.36
Transfer/sec: 96.77MB
```

A.4 Evaluation workflow

A.4.1 Major Claims

(C1): The system is able to work with other UAF defenses, as proven by the experiments **(E1)**.
(C2): The system can defend against heap out-of-bound bugs, as demonstrated by the experiments **(E2)**.
(C3): The system has a small time overhead, as shown by the experiments **(E3)** and **(E4)**.

A.4.2 Experiments

(E1): *[1 human-minutes + 10 compute-minutes]:* To show SHADOWBOUND can cooperate with other UAF defenses, we show how we use SHADOWBOUND + FFMalloc and SHADOWBOUND + MarkUS to compile nginx and run it, you can run the following command to achieve it:

```
$ TARGET=shadowbound-ffmalloc \
  docker compose up --build nginx-eval
$ TARGET=shadowbound-markus \
  docker compose up --build nginx-eval
```

After running the command, you can access the result at the following two files

- `artifact/nginx/results/shadowbound-ffmalloc.txt`
- `artifact/nginx/results/shadowbound-markus.txt`

Warning: You may encounter a Segmentation fault during the test. It is due to UAF defense issues, not ShadowBound. If this occurs, try running the test again. In our experience, such issues are relatively rare.
Result: You should get similar results as you see in the basic tests.

(E2): *[1 human-minute + 1 compute-hour]:* To show SHADOWBOUND meet the security requirements, we show how we use SHADOWBOUND to handle real world vulnerabilities.
Preparation: First, you should download our pre-built Docker image. If you want to check the building process, you can also build it yourself by following the instructions in our GitHub repository.

```
$ docker pull
    ↪ ghcr.io/cla7aye15i4nd/shadowbound/
    ↪ shadowbound-sec-eval:1.0.0
```

Execution: Enter the docker container and run test script:

```
$ docker run -it
    ↪ ghcr.io/cla7aye15i4nd/shadowbound/
    ↪ shadowbound-sec-eval:1.0.0
    ↪ /root/test.sh
```

Result: You should see results like which means all testcases passed:

```
[+] 2017-9164-9166
[+] 2017-9167-9173
[+] CVE-2006-6563
[+] CVE-2009-2285
[+] CVE-2013-4243
[+] CVE-2013-7443
[+] CVE-2014-1912
[+] CVE-2015-8668
[+] CVE-2015-9101
[+] CVE-2016-10270
[+] CVE-2016-10271
...
```

(E3): *[1 human-minute + 3 compute-hour]:* To show the performance of SHADOWBOUND, we evaluate SHADOWBOUND on the SPEC CPU 2017
Preparation: First, you should download our pre-built Docker image. If you want to check the building process, you can also build it yourself by following the instructions in our GitHub repository.

```
$ docker pull
    ↪ ghcr.io/cla7aye15i4nd/shadowbound/
    ↪ shadowbound-spec2017-eval:1.0.0
```

Execution: Enter the docker container and run test script:

```
$ docker run --privileged -it
    ↪ ghcr.io/cla7aye15i4nd/shadowbound/
    ↪ shadowbound-spec2017-eval:1.0.0
    ↪ /bin/bash
$ python3 /root/scripts/spectest.py | tee
    ↪ /root/spectest.log
```

Result: After running the command, you can check the result at `/root/spectest.log`. Please note that since our original experiment was conducted on a bare metal machine without running any other processes, it is normal for the overhead in the artifact to have some variance compared to the result in the paper.

(E4): *[1 human-minute + 2 compute-hour]:* To show the performance of SHADOWBOUND on the real world application, we evaluate SHADOWBOUND on the nginx and chakra.
Preparation: First, you should build Docker image. If you want to check

```
$ docker pull
    ↪ ghcr.io/cla7aye15i4nd/shadowbound/
    ↪ shadowbound-nginx:1.0.0
$ docker pull
    ↪ ghcr.io/cla7aye15i4nd/shadowbound/
    ↪ shadowbound-chakra:1.0.0
```

Execution: Enter the docker container and run test script:

```
$ ./artifact/nginx/test.sh
$ docker compose up chakra-eval
```

Result: After running the command, You can compare the result at `artifact/nginx/results/native.txt` and `artifact/nginx/results/shadowbound.txt` to see the nginx performance overhead. You can check the ChakraCore performance overhead at `artifact/chakra/results/shadowbound.txt`.

A.5 Notes on Reusability

Our tool is implemented atop LLVM 15, and we provide simple arguments (`-fsanitize=overflow-defense`) to use SHADOWBOUND, which shows high compatibility. In this artifact, we excluded some auxiliary experiments designed to strengthen our claims but which did not affect the paper's conclusions. The reason for this exclusion is the significant time these experiments require. For instance, integrating PUMM involves fuzzing and analyzing each program, a process that can take over 10 days.

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/usenixsec2024/.