CAMP: Compiler and Allocator-based Heap Memory Protection

Zhenpeng Lin, Zheng Yu, Ziyi Guo, Simone Campanoni,
Peter Dinda, and Xinyu Xing, *Northwestern University*

https://www.usenix.org/conference/usenixsecurity24/presentation/lin-zhenpeng

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: <CAMP: Compiler and Allocator-based Heap Memory Protection>

Zhenpeng Lin, Zheng Yu, Ziyi Guo, Simone Campanoni, Peter Dinda, and Xinyu Xing
{zplin, zhengyu2027, n7l8m4}@u.northwestern.edu
{simone.campanoni, pdinda, xinyu.xing}@northwestern.edu
Northwestern University

A Artifact Appendix

A.1 Abstract

The paper proposes a heap protection tool that prevents heap UAF and OOB while introducing mild overhead. The tool is composed of a set of compiler optimization and fast runtime support, which are implemented as an LLVM pass and tcmalloc-based allocator. The paper has two major claims over its security guarantee and overhead under a bunch of benchmarks.

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. The Docker environments are designed to reproduce the evaluation results of CAMP, covering both performance and security evaluations.

A.2 Description & Requirements

A.2.1 Security, privacy, and ethical concerns

All experiments are run in Docker and will not harm the computer.

A.2.2 How to access

All artifacts are available in https://github.com/cla7aye15I4nd/CAMP/tree/a74a3069adb4aeff2426bba1fd6391c7d1fbb405.

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
- Ubuntu 22.04

A.2.5 Benchmarks

It's preferable to utilize the SPEC CPU2016 and SPEC CPU2017 benchmarks. With them, you can manually employ CAMP for compilation. However, if you don't possess these benchmarks, you can alternatively make use of our Docker. Our Docker provides the SPEC binary compiled with CAMP, allowing you to reproduce the evaluation results.

A.3 Set-up

You can use the following command to download the Docker images for each experiment.

```
$ docker pull dataisland/camp-spec
$ docker pull dataisland/camp-juliet
$ docker pull dataisland/camp-nginx
$ docker pull dataisland/camp-chromium
$ docker pull n7l8m4/camp-u22:v2
$ docker pull n7l8m4/camp-u18:v2
$ docker pull n7l8m4/camp-sudo:v1
$ docker pull
    ↪ n7l8m4/camp-chrome-issue:v1
```

A.3.1 Installation

Create containers using the downloaded Docker images

A.3.2 Basic Test

After creating the corresponding Docker container, you can enter the Docker container which created from image `dataisland/camp-juliet` , and input the following command. The expected result is shown in Figure 1.

```
$ cd /root/camp-experiment/juliet
$ python3 test_fp.py 2>/dev/null
```

```
root@74fdab0f52e7:~/camp-experiment/juliet# python3 test_fp.py 2>/dev/null
[5371/5372]CWE416_Use_After_Free/CWE416_Use_After_Free__malloc_free_int64_t_63
[FINISHED]
root@74fdab0f52e7:~/camp-experiment/juliet# python3 test_fn.py 2>/dev/null
CWE122_Heap_Based_Buffer_Overflow : 3870
CWE415_Double_Free : 820
CWE416_Use_After_Free : 394
CWE761_Free_Pointer_Not_at_Start_of_Buffer : 288
[3870/898/0/3870]CWE122_Heap_Based_Buffer_Overflow/s11/CWE122_Heap_Based_Buffer_Overflow__sizeof_struct_84
[FAIL/IGNORE/ALL][0/899/3870] CWE122_Heap_Based_Buffer_Overflow
[820/820]CWE415_Double_Free/s02/CWE415_Double_Free__no_copy_const_01_good1
[FINISHED] CWE415_Double_Free
[394/394]CWE416_Use_After_Free/CWE416_Use_After_Free__return_freed_ptr_18
[FINISHED] CWE416_Use_After_Free
[288/288]CWE761_Free_Pointer_Not_at_Start_of_Buffer/CWE761_Free_Pointer_Not_at_Start_of_Buffer__char_listen_socket_84
[FINISHED] CWE761_Free_Pointer_Not_at_Start_of_Buffer
```

Figure 1: Juliet Results

A.4 Evaluation workflow

A.4.1 Major Claims

(C1): CAMP achieves good performance and memory overhead with the SPEC benchmark and is capable of handling real-world programs. This is substantiated by experiments (E2), (E3), and (E4).

(C2): CAMP able detect all memory corruption in juliet and CVE benchmarks which mentioned in our paper. This is proven by the experiments (E1) and (E5).

A.4.2 Experiments

(E1): *Security Evaluation on Juliet Dataset*
The environment is contained within the Docker image `dataisland/camp-juliet`.
How to: Download and create a container using the image `dataisland/camp-juliet`.
Preparation: None.
Execution: Refer the command below.
Results: If the command does not output `[FAIL]`, it means all test cases have passed.

```
$ cd /root/camp-experiment/juliet
$ python3 test_fp.py 2>/dev/null
$ python3 test_fn.py 2>/dev/null
```

(E2): *Performance Evaluation on NGINX*
The environment is contained within the Docker image `dataisland/camp-nginx`.
How to: Download and create a container using the image `dataisland/camp-nginx`.
Preparation: None.
Execution: Refer the command below.
Results: The expected output of the command should similar with Figure 2.

```
$ cd /root/camp-experiment/nginx
$ ./test_wrk.sh
```

(E3): *Performance Evaluation on Chromiumn* The environment is contained within the Docker image `dataisland/camp-chromiumn`.
How to: Download and create a container using the image `dataisland/camp-chromiumn`.
Preparation: None.

```
root@74fdab0f52e7:~/camp-experiment/nginx# ./test_wrk.sh
Running 1m test @ http://localhost:80/index.html
  8 threads and 100 connections
  Thread Stats   Avg      Stdev     Max   +/- Stdev
    Latency   636.81us  236.44us  21.03ms   98.08%
    Req/Sec    19.10k   642.65    34.22k    90.59%
  Latency Distribution
     50%  616.00us
     75%  627.00us
     90%  645.00us
     99%    0.95ms
  9129150 requests in 1.00m, 7.25GB read
Requests/sec: 151962.41
Transfer/sec:    123.62MB
Running 1m test @ http://localhost:80/index.html
  8 threads and 100 connections
  Thread Stats   Avg      Stdev     Max   +/- Stdev
    Latency   826.83us  171.93us  28.45ms   97.29%
    Req/Sec    14.62k   538.73    29.49k    91.53%
  Latency Distribution
     50%  806.00us
     75%  827.00us
     90%    0.87ms
     99%    1.08ms
  6987130 requests in 1.00m, 5.55GB read
Requests/sec: 116259.95
Transfer/sec:     94.58MB
```

Figure 2: Nginx Results

Execution: Refer to the command below. It is used to test the loading time of a single website. You can replace the website field with another website and you can replace the `camp` with `native` to test the loading time of native chrome.
Results: The command will generate a screenshot picture of corresponding website, and output the loading time. The overhead of CAMP should similar with what our paper reported.
The environment is contained within the Docker image `dataisland/camp-chromiumn`. To run the test, use the command provided below. Note that you can replace the specified website with another of your choice.

```
$ cd /root/chromiumn/src
$ export
    ↪ LD_LIBRARY_PATH=/root/CAMP/
    ↪ build/src/safe_tcmalloc/tcm
    ↪ alloc
$ time ./out/camp/chrome
    ↪ --disable-sync --disable-gpu
    ↪ --headless --screenshot
    ↪ --no-sandbox
    ↪ http://www.gmail.com
```

(E4): *Performance Evaluation on SPEC Benchmark* The environment is contained within the Docker image `dataisland/camp-spec`. To run the test, use the command provided below. You will be able to see the time and memory consuming of each testcases.

How to: Download and create a container using the image `dataisland/camp-chromiumn`.
Preparation: None.
Execution: Refer to the command below. `./camp.sh` and `./native.sh` is used to test the performance of program compiled by CAMP and NATIVE program on SPEC CPU2017 and SPEC 2006.
Results: The command will generate multi files which contain each testcases' time and memory consuming. The overhead of CAMP should similar with what our paper reported.

```
$ cd /root/camp-experiment/spec
$ ./camp.sh
$ ./native.sh
```

(E5): *Security Evaluation on CVE Benchmark*
How to: Download and create container `n7l8m4/*`
Preparation: None.
Execution: In every container, we have set up `test.sh`, `test_OOB.sh`, `test_UAF.sh`, in `/root` directory. Test can run the `test.sh` to automatically run the test cases.
Results: We have set up the results evaluation in test scripts. i.e. the test script can decide if the test succeeds or fails for protecting the cases. When succeed, automated test scripts will tell user the results, like "CAMP Protects OOB Successfully", "CAMP Protects UAF Successfully, poison detected" and etc, if it fails, test script will print "CAMP Failed".

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/usenixsec2024/.