Playing for K(H)eaps: Understanding and Improving Linux Kernel Exploit Reliability

Kyle Zeng, *Arizona State University;* Yueqi Chen, *Pennsylvania State University;* Haehyun Cho, *Arizona State University and Soongsil University;* Xinyu Xing, *Pennsylvania State University;* Adam Doupé, Yan Shoshitaishvili, and Tiffany Bao, *Arizona State University*

https://www.usenix.org/conference/usenixsecurity22/presentation/zeng

This artifact appendix is included in the Artifact Appendices to the Proceedings of the 31st USENIX Security Symposium and appends to the paper of the same name that appears in the Proceedings of the 31st USENIX Security Symposium.

August 10–12, 2022 • Boston, MA, USA

978-1-939133-31-1

A Artifact Appendix

A.1 Abstract

This artifact requires machines with x86_64 architecture. At least 2 logical cores and 2 GB RAM is required for running the experiments. Since the experiment is resource-consuming, more cores and RAM settings are recommended. The artifact has been containerized, so it runs on most Linux-based operating systems. It has been verified to work on Ubuntu-20.04.

Our paper is about empirically evaluating the reliability enhancement brought by kernel exploit stabilization techniques; the empirical experiment forms the foundation for our paper.

To validate the experiment, one can repeat the experiment included in the artifact and compare the result with what we present in the paper. Since our experiment result can be slightly affected by the underlying hardware, we expect the result on another machine to be slightly different from what is in the paper. However, the effect of each exploit stabilization technique should not change. In other words, if a technique improves exploit reliability for a specific CVE in the paper, it should behave the same in repeated experiments. However, the improvement may be slightly different.

A.2 Artifact check-list (meta-information)

- **Binary:** Compiled vulnerable Linux kernels are included. New vulnerable kernels can be compiled as well using scripts/kernel_builder/build_kernels.py.
- **Data set:** The artifact requires a dataset of vulnerable Linux kernels and corresponding exploits. They are included in exploit_env/CVEs/.
- **Run-time environment:** The artifact depends on "docker" software. It requires a Linux-based host OS to build the container image. It has been verified to work on Ubuntu-20.04. The OS inside the container is Ubuntu-18.04. root access on the host OS is required.
- **Execution:** The experiment should be run on a machine without other processes running. The existence of other processes may interfere with the experiment and affect the result.
- **Metrics:** The metric used in the experiment is the success rate of exploits.
- **Output:** The output of the experiment is the success rate of each exploit. The number of success/failure runs is saved in a JSON file in the output folder.
- **Experiments:** To prepare and run the experiment, one needs to 1. clone the artifact repository from https://github.com/sefcom/KHeaps, 2. build the docker image as instructed in README.md, and 3. run an evaluation experiment for each CVE as instructed.

  The expected result is included in the paper. We expect slightly different success rates for each exploit. However, the effect of each exploit stabilization technique should be the same. In other words, if a technique improves reliability in the paper, the behavior should stay the same in repeated experiments with a slightly different improvement. The same applies to the cases where techniques hurt exploit reliability.
- **How much disk space required (approximately)?:** We expect the whole experiment to take about 20GB disk space after disabling logging (the "-nl" option in "vuln_tester.py").
- **How much time is needed to prepare workflow (approximately)?:** We containerized the whole experiment. It takes about 10-15 minutes to build a disk image for the VM and docker image for the evaluation.
- **How much time is needed to complete experiments (approximately)?:** To complete the 2CPU+2GB RAM experiment (each VM configured with 2 virtual CPU and 2GB RAM), it requires 1680 CPU days. The time needed can be reduced by increasing the number of CPUs. For example, it can be finished in 42 days with a 40-core machine.
- **Publicly available (explicitly provide evolving version reference)?:** The artifact is publicly available at https://github.com/sefcom/KHeaps
- **Code licenses (if publicly available)?:** MIT license.
- **Data licenses (if publicly available)?:** MIT license.
- **Archived (explicitly provide DOI or stable reference)?:** Stable reference on GitHub: https://github.com/sefcom/KHeaps/tree/22b35da5f9f259f5cc8f349da9f791d9428295e4.

A.3 Description

A.3.1 How to access

Clone git repository from https://github.com/sefcom/KHeaps/tree/22b35da5f9f259f5cc8f349da9f791d9428295e4.

A.3.2 Hardware dependencies

N/A.

A.3.3 Software dependencies

The experiment requires a Linux-based OS to build. Ubuntu-20.04 is preferred.

One of the experiments requires nested-kvm parameter in kvm-intel kernel module. One can check whether it is enabled by checking /sys/module/kvm_intel/parameters/nested. If it is enabled, the pseudo file should return "Y".

The experiment depends on "docker" software.

A.3.4 Data sets

The dataset is included in the public KHeaps repository. It consists of two parts: 1. vulnerable kernels are pre-compiled and included in the repository. 2. kernel exploits are included in "poc" folders.

A.3.5 Models

N/A

A.3.6 Security, privacy, and ethical concerns

N/A

A.4 Installation

Use the following command to build the docker image.

1. git clone https://github.com/sefcom/KHeaps
2. cd KHeaps
3. cd scripts/create-image/ && ./create-image.sh && cd ../..
4. docker build -t kheap .

The above process takes about 10 minutes to finish.

At this stage, a docker image called "kheap" should be created. One can verify this by making sure its existence in the output of "docker images".

A.5 Experiment workflow

The experiment aims to evaluate the success rates of exploits against vulnerable kernels. For each CVE, it compiles all the corresponding exploits first and then launches VMs with the vulnerable kernel. It then copies exploits into the VMs using ssh and runs exploits inside the VMs until the VMs crash. The VM monitor will extract the crash logs and determine whether the exploits succeed or not. We regard an exploit as successful if the VM crashes at an attacker-controlled program counter, which demonstrates the control flow hijacking capability of the exploit.

A.6 Evaluation and expected results

Main claim: Exploits equipped with the combo technique outperforms realworld exploits in terms of reliability. This can be verified by running realworld exploits and combo exploits and comparing their success rates. In our evaluation, the success rates of realworld and combo exploits are 54.30% and 91.15% (67.86% improvement). In repeated experiments, we expect combo exploits to have at least 50% improvement over realworld exploits.

Key results:

- Defragmentation improves reliability for OOB exploits. We expect exploits equipped with Defragmentation technique to have a significantly higher success rate compared with baseline exploits. This can be verified by running baseline exploits and exploits equipped with Defragmentation technique.
- Defragmentation may hurt reliability for UAF or DF exploits. We expect that Defragmentation does not significantly improve reliability for UAF and DF exploits and significantly hurts the reliability for some of them. For example, CVE-2017-2636.
- Heavy workload hurts exploit reliability, but exploits can still achieve high success rates. This can be verified by running exploits in both idle and busy settings. One should observe exploit success rate degradation in busy settings and that more than half exploits equipped with Multi-Process Heap Spray can achieve more than 90% success rates.
- Multi-Process Heap Spray generally outperforms Single-Thread Heap Spray. This can be verified by running both Multi-Process Heap Spray and Single-Thread Heap Spray exploits. We expect Multi-Process Heap Spray to outperform Single-Thread Heap Spray in all settings with one exception: CVE-2017-6074 in idle settings, potentially also in busy settings.

A.7 Experiment customization

To evaluate exploits for a new CVE, one needs to add a new folder in "CVEs" folder and specify its maximum runtime in "setup.json".

To limit the evaluation to some specific exploits, one can add filters in "make_pocs" function in "vuln_tester.py" script.

A.8 Notes

N/A.

A.9 Version

Based on the LaTeX template for Artifact Evaluation V20220119.