USENIX Security '24 Artifact Appendix: <SeaK: Rethinking the Design of a Secure Allocator for OS Kernel>

A Artifact Appendix

We aim to apply for Available, Functional and Reproduced Badges.

A.1 Abstract

This artifact is applying for an Artifacts Available badge, an Artifacts Functional badge, and an Results Reproduced badge. For Available badge, all source code and scripts can be found in https://github.com/a8stract-lab/SeaK/tree/v1.1. For Functional badge, we will use CVE-2021-4154 as an example to demonstrate the workflow to create an eBPF program from scratch. For Reproduced badge, we have prepared LMbench and phoronix test suite to thoroughly reproduce the performance and memory overhead. All the experiments will be done in a virtual machine for convenience and detailed workflow is in the github repository.

A.2 Description & Requirements

In this section, we first describe whether reproducing our artifacts will risk the evaluator's machine security, followed by approaches to accessing our artifacts. Then, we describe hardware dependencies and software dependencies before listing the benchmarks.

A.2.1 Security, privacy, and ethical concerns

SeaK aims to protect the OS kernel through the eBPF ecosystem. To enable SeaK, the kernel needs additional eBPF helper functions before being compiled and installed, which is destructive to some extent. Therefore, to make evaluators feel safe, we prepared a kernel image and a root filesystem for evaluators. As such, evaluators can download the image and reproduce our results in an isolated environment, ensuring the safety and privacy of the host machine. The access for the image can be found in Section A.2.2. Furthermore, it is important to note that all vulnerabilities included in the artifact are publicly available and have been addressed in the mainstream kernel. Therefore, there are no security, privacy, or ethical concerns regarding the open-source community. The artifact builds upon resolved issues, and its purpose is to contribute to the knowledge and advancement of the field.

A.2.2 How to access

The complete artifacts are available in a public Github Repo https://github.com/a8stract-lab/SeaK/tree/v1.1 , which includes three main components: eBPF programs and corresponding scripts for evaluation, source code developed in SeaK, manuals and examples for evaluators to quickly understand the key idea of SeaK. The virtual machine file system image can be downloaded from https://tinyurl.com/mwsub255, which should be put under the directory '1-evalution'. Due to the space limit, we cannot list all details here. Instead, they are clearly stated in the repo for evaluators to follow. The artifacts provided in the Github Repo are sufficient for evaluators to reproduce.

A.2.3 Hardware dependencies

To completely reproduce SeaK, we recommend the following minimum hardware configurations: ① an Intel CPU with VT-X virtualization feature, ② 32GB or larger memory, and ③ at least 300GB disk space (for LLVM compilation).

A.2.4 Software dependencies

It is preferable to perform the evaluation on the Ubuntu Linux distro, especially the 22.04 desktop which is the same OS for SeaK development. The OS is supposed to include essential packages such as debootstrap, qemu-system-x86_64, openssh and wget. These packages are necessary for setting up the evaluation environment and conducting runtime evaluations. Besides, it is advised not to utilize Docker for the evaluation process because the artifact necessitates the use of two separate terminals - one for displaying the result when running the vulnerabilities programs or benchmarks and another for running bpf programs.

A.2.5 Benchmarks

The exploits for vulnerabilities used as test cases have been collected and provided in the Github Repo. We used Phoronix-benchmark and lmbench for performance measurement which is publicly available online

A.3 Set-up

In this section, we focus on the installation and testing of SeaK using the kernel image and a root filesystem we provided for the sake of ethics (See Section A.2.2). The evaluator can boot up the kernel using QEMU. Due to the space limit, we move the detailed instruction for reproducing SeaK from scratch in the Github Repo.

A.3.1 Functional

To present the functional of the SeaK, we take CVE-2021-4154 as an example, demonstrate the workflow to create an eBPF program from scratch. At first, system admin may receive a bug report showing the error sites and objects of the heap memory corruption. Then, system admin use the static analysis tools `hot\_bpf\_analyzer` to extract the alloction and release sites of those heap objects. After that, SeaK generate the BPF AA(atomic Alleviation) programs by taking the allocation and release sites as parameter. BPF AA programs are loaded in the kernel to seperate the location of struct file objects, and the exploitations will be prevented. The system will not be taken over, though there is proabilitity that the system may panic

A.3.2 Reproduce

To prove the results can be reproduced, we design 2 experiments to thoroughly analyze the performance and memory overhead of existing features and SeaK. To make the experiments process easier, we provides virtual machines to accelerate the evaluations, but the whole process may still take more than 20 hours.

A.3.3 Installation

None

A.3.4 Basic Test

After evaluators pull the github repository and run `evaluate.sh`, there will be 2 terminals pop up. In figure 1, the left terminal 1 boots up the virtual machine, and waits to be logged in, and the right terminal 2 has already logged in through ssh. Evaluators can login the virtual machine with the user name root and no password is needed. After that, evaluators can run ls command on either terminal, and there will be three directories and one tar file, including bpf-evaluation, POCs, scripts and lmbench.tar.gz. The bpf directory contains all compiled BPF prevention programs. The POCs directory contains the exploits of vulnerabilities. The scripts directory contains evaluation scripts that need evaluator to execute in the virtual machine. The file lmbench.tar.gz is one of the benchmarks.

A.4 Evaluation workflow

The specific details can be found in the Github Repo https://github.com/a8stract-lab/SeaK/tree/v1.1

A.4.1 Major Claims

(C1): The vulnerabilities can be protected from being exploited.
(C2): The overhead of the existing feature is the same as stated in the paper Section 3.2 and 3.3
(C3): The overhead of the SeaK is negligible as stated in paper Section 8.2 and 8.3

A.4.2 Experiments

(E1): [Sythesize bpf programs] [30 human-minutes + 5 compute-hour + 10GB disk]:
Execution: 1) analyze bug report to get the vulnerable, sensitive objects. 2) build up docker image 3) compile the specific LLVM compiler and the analyzer 4) compile the kernel with the specific LLVM compiler 5) run the static analyzer to get the allocation sites 6) generate and compile the BPF program
Results: a synthsized bpf program "hotbpf_effective" can be found under `2-source-code/linux-5.15.106/samples/bpf`.

(E2): [Test if the bpf programs can prevent the exploits] [30 mins human-hour]:
Execution: Simply execute `evaluate.sh` under "1-evaluation" and two terminals will pop up. On one of the terminals, run the bpf program which has just been compiled. On the other terminal, run the POCs we prepared.
Results: After the attack, in figure 2 we can see the system is still functioning, but the POCs cannot get `/bin/bash` shell.

(E3): [Test the overhead of existing features] [1 human-hour + 12 compute-hour]:
Execution: 1) run `evaluate-vanilla.sh` and run `EF_vanilla.sh`, then close the virtual machine 2) run `evaluate-C1.sh` and run `EF_C1.sh`, then close the virtual machine 3) run `evaluate-C2.sh` and run `EF_C2.sh`, then close the virtual machine 4) run `evaluate-C3.sh` and run `EF_C3.sh` 5) run `EF_analysis.sh`
Results: Result can be viewed in directory 'Results'. EF_lmbench.xlsx is the result of lmbench and EF_phoronix.xlsx is the result of phoronix. EF_memory_overhead.pdf shows the memory overhead of the existing feature when running lmbench.

(E4): [Test the overhead of SeaK] [24 compute-hour]:
Execution: At the beginning of this part, run `evaluate-SeaK.sh` to boot up the virtual machine. For this part, we provide two options. Because lmbench is a widely-used but very old benchmark (back to 1990s), some of the testing results may be unstable.

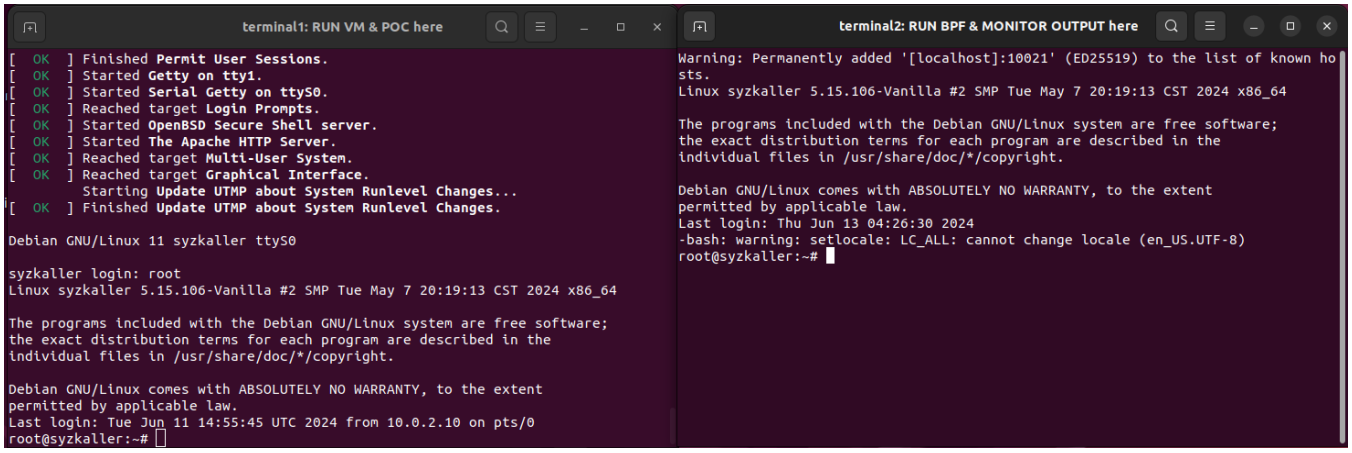

Figure 1: execute `evaluate.sh` to boot up the virtual machine, two terminals pop out.

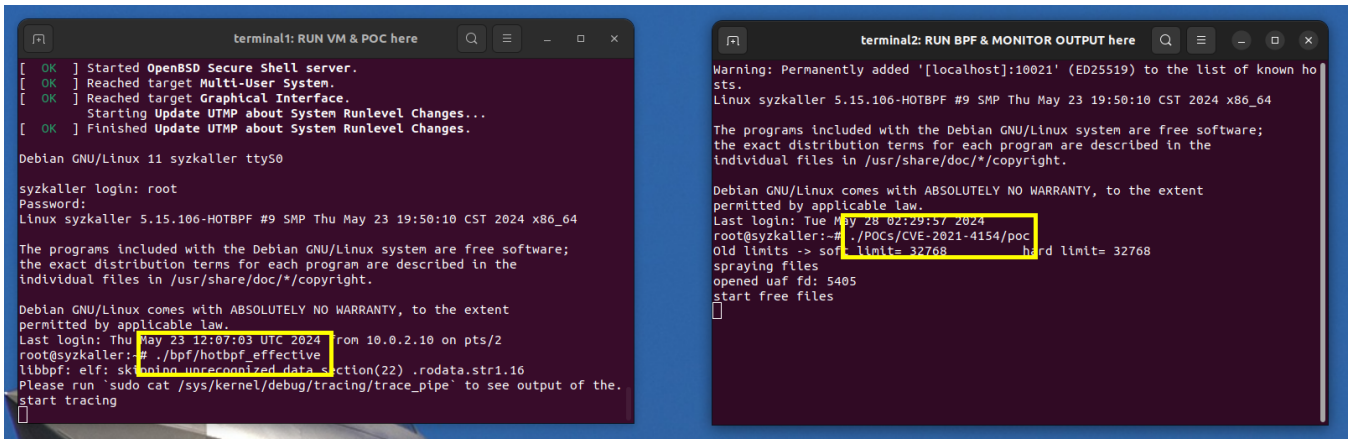

Figure 2: The functionality of SeaK, exploitation in the right terminal cannot corrupt the system that is protected by the SeaK AA eBPF program in the left terminal.

Option 1: Execute `SeaK.sh` to run the whole experiments which takes about 12 hours. This option runs lmbench 6 times (5 times for performance overhead, 1 time for memory overhead) and phoronix 1 time for each feature. In this option, most of the data can be stable and believable.

Option 2: Execute `SeaK-precise.sh` to run the whole experiments which cost about 24 hours. This option runs lmbench 11 times (10 times for performance overhead, 1 time for memory overhead) and phoronix 1 time for each feature. In this option, we can effectively get rid of most of the noise. The results can be more precise than option 1.

Results: Result can be viewed in directory 'results'. `SeaK_lmbench.xlsx` is the result of lmbench and `SeaK_phoronix.xlsx` is the result of phoronix. `SeaK_memory_overhead.xlsx.pdf` shows the memory-overhead of SeaK when running lmbench.

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/.