USENIX'23 Artifact Appendix: BoKASAN: Binary-only Kernel Address Sanitizer for Effective Kernel Fuzzing

Mingi Cho, Dohyeon An, Hoyong Jin, and Taekyoung Kwon, Yonsei University

A Artifact Appendix

A.1 Abstract

This artifact contains the source code of BoKASAN and the necessary data for the experiments. For Artifact Functional, two experiments are proposed, POC test and kernel fuzzing. The recommended environment for the experiment is Ubuntu 20.04 running on the machine with a multi-core x86_64 CPU and >=8GB of RAM. Using more CPU cores can reduce compile time. We expect artifact evaluation to require three human-hours and 12 compute-hours.

A.2 Description & Requirements

A.2.1 Security, privacy, and ethical concerns

None.

A.2.2 How to access

Access the Github repo at https://github.com/seclab-yonsei/bokasan/tree/usenix-ae

A.2.3 Hardware dependencies

- Multi-core x86_64 CPU (e.g., Intel i5, i7)
- >=8GB RAM
- >=100GB HDD/SSD

A.2.4 Software dependencies

- Ubuntu 16.04, 18.04, or 20.04
- gcc-6 or 7
- qemu-system-x86
- Syzkaller

A.2.5 Benchmarks

None.

A.3 Set-up

A.3.1 Installation

To install BoKASAN, download the target kernel and build it with the *'CONFIG_FUNCTION_TRACER'* flag set. Then, compile the loadable BoKASAN module according to the target kernel version. Finally, create a Linux image and insert the compiled BoKASAN module. Detailed installation instructions are described in README.md.

A.3.2 Basic Test

When the BoKASAN module is successfully compiled and the *.ko* file is created, run the target kernel using QEMU and load the module. If the installation is successful, we can find that the BoKASAN module is loaded on the target kernel using the *lsmod* command. We provide *script/qemu.sh* to run the QEMU.

A.4 Evaluation workflow

A.4.1 Major Claims

(C1): *BoKASAN detects out-of-bounds and use-after-free bugs in kernels to which KASAN is not applied. This is proven by the experiment (E1) described in Section 5.1 whose results are reported in Table 5.*

(C2): *BoKASAN detects out-of-bounds and use-after-free bugs when fuzzing kernels to which KASAN is not applied. This is proven by the experiment (E2).*

A.4.2 Experiments

(E1): *[POC test] [30 human-minutes + 1 compute-hour]: Reproduce Table 5 of the paper. Execute 15 POC codes on the target kernel.*
How to: Execute 15 POCs that trigger out-of-bounds and use-after-free bugs on the target kernel running on QEMU. The detailed process is described in *README.md*.
Preparation: Download and compile Linux kernel v4.19 without 'CONFIG_KASAN' flag. Make a Linux image including the BoKASAN module. This is prepared during the installation mentioned above. Then,

execute *compile.py* and *mount.sh* in the *poc/syz* directory. As a result, a Linux image containing POC binaries is created.

Execution: Run *scripts/qemu.sh* to boot the target kernel. After the target kernel boots, executes the POC binaries under the *poc_syz* directory. Each POC is located in the *"vuln type"_"vuln name"* directory as the executable binary named *repro_setpid*. Execute one of the POC, wait for about 15 seconds, and then terminate QEMU using the *ctrl a+x* command. Repeat the above steps to test all of the POCs.

Results: When BoKASAN successfully detects a bug, the message *"BUG: KASAN: ..."* is printed as *dmesg*.

(E2): *[Fuzzing test] [30 human-minute + 6 compute-hour]: Performing fuzzing on the kernel to which KASAN is not applied using Syzkaller.*

How to: Fuzzing the kernel to which KASAN is not applied using Syzkaller. The detailed process is described in *README.md*.

Preparation: Download Syzkaller and patch it using *syzkaller/syz.diff*. Then build Syzkaller following their guidelines. Compile target kernel with KCOV and without KASAN. Fuzzing Linux kernel 4.19 with the Linux image including BoKASAN.

Execution: Run the *scripts/run_fuzz.sh* to perform fuzzing.

Results: We can see the KASAN log in Syzkaller's web interface or in the results directory when BoKASAN detects OOB and UAF bugs.

A.5 Version

Based on the LaTeX template for Artifact Evaluation V20220926. Submission, reviewing and badging methodology followed for the evaluation of this artifact can be found at https://secartifacts.github.io/usenixsec2023/.