USENIX Security '24 Artifact Appendix: Endokernel: A Thread Safe Monitor for Lightweight Subprocess Isolation

Fangfei Yang
Rice University

Bumjin Im[1]
Amazon.com

Weijie Huang
Rice University

Kelly Kaoudis
Trail of Bits

Anjo Vahldiek-Oberwagner
Intel Labs

Chia-Che Tsai
Texas A&M University

Nathan Dautenhahn
Riverside Research

A Artifact Appendix

A.1 Abstract

The Endokernel is an intra-process security monitor that isolates memory at the granularity of subprocesses with low overhead. This appendix presents the test suites aimed at providing a reproducible environment for running the Endokernel, enabling users to utilize it and execute the benchmarks discussed in the paper. The artifacts include a test harness that leverages Docker for a containerized execution environment. You can also run our tests using the provided runq runtime or on bare metal systems. The Endokernel requires CPU support for Memory Protection Keys (MPK).

A.2 Description & Requirements

We provide several tools and pre-built binaries to assist you with your evaluation. Our Test Suite and related test scripts automatically download and compile all necessary tools. This Test Suite requires Ubuntu 20.10 to run. For those not using this system, we offer a comprehensive Docker image based on Ubuntu 20.10, which packages the fully compiled Test Suite for immediate use.

Optionally, we provide a modified Linux kernel necessary for tests that involve signals. This is not required for AE, but if you want to use Endokernel's signal features, you can use the runq runtime or install the kernel we provide on Ubuntu 20.10. However, please note that runq, which is based on QEMU, cannot perform all tests.

Ensure you have more than 50 GB of disk space available to run the benchmarks.

A.2.1 Security, privacy, and ethical concerns

The benchmarks themselves do not impact the external environment; however, we have observed instability in the filesystem leading to system crashes with the base kernel version we use. This issue is inherent to the original kernel and is not related to our patches. In our setup, which exclusively uses ext4 partitions, these instabilities are generally resolved upon rebooting. However, please be mindful of the potential effects caused by filesystem instability.

A.2.2 How to access

You can access all the materials required for the Artifact Evaluation through the following links:

https://github.com/endokernel/test/releases/tag/after-ae

A.2.3 Hardware dependencies

PC with MPK

A.2.4 Software dependencies

On the Ubuntu platform, to run our test suite on bare metal, you will need the following packages: `build-essential`, `git`, `cmake`, `libncurses-dev`, `gawk`, `flex`, `bison`, `openssl`, `libssl-dev`, `dkms`, `libelf-dev`, `libudev-dev`, `libpci-dev`, `libiberty-dev`, `autoconf`, `libdwarf-dev`, `libdw-dev`, `libaio-dev`, `libpcre3-dev`, `libtool`, `uuid-dev`, `libblkid-dev`, `apache2-utils`, `automake`, `libtool-bin`, `wget`, `dwarves`, `curl`, `apache2-utils`, `psmisc`, `tcl`.

If you are using Docker or runq, you only need docker and/or install our modified runq. When using only Docker, the host kernel should be at least 5.11.

Additionally, you will need to install Git Large File Storage (LFS) because some binary files are stored in the code repository via LFS.

A.2.5 Benchmarks

None

[1] Authored before joining Amazon.com

A.3 Set-up

A.3.1 Installation

Install/Build Docker Image for Test Suite You can checkout our Endokernel Test Suite and compile the Docker image yourself using with `make image`. The image will require approximately 10GB of disk space.

Or, you can download and load our Docker image with:

```
docker load < intravirt-env.tar.gz
```

The image is based on the old test suite script; you need to overwrite it (`./testcases/`) with the latest script (the binary hasn't changed).

The default Docker image only uses the dispatch_eiv (`nex-sud` in the paper) configuration, as we do not primarily compare between configurations. You can modify `script/iv.sh` to compile other configurations.

Setup Runq (Optional) If you are using runq, please ensure you have Docker functioning properly, then follow these steps and instructions in the runq repository:

1. Download the Runq v1.0.2 and extract it to `/var/lib/runq`.
2. Run the script `/var/lib/runq/qemu/mkcerts.sh` to configure the certificates.
3. Ensure that the `vhost_vsock` kernel module is enabled.

The runq environment has more complete signal compatibility, though this is not necessary for AE; instead, it causes errors in some tests.

```
docker run -ti --rm \
--security-opt=seccomp:unconfined \
intravirt-env
```

Build Test Suite on bare metal (Optional) Run the command `make build-prog`. By default, this command utilizes prebuilt binary files. Note that you need to place our test suite in the directory `/intravirt` to do this. Alternatively, you must set the `USE_PREBUILT` variable to `0` to disable the use of prebuilt binaries, as the glibc prefix is specified during the configuration process in the pre-built binaries.

Install Modified Kernel and Glibc (Optional) If you are using Ubuntu 20.10, you can also install our provided kernel and glibc to run tests designed for signals without using runq. You only need to install the kernel available at this link and the glibc from this link.

A.3.2 Basic Test

Initiate with the following command:

```
docker run -ti --rm \
--security-opt=seccomp:unconfined \
intravirt-env
```

If you want to use runq, add `-runtime runq`. If you are on bare metal, you can skip this step.

For bare metal and runq, you can verify that the current kernel version is Linux 5.9.8-arch1-cet by running:

```
uname -a
```

Next, navigate to the `testcases` directory and execute the command:

```
python ./nginx.py
```

You should observe benchmarks for both the baseline and Endokernel running.

Finally, go to the `/intravirt/result` directory. You should find generated CSV files where the results for baseline and dispatch_eph are similar and non-zero.

A.4 Evaluation workflow

A.4.1 Major Claims

(C1): Endokernel has been tested on various applications including lighttpd, Nginx, curl, SQLite, and zip, and corresponding performance metrics have been obtained. This is proven by the experiments (E1) whose results are reported in 6.2.2 with Figure 7.

(C2): Endokernel has been tested on LMBench for system call overhead and sysbench for thread scalability as microbenchmarks, and corresponding performance metrics have been obtained. This is proven by the experiments (E2) whose results are reported in 6.2.1 with Figure 4,5 and 6.

A.4.2 Experiments

(E1): *[Application Benchmarks] [30 human-minutes + about 1 computer-hours + 15GB disk]:* Benchmark applications under Endokernel and compare it with the baseline.
Preparation: Switch to the `testcases` directory.
For curl tests, navigate to the `www` directory and execute the following command to create a necessary file:

```
cd www
dd if=/dev/random of=1g.bin \
bs=1024 count=1048576
```

You may skip this step if you are using runq, as nginx under runq is unable to serve 1GB files, thereby rendering this test unexecutable.
Execution: Execute the following Python scripts to run the benchmarks:

```
python ./lighttpd.py
python ./nginx.py
python ./zip.py
python ./curl.py
python ./sqlite.py
```

In order to test files larger than 4k, you need to modify the datasizes in nginx.py
We have set the `tries` parameter relatively low. You can modify this setting in `variables.py`. Some tests modify `variable.tries`, which overrides the value set in `variables.py`. Adjustments to `tries` for these specific tests will need to be made individually.
Results: In the `result` directory, a .csv file will be created for each test/configuration conducted, which can be used to calculate overhead. Due to limitations imposed by runq, some tests may not run as expected.
Example Results:

```
cat nginx_dispatch_eiv.csv
0k,1k,2k,4k,
137.00,670.62,1209.93,2221.56,
133.37,658.62,1233.54,2180.99,

cat nginx_baseline.csv
0k,1k,2k,4k,
133.10,685.60,1227.32,2409.75,
133.08,694.01,1237.02,2319.34,

cat zip.csv
baseline,dispatch_eiv,
25.357,34.433,
16.935,33.200,

cat curl.csv
baseline,dispatch_eiv,
5.342,1.691,
9.400,6.506,
```

To better interpret these results, you can use `calc_overhead.py result_folder test_name` to convert it to the numbers in the figures.

(E2): *[Microbenchmarks using LMBench and sysbench] [10 human-minutes + about 10 computer-minutes + 1GB disk]:* Microbenchmarking Endokernel and compare it with the baseline.
Preparation: Switch to the `testcases` directory.
Execution: Execute the following Python scripts to run the benchmarks:

```
python ./lmbench.py # Figure 4
python ./file_bw.py # Figure 5
python ./sysbench.py # Figure 6
```

Results: In the `result` directory, a .csv file will be created for each test/configuration conducted, which can be used to calculate overhead. Due to limitations imposed by runq, some tests may not run as expected.
Example Results:

```
cat sysbench_baseline.csv
1,,2,,4,,8,,16,,
memory,fileio,memory,fileio,memory,fileio,
    memory,fileio,memory,fileio,memory,
    fileio
176.35,6859.56,217.47,6102.65,
138.53,6164.54,224.41,6287.81,

cat ../result/sysbench_dispatch_eiv.csv
memory,fileio,memory,fileio,memory,fileio,
    memory,fileio,memory,fileio,memory,
    fileio
136.06,6288.02,223.91,6217.23,
139.51,6252.23,220.61,6234.21,
```

To better interpret these results, you can use `calc_overhead.py result_folder test_name` to convert it to the numbers in the figures.
We noticed that sysbench performance drops with 16 threads when using kernel 6.8 with this test suite. We believe this is due to kernel version differences, as the same binary does not show this issue with a 5.9 kernel.

A.5 Notes on Reusability

Endokernel's signal virtualization layer and the design for multithread safety can be utilized by other projects to construct parts of their monitors. Additionally, some of Endokernel's syscall policies can also be adopted by other works. However, being limited by its status as a research prototype, these components may not be perfect or easily reusable as is. We are currently rewriting parts of the design using a C++ or Rust to enhance their portability and usability.

To utilize Endokernel effectively, you will likely need the Endokernel repository and the modified glibc rather than the related test suite. Endokernel can operate on a normal Linux supporting MPK if the application is not using signals. Installation can be conducted using the methods provided on the project page. Libraries related to isolation are located at libiso. By integrating these libraries into the code, Endokernel is enabled to protect corresponding entry points and memory using MPK.

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/.