Endokernl with Software Fault Isolation

Zhenghui Guo
Rice University

Jerry Zhang
Rice University

Abstract

Endokernel provides a new way to easily add efficient and extensible security architecture by using Intel technology Erim with memory protection keys(MPK) to implement memory isolation at the intra-process level, separating the trusted and untrusted domain set by the Erim. As one of the memory isolation mechanisms available from Intel for the x86 platform on the hardware side, MPK has some limitations concerning architecture dependency that hinder its universal adoption. In order to get rid of the limits traped by the hardware side, we'll apply another software mechanism, SFI, for memory isolation to achieve architecture independency and compare performance costs. In this project, we'll demonstrate the benefit of using SFI other than MPK in our work to build Endokernal in a portable way to suit a modern diverse architectural environment.

1 Introduction

The Endokernel introduces a new virtual machine abstraction for representing subprocess authority, enforced by an efficient self-isolating monitor that maps the abstraction to system-level objects (processes, threads, files, and signals)[1]. It also provides memory isolation using Intel hardware technology, MPK and Erim[5]. The MPK is a recent hardware-based technology proposed by Intel to enforce memory protection within the same address space, allowing it to separate and provide privileged memory access of the same process to its own address space. Compared to traditional inter-process memory isolation which requires a context switch and changing page table when switching between two processes, it also provides fast authorization check for all memory access with the PKRU register and the protection bits stored in each page table entry. However, there are several drawbacks to Erim implementation. 1) Erim is based on MPK which is a mechanism only available for x86 CPUs and lacks support for recently fast-growing ARM processors. 2) The MPK could still suffer from malicious attacks such as hackers injecting code to execute the update PKRU instruction.

To provide architecture-independent memory isolation and to further use the Endokernel in CPUs based on other architectures, such as ARM CPUs, we are working on a universal memory isolation technique for the Endokernel. We further explore software fault isolation(SFI)[6] as a new mechanism to separate memory access. The SFI splits an address space into different fault domains and prevents memory access from one fault domain to another by checking the target address both at compile time for static memory access and runtime for dynamic memory access. It's widely recognized that the SFI could be slower than MPK since it needs to check all dynamic memory access by comparing the target segment identifier with the current one at runtime. The cross-domain RPC communication that SFI proposed is also much slower than the WRPKRU instruction used by MPK because that includes extra instructions for saving and restoring registers and preparing call stubs and return stubs. Nevertheless, the question for our project is: *how much faster would MPK be over SFI and would there be some corner cases where the SFI would be faster than MPK?* Additionally, using the advantage of the SFI, a software-based technique that would be architecture-independent, we further install the Endokernel incorporated with SFI on ARM CPU processors and evaluate its performance there. There

are three goals in our project: **1**) **We will replace Erim(MPK) with SFI in Endokernel. 2**) **We are going to implement Endokernel in ARM. 3**) **We will use CPI+MPK to achieve a system call.**

2 Background

2.1 Endokernel

The operating system and kernel industry has been thinking about memory isolation as the complexity of operating systems and software grows. The most common and popular method for memory isolation is the paging and page table which provides process-level memory isolation. However, the need for more isolation beyond just achieving process-level isolation is growing. If users want to isolate a piece of memory, they have to create a new process that causes overhead when doing context switches and changing page tables. Not until recently, this brought attention to both the software and hardware industry and intel proposed MPK as new hardware primitive to support intra-process memory isolation and thread-local permission control on groups of pages[3]. The Endokernel further uses the MPK to support its memory isolation and protection mechanism, but the MPK has the downside of only supporting x86 CPUs and limiting the Endokernel's portability. We switch our attention to a more universal architecture-independent approach, SFI.

2.2 Software Fault Isolation

SFI uses inline reference monitors within a single memory space. The core technique of SFI is to allocate a logical enough "fault domain", and at the same time, check each memory access at compile time and runtime to ensure they do not go beyond their fault domain. The SFI also provides low-latency cross-fault-domain RPC calls to allow communications between fault domains and isolating syscalls via delegating all syscalls to a trusted domain with an arbitration code to determine the authority to make that call. Recently, Apple silicon has been built based on ARM architecture; Hardware like the Apple M1 chip has shown they are much more energy-efficient than x86, as apple silicon almost replaced the role of x86 in Mac, it already occupies a giant chunk of the computer market in recent years. Paying more attention to the Arm architecture seems like a mandatory task for future Endokernal development.

2.3 Code Pointer Integrity

Endokernal provides W xor X to limit reading and writing to improve system calls. This method is very good, but it increases the complexity of the policy. We try to use a very popular method to widely use CPI[2] to solve the problem of system calls. Traditionally methods such as CPI(code pointer integrity) is been widely used with more stability.

3 Methods and Plan

We plan to incorporate the current SFI implementation into the Endokernel without changing the interfaces that the Endokernel provides to the users but change the low-level memory access control only. We will provide a modified LLVM toolchain to inject SFI-related code into the compiled object, enforcing the memory access control by checking dynamic memory access and creating cross-domain RPC calls. The SFI proposes to enforce that only a single trusted domain with arbitration code could be able to make syscalls and other trust domains wanting to make syscalls should make cross-domain RPC to the trusted domain which will determine if that particular syscall could be performed. While the Endokernel also provides syscall isolation using Seccomp, easing the requirements for isolating syscalls in SFI. We plan to evaluate the possibility of ignoring syscalls isolation in the SFI and enforce that with the Seccomp proposed by Endokernel by analyzing different thread models. With the implemented SFI-based Endokernel, we plan to verify its portability by installing it on ARM-based processors with possibly a few modifications on the SFI LLVM since the instruction sets are different.

Our evaluation plan is to compare the performance

of the Erim-based Endokernel and the SFI-based Endokernel both in general programs such as curl, Lighthttpd, SQLite, etc and then analyze the corner cases where the SFI-based Endokernel would be faster than Erim-based Endokernel. The assumption is that the SFI-based Endokernel will be faster for programs with little dynamic memory access and cross-domain RPC communication. So that there is little overhead for doing dynamic memory access checks and cross-domain RPC calls, the main source of the overhead of the SFI. We plan to write a program with those properties and compare the performance of two versions of Endokernel running this program.

4 Milestones

1. Our team will work on replacing MPK with SFI and using the same benchmark method as the MPK version Endokernal to compare with our SFI version performance. We are also going to test the MPK version vs. SFI version in performance cost version in some best case worst, such us supposed a chance that all code is trusted, and all the memory accesses are static accesses that can be analyzed at compile time so that there is no overhead caused SFI during runtime. Our ideal goal is to down this around the beginning of November.

2. We will run our SFI version in the ARM architecture and probably run it in an M1 chip machine to see how performance will compare to the x86 platform. Theoretically, this can be down in mid of November.

3. Our last shot, if we still have time, we should consider implementing MPK + CPI in x86 first, then use an ARM version of MPK, Memory Tagging Extension (MTE)[4], to replace MPK implementation on ARM.

References

[1] Bumjin Im, Fangfei Yang, Chia-Che Tsai, Michael LeMay, Anjo Vahldiek-Oberwagner, and Nathan Dautenhahn. The endokernel: Fast, secure, and programmable subprocess virtualization. *arXiv preprint arXiv:2108.03705*, 2021.

[2] Volodymyr Kuznetzov, László Szekeres, Mathias Payer, George Candea, R Sekar, and Dawn Song. Code-pointer integrity. In *The Continuing Arms Race: Code-Reuse Attacks and Defenses*, pages 81–116. 2018.

[3] Soyeon Park, Sangho Lee, Wen Xu, Hyungon Moon, and Taesoo Kim. libmpk: Software abstraction for intel memory protection keys (intel {MPK}). In *2019 USENIX Annual Technical Conference (USENIX ATC 19)*, pages 241–254, 2019.

[4] Kostya Serebryany, Evgenii Stepanov, Aleksey Shlyapnikov, Vlad Tsyrklevich, and Dmitry Vyukov. Memory tagging and how it improves c/c++ memory safety. *arXiv preprint arXiv:1802.09517*, 2018.

[5] Anjo Vahldiek-Oberwagner, Eslam Elnikety, Deepak Garg, and Peter Druschel. Erim: Secure and efficient in-process isolation with memory protection keys. *arXiv preprint arXiv:1801.06822*, 2018.

[6] Robert Wahbe, Steven Lucco, Thomas E Anderson, and Susan L Graham. Efficient software-based fault isolation. In *Proceedings of the fourteenth ACM symposium on Operating systems principles*, pages 203–216, 1993.