Computer-Security-Project-Proposal

Title and Author List

Title: `libpmp`

Author List:

SID	Name
11913008	谢岳臻
11910104	王奕童
11912614	张睿豪
12012319	刘晟淇
11913003	李家奥

Problem Statement

What the problem is?

- Memory protection is more related to managing the access of memory pages to avoid bugs or prevent malicious behavior. It is usually implemented with the system call, for instance, the `mprotect` on Linux, because the modification of the page table requires privileged access. To implement it, Intel has proposed a new mechanism called **Memory Protection Key (MPK)** for x86 architecture, which uses the **protection key rights register (`PKRU`)** and corresponding instructions to maintain 16 protection keys. Those protection keys are used to control the access of page groups. Intel MPK also provides several system calls which implement the change of memory access privilege at the user level.
- However, there exist three hidden issues in MPK:
    - Protection-key-use-after-free problem.
    - Protection key limitation.
    - Inter-thread synchronization for multi-thread programs.
- To fix those issues, an open-source project called `libmpk` provides an abstraction of MPK, which fixes those three issues above.
- However, this is the overall solution for x86 architecture. We want to explore the memory protection mechanisms in other architecture.
- In RISC-V, a similar design is called **Physical Memory Protection (PMP)**. PMP uses several PMP entries to control the privilege of memory access (PMP is composed of configuration and address registers,

which are per-hardware-thread machine-mode control registers). After the exploration of PMP, we found that the second issue (Protection key limitation) exists in PMP.

- **Protection key limitation exists in `pmp`:** Unlike the Intel MPK, PMP uses 16 PMP entries instead of the protection key to manage the privilege of memory access for different memory regions. Since the number of PMP entries is limited, the isolation request for a specific memory region is also limited (for example, if 17 non-intersecting memory regions need isolating, from the settings of PMP, the content of one PMP entry will be overwritten).
- **Inter-thread synchronization exist in `pmp`:** From RISC-V Instruction Set Manual Volume II: Privileged Architecture (a manual), we find that `pmp` provides physical memory protection at a **hart** level. If one thread runs on a core, this thread has its own 16 PMP entries. If this thread is switched out from this hart, the content of PMP entries will be stored. However, suppose one thread A runs on core x, if A changes the value of some PMP entries, it will not be synchronized to other threads within the same process, which leads to the inter-thread synchronization problem.

Why it is important?

- **For Protection key limitation:** In practice, programs may make a request for the isolation of many memory regions. If PMP is directly used, some PMP entries may be overwritten, which is inconsistent with the configuration of the user program and may cause security issues.
- **For Inter-thread synchronization:** In Linux, programs uses the syscall `mprotect` to set the memory access privilege when allocating memory for one process. In RISC-V architecture, changing the memory access privilege can be implemented by setting specific values on PMP entries (consist of configuration registers and address registers). If the change in PMP entries cannot be synchronized with threads, it will cause confilct with the semantics of `mprotect` and may bring security vulnerabilities (for example, if thread A set one memory region to be not readable, however, thread B has the read privilege to the same memory region).

Related Work

> libmpk: Software Abstraction for Intel Memory Protection Keys

This work introduces the software abstraction of MPK, which is used to fix the three hidden issues.

- **`libmpk`'s solution to Protection key limitation:**
  - `libmpk` uses an application to virtualize the hardware protection key, it will call `mpk_mmap()` to create a new page group when a virtual key will be allocated to associate with the new group. The virtual key maintains the group's permission and used to hide the physical key. A cache-like structure is used to map the virtual key to hardware key, and it will make sure that the frequently updated virtual key will be mapped with a hardware key and the other will be switched out. Hence, the limitation is hidden by the virtual key.
  - `libmpk` implements its own `mpk_protect`, which calls `do_pkey_sync()` to do the inter-thread synchronization. In `do_pkey_sync`, the calling thread will send interrupt to other threads within a process and register the callback functions to change the content of PMP entries. When other threads are scheduled, they will respond to the interrupt and execute the callback functions to synchronize the PMP entries.

Proposed New Solution

- The goal of `libpmp` is to support unlimited isolation of memory regions either using software virtualization.
    - Limitation of number of PMP entry

      The solution is similar to that of `libmpk`. We will maintain a cache-like structure called mapping table to store the mappings between virtual entry and PMP entry, where virtual entry acts as virtual key in `libmpk` whose number has no limit. If a virtual entry is already associated with `PMP` entry, then user can directly access a certain part of memory. When an expected virtual entry doesn't associate with any `PMP` entries and all of `PMP` entries are mapped, then there will be replacement in mapping table to evict some virtual entries based on algorithms like LRU.
    - Inter-thread synchronization

      The proposed solution is similar to that of `libmpk`. If one thread change the content of PMP entries, it will send interrupt and register callback functions to other threads within one process. When other threads are scheduled, they will change the content.

Evaluation Plan

Our hypothesis when building `libmpk` was that `libmpk`, a software abstraction for PMP should implement the same function as the original PMP. Libmpk should limit untrusted code to access its own memory space.

1. Security Evaluation

We evaluate the benefit of `libpmp` from the memory protection and isolation first. For some applications, `libpmp` provides domain-based isolation to prevent the malicious code from executing privileged instructions. The permission for the particular memory space set by `libpmp` is globally safe, which disabled the inner code to access other memory space. Moreover, exploiting the memory corruption bug to leak or ruin sensitive data stored in other memory spaces is killed by segmentation fault resulting the lack of permission, which protected the memory space from untrusted access.

2. Microbenchmark

Out evaluation to `libpmp` uses several microbenchmarks to show the performance of APIs provided by `libpmp`.

Cache performance

Cache performance is mainly decided by cache hit/eviction rate and number of pmp entries in use. We will run several microbenchmarks to check the cache performance.

Memory overhead

In this part, our team will focus on the memory space and its internal data structure to maintain the memory space protection.

Synchronization latency

Context switch and inter-thread interaction will cause latency in the API invoking. We will evaluate `libpmp`'s efficiency by comparing its performance with the original PMP implementation.

3. Macrobenchmark

We measure the performance overhead of `libpmp` in several actual application by evaluating their performance.