Technical Report UW-CSE-2019-11-01

Developing security monitors on RISC-V

Case studies on HiFive Unleashed

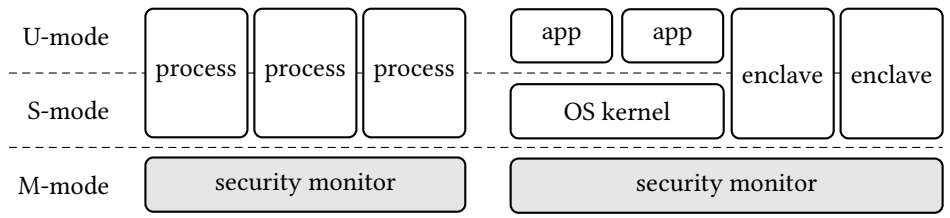
LUKE NELSON and XI WANG, University of Washington

This technical report documents ways of using hardware protection mechanisms afforded by RISC-V to develop security monitors. As case studies, we examine three publicly available security monitors: CertiKOS^s and Komodo^s from the Serval project [4]; and Keystone [3]. The security monitors target the HiFive Unleashed RISC-V development board. We also document hardware issues we have encountered, and their mitigations.

1 INTRODUCTION

The security monitors examined by this technical report target the HiFive Unleashed development board, which is based on the FU540-C000 SoC [5]. It includes five 64-bit RISC-V cores: one E51 core and four U54 cores. The E51 core implements two privilege levels, machine mode (M-mode) and user mode (U-mode), and the RV64I base integer instruction set with extensions "M" for integer multiplication and division, "A" for atomic instructions, and "C" for compressed instructions (RV64IMAC). In addition to the privilege levels and extensions supported by the E51 core, the U54 core implements supervisor mode (S-mode), and the "F" and "D" extensions for single- and double-precision floating point operations, respectively (RV64IMAFDC).

Figure 1 shows an overview of the examined security monitors. M-mode on the U54 core is appropriate for running the security monitors for the following reasons. First, since M-mode is the most privileged mode, running a security monitor in M-mode avoids the need to trusting lower-level software. Second, only the U54 core supports S-mode,which facilitates running an OS kernel or an enclave. Lastly, because the security monitors we study do not support multicore execution, they put all but one core to sleep upon booting.



(a) CertiKOS^s **(b)** Komodo^s and Keystone

Fig. 1. Security monitors on RISC-V.

2 PROTECTION MECHANISMS

The RISC-V ISA specification [7] describes hardware protection mechanisms that allow M-mode to isolate S- and U-mode execution. Below we briefly review two such mechanisms for physical memory isolation that are used by the security monitors.

Physical Memory Protection (PMP). PMP is a memory protection mechanism that allows M-mode to create and assign permissions to contiguous physical memory regions. PMP is configured by M-mode through two sets of registers. The PMP address registers $\texttt{pmpaddr}_0$–$\texttt{pmpaddr}_15$ specify the physical addresses of PMP regions. The PMP configuration entries 0–7 are packed in the PMP configuration register $\texttt{pmpcfg}_0$, and PMP configuration entries 8–15 in $\texttt{pmpcfg}_2$. The PMP

configuration entries specify physical memory access privileges, R (read), W (write), and X (execute), and the address-matching mode for PMP regions. The PMP unit on the U54 core supports only 8 PMP regions: writes to registers $\texttt{pmpaddr}_8$–$\texttt{pmpaddr}_{15}$ and $\texttt{pmpcfg}_2$ are ignored by the hardware.

PMP region addresses are interpreted in one of four ways, depending on the mode set by the corresponding configuration entry. The four modes are OFF (disabled), TOR (top of range), NA4 (naturally aligned four-byte region), and NAPOT (naturally aligned power-of-two region of 8 bytes or more). If PMP configuration entry i selects TOR, the entry specifies a region using both the preceding and the associated PMP address registers $[\texttt{pmpaddr}_{i-1}, \texttt{pmpaddr}_i)$ for $i > 0$, regardless of PMP configuration entry $i - 1$; or $[0, \texttt{pmpaddr}_i)$ for $i = 0$.

For any memory access in S- or U-mode, the CPU matches the physical address using PMP entries in the order from 0 to 15, with a lower-numbered entry taking a higher priority. The first PMP entry that matches the address determines whether the access is permitted using access privileges. If none match or the access is not fully contained within a single PMP region, the access fails. PMP entries can also be configured to enforce permissions for M-mode accesses, however, we do not use this feature as we assume M-mode can access any physical address.

Trap Virtual Memory (TVM). TVM allows M-mode to limit modifications to page tables by S-mode. Setting the TVM bit in the `mstatus` register traps accesses to the `satp` register (which holds the physical address of the page-table root) and execution of the `sfence.vma` instruction in S-mode.

3 CASE STUDIES

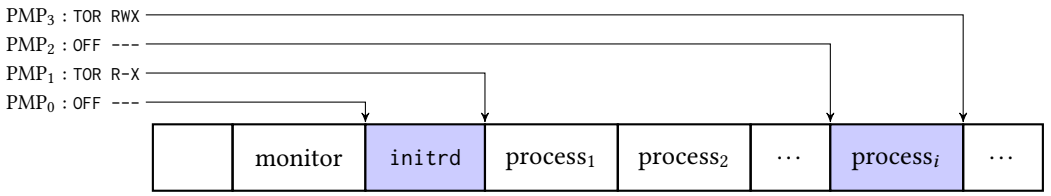


Fig. 2. Physical memory layout in CertiKOS^s (process_i is the currently running process).

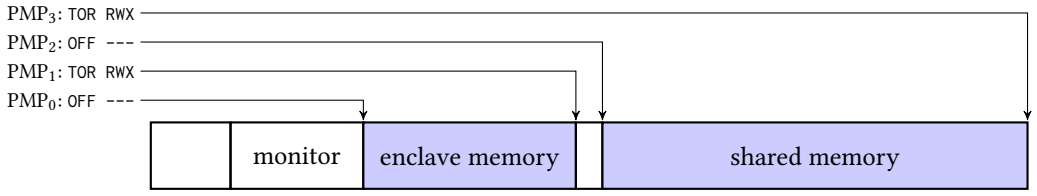
Process isolation using PMP. CertiKOS^s [4: §6.2] is a RISC-V port of the publicly available uniprocessor version of CertiKOS described by Costanzo et al. [1]. Each process runs in S-mode and can access a contiguous memory region with a designated quota. To enforce the memory quota, CertiKOS^s uses PMP to isolate memory for each process. It configures two PMP regions (Figure 2):

- A TOR region for user bootstrapping code (`initrd`), with RX permissions.
- A TOR region for the currently running process, with RWX permissions.

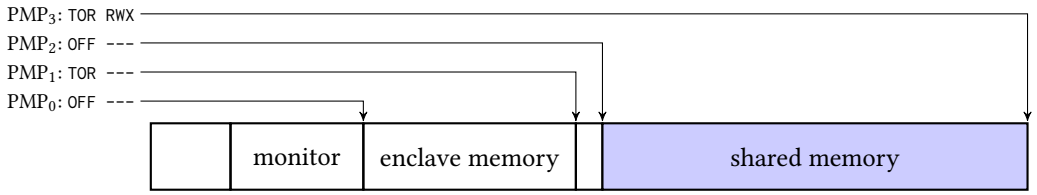
This configuration allows the process to access its own memory while preventing it from accessing the memory of other processes or the monitor. The process sets up its own page table and handles page faults in S-mode, without invoking the monitor.

Software enclaves using PMP and TVM. Komodo^s [4: §6.3] is a RISC-V port of the unverified version of Komodo [2]. It supports running a set of enclaves in S-mode, along with an untrusted OS. To enforce enclave isolation, Komodo^s uses PMP to isolate enclave memory from the OS and uses page tables to isolate enclaves from each other. It configures two PMP regions (Figure 3):

- A TOR region for shared memory, with RWX permissions.
- A TOR region for enclave memory, with RWX permissions during enclave execution only.



(a) PMP configuration for enclave execution.



(b) PMP configuration for OS execution.

Fig. 3. Physical memory layout in Komodo^S.

To enter an enclave from the OS, Komodo^S sets TVM to prevent the enclave from changing the page-table root, and enables the PMP region for enclave memory to allow the enclave to access its own private memory. The enclave's page table prevents the enclave from accessing the memory of other enclaves or modifying its page-table pages. To exit an enclave to the OS, Komodo^S clears TVM and disallows access to the PMP region for enclave memory.

Software enclaves using PMP. Keystone [3] is another software enclave system on RISC-V. Unlike Komodo^S, it uses only PMP for enclave isolation. It configures the following PMP regions (Figure 4):

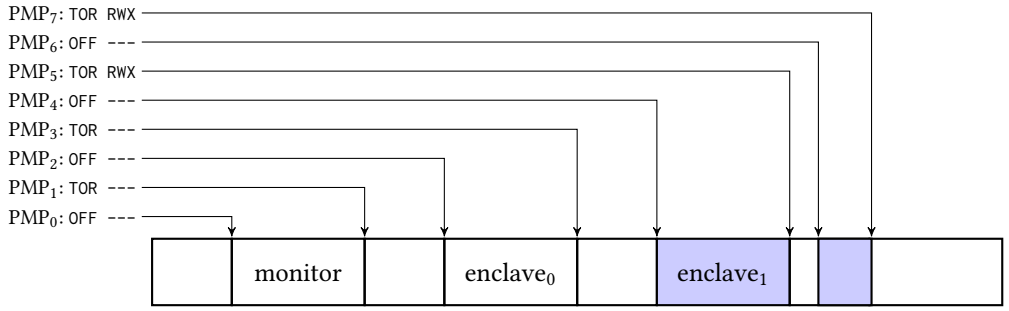
- A high-priority region that disallows access to the monitor memory.
- For each enclave, a secure region with RWX permissions during that enclave's execution and no permissions otherwise.
- A low priority region with RWX permissions that allows access to shared pages.

These PMP regions may be configured using either NAPOT or TOR. The number of enclaves supported by the system is limited by the number of available PMP entries (registers). Given 8 PMP entries on the U54, Keystone supports up to 4 enclaves (using NAPOT) or 2 enclaves (using TOR).

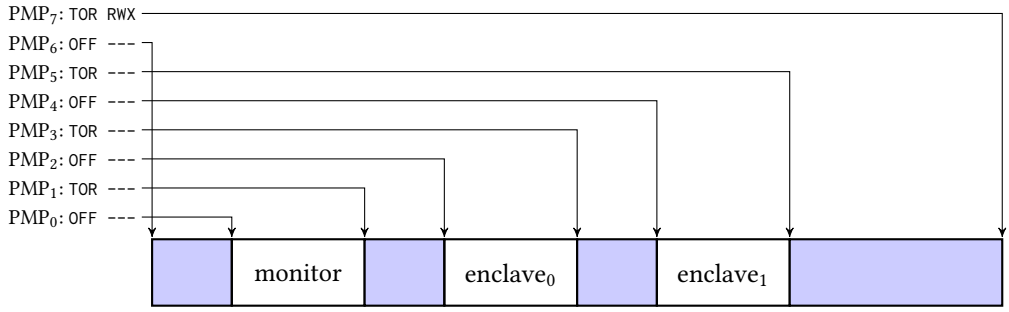
Each enclave has a secure region that contains memory private to that enclave, and a shared region it can use to communicate with the untrusted OS. Keystone ensures that the secure regions do not overlap with other secure regions or the monitor itself. During OS execution, the secure region of each enclave is set to prohibit access; the shared region is set to allow access to every address not prohibited by any higher-priority region. To run an enclave, Keystone allows access to the corresponding secure region, and modifies the bounds of the shared region to allow access only to the range granted by the OS.

4 PMP PERFORMANCE

The U54 core caches the results of PMP permission checks in the translation lookaside buffer (TLB), similar to how virtual address translations are cached. The simplest case is when a PMP region is aligned to virtual page boundaries, as the PMP permission checks can be cached together with the permission checks from the page table in a single entry. The U54 core, however, also supports PMP regions that are smaller than one page, or not aligned to page boundaries. Such regions, called *inhomogeneous* regions, cannot be cached in regular TLB entries which cover aligned pages. To



(a) PMP configuration for enclave execution (enclave_1 is the currently running enclave).



(b) PMP configuration for OS execution.

Fig. 4. Physical memory layout in Keystone (using TOR).

understand the performance impact of use of inhomogeneous regions, we communicated with CPU developers and learned that the U54 core has only a single, dedicated TLB entry specifically for such regions.

To test our understanding of this, we conducted an experiment that measures the number of TLB misses incurred by use of inhomogeneous PMP regions. The test runs in S-mode with paging enabled; it alternates 1-byte accesses two page-aligned addresses in a loop that runs 2^{12} times. The two addresses are each covered by a separate PMP region. We run the test code under three different PMP configurations for these regions.

(1) Two 4 KiB regions.
(2) One 4 KiB region, and one 8-byte (inhomogeneous) region.
(3) Two 8-byte (inhomogeneous) regions.

The first configuration incurs 2 TLB misses because each access can be cached in a regular TLB entry. The second configuration also incurs 2 TLB misses because the access to the inhomogeneous region can be cached in the specialized TLB entry. The third configuration incurs a number of misses proportional to the number of loop iterations (2×2^{12} in this case), because the single specialized entry for inhomogeneous regions thrashes between both regions.

These experimental results confirm that the U54 has only one TLB entry to cache inhomogeneous PMP permission checks. As performance will degrade with multiple such regions, we suggest to avoid using them. Note that this may be required by some CPUs that do not support inhomogeneous PMP regions (e.g., the U74 core [6]).

5 CPU BUGS

Superpages and PMP. The U54 MMU supports superpages up to 1 GiB; our initial prototypes used such superpages to simplify page table construction. We observed that accesses to superpages trigger PMP permission exceptions when the PMP region is smaller than the superpage, even if the addresses involved in the access are all allowed by the PMP region. We confirmed with hardware developers that this is an MMU bug and will be fixed in the next hardware revision. As a workaround, we use only 4 KiB pages, which do properly compose with PMP.

Performance counters. The U54 core implements a set of hardware performance counters (e.g., `cycle`). The counter-enable registers, `mcounteren` and `scounteren`, control whether S- and U-mode can access performance counters, respectively. Since these performance counters may leak information, we initially attempted to disable their accesses in S-mode by clearing the corresponding bits in `mcounteren`. However, we observed that the U54 core ignores `mcounteren` and does not raise any exception for such accesses. To work around this issue, we changed the context-switching code to save and restore performance counters.

ACKNOWLEDGMENTS

We thank Andrew Waterman and Jim Wilson for answering our questions about the CPU bugs.

REFERENCES

[1] David Costanzo, Zhong Shao, and Ronghui Gu. 2016. End-to-End Verification of Information-Flow Security for C and Assembly Programs. In *Proceedings of the 37th ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI)*. Santa Barbara, CA, 648–664.

[2] Andrew Ferraiuolo, Andrew Baumann, Chris Hawblitzel, and Bryan Parno. 2017. Komodo: Using verification to disentangle secure-enclave hardware from software. In *Proceedings of the 26th ACM Symposium on Operating Systems Principles (SOSP)*. Shanghai, China, 287–305.

[3] Dayeol Lee, David Kohlbrenner, Shweta Shinde, Dawn Song, and Krste Asanović. 2019. Keystone: An Open Framework for Architecting TEEs. https://arxiv.org/abs/1907.10119.

[4] Luke Nelson, James Bornholt, Ronghui Gu, Andrew Baumann, Emina Torlak, and Xi Wang. 2019. Scaling symbolic evaluation for automated verification of systems code with Serval. In *Proceedings of the 27th ACM Symposium on Operating Systems Principles (SOSP)*. Huntsville, Ontario, Canada, 225–242.

[5] SiFive. 2018. SiFive FU540-C000 Manual, v1p0. https://www.sifive.com/boards/hifive-unleashed.

[6] SiFive. 2019. SiFive U74 Manual, v19.08p0. https://www.sifive.com/cores/u74.

[7] Andrew Waterman and Krste Asanović (Eds.). 2019. *The RISC-V Instruction Set Manual, Volume II: Privileged Architecture*. RISC-V Foundation.