RISCV PMP Extension Proposal

Nick Kossifidis, Joe Xie, RISCV TEE WG

January 2019

1 Introduction

RISCV physical memory protection (PMP) provides a scheme to protect physical memory. M mode firmware can use PMP to isolate physcial memory between different sub-M modes. According to privilege spec 1.10 M mode access will only be checked by locked entries, while sub-M entries are usually dynamic and not locked by M mode so that the can be revoked/merged..etc, this effectively means M-mode always has access to sub-M modes memory.

Furthermore, it is impossible to create R/W entry for M mode data and X/R entry for M mode code. As spec indicates, locked entries will check both M and non-M modes, the M entries need to be locked to check M access, but meanwhile locked entries will open access to non-M modes.

Allowing M mode to unintentionally execute/access anywhere leads to several security exploits including privilege escalation. Software always has bugs in real world, suppose M mode firmware has a buffer overflow bug, a malicious H or S mode can exploit the bug and trick M mode to jump/return into malicious code in H/S mode's memory space via system call thus gains M privilege.

It is a common technique that hardware architecture to provide a mechanism to prevent unintentionally access to lower privilege modes' memory from high privilege modes. Such feature can be found in many other CPU architectures and even in current RISCV spec. Intel has SMAP (Supervisor Mode Access Prevention) and SMEP (Supervisor Mode Execution Prevention) starting from Boardwell. RISCV has SUM (permit Supervisor User Memory access) bit in SSTATUS to achieve similar purpose for S mode. However lacking of such mechanism in PMP exposes M mode to the risk of privilege escalation attack from sub-M mode.

This document proposes an enhancement to RISCV PMP called MMI (Machine Mode Isolation) to fix above problems.

2 Proposal

We propose to add one bit in mstatus called MMI (Machine Mode Isolation) to protect unintentional access from M mode. `mstatus` layout with MMI can be found in Figure 1 for RV32 and Figure 2 for RV64 and RV128, it takes `mstatus`.bit23, and does not present in `hstatus`, `sstatus` and `ustatus`.

Table 1 explains the the meaning of MMI combining with PMP lock, MMI is reset to 0.

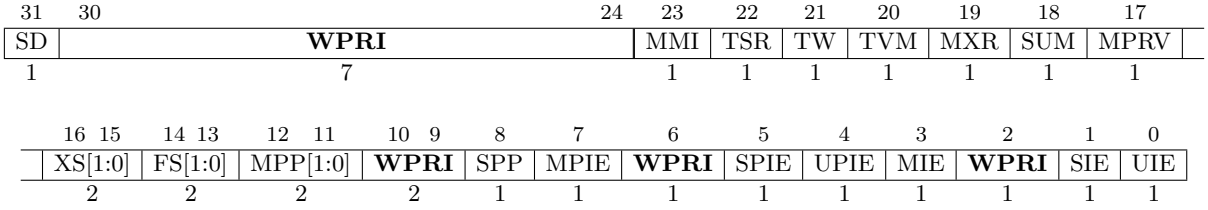

Figure 1: Machine-mode status register (`mstatus`) with MMI for RV32.

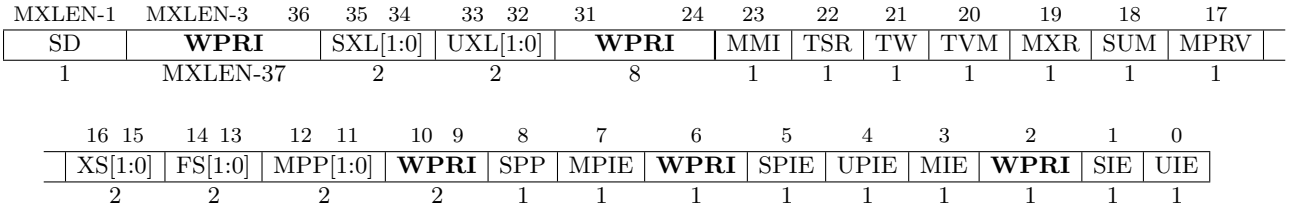

Figure 2: Machine-mode status register (`mstatus`) with MMI for RV64 and RV128.

`pmpcfg.L`	`mstatus`.MMI	Meaning
`0`	0	Temporary entry; enforced on sub-M modes; M-mode succeeds
`0`	1	Temporary entry; enforced on sub-M modes; M-mode fails
`1`	0	Locked entry; enforced on all modes
`1`	1	Locked entry; enforced on M-mode; sub-M modes fails

Table 1: Meaning of MMI and Lock

3 Explanation

Table 2 is an example PMP configuration with MMI.

The first 3 entries are locked, they are allocated for M code/rodata/data. Entry 4-15 are temporary entries, they are allocated for Sub-M mode memory.

M entries are placed on top and locked in this example to guarantee that M access will be checked even if M mode is compromised. These entries are locked during secure boot before M mode starts, one can make M code entry (entry 0) is executable-only, so that an adversary cannot steal the code even M is compromised.

According to Table 1, sub-M mode access will fail when MMI is set and checked when MMI is clear. Here sub-M mode must never be allowed to access to M entries, we don't want sub-M mode to have access, the security relies on M mode firmware to make sure MMI is set before leaving M to sub-M modes otherwise M data might be exposed to sub-M access.

Sub-M entries are not locked, according to Table 1 M mode access to sub-M entries will fail when MMI is set. M mode can clear MMI to gain access again.

Index	L	X	W	R	Meaning
0	1	1	0	0 or 1	M code entry, locked, sub-M has no access
1	1	0	0	1	M rodata entry, locked, sub-M has no access
2	1	0	1	1	M data entry, locked, sub-M has no access
3-15	0	0 or 1	0 or 1	0 or 1	Sub-M entries, not locked, M has no access by default

Table 2: Example PMP configuration

4 Back up

One potential problem with above proposal is that if MMI is accidentally left clear by M mode then M data will be exposed to sub-M access. This variant addresses the concern by having per-entry MMI, using reserved bit `pmpcfg`.bit6. Figure 3 shows the layout of a PMP configuration register with per entry MMI bit.

7	6	5	4 3	2	1	0
L (**WARL**)	MMI (**WARL**)	0 (**WARL**)	A (**WARL**)	X (**WARL**)	W (**WARL**)	R (**WARL**)
1	1	1	2	1	1	1

Figure 3: PMP configuration register format with MMI.

Table 3 explains the the meaning of MMI and lock in this variant. Obviously per-entry MMI adds more complexity to the architecture.

pmpcfg.L	pmpcfg.MMI	Meaning
0	0	Temporary entry; enforced on sub-M modes; M-mode succeeds
0	1	Temporary entry; enforced on sub-M modes; M-mode fails
1	0	Locked entry; enforced on all modes
1	1	Locked entry; enforced on M-mode; sub-M modes fails

Table 3: Meaning of per-entry MMI and Lock

Table 4 explains example PMP setting using per-entry MMI, MMI bit for M entries are all set and locked so that there's no way software can mess up MMI and expose M data to sub-M modes.

Index	L	MMI	X	W	R	Meaning
0	1	1	1	0	1	M code entry, locked, sub-M has no access
1	1	1	0	0	1	M rodata entry, locked, sub-M has no access
2	1	1	0	1	1	M data entry, locked, sub-M has no access
3-15	0	0/1	0/1	0/1	0/1	Sub-M entries, not locked, M has no access by default

Table 4: Example PMP configuration with per-entry MMI