计算机研究与发展
Journal of Computer Research and Development
DOI:10.7544/issn1000-1239.20200537
59(2):403-417, 2022

基于动态资源使用策略的SMT执行端口侧信道安全防护

岳晓萌[1,2] 杨秋松[1] 李明树[1]
[1](基础软件国家工程研究中心(中国科学院软件研究所) 北京 100190)
[2](中国科学院大学 北京 100049)
(xiaomeng@iscas.ac.cn)

SMT Port Side Channel Defending Method Based on Dynamic Resource Usage Strategy

Yue Xiaomeng[1,2], Yang Qiusong[1], and Li Mingshu[1]
[1](*National Engineering Research Center for Fundamental Software* (*Institute of Software*, *Chinese Academy of Sciences*), *Beijing* 100190)
[2](*University of Chinese Academy of Sciences*, *Beijing* 100049)

Abstract Simultaneous multi-threading (SMT) technology is one of the important micro-architecture optimization technologies to improve thread-level parallelism. SMT can realize two logical cores on one physical core and improve the overall performance of the processor. However, some timing channel security problems represented by sharing execution ports in SMT environment appeared. A port timing channel attack defending method is proposed based on dynamic resource usage strategy in SMT environment. Dynamic strategy adjustment algorithm is designed for different processing modes of data structure resources, and improved processor port binding and scheduling selection algorithm are adopted to protect the port side channel attack in SMT environment. Defending method used modular design has realized the port conflict matrix, branch filters and dynamic resource editor strategy. Respectively judgment model for port conflict, branch information filtering and SMT dynamic resource use strategy changes, the final modification strategy can be directly applied to the execution port binding and scheduling algorithm. The defending method in this paper can achieve the effect of close SMT technology and reduce the performance cost greatly. At the same time, its hardware cost is controllable. Therefore, the method proposed in this study has high application value.

Key words SMT; timing channel; side channel; execution port; security defending

摘 要 同时多线程(simultaneous multi-threading, SMT)技术是提升线程级并行度的重要微架构优化技术之一,SMT技术能够在1个物理核上实现2个逻辑核,提升处理器的整体性能.然而,以共享执行端口为代表的SMT环境下特有的时间侧信道安全问题也陆续出现.提出了一种基于动态资源使用策略的SMT环境下执行端口时间侧信道攻击防护方法,基于SMT技术对数据结构资源的不同处理方式设计动态策略调整算法,通过改进处理器端口绑定及调度选择算法以防护SMT环境下执行端口时间侧信道攻击.防护设计实现了端口冲突矩阵、分支过滤器和动态资源使用策略修改器3个组件,该方法在

收稿日期:2020-07-07;**修回日期**:2021-03-12
基金项目:"核高基"国家科技重大专项基金项目(2014ZX01029101-002);中国科学院战略性先导科技专项(XDA-Y01-01)
This work was supported by the National Science and Technology Major Projects of Hegaoji (2014ZX01029101-002) and the Strategic Priority Research Program of Chinese Academy of Sciences (XDA-Y01-01).

防护有效性上可以达到关闭 SMT 技术的防护效果且性能开销大大降低,同时硬件开销可控,具有较高的应用价值.

关键词 同时多线程;时间信道;侧信道;执行端口;安全防护

中图法分类号 TP309.2

随着计算机技术的发展和数据处理需求的增大,处理器厂商一直致力于提升处理器的并行处理能力.早期处理器发展重点是提升处理器内部的指令并行度,当处理器指令并行度提升到了一定瓶颈后,处理器厂商开始提升线程并行度来提升处理器的整体性能,这也推动了多核处理器的出现.

同时多线程(simultaneous multi-threading, SMT)技术是实现线程级并行的技术之一,其通过增加少量硬件资源,把 1 个物理核映射成 2 个逻辑核,同时运行的线程可共享处理器的资源.因为线程执行总有空闲或者等待的时间,SMT 环境下当一个线程进入空闲或等待,另一个线程可以继续执行,从而更加合理地使用处理器资源,进而达到比单线程超标量处理器更好的指令吞吐量和资源利用率.已实现 SMT 技术的处理器厂商包括 Intel, AMD, IBM 等,其中以 Intel 最早于 2002 年的 Pentium4 处理器上使用的超线程(hyper-threading, HT)技术[1]最为典型.本文中的 SMT 技术均以 Intel 提出的超线程技术作为主要参考.

从 Intel 公司推出的 HT 技术商用开始,研究人员就开始针对 SMT 技术进行安全问题的挖掘和研究,因为 SMT 技术增加了很多处理器微架构安全问题利用的场景和机会,因此有研究人员评价 SMT 技术是"廉价的硬件并行意味着廉价的安全性"[2].

在 SMT 技术设计下,处理器执行资源是完全被 2 个线程共享的,2 个线程间共享一样的执行端口及端口内的执行单元,很容易构造资源的竞争,这种竞争可以被攻击者检测和利用.

2006 年,Wang 等人[3]首次将执行单元的竞争应用于隐蔽信道.2016 年,Covert shotgun 项目[4]提出一个自动化挖掘 SMT 环境下执行单元隐蔽信道的框架.在执行端口及执行单元的隐蔽信道攻击研究过程,逐步出现了基于相同技术原理的侧信道攻击技术研究.2018 年,Aldaya 等人[5]在 Intel Skylake 和 Kabylake 架构上利用其 SMT 技术开启后执行端口竞争问题提出了 PortSmash 攻击.2019 年,IBM 的研究团队提出了一种叫 SMoTherSpectre[6]的新型"Spectre"类型攻击,其使用 SMT 环境下执行端口的竞争构建了投机代码重用的攻击场景,可以从受害者线程获取私密信息.

在针对上述 SMT 环境下共享执行端口或执行单元的时间侧信道攻击,研究人员也提出了一些防护方法.Percival 等人[7]在早期提出可以禁用 SMT 来防范此类攻击,其可以彻底解决执行单元或执行端口双线程共享产生的时间侧信道问题,但这种方式失去了 SMT 技术本身带来的收益,会导致相应的性能损失.Hu[8]在提出通过将噪声添加到与进程相关的所有时间信息中,达到降低时序信道带宽的目的进行防护,其通过修改 RDTSC 或 RDTSCP 指令,将噪声加入到指令执行的返回值中,就可以达到降低攻击者获取时间信息精度的目的,但这种方式也会直接影响相应被修改指令的功能准确性,影响正常程序的使用.2019 年,Zhang 等人[9]提出了名为 DDM(demand-based dynamic mitigation)的防护方法,其使用软件的手段动态地基于需求关闭 SMT 技术,进而达到防护基于 SMT 技术执行资源共享的侧信道攻击的目的.然而 DDM 是通过软件手段动态关闭 SMT 技术来达到防护目的,尚未有从处理器微架构角度防护的案例.

本文提出了一种针对现代处理器 SMT 技术执行端口时间侧信道攻击的防护方法.主要贡献有 2 点:

1) 提出动态调整双线程共享的执行端口资源使用策略,防护 SMT 环境下执行端口双线程共享产生的时间侧信道攻击的方法,该方法将 SMT 技术对不同数据结构的处理方式应用到侧信道安全防护中,通过对数据结构共享策略的改变来达到防护效果,可以在 SMT 技术合理利用共享数据结构的同时达到安全防护的目的;

2) 提出并设计了一种基于记录分支预测错误刷新及执行端口冲突历史的防护方法,可以有效防护以 SMotherSpectre 为代表的组合使用诱导分支预测投机执行的 Spectre 机制及执行端口冲突时间侧信道攻击.

1 相关工作

在 20 世纪初期 SMT 技术问世之后,研究人员

就开始在 SMT 技术上挖掘安全问题和防护方法.Koç 等人[10]在 2009 年总结了当时已有的基于 SMT 的攻击技术和应用案例,对 SMT 技术导致的安全问题进行了系统归纳,也汇总了当时针对 SMT 技术安全问题的缓解手段.2018 年,Ge 等人[11]总结了 2002—2018 年期间各个系统层级由于资源共享导致的时间信道安全问题及缓解措施,作者提出基于 SMT 技术产生的硬件线程级时间信道安全问题相对于跨核、跨处理器的时间信道安全问题是更难应对和防护的.SMT 环境下除缓存结构外的其他共享数据结构很难有好的防护效果,其中最突出的是 SMT 环境下执行端口双线程共享导致的时间侧信道安全问题.

SMT 环境下执行端口时间侧信道攻击路径核心环节是冲突的构建和对时间的度量,这也是时间信道利用的 2 个基本要素(简称为冲突和时间).表 1 描述了已有针对 SMT 共享执行端口的时间侧信道防护方法:

Table 1 Timing Channel Defending Method Under SMT Execution Port Shared

表 1 SMT 环境下共享执行端口的时间信道防护策略及方法

SMT 共享资源	防护案例	防护要素		防护策略	防护方法	防护实现
		时间	冲突			
执行端口	文献[7,9,12]	√		不产生冲突	进程隔离	SMT 技术禁用/DDM
	文献[8,13-14]		√	使时间度量手段精度变差	增加时间噪声	修改时间度量指令

注:"√"表示属于该防护要素.

1.1 冲突要素防护

从冲突要素的防护策略角度,针对 SMT 技术下执行端口双线程共享产生的时间侧信道安全问题目前主要有 2 种防护手段:将 SMT 技术禁用以及动态关闭 SMT(DDM[9]).

Percival[7]在早期提出数据缓存可以在 SMT 环境下被利用于构造隐蔽信道和侧信道攻击,建议可以通过禁用 SMT 来防范此类攻击.当前禁用 SMT 的方案在实践中已经得到使用,例如微软 Microsoft Azure[12]的公共云服务上就禁用 SMT 技术.除了直接禁用 SMT 技术,DDM[9]可以基于上层软件的配合来达到防护目的.DDM 是一种基于需求的动态 SMT 瞬态攻击防护技术,其实现了进程级的防护粒度,当安全进程在运行时,如果有防护范围内的恶意进程尝试分配到和安全进程一样的物理核时会被 OS 拒绝.当安全进程在一个物理核中运行时,另一个相同物理核的进程可以通过 HLT 指令的系统调用完成 SMT 状态的关闭,保证安全进程能够在等同于关闭 SMT 的状态下运行,从而达到防护 SMT 技术下共享执行端口侧信道攻击的目的.DDM 的防护方式本质上也是通过关闭 SMT 技术达到防护效果,但是其动态开关的方式会一定程度上降低彻底关闭 SMT 的性能损失.DDM 使用 SPEC2000 进行性能评估约产生 8%左右的性能损失.

1.2 时间要素防护

从时间要素的防护策略角度,可以通过修改时间度量指令的方式增加时间噪声来防护针对 SMT 技术下执行端口共享产生的时间侧信道攻击.

增加时间噪声是指在时间度量方式和手段上进行噪声添加.Hu[8]在 1991 年就提出了模糊时间(fuzzy time)技术,通过将噪声添加到与进程相关的所有时序信息中,最终达到降低时间信道带宽的目的.文章中通过修改 RDTSC 或 RDTSCP 指令在时间信息中添加噪声,此指令的功能是通过读取处理器内部的时间戳计数器(time stamp counter, TSC)来度量运行时间,通过修改指令的实现方式,将噪声加入到指令执行的返回值中,就可以达到降低攻击者获取时间精度的目的.

Vattikonda 等人[13]通过修改 Xen 虚拟机监控器(hypervisor),将噪声加入到 RDTSC 指令的返回值中.Martin 等人[14]发现如果处理器内部时间度量变得不精确,可以有效防护时间侧信道攻击.作者修改了 x86 体系架构下的 RDTSC 指令的硬件实现,使得 RDTSC 指令会暂停所有执行操作直到预设的时间阈值 T 满足,然后通过产生一个 0 到时间阈值 T 的随机数返回给指令使用者来达到模拟时间的效果.

1.3 小 结

当前除了禁用 SMT 技术[7]、动态关闭 SMT[9]以及修改时间度量指令(RDTSC/RDTSCP)[8,13-14]的方法外,还没有其他可以针对 SMT 环境下执行端口时间侧信道攻击的防护手段.已有的防护手段都有其明显的缺陷,禁用或动态关闭 SMT 技术会损失 SMT 技术带来的性能收益;修改指令语义会弱化指令功能,从而影响其他使用时间度量指令应用

的正常使用.综合已有的防护方法,目前针对 SMT 环境下执行端口时间侧信道攻击效果较好的防护策略是动态关闭 SMT 技术(DDM[9]),还没有公开的基于处理器微架构的防护方案.

2 背景知识

2.1 SMT 微架构特征及安全挑战

处理器在开启 SMT 技术后,处理器内部微架构会使用相关的 SMT 功能特性,从而完成微架构功能的转变,进入到同时多线程模式,可以在同一时刻利用不同执行端口发射执行不同线程的指令.

如图 1 所示,SMT 技术区别于其他的多线程技术的核心是可以在同一个流水级执行来自几个不同线程的微操作(或称 Uop、微指令).普通的超标量单核处理器同时只能运行 1 个线程,在局部时间内流水线资源还会处于空闲状态(例如取指令阶段);对于普通的多核处理器,其会使用多个物理核来运行多个线程,是普通单核处理器的简单叠加;粗粒度和细粒度多线程技术使用时间片的方式分配资源使用,资源会在一段时间内分配给独立的 1 个线程使用,同一时间资源只会分配给 1 个线程使用;SMT 技术区别于粗/细粒度的多线程技术,其可以灵活分配处理器内部资源,大部分数据结构资源被多个线程共享且同一周期可以调度多个线程的操作同时执行,最大化得提高资源利用率.

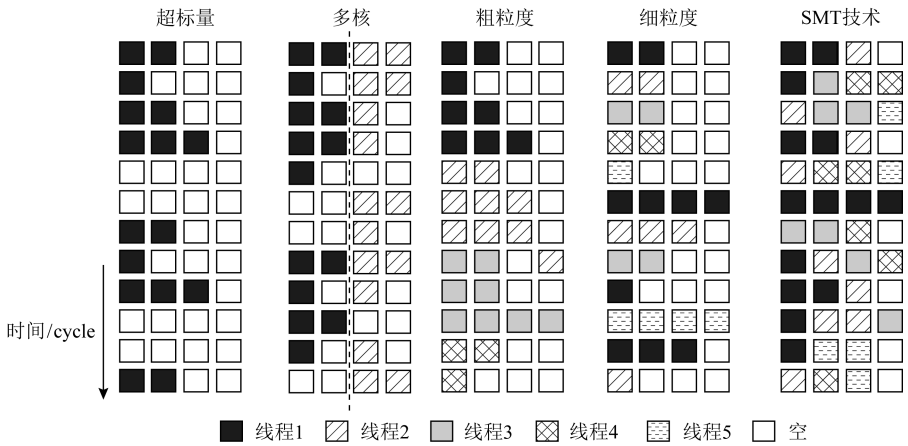


Fig. 1 SMT feature

图 1 SMT 技术特征

SMT 技术原理(以 Intel 的超线程技术为例)主要包含 3 个部分:

1) 制定资源共享策略.SMT 技术可以使用资源复制、资源分割和资源共享的方式来实现数据结构资源的分配.以图 2 为例,Intel SMT 技术中的 ITLB 和 IP 结构采用了资源复制的处理策略,微操作队列结构采用了资源分割的处理策略,追踪缓存结构采用了资源共享的处理策略.

2) 设置线程选择点.使用 SMT 技术的处理器微架构需要在合适的流水线位置增加线程选择点,其作用是在共享资源的使用上进行 2 个线程的选择及切换.合理的线程选择点设置和选择算法使用能够有效提高双线程共享资源的利用率以及双线程的公平性.

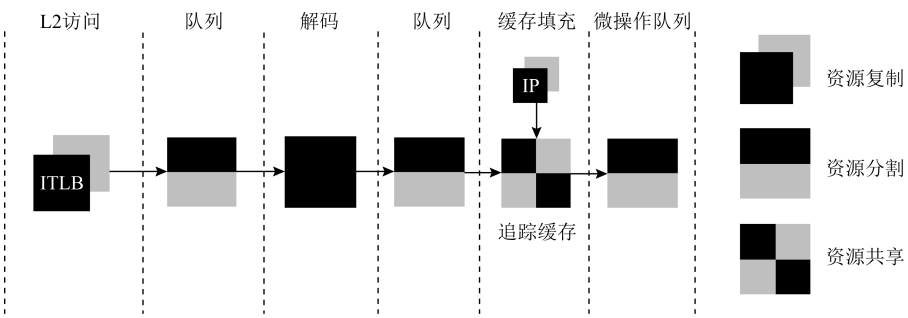


Fig. 2 Intel SMT front end resource

图 2 Intel SMT 前端资源的处理方式

3) 保证线程公平性.处理器微架构在实现 SMT 技术时需要注重线程间的公平性,线程选择点需要使用相应的公平算法,避免单一线程由于长时间得不到资源而饿死的情况出现.

SMT 技术中资源复制和资源分割的处理方式使得 2 个线程在处理器内部会公平地使用自己的私有资源,不会对另外线程的资源产生影响,因此基于共享资源竞争的时间信道攻击方式对该类型的微架构组件无效.对于资源共享的处理方式,2 个线程在处理器内部会产生竞争,容易被攻击者利用,例如针对 SMT 环境下执行端口双线程共享的技术原理,控制执行指令流的类型可以构建端口冲突,攻击者可以通过时间信道传递信息获取另一线程运算逻辑或指令类型的使用状态.

2.2 SMT 技术的多端口调度算法

本节通过剖析 SMT 技术的多端口调度算法的微架构实现,来描述 SMT 环境下执行端口时间侧信道攻击原理,如图 3 所示:

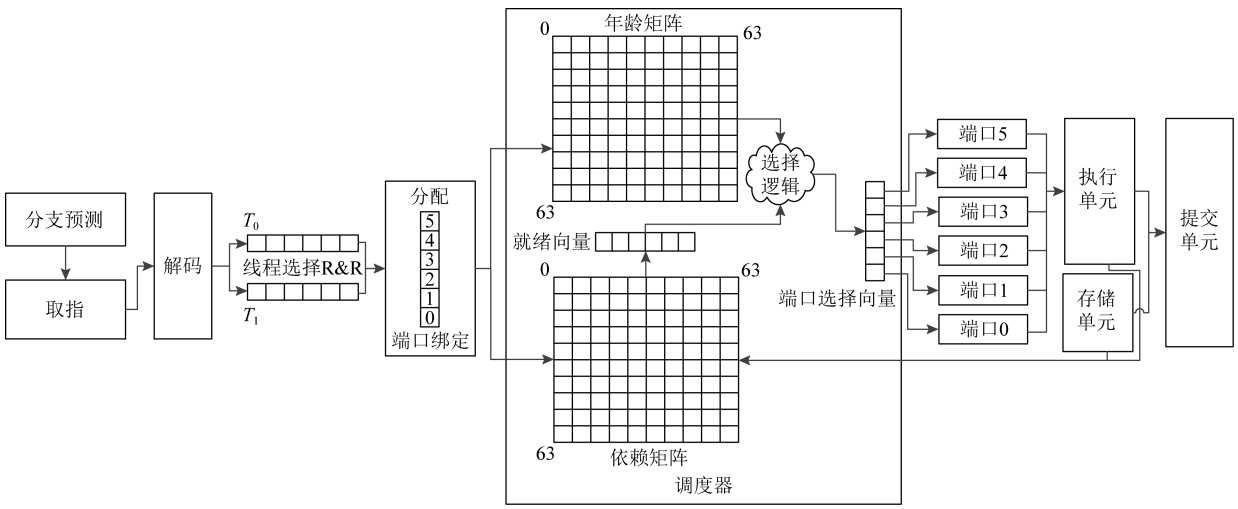


Fig. 3 SMT multi-port schedule micro-architecture

图 3 SMT 技术的多端口调度微架构

在超标量处理器微架构设计中,指令集架构中描述的宏指令会经过取指、译码的流程形成对应的微操作.微操作在进行指令执行前,会通过指令解析完成对应执行端口的绑定.在关闭 SMT 技术的情况下,处理器流水线内部只有 1 个线程,完成端口绑定的微操作会通过对应的端口进入执行单元;在开启 SMT 技术的情况下,处理器解码系统和乱序执行系统交互过程中,因为在重命名和分配微架构设计中约束了同一时刻只能处理来自于同一个线程的微操作,因此,同一时刻也只有 1 个线程的微操作可以进入调度器.在调度器微架构设计中为了更快地调度和分发微操作,其并不关心微操作的线程信息,只考虑其是否就绪("就绪"指的是某条微操作已满足被选择并发射到执行端口的条件),如果有同时就绪的微操作则年龄更老的优先发射,因此,调度器可以同时在不同执行端口发射 2 个线程的微操作.

如果 2 个线程在调度器内的微操作指向同一个端口,那么由于分配时的先后关系,2 个线程的微操作会自然携带不同的年龄信息.调度器中有 2 个关键数据结构,一个是年龄矩阵,另一个是就绪向量,虽然 2 个线程的微操作即使都处于就绪状态,依赖于年龄矩阵的先后关系,调度器会先发射更"老"的线程的微操作进入相应端口,那么已经就绪但由于较"年轻"导致无法发射到相同端口的另外线程的微操作就产生了延迟,此时就叫作出现了线程间的端口冲突.

假设线程 T_0 和 T_1 是 1 个物理核上分配的 2 个逻辑核,按照 SMT 微架构特征的描述,其按照线程公平性原理进行执行端口的分配使用,分配策略如算法 1 所示.

算法 1. 端口分配策略 $Alloc(T_{xi},P,C,A_{xi})$.

输入:准备分配线程端口 T_{xi},$x=\{0,1\}$,执行端口 $P=\{p_0,p_1,\cdots,p_{n-1}\}$,$i=\{0,n\}$,时钟周期 C;

输出:线程分配端口 A_{xi}.

① 时钟周期 C 执行:

② for ($0\leqslant i<n$)

③ if ($T_{xi}==p_i$ && $T_{(1-x)i}==p_i$)

④ $A_{xi}=p_i$;

⑤ else

⑥ $A_{xi}=p_i$;
⑦ $A_{(1-x)i}=P\backslash p_i$;
⑧ end if
⑨ end for
⑩ 时钟周期 $C+1$ 执行:
⑪ for ($0\leqslant i<n$)
⑫ if ($T_{xi}==p_i$ && $T_{(1-x)i}==p_i$)
⑬ $A_{(1-x)i}=p_i$;
⑭ else
⑮ $A_{(1-x)i}=p_i$;
⑯ $A_{xi}=P\backslash p_i$;
⑰ end if
⑱ end for

上述分配策略会产生 2 种极端情况,当 T_0 和 T_1 都处于完全流水线执行的状态时,一种极端情况是如果 T_0 和 T_1 一直使用不相关的执行端口,那么 T_0 和 T_1 不会有任何的执行端口冲突;另一种极端情况是如果 T_0 和 T_1 一直使用相同的某一个执行端口,那么 T_0 和 T_1 需要轮流使用该执行端口,吞吐量是上一种极端情况的一半.

因此,某线程在特定的执行端口下执行指令并且度量其执行程序的时间就可以推断同一端口或执行单元下另一线程的执行情况.

3 防护目标

2018 年的 PortSmash[5] 和 2019 年的 SMoTherSpectre[6] 是近期利用 SMT 技术执行端口双线程共享产生的时间侧信道构建攻击的案例.

PortSmash[5] 针对 Intel Skylake 的执行微架构(如图 4 所示)设计了 3 组冲突指令流,分别适用于端口 1、端口 5、端口 0156 的冲突构建,端口 1、端口 5 的冲突构建指令选用了长延迟指令,端口 1 选用整型乘法(INT MUL)执行单元执行的 crc32 指令,端口 5 选用了向量交织(VEC SHU)执行单元执行的 vpermd 指令,端口 0156 的重构构建指令选用了单周期指令,使用最常见的算术逻辑(INT ALU)执行单元执行的 ADD 指令.

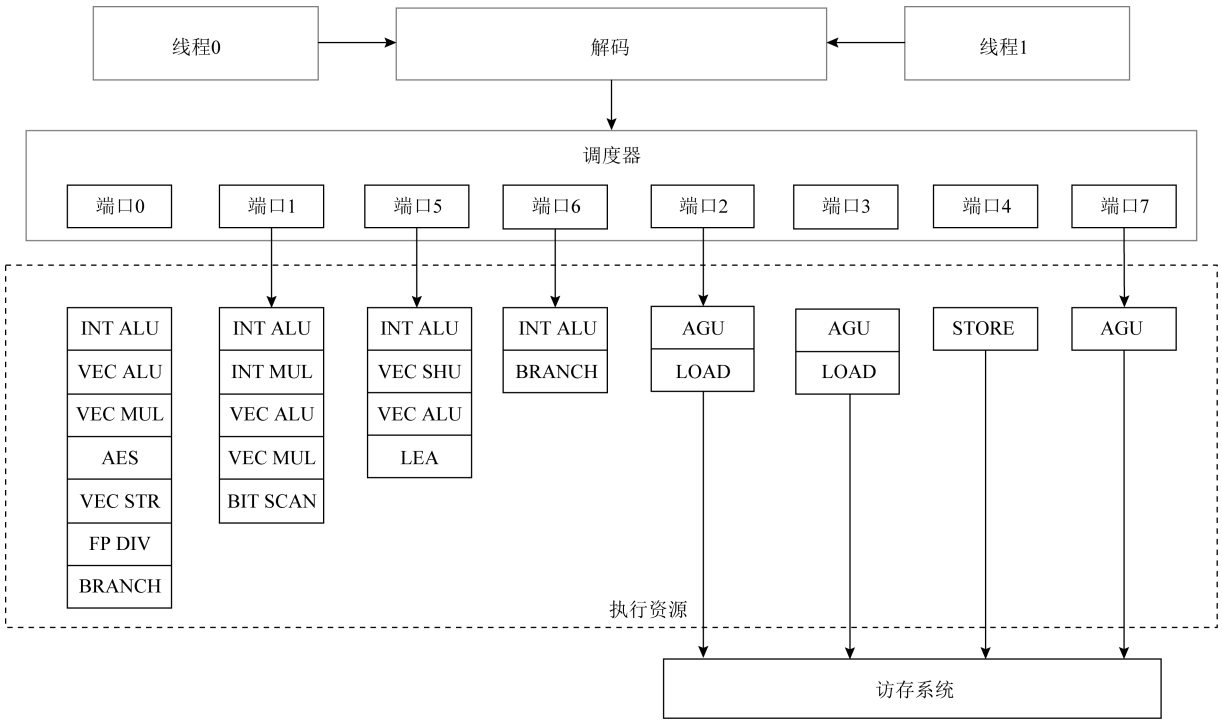


Fig. 4 Intel Skylake execution port distribution

图 4 Intel Skylake 执行端口分布

SMoTherSpectre 除了采用同 PortSmash 类似的冲突构造指令来构建用于受害者泄露信息的 SMoTher Gadget 和用于攻击者度量时间信息 Time SMoTher Gadget 外,其在冲突构建阶段使用了 Spectre[15] 分支诱导的方式来加强冲突构建的成功率,如图 5 所示.受害者的 BTI Gadget 和攻击者的 Poison BTB 组件的作用是构建分支诱导场景,攻击者通过训练处理器分支预测单元的分支目标缓

存(branch target buffer, BTB)结构来诱导受害者投机执行 SMoTher Gadget 的程序,当受害者程序被分支诱导后会进入 SMoTher Gadget 中,该组件通过分支跳转的形式构建了 2 组冲突指令,在 POC 示例中,分支指令后使用了 popcnt 指令,分支目标使用了 ror 指令,同时攻击者程序中的 Time SMoTher Gadget 使用了大量的 ror 指令,2 个线程同时执行 ror 指令在 Intel Skylake 微架构中会造成端口 6 的冲突,从而达到构建冲突的目的.

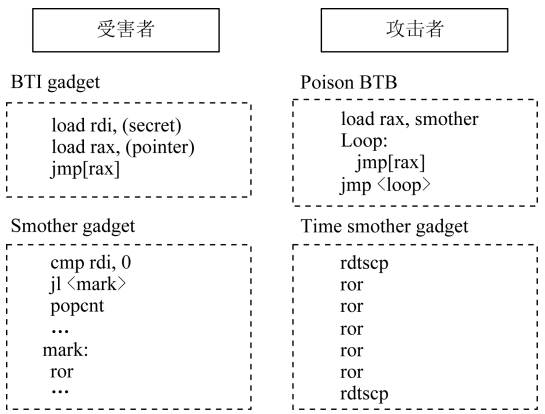


Fig. 5 SMoTherSpectre attack gadget

图 5 SMoTherSpectre 攻击组件示意图

PortSmash[5] 和 SMoTherSpectre[6] 均使用了 rdtsc/rdtscp 指令来作为时间度量的手段,通过在攻击者程序冲突指令执行前后使用 rdtsc/rdtscp 指令可以得出冲突指令的执行时间,从而获取冲突程度或冲突次数的信息.

本文提出的基于动态资源使用策略的 SMT 环境下执行端口时间侧信道攻击防护方法目标是通过执行端口资源使用策略的动态调整,破坏上述 2 种攻击方式在端口冲突构建环节中使用的技术原理,从而切断通过改变指令类型来构建端口冲突进而通过时间信道传递信息的攻击路径,达到防护上述 2 种攻击方式及使用同类技术原理的其他攻击的效果.

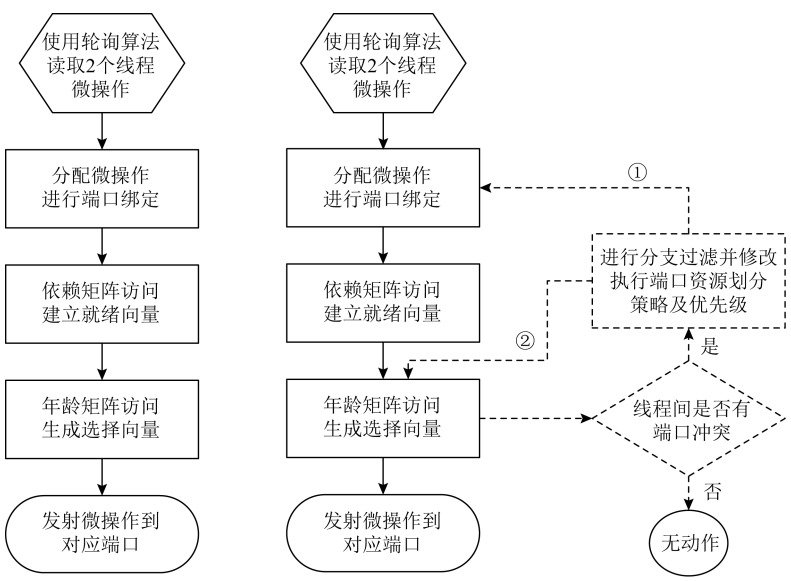
4 防护设计与实现

4.1 设计概述

本文从微架构角度针对 SMT 环境下执行端口时间侧信道攻击设计了一种软件不可见的防护机制,图 6 描述了本文设计的防护设计对 SMT 技术的多端口调度算法的改进.

在 SMT 技术的多端口调度标准算法中,2 个线程通过轮询算法交替选择线程内的微操作送入分配单元,分配单元完成微操作的端口绑定,单端口执行的微操作绑定对应执行端口,多端口执行的微操作采用循环选择算法绑定执行端口.微操作进入调度器后通过依赖矩阵建立就绪向量,通过年龄矩阵生成最终的选择向量,调度器按照当前端口上绑定的已就绪的微操作的年龄顺序依次选择微操作发射到对应端口上.

本文的防护设计对 SMT 技术的多端口调度算法进行了扩充,详细的微架构示意图如图 7 所示.



(a) 标准算法 (b) 增加防护设计算法

Fig. 6 Modification of multi-port scheduling algorithm by protection design

图 6 防护设计对多端口调度算法的改动

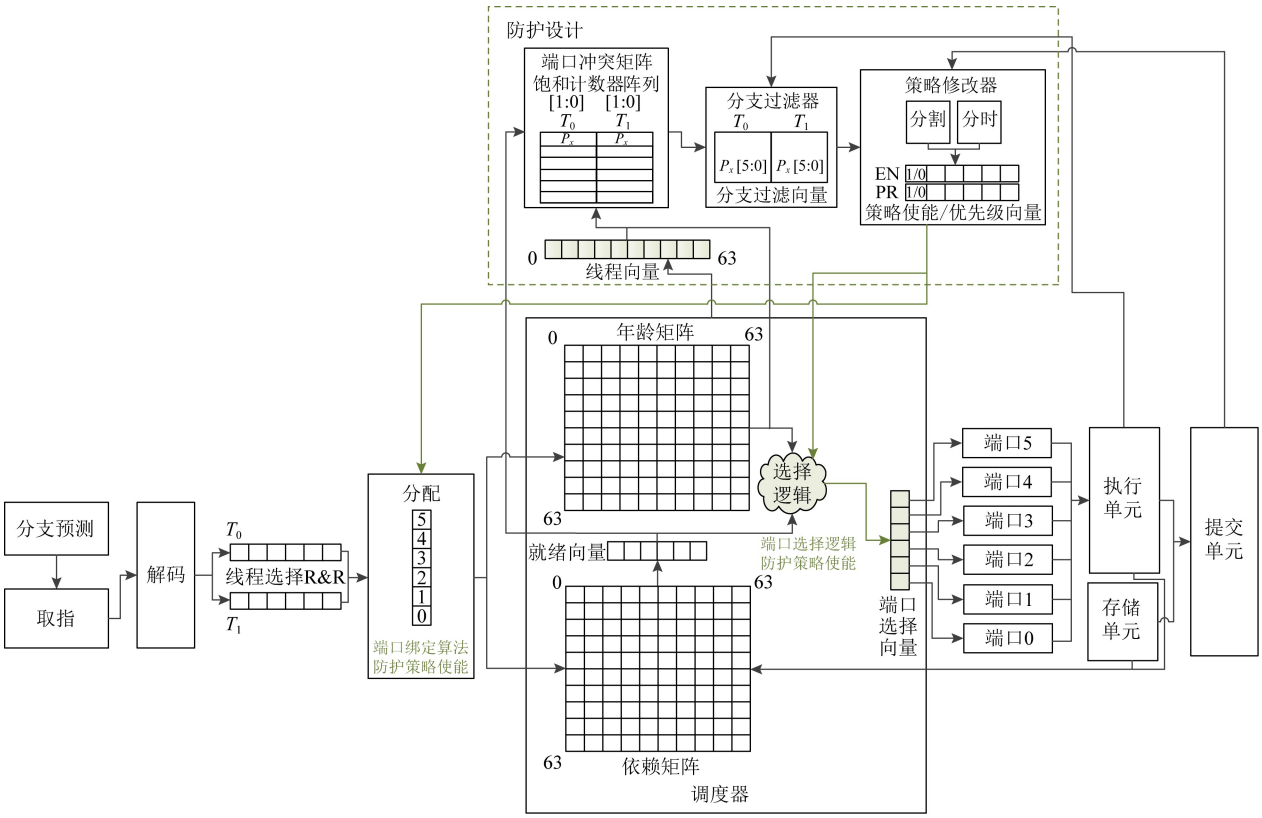


Fig. 7 The micro-architecture of protection design

图 7 防护设计微架构图

在年龄矩阵访问生成最终的选择向量后增加冲突检测的检查环节,本文在原有的依赖矩阵和年龄矩阵之外,新增了端口冲突矩阵的数据结构,该数据结构提取依赖矩阵的就绪向量信息、年龄矩阵的选择向量信息以及调度器的线程向量信息,通过分线程的饱和计数器阵列来记录线程间的端口冲突情况.如果检测到冲突,则将冲突信息提供给分支过滤和策略修改器进行下一步的防护算法应用;如果没有检测到冲突则不会产生额外动作,不会影响正常的调度算法.分支过滤器对分支指令产生的端口冲突信息进行过滤并记录分支刷新产生时的冲突信息,动态资源使用策略修改器(简称:策略修改器)生成策略使能向量和优先级向量进行防护算法的应用.

防护算法应用会改变标准的 SMT 技术的多端口调度算法,主要分 2 个方面:

① 防护策略类型、策略端口使能向量伴随指令操作码会提供给分配单元.分配单元接收到分割的策略类型和策略端口使能向量时,会在原端口绑定算法的基础上将多端口执行的微操作采用的循环选择算法改为端口分割算法.当匹配到微操作的操作码一致时,不同线程的微操作固定使用分开的执行端口.

② 防护策略类型、策略端口使能向量、优先级向量伴随指令操作码会提供给调度器选择逻辑.调度器选择逻辑接收到分时策略、策略端口使能向量以及优先级向量时,会改变原始完全按照年龄矩阵的选择算法,而是基于优先级向量进行优先级选择,即使高优先级线程微操作在年龄矩阵中记录的年龄比另外的线程年轻,其也会被优先选择.当匹配到微操作操作码一致时,选择逻辑按照优先级在对应端口中优先选择优先级高的线程微操作进行分发.

本文的防护设计微架构实现共包含 3 个子模块,分别是端口冲突矩阵、分支过滤器及策略修改器.在整个防护设计中,通过对端口冲突的连续检测形成端口冲突向量;然后通过分支过滤器过滤掉分支类型微操作并且针对 SMoTherSpectre 此类攻击特征进行分支刷新后的优先级调整;最后通过策略修改器进行处理器分配单元及调度器选择逻辑的算法策略修改,消除由于执行端口冲突产生的时间信道,进而防护 SMT 环境下利用双线程共享执行端口进行的时间侧信道攻击.

4.2 端口冲突矩阵

为了记录 SMT 技术 2 个线程的端口冲突,本文的防护方法需要设计实现用于记录 SMT 技术 2 个线程的端口冲突矩阵.

端口冲突矩阵由 2 个线程独立的 2 维数组组成,数组内部的每一行代表 1 个端口,按照图 7 中描述,存在 6 个端口,分别是 P_0 到 P_5,每一个端口都有一个 2 b 的饱和计数器,该计数器的初始值为 0,当某线程因为绑定的端口同另一个线程相同但由于年龄先后的原因而导致延迟发射时,计数器加 1,当某线程绑定的端口连续 2 次发射均没有冲突时,计数器减 1.

判定某线程同另一个线程由于端口资源冲突导致延迟发射的算法如下:

算法 2. 端口冲突延迟发射 $ConflictCounterSet(T_{xi},S_{xi},P,C,Conf_i[1:0])$.

输入:就绪线程端口 T_{xi},选择线程端口 S_{xi},$x=\{0,1\}$,执行端口 $P=\{p_0,p_1,\cdots,p_{n-1}\}$,$i\in\{0,n\}$,时钟周期 C;

输出:冲突饱和计数器 $Conf_i[1:0]$.

① 时钟周期 C 执行:
② for ($0\leqslant i<n$)
③ if ($T_{xi}==p_i$ && $T_{(1-x)i}==p_i$)
④ if ($S_{xi}==p_i$)
⑤ $Conf_i[1-x]=1$;
⑥ else if($S_{(1-x)i}==p_i$)
⑦ $Conf_i[x]=1$;
⑧ end if
⑨ else
⑩ $S_{xi}=p_i$;
⑪ $S_{(1-x)i}=P\backslash p_i$;
⑫ end if
⑬ end for

判定某线程绑定的端口连续 2 次发射均没有冲突的算法如下:

算法 3. 端口冲突取消 $ConflictCounterReset(T_{xi},S_{xi},P,C,Conf_i[1:0])$.

输入:就绪线程端口 T_{xi},选择线程端口 S_{xi},$x=\{0,1\}$,执行端口 $P=\{p_0,p_1,\cdots,p_{n-1}\}$,$i\in\{0,n\}$,时钟周期 C;

输出:冲突饱和计数器 $Conf_i[1:0]$.

① 时钟周期 C 执行:
② for ($0\leqslant i<n$)
③ if (~($T_{xi}==p_i$ && $T_{(1-x)i}==p_i$) && $S_{xi}==p_i$)
④ $Flag_i[x]=1$;
⑤ else if (~($T_{xi}==p_i$ && $T_{(1-x)i}==p_i$) && $S_{(1-x)i}==p_i$)
⑥ $Flag_i[1-x]=1$;
⑦ else
⑧ $Flag_i[x]=0$;
⑨ $Flag_i[1-x]=0$;
⑩ end if
⑪ end for
⑫ 时钟周期 $C+1$ 执行:
⑬ for ($0\leqslant i<n$)
⑭ if (~($T_{xi}==p_i$ && $T_{(1-x)i}==p_i$) && $S_{xi}==p_i$ && $Flag_i[x]==1$)
⑮ $Conf_i[x]=0$;
⑯ else if (~($T_{xi}==p_i$ && $T_{(1-x)i}==p_i$) && $S_{(1-x)i}==p_i$ && $Flag_i[1-x]==1$)
⑰ $Conf_i[1-x]=0$;
⑱ else
⑲ $Conf_i[x]=Conf_i[x]$;
⑳ $Conf_i[1-x]=Conf_i[1-x]$;
㉑ end if
㉒ end for

因为 SMT 技术的公平性设计,冲突会在 2 个线程中间交替出现,当 2 个线程的饱和计数器全为 1 时,则判断该端口存在双线程的冲突.冲突端口的信息会以独热码(one-hot)的形式作为冲突向量给到分支过滤器.

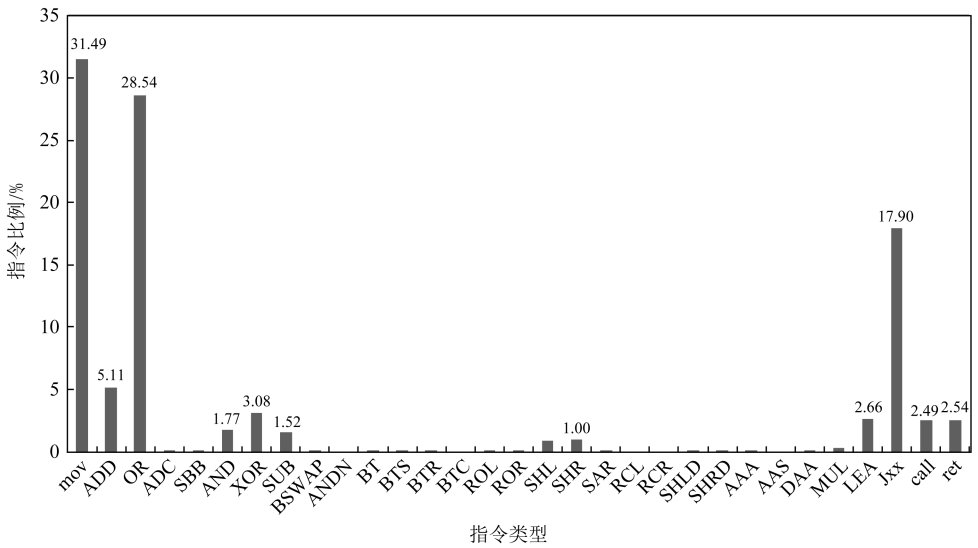
4.3 分支过滤器

在 SMoTherSpectre[6] 中,攻击者通过 Spectre 分支诱导的方式诱导受害者进入 SMoTher Gadget,SMoTher Gadget 包含受害者投机执行的代码片段.因为是诱导投机执行代码,后续会产生分支刷新刷掉该指令流,该指令流使用了攻击者构造的端口冲突代码,长时间执行相同端口的微操作,而这些微操作同攻击者 Time SMoTher Gadget 中的指令冲突与否则会产生对应的时间信道.

分支过滤器的作用是当接收到冲突向量后进行包含分支类型微操作执行单元端口的冲突信息进行过滤并针对 SMoTherSpectre 中使用 Spectre 诱导方式进行冲突构建和安全信息提取的攻击方式进行专门的防护.

分支过滤器除了接收端口冲突矩阵的冲突向量外,还接收译码过后的微操作信息.识别分支指令的目的是对分支指令进行冲突过滤,其他信息会解码后给到后面的策略修改器用于防护策略的算法应用.

对分支指令进行过滤的原因是在普通应用程序中,分支指令占比较高,且在实际攻击应用时因为分支指令执行会产生流水线刷新行为,攻击者不会使用分支指令作为执行端口冲突的构建对象.如图 8 所示,本文选取了常用的 Linux 指令(mount,cat,ls,sh,su,hostname,login,password,id,which),其分支指令类型的占比接近 23%.以 Intel Skylake 微架构为例,其分支执行单元在端口 0 和端口 6,如果一旦端口 0 和端口 6 检测到端口冲突,且分支过滤器获取微操作类型是分支时,过滤器将过滤掉该冲突信息.



注:频度小于1%的指令不显示具体数值

Fig. 8 Linux typical application instruction frequency statistics

图 8 Linux 典型应用指令频度统计

除了分支指令过滤的功能外,分支过滤器还可以通过记录分支刷新的方法来防护使用 Spectre 诱导方式进行冲突构建和安全信息提取的攻击.具体算法流程如下:

算法 4. 分支刷新优先级设置 *BranchFlushPriorSet*(F_x,S_{xi},P,$Conf$[1:0]).

输入:选择线程端口 S_{xi},分支刷新线程 F_x,$x=\{0,1\}$,执行端口 $P=\{p_0,p_1,\cdots,p_{n-1}\}$,$i\in\{0,n\}$,冲突饱和计数器 $Conf$[1:0];

输出:线程优先级 $Prior_x$,$x=\{0,1\}$.

① for ($0\leqslant i<n$)
② if ($S_{xi}==p_i$ && $F_x==1$ && |$Conf$)
③ $Prior_x$--;
④ else if($S_{(1-x)i}==p_i$ && $F_{1-x}==1$ && |$Conf$)
⑤ $Prior_{1-x}$--;
⑥ end if
⑦ end for

线程对某端口的分时调度优先级降低后,仅当另一线程不使用该端口时才调度本线程绑定该端口的微操作,该优先级策略会覆盖策略修改器的优先级策略.

针对 SMoTherSpectre[6] 攻击,分支过滤器通过对分支刷新后冲突端口的记录并降低其优先级的方式可以使得受害者投机执行的 SMoTher Gadget 指令流中的端口冲突指令无法同攻击者 Time SMoTher Gadget 的指令冲突,进而在攻击者的时间度量范围内无法观测到冲突出现,破坏其在信息提取阶段的关键信息,使得 SMoTherSpectre 此类攻击方式失效.

4.4 动态资源使用策略修改器

经过分支过滤器过滤后,最终的端口冲突向量及分支过滤后的刷新冲突记录向量会进入动态资源使用策略修改器,策略修改器用于生成对 SMT 技术的多端口调度算法的防护逻辑并完成新防护算法的应用.

策略修改器主要应用的策略有 2 种,端口分割策略和端口分时策略.为了保证 2 个线程之间的

公平性,默认分时优先级选择采用 Ping-Pong 原则,且该默认优先级受分支过滤器的控制.策略修改器会产生策略类型、策略端口使能向量、优先级向量.策略类型、策略端口使能向量、优先级向量伴随微操作的操作码会同时提供给分配单元和调度器选择逻辑.

策略修改器接收来自于提交单元的刷新信号,当出现中断、事件、异常等单线程事件或 SMT 下线程唤醒、线程睡眠、线程切换等双线程事件时,策略修改器重置,恢复原始的双线程共享多端口调度算法.

以 Intel Skylake 微架构的多端口的执行单元 INT ALU 以及单端口的执行单元 INT MUL 为例,给出策略修改器的算法.

算法 5. 资源策略修改 $ResourceModify(T_{xi}, Prior_x, P, S_{xi})$.

输入:就绪线程端口 T_{xi},线程优先级 $Prior_x$,$x=\{0,1\}$,执行端口 $P=\{p_0,p_1,\cdots,p_{n-1}\}$,$i\in\{0,n\}$;

输出:选择线程 S_{xi},$x=\{0,1\}$.

① 多端口执行单元(INT ALU):
② for ($i\in\{0,1,5,6\}$)
③ if ($T_{xi}==p_i$ && $T_{(1-x)i}==p_i$)
④ $S_{xi}=p_m(m\in\{0,1\})$;
⑤ $S_{(1-x)i}=p_n(n\in\{5,6\})$;
⑥ else
⑦ $S_{xi}=p_i$;
⑧ $S_{(1-x)i}=P\backslash p_i$;
⑨ end if
⑩ end for
⑪ 单端口执行单元(INT MUL):
⑫ if ($T_{xi}==p_1$ && $T_{(1-x)i}==p_1$ && $Prior_x>Prior_{1-x}$)
⑬ $S_{xi}=p_1$;
⑭ else if($T_{xi}==p_1$ && $T_{(1-x)i}==p_1$ && $Prior_x<Prior_{1-x}$)
⑮ $S_{(1-x)i}=p_1$;
⑯ else
⑰ $S_{xi}=p_i$;
⑱ $S_{(1-x)i}=P\backslash p_i$;
⑲ end if

在分配单元,策略修改器将多端口执行的微操作采用的循环选择算法改为端口分割算法.在调度器选择逻辑,策略修改器将完全按照年龄矩阵的选择算法改为基于优先级的选择算法.如果分支过滤器的端口优先级向量使能,该使能信息会覆盖分时优先级 Ping-Pong 选择器结果,固定匹配线程的优先级低于另一线程.

4.5 小 结

本文提出的基于动态资源使用策略的 SMT 环境下执行端口时间侧信道攻击防护方法,可以将 SMT 技术对数据结构分割和分时处理的方式应用到 SMT 环境下执行端口时间侧信道安全防护中,通过对数据结构共享策略和算法的改进来达到防护效果,可以在 SMT 技术合理利用共享数据结构的同时达到安全防护的目的.

在破坏 SMT 技术下执行端口时间侧信道攻击路径上,本防护机制可以同时作用于攻击路径的冲突构建和时间度量 2 个环节上.

首先,防护机制从冲突要素入手,动态资源使用策略会作用于 SMT 技术的多端口分配和调度选择算法上,使得 SMT 环境下 2 个线程不发生同样类型的端口冲突,达到冲突要素防护中不产生冲突防护策略的硬件隔离效果.

其次,本文提出的防护机制在时间要素上同样具有作用,其对时间的干扰主要体现在针对 SMoTherSpectre[6] 类型攻击上,区别于 PortSmash 攻击,SMoTherSpectre 攻击者在进行时间信道的度量时,冲突产生的时间信息及无冲突产生的时间信息都有作用.PortSmash 攻击对时间信息的度量完全基于冲突时间来完成,而 SMoTherSpectre 通过时间度量采集的时间即使没有冲突也可以用于私密信息的分析.通过本文的防护算法应用,从 SMoTherSpectre 攻击者角度,其度量到的时间信息不再准确.从时间要素防护的角度,这也是破坏时间度量精度的一种方式.

5 防护评估

5.1 评估方法概述

本文使用 Gem5[16] 模拟一个高性能多执行端口的超标量处理器作为本文防护方法的防护有效性、性能开销的评估平台,其配置参数如表 2.对标的防护方式是目前针对 SMT 环境下执行端口双线程共享的时间侧信道攻击最有效的防护策略—关闭 SMT 技术.

Table 2 Gem5 Simulator Configuration Parameters

表 2 Gem5 模拟器配置参数

参数名	配置
ISA	ALPHA
Frequency/GHz	1.0
ProcessorType	Out-Of-Order
Fetch Width	4
Fetch Buffer Size/B	16
Decode Width	4
Rename Width	4
Dispatch Width	6
Issue Width	8
Commit Width	4
PhysIntRegs	128
IQ Entries	36
ROB Entries	128
LQ Entries	48
SQ Entries	36
L1 ICache/KB	32
L1 DCache/KB	32
L2 Cache/KB	256
L3 Cache/MB	2
Memory/GB	8

对于硬件开销的评估,本文使用华力 HLMC 40 GP 工艺进行防护方法的硬件开销评估,主要包含面积及时序.

对于防护有效性评估,本文选用了 PortSmash 和 SMoTherSpectre 两篇文章中的 POC(proof of code)代码作为实验目标,分析本文提出的防护设计对这 2 种攻击方式的防护效果.

对于性能评估,本文选用 SPEC CPU 2006 测试集[17]作为性能评估的参考,主要评估防护方法对性能的影响以及同关闭 SMT 技术产生的性能影响进行对比.本文将采用 12 组 SPEC CPU 2006 INT 测试集.通过在 Gem5 上运行测试程序,分别统计关闭 SMT 技术、正常开启 SMT 技术、开启 SMT 技术并且使用本文的防护方法 3 个场景的执行周期数值,然后进行 SPEC CPU 2006 INT 程序的整体性能对比,执行周期数越大,性能越差,周期数越小,性能越好.

本文 SPEC CPU 2006 INT 程序在 Gem5 ALPHA 架构下 12 个测试程序可成功运行 7 个,错误原因均为 Gem5 模拟器对系统调用支持情况的缺陷导致.表 3 是 SPEC CPU2006 整型计算基准程序在本实验下的运行情况:

Table 3 Running Condition of SPEC CPU2006 Integer Base Programs

表 3 SPEC CPU2006 整型计算基准程序运行情况

测试程序	类型	语言	可运行	错误原因
400.perlbench	整型	C	否	fatal:syscall osf_syscall(#0) unimplemented.
401.bzip2	整型	C	是	
403.gcc	整型	C	否	fatal:syscall osf_syscall(#0) unimplemented.
429.mcf	整型	C	是	
445.gobmk	整型	C	否	fatal:syscall osf_syscall(#0) unimplemented.
456.hmmer	整型	C	是	
458.sjeng	整型	C	否	fatal:syscall osf_syscall(#0) unimplemented.
462.libquantum	整型	C	是	
464.h264ref	整型	C	是	
471.omnetpp	整型	C++	是	
473.astar	整型	C++	是	
483.xalancbmk	整型	C++	否	fatal:syscall osf_syscall(#0) unimplemented.

对于硬件开销评估,本文提出的防护机制使用 RTL(register-transfer level)进行代码实现.完成设计实现后的防护设计代码在 HLMC 40 GP 工艺下使用新思科技的 ASIC 设计流程和工具进行面积和时序的评估.

5.2 防护有效性评估

本节防护有效性评估针对 PortSmash 和 SMoTherSpectre 两个攻击案例进行,在禁用[7]或动态关闭 SMT[9]技术后,2 个攻击案例均无法实现,因此,禁用或关闭 SMT 的防护效果为最佳,本文的防护设计目的是尽量接近禁用或关闭 SMT 技术的防护效果.

PortSmash 参考其针对 Intel Skylake 微架构的 POC 代码片段,攻击者循环执行某端口指令,受害者同样会使用该端口指令,也会使用其他端口指令.为了使攻击效果明显,设计受害者循环进行相同操作,攻击者也循环进行相同操作且攻击者和受害者同时在开启 SMT 技术的 2 个逻辑核上执行.SMoTherSpectre 选用其攻击模型中使用的 POC 代码片段类型,受害者运行程序中包含了攻击者植入的一个 SMOTHER 代码片段,正常运行过程中,受害者不会运行 SMOTHER 代码片段,其在加载完核心参数后会跳走,但由于攻击者使用了 Spectre 的分支预测诱导机制将受害者跳转指令的目标地址诱导进 SMOTHER 代码片段中,进而实施执行端口冲突

的时间侧信道攻击,其技术原理同 PortSmash 一致.

将攻击者和受害者的运行代码使用 gcc 编译器编译成可执行二进制文件,其中攻击者可以使用同受害者端口一致的指令类型,也可以使用同受害者端口不同的指令类型.首先使用原始 Gem5 模拟器,开启 SMT 选项,且同时加载 PortSmash 攻击的攻击者和受害者 POC 代码的二进制可执行文件,配置不同冲突次数和频度,通过原始 Gem5 模拟攻击者通过端口冲突获取受害者信息的攻击路径,并统计运行时间来提取端口冲突时间信息;然后使用增加防护设计后的 Gem5 模拟器,进行同样的操作.在本实验过程中,为了测试不同冲突频度产生的冲突效果,选取了 60～30 720 共 10 组不同量级的指令冲突频度进行测量.

图 9 为防护效果评估对比,SMTwC 表示 SMT 下端口冲突的攻击者执行周期;SMTwoC 表示 SMT 下端口无冲突的攻击者执行周期;SMTwDwC 表示 SMT 下增加防护设计后有端口冲突的攻击者执行周期;SMTwDwoC 表示 SMT 下增加防护设计后端口无冲突的攻击者执行周期.表 4 详细给出了 Gem5 输出的时钟周期数据.

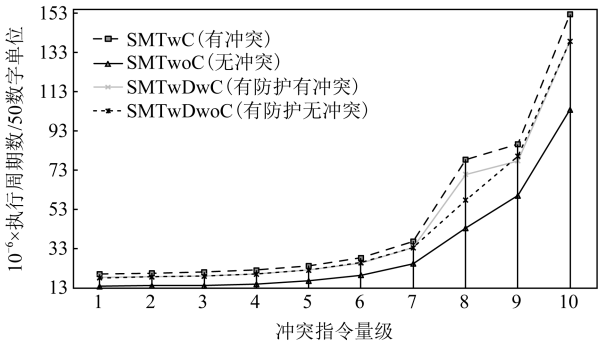


Fig. 9 Evaluation of protective effect

图 9 防护效果评估

从图 9 中的 SMTwC 和 SMTwoC 的对比可以看出,在 SMT 技术下 PortSmash 和 SMoTherSpectre 攻击中攻击者使用的攻击指令同受害者同端口指令出现冲突时其冲突造成的时间延迟和冲突规模呈线性关系,而不产生冲突时,其指令延迟信息也呈现明显的线性关系,这样通过时间延迟的不同可以提取出受害者程序执行的指令类型和次数.

Table 4 Effect of Defending Method

表 4 防护效果数据

冲突指令量级	执行周期数/50 数字单位			
	SMTwC(有冲突)	SMTwoC(无冲突)	SMTwDwC(有防护有冲突)	SMTwDwoC(有防护无冲突)
60	19 893 500	13 554 000	18 326 500	18 009 000
120	20 369 250	13 901 650	18 609 000	18 418 000
240	20 879 650	13 892 650	19 013 000	18 781 500
480	21 824 550	14 743 850	19 893 000	19 834 500
960	24 083 950	16 377 350	21 892 000	21 858 000
1 920	28 123 700	19 368 750	25 369 000	25 665 000
3 840	36 544 750	25 217 250	33 280 000	33 284 000
7 680	78 300 200	43 303 850	70 818 500	57 683 500
15 360	86 156 950	59 845 500	77 720 500	80 082 500
30 720	152 462 750	103 725 750	138 660 500	138 662 500

从图 9 中的 SMTwDwC 和 SMTwDwoC 的对比可以看出,增加防护设计后,由于冲突端口随着冲突指令量级增大会使能分割或分时的动态资源使用策略,攻击者程序即使存在同受害者的指令冲突且冲突造成的时间延迟和冲突规模呈明显的线性关系,但是使用同受害者不产生冲突的指令类型其造成的时间延迟和冲突规模也呈明显的线性关系且同有冲突时趋于一致.通过该实验数据表明,攻击者使用 PortSmash 和 SMoTherSpectre 类型的攻击方式通过度量冲突时间来采集时间信道过程中会有明显的误差和干扰存在,无法判断受害者使用的指令类型和频度,可以达到禁用或动态关闭 SMT 技术的防护效果,进而防止受害者指令执行端口、指令执行次数及指令类型信息的泄露.

5.3 性能开销评估

本文防护设计的性能评估选用 SPEC CPU 2006 测试集作为性能评估的参考,主要评估防护方法对性能的影响以及同禁用或动态关闭 SMT 技术

(DDM[9])产生的性能影响进行对比.

1) 评估样本.7 个 SPEC CPU 2006 INT 测试程序.

2) 评估方式.使用总执行周期数据指标作为性能优劣的核心评估参数.

3) 评估目标.对比禁用 SMT 技术、正常开启 SMT 技术、开启 SMT 技术并且使用本文的防护方法 3 个场景的程序总执行周期数据(其中 401 和 471 的程序规模较大,其统计的时钟周期数进行等比例缩减,缩减比例 100:1),分析本文的防护方法对性能的影响程度.

4) 对比数据(如图 10 所示).

① NOSMT——关闭 SMT 技术后 Gem5 运行 7 组 SPEC CPU 2006 INT 测试程序的执行周期数;

② SMT——正常开启 SMT 技术后 Gem5 运行 7 组 SPEC CPU 2006 INT 测试程序的执行周期数;

③ DEFENCE——开启 SMT 技术并且使用本文的防护方法下 Gem5 运行 7 组 SPEC CPU 2006 INT 测试程序的执行周期数.

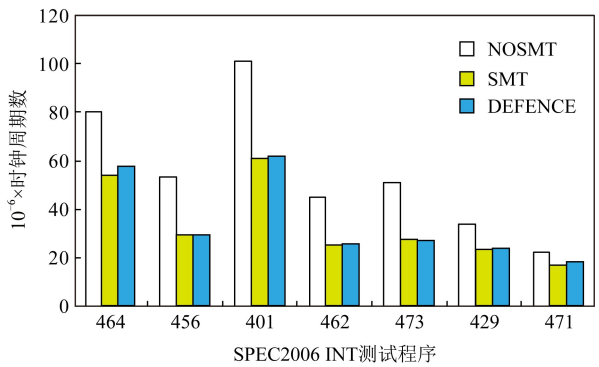


Fig. 10 Performance comparison

图 10 性能对比

5) 评估结论.使用本文的防护设计后 SPEC CPU 2006 INT 测试集测试出的执行性能平均下降 2.98%,性能下降最明显的是 471,下降比例为 7.77%,也优于 DDM[9] 动态关闭 SMT 的性能表现(SPEC2000 评测约 8%的性能损失),且以 473 为代表的测试程序在增加防护设计后性能没有下降,且有 0.5%的性能提升.

综上可以得出,本文防护方法对性能的影响整体可控,且相比于禁用或动态关闭 SMT 技术有明显的性能优势.

5.4 硬件开销评估

本文描述的防护机制使用 Verilog 语言进行 RTL 实现.然后使用 HLMC 40 GP 工艺和新思科技的 ASIC 设计流程和工具进行面积和时序的评估.

本文防护机制设计为 1 个独立的单元,命名为 smt_defend,该单元支持针对 6 个执行端口,64 个调度器项的处理器微架构设计.

本文使用新思科技 DC-Compiler 工具,设置时钟约束为 2 GHz,clk_uncertainty 参数值为 100 ps.本文防护设计的时序和面积数据如表 5 所示:

Table 5 Hardware Overhead Evaluation

表 5 硬件开销评估

模块	面积开销/μm^2	时序开销(reg2reg)/ps	时序开销(in2reg)/ps	时序开销(reg2out)/ps
smt_defend	4 877	357	381	300
conflict_matrix	4 607			
branch_filter	135			
policy_modifier	132			
int exec(对比)	302 979			

经过综合工具实现,smt_defend 单元总面积为 4 877 μm^2,在时序方面,3 类时序数据的最差路径分别为 reg2reg(寄存器到寄存器)时序最差路径为 357 ps;in2reg(输入到寄存器)时序最差路径为 381 ps;reg2out(寄存器到输出)时序最差路径为 300 ps.在 2 GHz的时钟约束下无任何时序违例.

3 个子模块 conflict_matrix, branch_filter, policy_modifier 的面积数据分别为 4 607 μm^2, 135 μm^2, 132 μm^2.

为了进行面积和时序的对比,本文使用 DC-Compiler 工具在相同的设置和约束下分析了 6 个执行端口整型执行单元的整体面积约为 302 979 μm^2,本防护设计单元的面积为其 1.6%,硬件开销可控.

相比于处理器内部的基本单元,本文描述的防护机制实现在硬件开销上可控,且该防护机制在流水线设计上,并不影响直接影响 SMT 技术的多端口调度算法时序,在时序角度上不存在设计负担.

6 结 论

本文针对 SMT 环境下执行端口时间侧信道攻击防护的一些相关工作进行了系统的分析、归纳.同时,本文提出了一种基于动态资源使用策略的 SMT 环境下执行端口时间侧信道攻击防护方法,在 SMT 技术对共享数据结构的处理算法上进行改进,将不同的资源处理策略应用到 SMT 环境下执行端口时间侧信道安全防护中,通过对数据结构共享策略和算法的改进来达到防护效果,可以在 SMT 技术合

理利用共享数据结构的同时达到安全防护的目的.经过充分评估,本文方法从防护有效性上针对已有的攻击方式有很好的防护效果,基本和关闭 SMT 技术的防护方式等效;在性能开销上远远好于关闭 SMT 技术的防护方法;在硬件开销角度,本文防护方法的设计简洁,面积和时序影响小,具有一定的实用价值.

作者贡献声明:岳晓萌提出可动态调整双线程共享的执行端口资源使用策略的防护方法,并进行了实验;杨秋松指导并优化动态资源使用防护策略的算法,审阅文章内容,提出改进建议;李明树指导研究方向和技术方法,审阅文章内容,提出改进建议.

参 考 文 献

[1] Marr D T, Binns F, Hill D L. Hyper-threading technology architecture and microarchitecture [J]. Intel Technology Journal, 2002, 6(1): 1-12

[2] Acıçmez O, Seifert J P. Cheap hardware parallelism implies cheap security [C] //Proc of the Workshop on Fault Diagnosis & Tolerance in Cryptography. Piscataway, NJ: IEEE, 2007: 80-91

[3] Wang Zhenghong, Lee R. Covert and side channels due to processor architecture [C] //Proc of the 22nd Annual Computer Security Applications Conf. Piscataway, NJ: IEEE, 2006: 473-482

[4] Fogh A. Automatically finding SMT covert channels [EB/OL]. 2016 (2016-09-27) [2019-10-12]. https://cyber.wtf/2016/09/27/covert-shotgun/

[5] Aldaya A, Brumley B, Hassan S U, et al. Port contention for fun and profit [C] //Proc of the Symp on Security and Privacy. Piscataway, NJ: IEEE, 2019: 19-23

[6] Bhattacharyya A, Sandulescu A, Neugschwandtner M, et al. SMoTherSpectre: Exploiting speculative execution through port contention [C] //Proc of the 2019 ACM SIGSAC Conf on Computer and Communications Security. New York: ACM, 2019: 785-800

[7] Percival C. Cache missing for fun and profit [J/OL]. (2015-02-08) [2019-12-16]. http://www.daemonology.net/papers/htt.pdf

[8] Hu Weiming. Reducing timing channels with fuzzy time [C] //Proc of the 1991 IEEE Computer Society Symp on Research in Security and Privacy. Piscataway, NJ: IEEE, 1991: 8-20

[9] Zhang Yue, Zhu Ziyuan, Meng Dan. DDM: A demand-based dynamic mitigation for SMT transient channels [J]. arXiv preprint, arXiv:1910.12021, 2019

[10] Koç Ç K. Microarchitectural attacks and countermeasures [M] //Cryptographic Engineering. Berlin: Springer, 2009: 475-504

[11] Ge Qian, Yarom Y, Cock D, et al. A survey of microarchitectural timing attacks and countermeasures on contemporary hardware [J]. Journal of Cryptographic Engineering, 2016, 8(1): 1-27

[12] Marshall A, Howard M, Bugher G, et al. Security best practices for developing windows azure applications [J/OL]. Microsoft Corporation, 2010-06 [2019-07-12]. http://citeseerx.ist.psu.edu/viewdoc/download;jsessionid=0658D6B60EE42435E728A049FC4C0E40?doi=10.1.1.455.6671&rep=rep1&type=pdf

[13] Vattikonda B, Das S, Shacham H. Eliminating fine grained timers in Xen [C] //Proc of the 3rd ACM Workshop on Cloud Computing Security Workshop. New York: ACM, 2011: 41-46

[14] Martin R, Demme J, Sethumadhavan S. TimeWarp: Rethinking timekeeping and performance monitoring mechanisms to mitigate side-channel attacks [J]. ACM SIGARCH Computer Architecture News, 2012, 40(3): 118-129

[15] Kocher P, Genkin D, Gruss D, et al. Spectre attacks: Exploiting speculative execution [J]. Communications of the ACM, 2020, 63(7): 93-101

[16] Binkert N, Beckmann B, Black G, et al. The gem5 simulator [J]. ACM SIGARCH Computer Architecture News, 2011, 39(2): 1-7

[17] Henning J L. SPEC CPU2006 benchmark descriptions [J]. ACM SIGARCH Computer Architecture News, 2006, 34(4): 1-17

Yue Xiaomeng, born in 1989. PhD. His main research interests include operating system, computer architecture and system security.

岳晓萌,1989 年生.博士.主要研究方向为操作系统、计算机架构和系统安全.

Yang Qiusong, born in 1977. PhD, professor, PhD supervisor. His main research interests include operating system, software engineering and system security.

杨秋松,1977 年生.博士,教授,博士生导师.主要研究方向为操作系统、软件工程和系统安全.

Li Mingshu, born in 1966. PhD, professor, PhD supervisor. Fellow of CCF. His main research interests include operating system, software engineering and distributed system.

李明树,1966 年生.博士,教授,博士生导师,CCF 会士.主要研究方向为操作系统、软件工程和分布式系统.