基于边界检测的安全数据预取方案

吝常青[1,2], 田 鑫[1], 侯 锐[1], 孟 丹[1]

[1] 中国科学院信息工程研究所 北京 中国 100093
[2] 中国科学院大学 网络空间安全学院 北京 中国 100049

摘要 为了不断提升微处理器的性能, 现代微处理器当中包含了越来越多用于性能优化的部件, 比如高速缓存, 分支预测器, 数据预取器等, 这些性能优化部件在给微处理器带来可观的性能提升的同时, 也引入了一定的安全隐患。比如高速缓存引入的侧信道, 分支预测引入的“幽灵”漏洞等等, 与上述两个性能优化部件类似, 数据预取也存在安全隐患, 然而却未引起足够的重视。数据预取的根本目的在于提升高速缓存命中率, 主要通过观察程序的访存行为规律提前将所需的数据加载到高速缓存当中, 是现今高性能微处理器当中重要的微处理器性能优化技术。近来有研究表明, 数据预取会引入侧信道, 造成信息泄露, 对微处理器的整体安全性造成了一定的威胁, 然而目前却鲜有关于如何对数据预取安全缺陷进行防御的相关研究。性能优化部件之所以引入安全风险的根本在于其具有推测性, 当推测的处理器行为与实际的行为不符时, 便会在处理器内部遗留下“脏数据”, 这些“脏数据”有可能来自于越权或者越界访问。本文重点分析了硬件数据预取目前面临的安全风险及其产生原因, 提出了安全的数据预取行为规范, 在开源处理器 BOOM(Berkeley Out of Order Machine)上实现了基于指令指针(instruction pointer)的步距预取器, 同时依据上述安全的数据预取行为规范, 实现了具有边界检测功能的安全数据预取系统, 最后对其安全性和性能开销进行了简要评估。

关键词 数据预取器; Cache 侧信道攻击; 信息泄露
中图法分类号 TP309 DOI 号 10.19363/J.cnki.cn10-1380/tn.2022.01.08

Security Data Prefetching Scheme Based on Boundary Detection

LIN Changqing[1,2], TIAN Xin[1], HOU Rui[1], MENG Dan[1]

[1] Institute of Information Engineering, Chinese Academy of Sciences, Beijing 100093, China
[2] School of Cyber Security, University of Chinese Academy of Sciences, Beijing 100049, China

Abstract In order to continuously improve the performance of microprocessors, modern microprocessors contain more and more components for performance optimization, such as caches, branch predictors, data prefetchers, etc., these performance optimization components speed up the microprocessor’s performance. At the same time of considerable performance improvement, certain security risks have been introduced also. For example, side channels introduced by cache, "spectre" vulnerabilities introduced by branch prediction, etc., similar to the above two performance optimization components, data prefetching also has security risks, but haven’t attracted enough attention yet. The fundamental purpose of data prefetching is to increase the cache hit rate. It is mainly used to load the required data into the cache in advance by observing the memory access rules of the program. It is an important microprocessor performance optimization technology among today's high-performance microprocessors. Recent studies have shown that data prefetching will introduce side channels, causing information leakage, and posing a certain threat to the overall security of microprocessors. However, there are currently few related studies on how to defend against security flaws in data prefetching. The fundamental reason why performance optimization components introduce security risks is that they are speculative. When the speculative processor behavior does not match the actual behavior, “dirty data” will be left in the processor. These “dirty data” may come from out of bounds access or unauthorized access. This article focuses on analyzing the current security risks faced by hardware data prefetching and their causes, and proposes a safe data prefetching code of conduct, and implements an instruction pointer based on the open-source processor BOOM (Berkeley Out of Order Machine). At the same time, according to the above-mentioned safe data prefetching behavior specification, a secure data prefetching system with boundary detection function is realized. Finally, a brief evaluation of its security and performance overhead is carried out.

Key words hardware prefetch; microarchitectural side-channel attacks; information leakage

通讯作者: 侯锐, 博士, 研究员, Email: hourui@iie.ac.cn。
本课题得到中国科学院战略性先导科技专项(No. XDC02010200)资助。
收稿日期: 2020-01-08; 修改日期: 2020-02-24; 定稿日期: 2021-11-16

1 引言

微处理器是现代计算机系统的核心, 它的性能优化一直是微体系结构研究的重点。现代微处理器的处理速度越来越快, 而存储器访问速度的提升并没有随着微处理器同等提升, 这种现象导致了微处理器的运行速度和存储器访问速度之间的差距越来越大, 从而严重地影响了整个处理器的性能。为了解决上述速度差问题, 体系结构设计者引入了缓存(Cache)的概念, 根据程序访存的时间局部性和空间局部性原理, 将近期访问过的数据存入 Cache 当中。

随着微处理器的处理速度和访存速度之间的差距进一步加大, Cache 缺失率对处理器整体处理速度的影响日益加剧, 尤其是 Cache 并不能解决首次访存缺失。为了更进一步提高 Cache 命中率, 提高处理器的整体性能, 数据预取技术[1]应运而生。数据预取技术是在微处理器对数据的需求产生之前预测到数据需求, 提前将数据从主存当中取回放入 Cache, 从而避免首次访存缺失, 有效地提高了 Cache 命中率。

高速缓存和数据预取的出现, 虽然降低了微处理器数据访存延迟, 却存在被恶意利用进行敏感信息窃取的可能性[2]。例如, 攻击者可以利用 Cache 的共享特性构造侧信道攻击。

近几年, 有研究者利用软件预取指令的不安全特性[3]和硬件预取器的微体系结构特征[4-5]进行了一系列攻击的尝试, 证明了数据预取确实存在安全缺陷, 但是鲜有针对数据预取技术的安全防御研究。本文主要分析总结了硬件数据预取的安全问题, 提出了新的安全数据预取方案, 主要贡献如下:

- 总结了目前硬件数据预取面临的安全威胁, 并重点分析了基于指令指针的步距预取器引发的侧信道攻击成因;
- 为了解决硬件数据预取目前的安全缺陷, 提出了基于边界检测的安全预取行为规范;
- 在微体系结构级别, 以基于指令指针的硬件数据预取器为基础, 实现了一个安全数据预取系统。

2 背景知识

针对数据预取的攻击, 往往需要利用 Cache 访问命中与缺失造成的时间差异信息, 同时需要借助共享 Cache 来构造侧信道。本章主要介绍两类典型的 Cache 侧信道攻击和基于指令指针的步距预取器的工作原理。

2.1 Cache 侧信道攻击

Cache 作为提升微处理器性能的主要部件, 可以有效地降低平均访存延迟, 同时它也是被攻击的主要部件。

在主流处理器微架构设计当中, Cache 是由多个 Cache Block 组成, 采用多路组相联结构, 即多行(组)多列(路)的结构。微处理器当中一般存在多个 Cache, 它们基本都按照存储层次结构来组织, 典型的 Data Cache 主要包括 L1 Cache, L2 Cache, L3 Cache 等, L1 Cache 又细分为 Data Cache 和 Instruction Cache。

当处理器需要从主存读取数据时, 首先查看需要的数据是否存在于 L1 Cache 当中。如果存在, 则直接将数据返回给处理器(称为访存命中); 否则, 需要向下一级的 Cache 或者主存储器请求数据(称为访存缺失)。

访存命中和缺失情况下的数据请求完成时间不同, 攻击者可以利用这种时间差异信息来推测秘密信息, 这类攻击被称为 cache-timing 攻击, 这种攻击也是最早的 Cache 侧信道攻击。

基于 cache-timing 的侧信道攻击当中, 最具有代表性且攻击能力较强的是 Prime+Probe[6]和 Flush+Reload[7]的攻击方式。

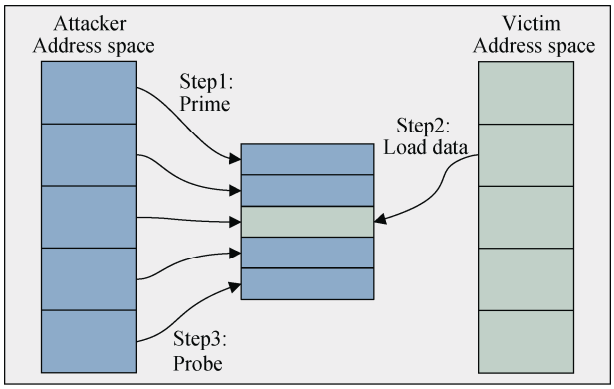

图 1 Prime+Probe
Figure 1 Prime+Probe

Prime+Probe: 在 Prime+Probe 攻击方式当中, 攻击者首先使用自己的数据或者代码对整个 Cache 进行填充(Prime), 之后等待受害者程序运行, 当受害者程序运行完成后, 攻击者使用相同的数据或代码对 Cache 重新填充, 并测量重新填充的时间。

如果受害者程序访问到攻击程序填充的组, 会将攻击者之前填充的数据替换出 Cache, 攻击者重新

填充的时候会造成访存缺失, 重填时间较长; 而未被受害者程序访问的组不会发生数据块的替换, 攻击者重新填充的时候, 则重填时间较短。

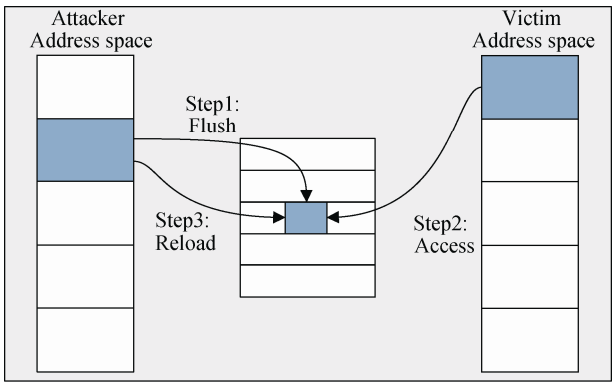

图 2 Flush+Reload
Figure 2 Flush+Reload

Flush+Reload: Flush+Reload 侧信道攻击方式粒度更细, 可以精确到具体的 Cache 块, 所以被认为是最具有破坏力的一种侧信道攻击。

Flush+Reload 主要是利用攻击者程序和受害者程序之间存在共享内存或者共享库的特性, 攻击者首先通过 clflush 指令将指定的数据块替换出 Cache, 等待受害者程序运行, 然后测量之前替换出 Cache 的数据块的重新加载时间, 根据时间的长短来判断受害者程序是否使用了指定的数据块。如果受害者程序使用了指定的数据块, 那么必定会将指定的数据块加载到 Cache 当中, 攻击者重新加载的时间必然较短, 否则需要从主存加载, 需要较长时间。

2.2 硬件数据预取

目前主流处理器通过两种方式支持数据预取, 一种是基于软件的预取, 需要编译器支持; 另一种是基于硬件的预取, 通过在 Cache 层级当中添加数据预取单元, 来完成数据预取功能。

硬件数据预取器是微体系结构级别的一个部件, 专门负责数据预取, 它对应用程序而言是透明的, 无需额外的预取指令和编译器修改。现今主流的处理器中都实现了基于指令指针的步距预取器, 比如 Intel 处理器的 Sandy Bridge 微架构以及后续的微架构[8], 因此下文将详细介绍基于指令指针的步距预取器的工作原理。

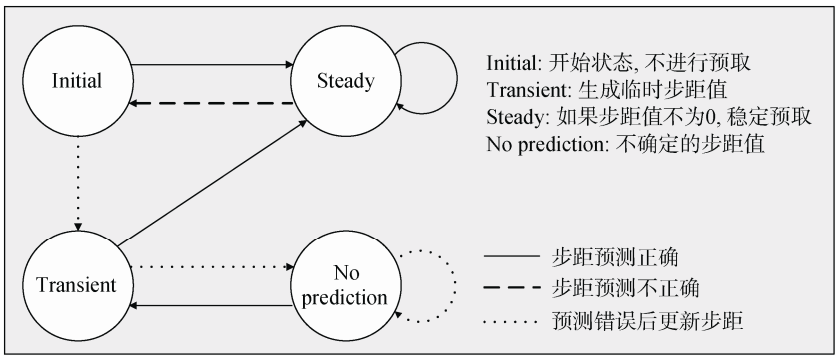

图 3 RPT 表项的状态转换图
Figure 3 State transition graph for RPT entries

基于指令指针的步距预取: 在大多数应用程序中, 数据的访问行为往往是跨越式的, 基于指令指针的步距预取通过记录同一个访存指令连续访存地址之间的步距来学习程序的访存行为。假设访存指令 *mem_i* 在连续 3 次访存中的访存地址分别是a_1, a_2, a_3。当$a_2 - a_1 = \Delta \neq 0$时, 步距预取器会将步距初始化为$\Delta$, 同时对下一个访存地址进行预测$A_3 = a_2 + \Delta$, 若$A_3 = a_3$, 则表明预测成功, 并继续以这个步距对后续的访存地址进行预测, 直到$A_n \neq a_n$。

基于指令指针的步距预取需要保存访存历史地址信息和最后一次预测成功的Δ值, 由于不可能将程序执行中的每一条访存指令信息寄存下来, 所以设计了访问预测表[1], RPT(Reference Prediction Table), 专门记录程序近期的访存指令相关信息, RPT 的结构如图 4 所示。

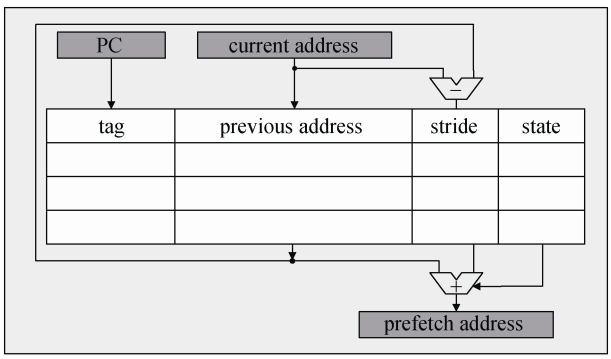

图 4 RPT 结构图
Figure 4 The organization of the RPT

RPT 表的每一项包含访存指令的 PC, 上一次访存的地址信息, 步距以及状态信息。RPT 表由访存指令的 PC 值进行索引, 当访存指令 *mem_i* 第一次执行时, 将从 RPT 中为其分配一个空闲的项, 并填写对应的指令 tag, 访存地址, 同时将这一项的状态置为 initial 状态, stride 值设为 0。当 *mem_i* 指令再次被执行, 并在 RPT 表中命中时, 则将当前的访存地址与从 RPT 表中读出的 previous address 相减, 得到的 stride 值填入 RPT 表当中, 同时用当前的访存地址更新 previous address, 最后将这一项的状态置为 transient, 此时的步距预取器处于不稳定状态。当 *mem_i* 指令第 3 次被执行, 并在 RPT 表中再次命中时, 将当前的访存地址和 RPT 表中旧的 previous address 相减得到新 stride 值, 新的 stride 值和 RPT 表中旧的 stride 值进行比较, 如果相等则将状态置为 steady, 步距预取器处于稳定预取状态; 否则更新 stride 的值, 数据预取器继续处于 transient 状态。RPT 表的状态迁移过程如图 3 所示。

3 基于数据预取的攻击

在 2012 年, Sarani Bhattacharya 等人利用侧信道脆弱性因子(SVF)对数据预取的安全性进行理论评估[9], 发现数据预取会影响恒定时间密码算法的抗侧信道攻击能力, 可能会造成信息泄露。近几年, 陆续有攻击者利用商用处理器的数据预取技术实施攻击。比如 Daniel Gruss 等利用预取指令绕过 ASLR 获取关键函数的地址信息[3], Youngjoo Shin 等利用硬件预取器的微体系结构特征[4]进行侧信道攻击, Patrick Cronin 等利用 RPT 表实施隐蔽信道攻击[5]。本章将详细分析基于指令指针的步距预取器造成侧信道攻击的成因, 同时对数据预取存在的安全问题进行简要总结。

3.1 数据预取侧信道

基于硬件数据预取进行侧信道攻击的方式主要是利用硬件数据预取器预取了并非程序真正所需的数据, 这些数据遗留在 Cache 当中成为了脏数据, 这些脏数据可能会构成 Cache 侧信道, 进而被攻击者利用窃取秘密信息。简单起见, 下文以攻击基于椭圆曲线的密码算法[10]为例进行介绍。

3.1.1 预取器的触发

标量点乘法是基于椭圆曲线的加密算法的核心运算, 具体地, 给定椭圆曲线上的一个点 P 和一个大整数 k, 将点 P 与自身相加 k 次。基于椭圆曲线的二进制域平方运算是标量点乘法当中重要且常见的运算形式。

算法 1. 二进制有限域平方运算

输入: $X \in GF(2^m)$

输出: $X^2 \in GF(2^m)$

1: $(X_{w-1}, X_{w-2}, \cdots, X_0) \leftarrow X$
2: *for* $i \leftarrow w-1$ *to* 0 *do*
3: $(x_{15}, x_{14}, \cdots, x_0) \leftarrow X_i$;
4: $W_{15} \leftarrow SQR\_tb[x_{15}] << 56$; $W_{14} \leftarrow SQR\_tb[x_{14}] << 48$;
5: $W_{13} \leftarrow SQR\_tb[x_{13}] << 40$; $W_{12} \leftarrow SQR\_tb[x_{12}] << 32$;
6: $W_{11} \leftarrow SQR\_tb[x_{11}] << 24$; $W_{10} \leftarrow SQR\_tb[x_{10}] << 16$;
7: $W_9 \leftarrow SQR\_tb[x_9] << 8$; $W_8 \leftarrow SQR\_tb[x_8]$;
8: $W_7 \leftarrow SQR\_tb[x_7] << 56$; $W_6 \leftarrow SQR\_tb[x_6] << 48$;
9: $W_5 \leftarrow SQR\_tb[x_5] << 40$; $W_4 \leftarrow SQR\_tb[x_4] << 32$;
10: $W_3 \leftarrow SQR\_tb[x_3] << 24$; $W_2 \leftarrow SQR\_tb[x_2] << 16$;
11: $W_1 \leftarrow SQR\_tb[x_1] << 8$; $W_0 \leftarrow SQR\_tb[x_0]$;
12: $Y_{2i+1} \leftarrow W15 \vee W14 \vee \ldots \vee W8$;
13: $Y_{2i} \leftarrow W7 \vee W6 \vee \ldots \vee W0$;
14: *end for*
15: $Y \leftarrow (Y_{2w-1}, Y_{2w-2}, \cdots, Y_1, Y_0)$
16: *return* $Y \bmod f$

(注: SQR_tb 为提前计算好的查找表)

因为基于椭圆曲线的二进制域平方运算非常重要且很耗时, 所以有很多针对平方运算的计算效率进行优化改进的研究。最常见的优化方法是基于查找表的方式, 提前将结果计算好, 存储于一张表中, 程序在运行时无需实时计算, 直接使用查表的方式得到计算结果[11]。如算法 1, 即是 OpenSSL(1.1.0g) 中使用查找表方式实现的基于椭圆曲线的二进制域平方运算。其中 SQR_tb 为提前计算好的查找表, 是一个具有 16 项 64bit 数据的列表, 当输入一个二进制形式的数据X进行运算时, 首先将其以 64bit 为单位划分为一个序列$(X_{w-1}, X_{w-2}, \ldots, X_0)$, 在每一次循环开始, 每一个序列中的元素$X_i$又被拆分为包含 16 个 4bit 的数据序列$(x_{15}, x_{14}, \ldots, x_0)$, 此序列中的每一个元素都被作为 SQR_tb 的索引使用, 每一次循环结束X_i都被转换为两个连续的 64bit 数据: Y_{2i}和Y_{2i+1}。

从算法 1 中可以看出, 基于查找表的二进制域平方算法主结构是一个循环结构, 其中每一次循环的X_i值按 4 比特被等分为 16 个索引值x_i作为查找表的索引, 如果相邻几次循环的X_i值中用于索引的x_i存在固定的差值, 比如连续几次循环的x_{15}, 则有可能会触发基于指令指针的步距预取器进行数据预取。

3.1.2 侧信道的形成

在程序运行基于椭圆曲线的二进制域平方算法时, 如果在索引查找表过程中访存步长有规律可循, 则会触发基于指令指针的步距预取器进入数据预取

状态，然而数据预取器没有查找表的边界信息，所以很可能会将不属于查找表的数据越界预取到 Cache 当中。

算法 2. 无分支的蒙哥马利阶梯算法

输入：椭圆曲线点 P 的 x 轴坐标值和大整数 k

输出：点 $Q=kP$ 的仿射坐标

$(k_{l-1}, \ldots, k_1, k_0) \leftarrow k$
$X_1 \leftarrow x, Z_1 \leftarrow 1, X_2 \leftarrow x^4 + b, Z_2 \leftarrow x^2$
for $i \leftarrow l-2$ *to* 0 *do*
  $\beta \leftarrow k_i$
  $Const\_Swap(X_1, X_2, \beta), Const\_Swap(Z_1, Z_2, \beta)$
  $Madd(X_2, Z_2, X_1, Z_1), Mdouble(X_1, Z_1)$
  $Const\_Swap(X_1, X_2, \beta), Const\_Swap(Z_1, Z_2, \beta)$
end for
return $Q = Mxy(X_1, Z_1, X_2, Z_2)$

为了防御侧信道攻击，基于椭圆曲线的各类密码算法在具体实现时，都具有抗 Cache 侧信道攻击的能力。以标量点乘法为例，OpenSSL(1.1.0g)在实现基于椭圆曲线的 DH 密码算法时，其中的标量点乘法采用的是无分支蒙哥马利阶梯算法，属于恒定时间密码算法，可以防御 Cache 侧信道攻击[12]。

算法 2 即无分支蒙哥马利阶梯算法的伪代码实现，其中$Const_Swap(X, Y, \beta)$在β为 1 时对X和Y进行交换。$Madd$和$Mdouble$是常量时间的模加和模倍乘算法。

从中可以看到，模加的参数和模倍乘的参数都依赖于k值每一比特的变化，同时模加和模倍乘算法当中平方算法的参数依赖于模加和模倍乘的输入参数，而二进制域平方算法的参数影响着数据预取器的行为，因此k值的每一比特都影响着数据预取器的行为，而k值在基于椭圆曲线的密码算法当中被作为秘密信息。

当数据预取器被二进制域平方算法触发，进而预取到查找表周边的数据时，可以通过分析查找表周边数据在 Cache 中的状态(利用 Flush+Reload 方法)，推测出 k 的具体数值。这里的 Cache 侧信道信息泄露并非由密码算法本身导致，而是由数据预取引发，无分支的蒙哥马利算法并不能防御此类侧信道攻击。

3.2 数据预取的安全问题

除了上文所述的 Cache 侧信道问题，硬件数据预取还存在其他方面的安全隐患，本小节对硬件数据预取存在的安全隐患进行简要总结。

硬件数据预取未考虑上下文切换：利用硬件数据预取器可以进行隐蔽信道攻击，原因在于基于指令指针的步距预取器的 RPT 表的状态在上下文切换时不会清除，同时预取器的 RPT 表项数有限，当发生上下文切换时，切换后进程的访存流也有可能会触发预取器工作并且在 RPT 表已满时，会覆盖 RPT 表中属于切换前进程的项，当切换前进程再次切换回并继续之前的访存操作时，若 RPT 表中属于原先进程的项未被覆盖，则不需要训练数据预取器直接开始预取操作即可，否则需要再次对数据预取器进行训练，这会导致特定 Cache 块的访存时间存在差异[5]，比如图 5 的数据块 A。针对这一问题只需要在上下文切换时，对数据预取器的状态进行保存和恢复即可。

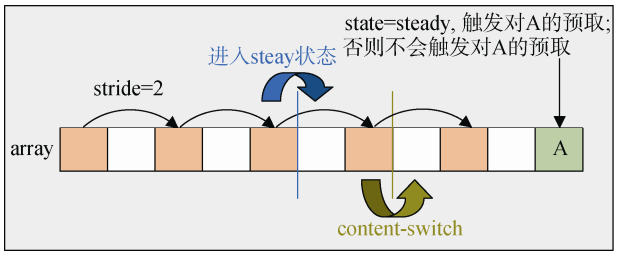

图 5 构造隐蔽信道的原理

Figure 5 The schematic of covert channel

硬件数据预取缺少不可预取边界信息：和软件预取指令不同的是，硬件预取器的行为不是由用户直接控制的。硬件数据预取器在学习到稳定的访存模式(比如步距)后，会使用这个模式对访存地址进行预测，不断地将预测的数据块从主存取到 Cache 当中，硬件数据预取器不具有识别目标区域是否可预取的能力。比如 3.1 小节分析的攻击场景，基于指令指针的步距预取器不知 SQR_tb 之外的数据不可预取。本文在后面章节设计一个具备边界检测功能的数据预取方案来尝试解决这一问题。

4 威胁模型

由于硬件数据预取器造成的隐蔽信道攻击成因简单，较易解决，而硬件数据预取引入侧信道的基本原理还未有充分研究，所以本文主要侧重于后者进行防御。

为了阐明本文中防御策略的适用场景和保护范围，在此对攻击者和被攻击系统提出以下几点假设：

- 攻击者具有在被攻击系统上以普通权限运行任何代码的能力；
- 被攻击系统不存在其他的提权漏洞，可被攻击者用来进行权限提升；
- 被攻击的密码算法本身是恒定时间的，具有抗

Cache 侧信道攻击的能力;

- 攻击者不具有控制微体系结构部件的能力。

5 安全数据预取方案

基于第三章节对数据预取引入侧信道的成因进行分析的基础上, 在本章节, 设计一个具有安全边界检测功能的数据预取方案, 同时以基于指令指针的步距预取器为基础, 对设计的方案进行实现。

硬件数据预取具有以下微体系结构特征: 硬件数据预取请求的数据块有时并非程序真实所需的数据块, 这种数据块会产生预取侧信道。因此, 需要对硬件数据预取器将要预取的数据块地址进行合理检测, 以判断是否会引发侧信道。

5.1 安全预取行为规范

预取触发块: 触发预取的 Cache 块, 这里称之为预取触发块。预取器会检测程序的访存行为, 若发现存在固定的访存特征, 则以这个特征对后续的访存地址进行预测, 预取触发块是构成固定访存特征的 Cache 块。

预取触发块的数量和硬件数据预取器的具体实现相关。在基于指令指针的步距预取策略中, 触发预取器对访存地址进行预测需要多个 Cache 块满足一定的访存特征, 所以预取触发块一般包含多个 Cache 块。预取触发块的内存地址是程序正常的访存请求地址。

预取请求块: 与预取触发块相对应的是预取请求块, 当数据预取器检测到程序的访存行为具有一定特征之后, 会对后续的访存地址进行预测, 这种被预测的访存地址对应的 Cache 块是预取请求块。

无论采用哪种硬件数据预取策略, 预取请求块都只有一个。当预取请求块恰好是程序所需的数据块时, 就表明产生了一次有效的预取; 当预取请求块并非程序本身所需的数据块, 不仅仅会造成 Cache 污染, 还可能会导致信息泄露, 触发侧信道攻击。

预取不可越边界: 从第三章可以看到硬件数据预取在抗 Cache 侧信道攻击的密码算法中引入了预取侧信道, 造成密钥的泄露。预取侧信道形成的根本原因在于进行了越界预取操作, 数据预取从密钥敏感区预取到了非密钥敏感区。这里将密钥敏感区称为安全区域, 非密钥敏感区称为非安全区域, 不同安全级别区域之间的边界即为预取不可越边界。

在上文提出的三个概念的基础上, 接下来, 本文从安全的角度出发, 重新审视数据预取的设计原则, 提出一个安全的数据预取应该遵循的行为规范。

安全预取行为规范: 预取侧信道形成的原因在于预取触发块和预取请求块不属于同一个安全级别的内存区域, 预取触发块位于安全区, 其触发的预取行为却对非安全区的数据块进行了预取操作, 从而在非安全区留下了可以被用来构造 Cache 侧信道的访存时差信息。

从安全设计的角度来看, 若硬件数据预取在安全区域被触发, 则表明程序希望预取的数据一定属于安全区域, 反之亦然。因此, 一个安全的数据预取需要遵守下述规范: 预取触发块和预取请求块必须位于同一个安全级别的内存区域中。

5.2 安全数据预取系统

本小节设计一个安全数据预取系统, 依照上述安全预取行为规范进行实施, 此系统分为三大模块: 异常行为检测器, 数据预取器和预取控制器。

异常行为检测器保存系统定义的预取不可越边界地址信息, 通过检测预取请求块和预取触发块的访存地址信息来判断预取行为是否符合规范; 数据预取器采用的是基于指令指针的步距预取设计方案, 根据程序访存指令在多次访存中的步距特征来预测下一个访存地址, 并对其进行预取操作; 预取控制器接收异常行为检测器产生的控制信号, 根据控制信号的类型决定是否取消预取请求。

5.2.1 边界地址寄存器

对于加密算法, 抑或是有安全需求的应用程序而言, 开发者清楚安全区域的范围, 但是在微体系结构级别, 执行数据预取的部件是数据预取器, 其并不能直接获取到系统软件级别的边界地址信息。因此, 为了在微体系结构级别获得安全区域边界地址信息, 需要加入一组内存映射寄存器。在此暂且假设安全区域只有一个, 边界地址寄存器为: *bottom_boundary* 和 *top_boundary*。

为了防止攻击者对边界寄存器的值随意修改, 边界地址寄存器的权限应当设为特权级别, 普通用户只有读取边界寄存器的能力, 而不具备修改能力。根据前面的危险模型可知, 攻击者只具有普通用户权限, 因此攻击者不具有修改边界寄存器的能力。

安全数据预取系统的关键在于数据预取器可以获得安全区域的精确边界信息, 与 ARM 提出的 trust zone 概念不一样的是, 这里的安全区域很多情况下并不是页对齐的, 有时安全区域小于一个页面大小, 比如上述的 SQR_tb, 这就很不适合以页为粒度对数据预取器的行为进行约束。

以 RISC-V 指令集为例, 由于通过指令操作 CSR 寄存器具有极大的灵活度和便捷性, 所以, 只需在微体系结构级别添加一组 CSR 寄存器作为边界地址

寄存器, 就可以在系统软件层面通过 *csrr* 和 *csrw* 指令对安全边界进行精确设置。这里将边界地址寄存器对应的 CSR 寄存器设置为 S(supervisor)级别。

边界地址寄存器中保存的是物理地址, 为了将安全区域的边界物理地址顺利写入边界地址寄存器, 需要添加一个系统函数 *boundary_set()*, 其参数为安全边界的虚拟地址, *boundary_set()*函数通过查询进程对应的 pagemap 文件就可以获得安全边界的物理地址, 此函数在进程启动时只被调用一次, 因此其性能开销可以忽略不记, 同时程序结束时调用 *boundary_clean()*系统函数, 两个系统函数都带有调用程序的 *pid*。

假设在现实场景下, 攻击者获得了系统的 root 权限, 为了防止其对边界信息进行破坏, 在硬件级别为 CSR 边界寄存器绑定两个 1 比特状态寄存器: set_state, clean_state 和一个 pid 寄存器。更新边界信息时, set_state 置为 1, 同时更新 pid 寄存器的值; 清除边界信息时, clean_state 置为 1, 同时清除 set_state 的值。只有当 set_state 为 0 时, 才可以更新边界信息, 也只有当发出清除边界信息程序的 *pid* 等于 *pid* 寄存器中的值时, 才可以清除边界寄存器中的信息, 这样可以在一定程度上避免攻击者拿到 root 权限后对边界信息进行破坏。

5.2.2 数据预取器

数据预取采用基于指令指针的步距预取方案, 在 2.2 章节的预取方案基础之上, 数据预取器还需要与异常行为检测器进行交互, 提供给异常行为检测器进行越界检测所需的地址信息, 同时还需负责初始化和启动异常行为检测器。

数据预取器共有 4 种状态: initial, transient, steady, no prediction。当处于 steady 状态时, 就具有了稳定预取的功能, 这时, 数据预取器除了向 Cache 发送预取访存请求之外, 还需要更新异常行为检测器当中的基准地址和启动越界检测。

将进入 steady 状态之后的预取触发块地址发送给异常行为检测器, 对异常行为检测器的基准地址进行更新, 当数据预取器再次进入 steady 状态(步距的变化将导致预取器的状态回退到非 steady 状态), 需要对异常行为检测器的基准地址再次进行更新。为了进行预取越界检测, 需要将预取请求块的访存地址发送给异常行为检测器和基准地址进行比较, 以判断是否发生了非安全预取行为。

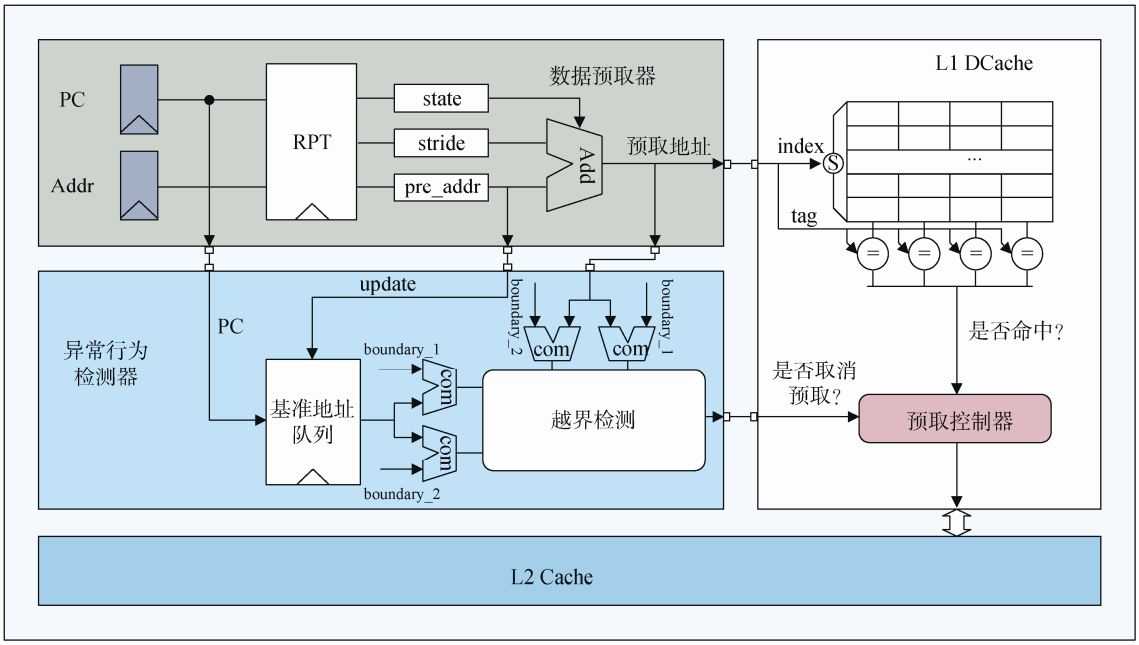

图 6 安全数据预取系统设计图

Figure 6 The design overview of security data prefetch system

5.2.3 异常行为检测器

为了对数据预取的非安全行为进行精确的检测, 需要对预取请求块地址和预取触发块地址进行有效的比较, 这里将异常行为检测器进一步划分为两个子模块: 基准地址队列和越界检测模块。

基准地址队列: 基准地址队列保存数据预取器进入 steady 状态后的预取触发块地址信息, 这里为了避免攻击者对数据预取器进行恶意训练来绕过检测, 基准地址队列只保存触发预取器进入 steady 状态的预取触发块的地址, 在数据预取器每次进入

steady 状态之后立即更新此队列, 基准地址队列的每一项与 RPT 的每一项一一对应。

越界检测模块: 当数据预取器进入 steady 状态之后, 就启动越界检测模块进入工作状态。处于工作状态的越界检测模块对数据预取器发送过来的预取请求块地址进行检测, 判断其是否发生非安全预取行为。

检测过程: 异常检测模块从基准地址队列中读出基准地址, 将其和两个边界地址寄存器作比较, 判断基准地址位于哪个区域, 同时对预取请求块地址做同样比较, 判断预取请求块地址所在区域和基准地址所在区域是否为同一个区域, 若非, 则认定为非安全预取行为, 向预取控制器发送取消预取信号。具体比较过程采用如下步骤:

Step 1: 将基准地址依次和 *bottom_boundary*, *top_boundary* 作比较, 若相等, 则向预取控制器发送预取取消信号; 否则产生两比特的比较结果, *base_compare_results*[1,0], 每一比特分别表示与两个边界寄存器的比较结果, 0 代表小于, 1 代表大于;

Step 2: 同 step 1, 将预取请求块地址和边界地址寄存器作相同的比较, 若相等, 则向预取控制器发送预取取消信号; 否则得到两比特的比较结果, *predict_compare_results*[1,0];

Step 3: 将 step 1 和 step 2 得到的比特序列进行按位同或操作。得到比较结果 *compare_results*[1,0], 最后将 *comapre_results*[0]和 *compare_results*[1]相与得到一比特最终结果 result;

Step 4: 若 result 为 1, 则向预取控制器发送预取请求信号; 若 result 为 0, 则向预取控制器发送预取取消信号。

考虑到当安全区域的尺寸大于系统页的大小时, 存在连续虚拟页面对应不连续物理页面的情况, 为了避免跨页预取违背上述的安全预取行为规范, 需要在进行异常行为检测之前取消跨页预取请求。

5.2.4 预取控制器

预取控制器嵌入 L1 Data Cache 当中, 负责检测预取请求是否在 Cache 命中, 同时根据异常行为检测模块检测的结果来决定是否向下一级存储系统发送预取请求。

预取请求进入 L1 Data Cache 之后, 会产生两种结果, 访存命中和访存缺失。对于访存命中而言, 预取控制器不再向下一级存储系统发送预取请求, 同时也会忽略异常行为检测器的控制信号; 若访存缺失, 则需要根据异常行为检测器的检测结果, 做出进一步判断。

异常行为检测器发送给预取控制器的信号有两种, 预取请求信号和预取取消信号。若预取控制器检测到了访存缺失, 并且收到了预取信号, 那么预取控制器向下一级存储系统发出区预取请求; 否则, 取消向下一级的预取请求。

5.3 小结

根据文献[13]所述, 现今数据预取的设计方案对数据预取可能引发的安全问题缺乏充足的考虑。上文以基于指令指针的步距预取为基础, 设计了一个安全数据预取系统, 此系统不仅可以确保基于指令指针的数据预取器的安全性, 也可以通过适当的改动应用于其他的数据预取器。

如上文所述, 硬件数据预取是一种推测行为, 在微体系结构级别, 但凡具有推测功能的部件都有一定的错误率, 比如分支预测器, 这种错误容易引入安全问题[14]。硬件数据预取在预测错误的情况下, 会将并非程序所需的数据预取到 Cache 当中, 改变 Cache 状态, 造成信息泄露, 为了避免这种情况的发生, 需要根据上述方案对硬件数据预取的行为加以限制。

6 实验与安全性评估

本章节首先在开源处理器 BOOM 上, 实现了基于指令指针的步距预取器, 并依据本文所述攻击原理成功泄露出加密算法密钥。其次在存在预取侧信道问题的 BOOM 处理器上, 依照 5.2 节所述的方案实现了安全数据预取系统, 并对其安全性和性能开销进行了评估。

6.1 实验环境配置

6.1.1 BOOM 处理器配置信息

BOOM 开源处理器采用四取指四译码的超标量结构, 具有十级流水线以及乱序推测执行机制。其存储层次分为两级, 一级 Cache 为私有的 L1 Data Cache 和 L1 Instruction Cache, 二级 Cache 为共享的 L2 Cache。其详细配置如表 1 所示

表 1 BOOM Cache 的配置信息

Table 1 The Cache parameters of the BOOM

Level	Type	Size	Associativity	Share?	Line Size
L1	Data	32KiB	8-way	Private	64Byte
L1	Instruction	32KiB	8-way	Private	64Byte
L2	Hybrid	1MiB	16-way	Shared	64Byte

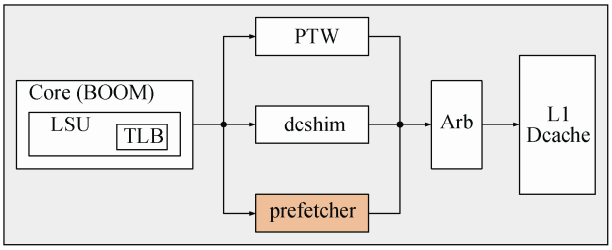

图 7　预取器在 BOOM 中的位置

Figure 7　The location of prefetcher in BOOM

在 BOOM 上实现基于指令指针的步距预取器时, 它与 BOOM 开源处理器中的 LSU 和 PTW 共享 L1 Data Cache 的请求端口, 通过监测 LSU 的访存请求历史来训练数据预取器的步距信息, 其 RPT 表的项数为 32 项, 其中保存的访存地址信息为物理地址, 数据预取器在 BOOM 当中的位置如图 7 所示。

6.1.2　实验评估

安全性评估: 对于安全数据预取系统, 需要确认其是否依然会产生预取侧信道。加密算法存在这个特性, 当输入不同密钥时, 若系统中存在 Cache 侧信道, 则加密运算产生的 Cache 活动状态不一致。基于此, 本文选择 OpenSSL 中的 EC_POINT_mul 密码算法函数用于侧信道检测, 对其输入不同的密钥, 采用 Youngjoo Shin[4]等人提出的侧信道检测方法, 测试安全数据预取系统中是否依然存在侧信道。由于 OpenSSL 目前不支持 RISC-V 交叉编译, 为了方便测试, 本文将 EC_POINT_mul 调用的相关函数提取出来。

性能开销评估: 由于安全数据预取系统具有越界检测功能, 可能会降低加密运算的速度, 因此需要评估安全数据预取系统对加密运算速度的影响, 这个评估可以通过对比单次加密运算在实现越界检测功能和未实现越界检测功能系统上的完成时间来实现, 这里采用的加密函数为 OpenSSL 中的 *EC_POINT_mul* 函数; 同时, 为了确认越界检测不会干扰正常的预取功能, 需要对安全数据预取系统中的预取功能进行评估, 这里选择了几个典型的可以频繁触发数据预取的程序, 包括 stream[15]等, 通过对比安全数据预取系统中的预取器和未实现越界检测的数据预取器为这些程序带来的加速比可以实现对预取功能的评估。

硬件开销评估: 考虑到安全数据预取系统中预取控制器内嵌于 L1 Data Cache, 可能会对访存路径的时序造成影响, 同时异常检测器包含基准地址队列和越界检测逻辑, 可能会引入较大的面积开销, 因此需要对预取控制器的访存路径时序和异常检测器的面积开销进行评估。基于 SMIC 28nm 技术, 使用 TT 工艺角, 通过 Design Compiler 对安全数据预取系统的 Register-Transfer Level (RTL)代码实现进行综合, 可获得访存路径的时序变化和异常行为检测器的面积开销数据。

6.2　实验结果分析

在对评估结果进行分析之前, 首先需要明确对实验数据的分析方法。为了更好地分析数据, 这里定义两个预取系统, 一个是非安全数据预取系统, 仅实现了基于指令指针的步距数据预取器; 另一个是安全数据预取系统, 不仅实现了数据预取的功能, 还实现了预取不可越边界检测功能。通过对比两个预取系统的测试结果数据, 可以获得安全数据预取系统的安全性和性能开销情况。

6.2.1　安全性评估结果

由 Youngjoo Shin[4]等人提出的侧信道检测方法可知, 与加密算法可执行程序相关的每一个 Cache Line 都有一个密钥可区分度, 若其值为 1, 则表示此 Cache Line 的活动状态在不同密钥下具有很强的区分度; 若为 0, 则表示不具有区分度, 此值可用于判断 Cache Line 是否存在侧信道。

在非安全数据预取系统上, 根据 Youngjoo Shin[4]等人提出的侧信道检测方法对 Cache 侧信道进行检测, 发现有两个 Cache Line 的密钥可区分度值接近于 1, 这两个 Cache Line 与 SQR_tb 在内存中的位置关系如图 8 所示。

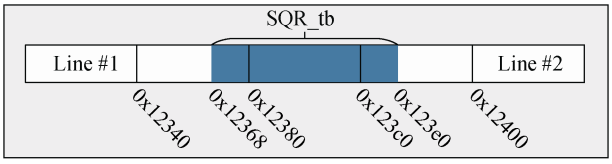

图 8　SQR_tb 的内存布局

Figure 8　Memory layout of SQR_tb

使用同样的方式, 对安全数据预取系统进行侧信道检测, 得到了图 8 中 *Line #1* 和 *Line #2* 的密钥区分度, 都为 0, *Line #1* 和 *Line #2* 在非安全数据预取和安全数据预取上的密钥区分度结果如表 2 所示。

从表 2 中可以看出, *Line #1* 和 *Line #2* 在非安全数据预取系统中存在侧信道, 而在安全数据预取系统中不存在侧信道。

从上述结果可以知道, 由非安全数据预取引发的侧信道恰好位于密钥敏感区域(SQR_tb)的两侧, 表明非安全数据预取跨域预取不可越边界进行了预取操作, 而安全数据预取系统通过越界检测抑制了

越界预取, 从而解决了数据预取引发的侧信道问题。

表 2 密钥可区分度对比

Table 2 Key distinguishability comparison

	Line #1	Line #2
非安全数据预取	0.934	0.952
安全数据预取	0	0

为了更进一步验证安全数据预取系统的安全性, 随机挑选一个密钥作为目标密钥(target_key), 利用侧信道 *Line #1* 和 *Line #2*, 在非安全数据预取系统和安全数据预取系统上对目标密钥进行恢复。图 9 列出了目标密钥和猜测密钥(test_key)在 *Line #1* 和 *Line #2* 上的命中率差值, 差值越小表明猜测值越接近目标值, 从图 9 中可以看出, 在非安全数据预取系统中, 没有开启越界检测, 猜测密钥和目标密钥的命中率差值具有明显的区分度, 当命中率差值最小时, 猜测密钥和目标密钥一致(0xa5); 在安全数据预取系统中, 开启了越界检测, 猜测密钥和目标密钥的命中率差值不再具有区分度, 因为预取控制器抑制了越界预取, 因此 *Line #1* 和 *Line #2* 的命中率全部为 0。

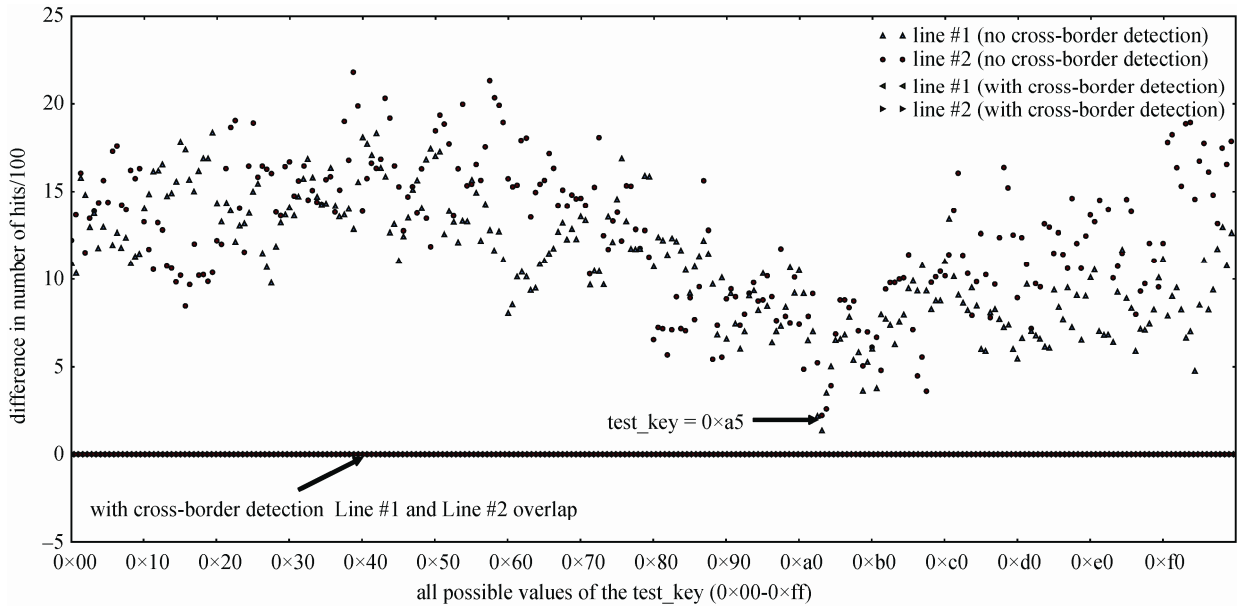

图 9 test_key 与 target_key 在 Line #1 和 Line #2 上的状态序列的值均差

Figure 9 Test_key and target_key's mean difference in the state sequence on Line # 1 and Line # 2

根据上述分析, 可以确定本文设计的安全数据预取系统可以有效防止硬件数据预取引入的预取侧信道安全问题, 进而避免攻击者对加密算法密钥进行窃取。

在以上实验中, 为了展示预取侧信道的安全隐患, 本文挑选了抗 Cache 侧信道分析的加密算法, 这类加密算法已被证明具有抵抗 Cache 侧信道分析的能力[14]; 同时, 由于数据预取是间接触发了 Cache 侧信道, 并且引发侧信道的数据区域并不涉及密钥运算, 目前已知的针对 Cache 侧信道的检测方案不再适用于数据预取侧信道检测[4], 因此采取了 Youngjoo Shin[4]等人提出的针对预取侧信道的新型检测方案。

6.2.2 性能开销评估结果

为了测试安全数据预取系统对不同长度密钥加密运算速度的影响, 本文选择了 64bit, 128bit 和 256bit 3 种长度的密钥, 分别统计其一次加密运算所需的时间, 并以非安全数据预取系统下的运算时间为基准进行归一化处理, 得到的结果如图 10 所示。

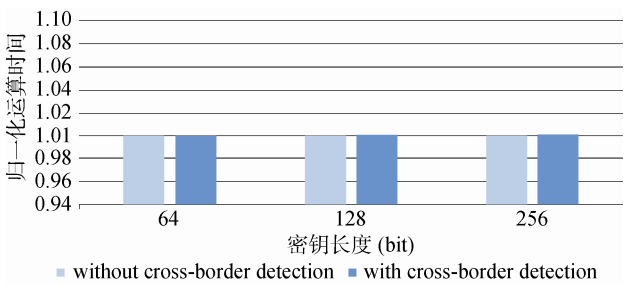

图 10 单次加密运算所需时间对比

Figure 10 Comparison of the time required for a single encryption operation

为了验证安全数据预取系统对正常的数据预取请求不存在影响, 本文选择了内存带宽测试程序 stream, 微控制器 CPU 性能测试基准程序 CoreMark[16]和 SPEC CPU 2006 的一些易触发数据预

取的测试程序(462.libquantum, 456.hmmer, 445.gobmk, 464.h264ref), 测试这些程序在安全数据预取系统和非安全数据预取系统中各自获得的加速比, 并以非安全数据预取系统得到的加速比为基准对结果进行归一化处理, 如图 11 所示。

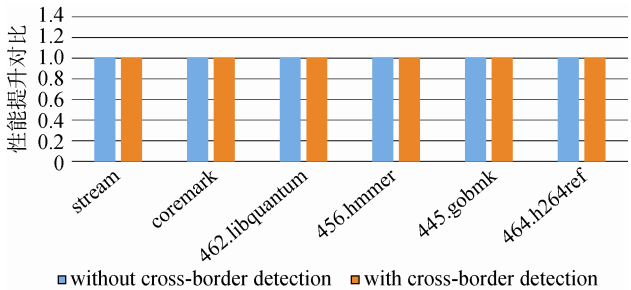

图 11 安全数据预取系统和非安全数据预取系统对程序的性能提升对比

Figure 11 Comparison of improvement in program performance between secure data prefetch system and non-secure data prefetch system

从上面的两个图中可以看出, 安全数据预取系统的越界检测对加密运算速度带来的影响微乎其微, 可以忽略不记, 同时, 安全数据预取系统对程序正常的预取请求不存在影响。因为在安全数据预取系统中, 越界检测和预取请求的访存是并行的, 并不会引起流水线的阻塞; 同时越界检测仅仅会抑制跨安全级别区域的数据预取请求, 对正常的数据预取请求没有影响, 所以上述的分析结果是真实的。

文献[4]中提出了另外一种预取侧信道解决方法, 也是目前最新版 OpenSSL[17]正在使用的方法: 对于二进制域平方算法采用实时计算的方式, 舍弃 SQR_tb 查表法。但是这种方式会引入较大的性能损失, 大致为 4%~8%[4], 远远高于本文提出的方案。

为了评估能耗开销, 我们基于 SMIC 28nm 技术, 使用 TT 工艺角, 利用 Design Compiler 对数据预取系统添加边界检测逻辑前后的能耗进行了对比。其中未实现边界检测功能的数据预取系统的动态功耗为: 44.5016 mW, 静态功耗为: 67.7680 μW; 实现了边界检测功能的数据预取系统的动态功耗为: 46.72mW, 相比增加了 4.098%, 静态功耗为: 70.83 μW, 相比增加了 4.5%。通过分析可以发现, 由于安全功能引入的功耗开销可以忽略不记。

6.2.3 硬件开销评估结果

通过 Design Compiler 对非安全数据预取系统和安全数据预取系统的 RTL 实现分别进行综合, 得到了预取控制器对访存路径的时序影响和异常行为检测器的面积开销数据。

预取控制器内嵌于 L1 Data Cache 中, 导致访存路径的时序从 0.631ns 提升到了 0.637 ns, 增加了 0.9%。异常行为检测器的面积开销仅为 978 μm^2, 与 8 路组相联 32 KB 的 L1 Data Cache 相比(0.219 mm^2), 异常行为检测器仅占据其 0.04%的面积开销。

通过 DC 综合结果的对比, 我们发现异常行为检测器引入的面积开销与 L1 DataCache 的面积开销相比可以忽略不记, 不会产生较大的制造成本, 因此本方案具有一定的实用价值。

7 总结与展望

硬件数据预取具有推测预取数据的性质, 导致其易引发 Cache 侧信道相关安全问题, 然而目前还未有针对硬件数据预取的安全防御研究, 本文通过在分析研究硬件数据预取安全缺陷的基础上, 引入了预取不可越边界的概念, 同时提出了一种安全预取行为规范, 基于此规范, 设计了安全数据预取系统, 并以基于指令指针的步距预取器为基础, 在开源处理器 BOOM 上对安全数据预取系统进行了实现, 最后对其安全性和性能开销进行了评估。

本文提出的安全数据预取系统, 可以应用于其他的硬件数据预取器, 而非仅仅是基于指令指针的步距预取, 同时借助 CSR 寄存器配置的灵活性(非 RISC-V 体系结构中可以采用内存映射寄存器), 可以实现对多个安全区域进行保护, 具有可扩展性。

对于硬件数据预取的防御研究, 本文仅做了初步尝试, 依然有很多问题值得思考, 期待未来有更好的研究点和解决方案出现。

参考文献

[1] Vanderwiel S P, Lilja D J. Data Prefetch Mechanisms[J]. *ACM Computing Surveys*, 2000, 32(2): 174-199.

[2] Ge Q, Yarom Y, Cock D, et al. A Survey of Microarchitectural Timing Attacks and Countermeasures on Contemporary Hardware[J]. *Journal of Cryptographic Engineering*, 2018, 8(1): 1-27.

[3] Gruss D, Maurice C, Fogh A, et al. Prefetch Side-Channel Attacks: Bypassing SMAP and Kernel ASLR[C]. *The 2016 ACM SIGSAC Conference on Computer and Communications Security*, 2016: 368-379.

[4] Shin Y, Kim H C, Kwon D, et al. Unveiling Hardware-Based Data Prefetcher, a Hidden Source of Information Leakage[C]. *The 2018 ACM SIGSAC Conference on Computer and Communications Security*, 2018: 131-145.

[5] Cronin P, Yang C M. A Fetching Tale: Covert Communication with the Hardware Prefetcher[C]. *2019 IEEE International Symposium on Hardware Oriented Security and Trust*, 2019: 101-110.

[6] Osvik D A, Shamir A, Tromer E. Cache Attacks and Countermeasures: The Case of AES[M]. Topics in Cryptology – CT-RSA 2006.

Berlin, Heidelberg: Springer Berlin Heidelberg, 2006: 1-20.

[7] Yuval Yarom, Katrina Falkner. FLUSH+ RELOAD: a high resolution, low noise, L3 cache side-channel attack[C]. *The 23rd USENIX Security Symposium*, 2014: 719-732.

[8] Intel 64 and IA-32 Architectures Optimization Reference Manual[EB/OL]. 2011

[9] Bhattacharya S, Rebeiro C, Mukhopadhyay D. Hardware Prefetchers Leak: A Revisit of SVF for Cache-Timing Attacks[C]. *2012 45th Annual IEEE/ACM International Symposium on Microarchitecture Workshops*, 2012: 17-23.

[10] Coron J S. Resistance Against Differential Power Analysis for Elliptic Curve Cryptosystems[C]. *International Workshop on Cryptographic Hardware and Embedded Systems*, 1999: 292-302.

[11] Aranha D F, López J, Hankerson D. Efficient Software Implementation of Binary Field Arithmetic Using Vector Instruction Sets[C]. *International Conference on Cryptology and Information Security in Latin America*, 2010: 144-161.

[12] Okeya K, Kurumatani H, Sakurai K. Elliptic Curves with the Montgomery-Form and Their Cryptographic Applications[C]. *International Workshop on Public Key Cryptography*, 2000: 238-257.

[13] Mittal S. A Survey of Recent Prefetching Techniques for Processor Caches[J]. *ACM Computing Surveys*, 2016, 49(2): 1-35.

[14] Kocher P, Horn J, Fogh A, et al. Spectre Attacks: Exploiting Speculative Execution[C]. *2019 IEEE Symposium on Security and Privacy*, 2019: 1-19.

[15] https://www.cs.virginia.edu/stream/

[16] https://www.eembc.org/coremark/

[17] https://github.com/openssl/openssl/blob/master/crypto/bn/bn_gf2m. c

吝常青 于 2017 年在西安电子科技大学信息安全专业获得学士学位。现在中国科学院信息工程研究所网络空间安全专业攻读硕士学位。研究领域为计算机体系结构安全。研究兴趣包括: 微处理器设计, 微体系结构安全。Email: linchangqing@iie.ac.cn

田鑫 于 2019 年在西安电子科技大学软件工程专业获得硕士学位。现任中国科学院信息工程研究所信息安全国家重点实验室工程师。研究领域为大规模集成电路设计。研究兴趣包括: 微处理器设计, FPGA 专用电路设计。Email: tianxin@iie.ac.cn

侯锐 于 2007 年在中国科学院计算技术研究所计算机系统结构专业获得博士学位。现任中国科学院信息工程研究所信息安全国家重点实验室研究员。研究领域为计算机体系结构, 研究兴趣包括处理器芯片设计, 计算机体系结构安全和人工智能安全。Email: hourui@iie.ac.cn

孟丹 于 1995 年在哈尔滨工业大学计算机体系结构专业获得博士学位。现任中国科学院信息工程研究所所长。研究领域包括计算机系统安全,大数据与云计算等。研究兴趣包括计算机系统安全,云计算安全等。Email: mengdan@iie.ac.cn