湖南大學

题　　目： 基于边信道的计算机硬件攻击小综述

姓　　名： 唐　　斌

学　　号： S191000880

专　　业： 计算机科学与技术专业

系　　别： 网络空间安全系 1901

2019 年 12 月 24 号

1.引言

近年来，随着各种各样针对硬件的攻击方式被专家学者提出人们不得不重新考虑硬件、体系结构等方面的安全性。在这些攻击中，大都用到了一种被称为边信道攻击的技术，通过这种技术，我们可以读取保存在某种微架构等状态中的秘密值。不同于一般的攻击方式，边信道攻击从那些在逻辑上看似与信息泄露无关的信号上下手，从而获取有用信息。

边信道攻击最早在 1996 提出，针对密码破解。传统的密码破解方法基于算法分析，效率低、成本高，利用边信道攻击，能够从功耗、时间等地方获取有用信息，从而实现密码破解。边信道攻击一直都存在，最常见的一种边信道攻击是 SQL 注入攻击中基于时间延迟的攻击，此外，还有存在针对系统层面的边信道攻击，例如，通过 CPU 缓存，可以监视到用户在浏览器中进行的操作。

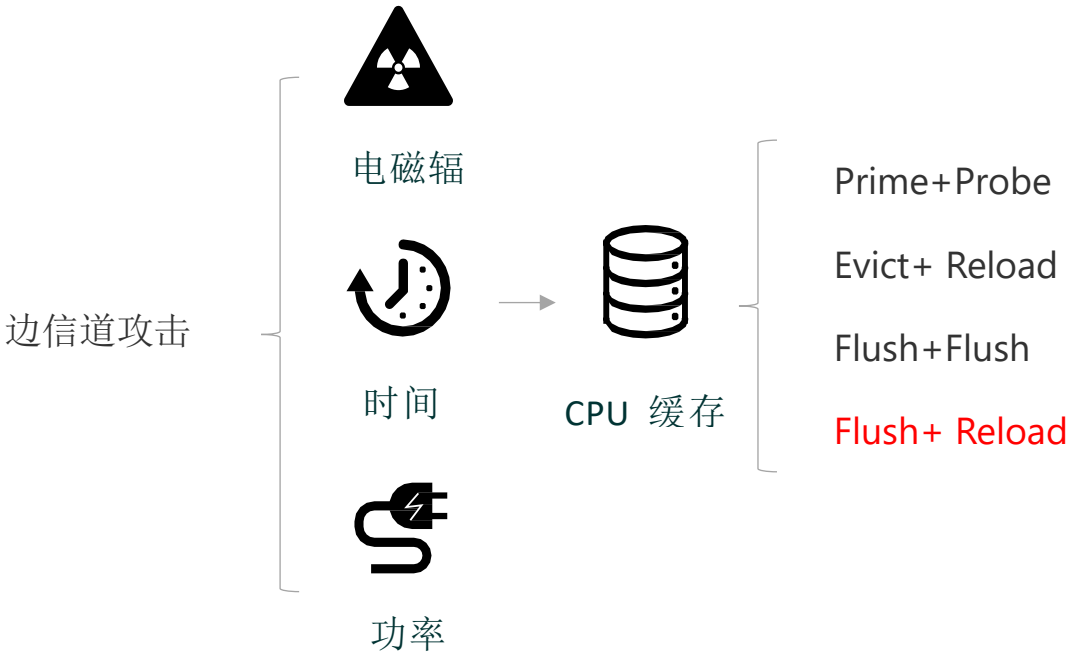
边信道攻击作为一种工具，在微架构、芯片级等漏洞中能够发挥巨大潜力，从而对已经存在的体系造成严重影响。

2.边信道攻击

边信道攻击：side channel attack，简称 SCA，也称为侧信道攻击。不同于一般的攻击形式，这是一种对加密电子设备在运行过程中的时间消耗、功率消耗或电磁辐射之类的侧信道信息进行利用，从而获取密码信息的方法。这类攻击的有效性远高于传统针对密码算法进行数

学分析或暴力破解的方法，给加密设备带来了严重的威胁。

其大致分类如下：



2.1 一般分类

2.1.1 缓存攻击

通过获取对缓存的访问权而获取缓存内的一些敏感信息，例如攻击者获取云端主机物理主机的访问权而获取存储器的访问权；

2.1.2 计时攻击

通过设备运算的用时来推断出所使用的运算操作，或者通过对比运算的时间推定数据位于哪个存储设备，或者利用通信的时间差进行数据窃取；

2.1.3 基于功耗监控的旁路攻击

同一设备不同的硬件电路单元的运作功耗也是不一样的，因此一个程序运行时的功耗会随着程序使用哪一种硬件电路单元而变动，据

此推断出数据输出位于哪一个硬件单元，进而窃取数据；

2.1.4 电磁攻击

设备运算时会泄漏电磁辐射，经过得当分析的话可解析出这些泄漏的电磁辐射中包含的信息（比如文本、声音、图像等），这种攻击方式除了用于密码学攻击以外也被用于非密码学攻击等窃听行为，如 TEMPEST 攻击（例如范·埃克窃听、辐射监测）；

2.1.5 声学密码分析

通过捕捉设备在运算时泄漏的声学信号捉取信息（与功率分析类似）；

2.1.6 差别错误分析

隐密数据在程序运行发生错误并输出错误信息时被发现；

2.1.7 数据残留

可使理应被删除的敏感数据被读取出来（例如冷启动攻击）；

2.1.8 软件初始化错误攻击

现时较为少见，Row hammer 是该类攻击方式的一个实例，在这种攻击实现中，被禁止访问的存储器位置旁边的存储器空间如果被频繁访问将会有状态保留丢失的风险；

2.1.9 光学方式

即隐密数据被一些视觉光学仪器（如高清晰度相机、高清晰度摄影机等设备）捕捉。

2.2 已实现的边信道攻击

2.2.1 通过 CPU 缓存来监视用户在浏览器中的快捷键及鼠标操作

这种攻击对最新型号的英特尔 CPU 有效，如 Core i7，需运行在支持 HTML5 的浏览器上。带有恶意 JS 的网页在受害者电脑上执行后，会收集与之并行的其它进程的信息，有了这个信息，攻击者可以绘制内存对按下按键和鼠标移动的反应情况，进而重塑用户使用情景。（哥伦比亚大学）

2.2.2“听译”电子邮件密钥

通过智能手机从运行 PGP 程序的计算机中“听译”密钥。这项最新的密钥提取攻击技术，能够准确地捕捉计算机 CPU 解码加密信息时的高频声音，并提取密钥。（同时提出 GnuPG 来应对这种攻击）

2.2.3 非智能手机+恶意软件+目标 PC

从采购供应链下手，将特制难以检测的恶意软件植入电脑，该软件会强制计算机的内存总线成为天线，通过蜂窝频率将数据无线传输到手机上。攻击者将接受和处理信号的软件嵌入在手机的固件基带中，这种软件可以通过社会工程攻击、恶意 App 或者直接物理接触目标电话来安装。

2.2.4 用手触碰电脑即可破解密码

电脑 CPU 运算时造成“地”电势的波动，用手触碰笔记本电脑的外壳，接着再测量释放到皮肤上的电势，然后用复杂的软件进行分析，最终得到计算机正在处理的数据。例如：当加密软件使用密钥解密时，监测这种波动就可得到密钥。[1]（GnuPG 可以应对这种攻击）

2.3 针对 CPU 缓存的几种边信道攻击

2.3.1 Prime+Probe

步骤 1. Prime: 攻击者用预先准备的数据填充特定多个 cache 组；

步骤 2. Trigger: 等待目标虚拟机响应服务请求，将 cache 数据更新；

步骤 3. Probe: 重新读取 Prime 阶段填充的数据，测量并记录各个 cache 组读取时间。

2.3.2 Flush+Reload

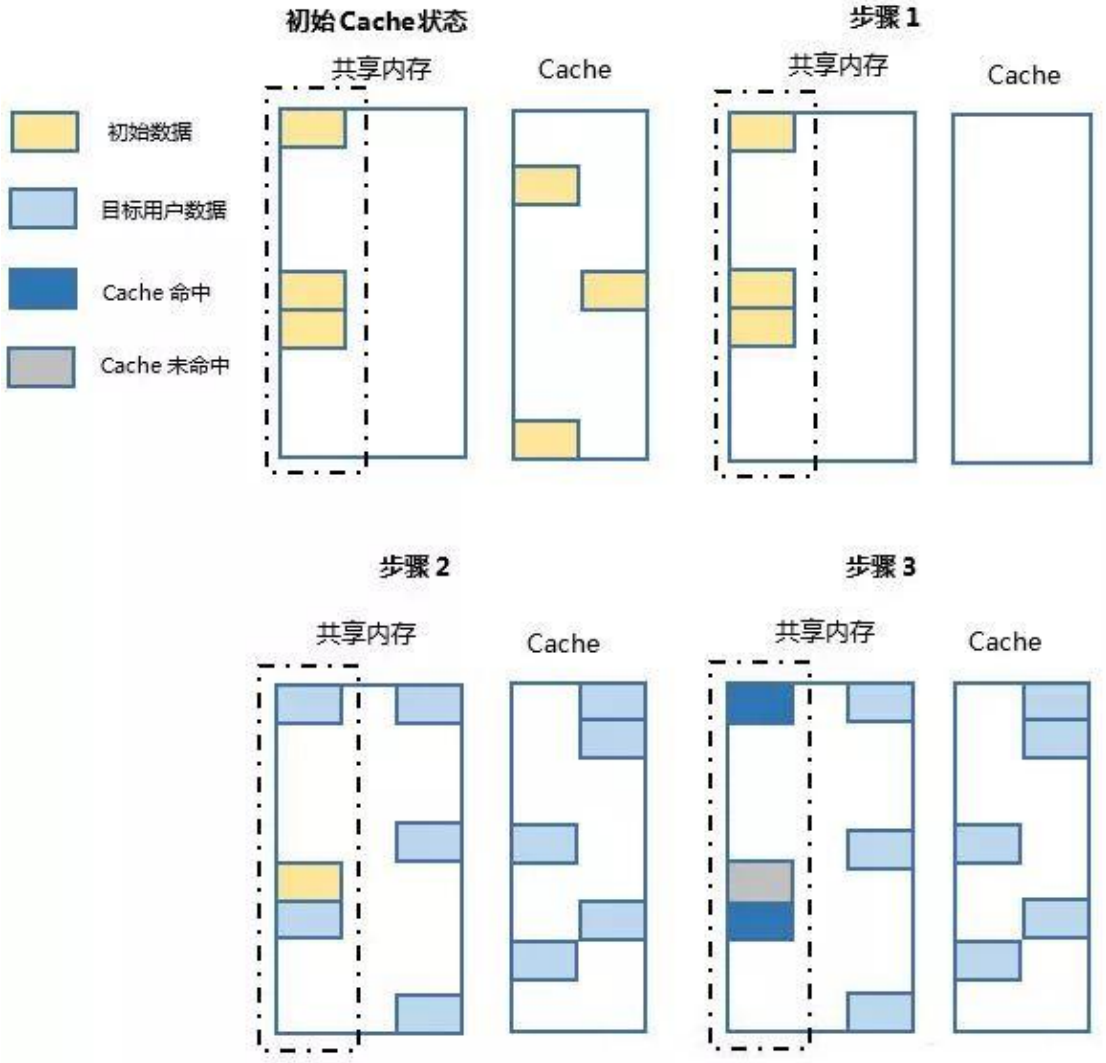
Flush-Reload 方法是 prime-probe 方法的变种，基于共享内存实现，是一种跨内核、跨虚拟机的 Cache 探测方法。在 Flush 阶段，攻击者将监控的内存块从 cache 中驱逐出去，然后在 Trigger 阶段等待目标用户访问共享内存。在 Reload 阶段，攻击者重新加载监控的共享内存块。如果在等待的期间，目标虚拟机访问过的内存块不需要重新加载，时间将会较短，因为这些数据已经被缓存在 cache 中。根据加载时间长短，可判定目标虚拟机访问过的数据。

Flush-Reload 具体步骤如下：

步骤 1. Flush：将共享内存中特定位置映射的 cache 数据驱逐；

步骤 2. Trigger：等待目标虚拟机响应服务请求，更新 Cache；

步骤 3. Reload：重新加载 Flush 阶段驱逐的内存块，测量并记录 cache 组的重载时间。



2.3.3 Evict+Reload

Evict+Reload 和 Flasu+Reload 的攻击流程基本一致，只是，通过驱逐（eviction）的方式，即从从 Cache 中移出一个行从而为新数据腾出空间的过程，用于替代 Flush+Reload 中的 Flush 指令缺失的情况。

2.3.4 Flush+Flush

与 Flush-Reload 不同的是，Flush-Flush 攻击是基于 clflush 指令执行时间的长短来实施攻击的。如果数据没在 Cache 中则 clflush 指令执行时间会比较短，反之若有数据在 cache 中则执行时间会比较长。与其它 Cache 攻击不同，Flush Flush 侧信道攻击技术在整个攻击过程中是不需要对内存进行存取的，因此该攻击技术更加隐蔽。然而根据作者的经验，由于有无数据情况下 Cache 被 flush 的时间差别其实并非特别明显，因此在攻击过程中数据判断的精度并不高。

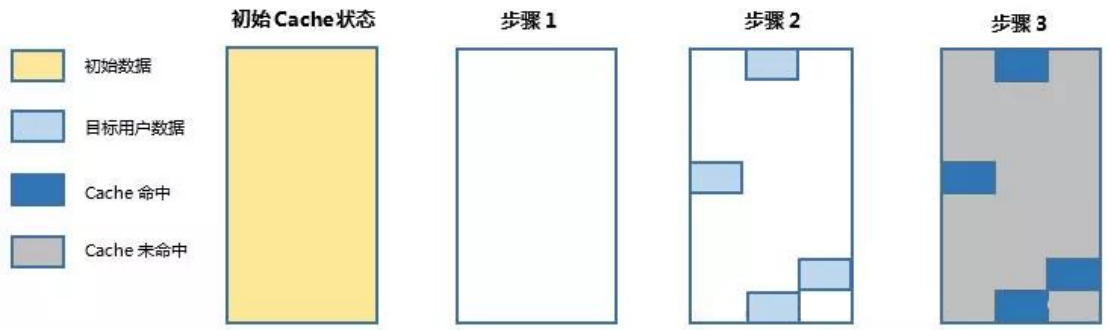
Flush-Flush 具体步骤如下：

步骤 1：通过 flush 清空 Cache 原始数据；

步骤 2：等待目标进程运行，更新 Cache，并刷新共享缓存行，测量刷新时间；

步骤 3：根据测量时间判断原始数据是否被缓存。

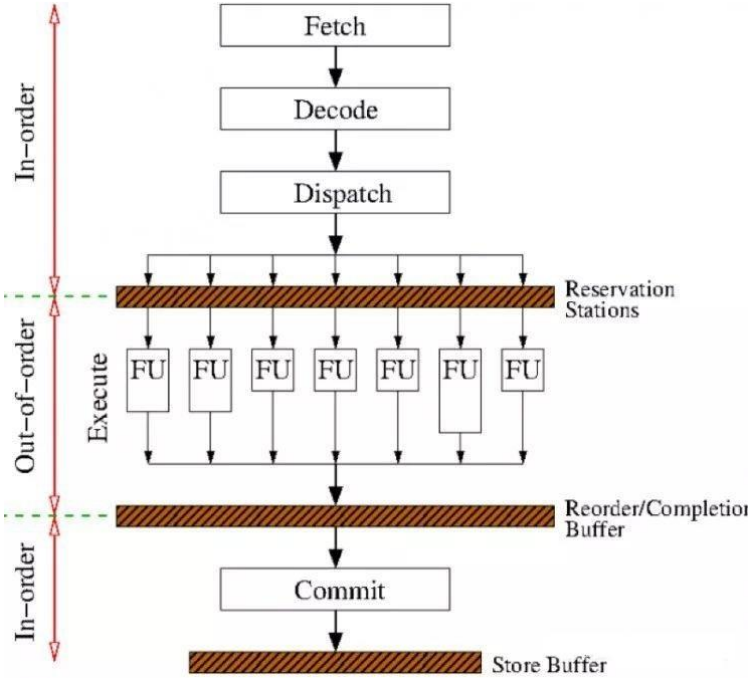
该方法攻击原理如图所示：



3. 测信道如何应用到硬件攻击

3.1 熔断攻击[2]

乱序执行可以简单的分为三个阶段，如下图所示：



每个阶段执行的操作如下：

1）获取指令，解码后存放到执行缓冲区（保留站）；

2）乱序执行指令，结果保存在一个结果序列中；

3）退休期，重新排列结果序列及安全检查（如地址访问的权限检查），提交结果到寄存器。

Meltdown 利用代码：

```
1    ; rcx = kernel address, rbx = probe array
2    xor rax, rax
3    retry:
4    mov al, byte [rcx]        //step 1
```

```
5    shl rax, 0xc                    //rax*4096
6     jz retry
7    mov rbx, qword [rbx + rax]
```

结合 Meltdown 利用的代码片段来看，Meltdown 漏洞的利用过程有 4 步：

1) 指令获取解码；

2) 乱序执行 3 条指令，line 5 和 line 7 要等 line 4 中的读取内存地址的内容完成后才开始执行，line 7 会将要访问的 rbx 数组元素所在的页加载到 CPU Cache 中；

3) 对 2)的结果进行重新排列，对 line 4、line5、line7 这 3 条指令进行安全检测，发现访问违例，会丢弃当前执行的所有结果，恢复 CPU 状态到乱序执行之前的状态，但是并不会恢复 CPU Cache 的状态；

4) 通过缓存测信道攻击，可以知道哪一个数组元素被访问过，也即对应的内存页存放在 CPU Cache 中，从而推测出内核地址的内容.

3.2 幽灵攻击

与 Meltdown 类似，Spectre 的原理是，当 CPU 发现分支预测错误时会丢弃分支执行的结果，恢复 CPU 的状态，但是不会恢复 CPU Cache 的状态，利用这一点可以突破进程间的访问限制，从而获取其他进程的数据。

幽灵攻击代码示例[3]：

```
if (x < array1_size) {
```

```
    y = array2[array1[x] * 4096];
    // do something detectable when
    // speculatively executed
}
```

具体攻击过程可以分为三个阶段：

1）训练 CPU 的分支预测单元使其在运行利用代码时会进行特定的预测执行；

2）预测执行使得 CPU 将要访问的地址的内容读取到 CPU Cache 中；

3) 通过缓存测信道攻击，可以知道哪一个数组元素被访问过，也即对应的内存页存放在 CPU Cache 中，从而推测出地址的内容。

4. 研究进展

熔断攻击和幽灵攻击提出之后，先后出现了一系列利用计算机硬件设计时遗留下来的缺陷进行攻击的方法。其分类如下：

- 芯 片：针对 PUF 的攻击
- 微 架 构：
  - Meltdown（乱序执行）
  - Spectre （分支预测）
  - MDL（微架构采样）
    - ZombieLoad
    - Fallout
    - RIDL
  - RAMBleed（内存）
- 系 统：通过操作系统漏洞泄露信息
- 应 用：基于时间延迟的 SQL 注入

4.1 Foreshadow[4]

Foreshadow 是针对 Intel 处理器 SGX 技术的攻击，它使攻击者能够窃取存储在个人计算机或第三方云中的敏感信息。Foreshadow 有两种版本，一种是旨在从 SGX 安全区提取数据的原始攻击，另一种是影响虚拟机（VM），虚拟机管理程序（VMM），操作系统（OS）内核内存和系统管理模式（SMM）内存的下一代版本。

从较高层次来说，SGX 是现代 Intel CPU 的一项新功能，即使整个系统都在攻击者的控制之下，它也可以使计算机保护用户的数据。虽然以前认为 SGX 可以抵抗推测性执行攻击（例如 Meltdown 和 Spectre），但 Foreshadow 演示了如何利用推测性执行来读取 SGX 保护的内存的内容以及提取机器的专用证明密钥。更糟的是，由于 SGX 的隐私功能，一份证明报告无法与其签名人的身份联系在一起。 因此，仅需使用一台受损的 SGX 计算机即可破坏对整个 SGX 生态系统的信任。

4.2 MDS（微架构数据采集）

4.2.1 ZombieLoad[5]

在 Meltdown，Spectre 和 Foreshadow 之后，作者发现了现代处理器中的更多关键漏洞。 ZombieLoad 攻击允许在计算机访问敏感数据和密钥时窃取它们。尽管程序通常只能看到自己的数据，但是恶意程序可以利用内部 CPU 缓冲区来掌握当前由其他正在运行的程序处理的机密。这些秘密可以是用户级别的秘密，例如浏览器历史记录，网

站内容，用户密钥和密码，也可以是系统级别的秘密，例如磁盘加密密钥。

ZombieLoad 的新变体可以对耐 MDS 的 CPU 进行攻击。在 2019 年 11 月 14 日，作者提出了 ZombieLoad 的新变体，可以对 CPU 进行攻击，其中包括针对芯片中 MDS 的硬件缓解措施。使用 Variant 2（TAA），数据仍然可以在诸如 Cascade Lake 之类的微体系结构上泄漏，而其他 MDS 攻击（如 RIDL 或 Fallout）是不可能的。此外，作者表明，结合微码更新作为基于 MDS 攻击的对策，基于软件的缓解措施是不够的。

4.2.2 RIDL[6]

RIDL 显示，攻击者可以利用 MDS 漏洞在实际环境中发起攻击并泄漏敏感数据。通过分析对 CPU 流水线的影响，作者开发了各种实用的漏洞利用方法，它们从不同的内部 CPU 缓冲区（例如，行填充缓冲区和加载端口）泄漏运行中的数据，这些缓冲区在 CPU 从内存加载或存储数据时使用。实验表明，RIDL 可以在搭载最新英特尔 CPU 的计算机上运行非特权代码，无论是使用共享的云计算资源，还是在恶意网站或广告上使用 JavaScript，都可以跨越任何安全边界从同一台计算机上运行的其他程序窃取数据： 其他应用程序，操作系统内核，其他 VM（例如在云中），甚至是安全（SGX）区域。

4.2.3 Fallout[7]

Fallout 表明攻击者可以从存储缓冲区泄漏数据，每次 CPU 流水线

需要存储任何数据时都会使用该存储缓冲区。更糟糕的是，没有特权的攻击者随后可以从 CPU 的存储缓冲区中选择泄漏的数据。实验证明，Fallout 可用于打破内核地址空间布局随机化（KASLR），以及泄漏由操作系统内核写入内存的敏感数据。具有讽刺意味的是，与较早的硬件相比，英特尔在最新的 Coffee Lake Refresh i9 CPU 中采用的最新硬件对策可防止 Meltdown 崩溃，但是，这使他们更容易受到 Fallout 的影响。

4.3 RAMBleed[8]

熔断攻击和幽灵攻击的主要提出者，他们的最新工作将发表在 2020 年的 S&P（security&privacy）会议上，题为“RAMBleed: Reading Bits in Memory Without Accessing Them”。

Rowhammer 是 DRAM 单元中的一个存在的问题，它可使无特权的攻击者翻转存储模块上相邻行中的位值。先前的工作已经将此方法用于跨安全边界的各类型错误攻击，攻击者在其中翻转难以访问的位，通常会导致特权升级。虽然攻击者已经可以通过常规的写操作来修改其私有内存，但是，人们普遍认为，攻击者自己的私有内存中的位翻转不会带来安全隐患。通过采用 Rowhammer 作为读取边信道，作者证明了这种假设是不正确的。更具体地说，作者展示了无特权的攻击者如何利用 Rowhammer 引起的位翻转和附近行中的位之间的数据依赖性来推断这些位，包括属于其他进程和内核的值。因此，这项工作的主要贡献是表明，Rowhammer 不仅威胁到完整性，而且也威胁到

机密性。此外，与需要持续进行位翻转的 Rowhammer 写端通道相反，即使 ECC 存储器检测并纠正了每个位翻转，作者的读取通道也可以成功。与以前的 Rowhammer 攻击不同，RAMBleed 攻击不需要使用大页面，并且可以在默认配置下在 Ubuntu Linux 上运行。

4.4 国内研究现状

清华大学魏少军、刘雷波团队提出一种基于动态可重构计算处理器实时监控的至强®内核硬件安全增强服务器 CPU 芯片。[9]

随着计算机和半导体技术的发展，CPU 芯片已经成为高度复杂的芯片，其设计、制造、封装及测试等过程涉及到全球化的产业分工，要对所有环节实施有效监管几乎不可能。此外，要在组成 CPU 的数十亿到上百亿颗晶体管中发现仅由数十颗晶体管就可以组成的恶意硬件也是不可能完成的任务。而人为疏忽或技术限制而造成的硬件漏洞，更是难寻踪迹，防不胜防。传统的通过检查 CPU 芯片的设计源码、网表、版图、管芯来查找恶意硬件和硬件漏洞的方法就如同大海捞针，完全不可行。2016 年，清华大学魏少军团队提出了基于高安全、高灵活可重构芯片架构的“CPU 硬件安全动态监测管控技术”。该技术通过动态、实时监控 CPU 运行过程中的“合法行为”来发现“非法行为”，从根本上克服了传统的 CPU 安全隐患、技术漏洞难以被监测和发现的困难。硬件木马、硬件漏洞（如“熔断”、“幽灵”）、硬件后门以及恶意利用硬件前门的行为，都能被该技术迅速发现并根据需要进行管控。

5．总结

通过对边信道攻击类论文的调研，以及其在不同场景的应用，按照级别、场景、实例、危害程度、修复的难易程度、研究前景，我将其总结为下表：

级别	场景	实例	危害	修复	前景
芯片级	硬件	针对 puf 的攻击、RAMBleed	严重	难	好
微架构级	微架构、体系结构	Meltdown、Spectre、Foreshadow、MDL	严重	难	好
系统级	操作系统	通过浏览器，获取用户鼠标活动	一般	较容易	较好
应用级	应用	Sql 盲注	一般	容易	一般

对边信道攻击的学术研究起于 1996-1999 年，传统的密码分析技术往往基于数学算法，效果很不理想，边信道攻击是一种不依赖于软件环境的攻击方法，它主要依靠一些物理信号现来判断电路中正在处理的数据，这种攻击方式比传统的方法更致命，更重要的一点，基于软件的漏洞利用可以通过升级、打补丁等方式进行修复，如果要对基于硬件信号进行利用的漏洞进行修复，就变得十分困难。

早在 1998 年，有关利用 Cache 命中率进行密钥分析的思路就已经

被提出来。在过去十年，利用旁道技术对 x86 CPU 实施的 Cache 攻击已经被证明是一项极其强大的攻击方法。2017 年，随着幽灵攻击（meltdown）和熔断攻击（spectre）的提出，利用旁信道技术针对 CPU 进行攻击的研究受到越来越多的人关注。

伴随着中美贸易战的持续进行，发展中国自己的芯片产业变得尤为重要。如果要发展好自己的芯片产业，除了在芯片核心技术上进行研究，还必须要保证芯片安全性，如果在芯片已经投入生产并且应用在国内大量厂商、政务机关之后，才被发现存在漏洞，这对芯片生产公司而言，这是一场巨大的灾难。在芯片投入生产前尽可能多地考虑安全问题，才能避免重复 Intel 目前的这种困境。

虽然清华大学魏少军团队提出的基于高安全、高灵活可重构芯片架构的“CPU 硬件安全动态监测管控技术”已经投入生产，但是，针对芯片安全的研究仍然还有很大的空间和潜力。

参考文献

[1]Genkin D, Pipman I, Tromer E. Get your hands off my laptop: Physical side-channel key-extraction attacks on PCs[J]. Journal of Cryptographic Engineering, 2015, 5(2): 95-112.

[2]Lipp M, Schwarz M, Gruss D, et al. Meltdown[J]. arXiv preprint arXiv:1801.01207, 2018.

[3] Kocher P, Horn J, Fogh A, et al. Spectre attacks: Exploiting speculative execution[C]//2019 IEEE Symposium on Security and Privacy (SP). IEEE, 2019: 1-19.

[4] Van Bulck J, Minkin M, Weisse O, et al. Foreshadow: Extracting the keys to the intel {SGX} kingdom with transient out-of-order execution[C]//27th {USENIX} Security Symposium ({USENIX} Security 18). 2018: 991–1008.

[5]Schwarz M, Lipp M, Moghimi D, et al. ZombieLoad: Cross-privilege-boundary data sampling[J]. arXiv preprint arXiv:1905.05726, 2019.

[6]van Schaik S, Milburn A, Österlund S, et al. RIDL: Rogue In-Flight Data Load[J]. S&P (May 2019), 2019.

[7]Canella C, Genkin D, Giner L, et al. Fallout: Leaking Data on Meltdown-resistant CPUs[C]//Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security. ACM, 2019: 769-784.

[8]Kwong A, Genkin D, Gruss D, et al. RAMBleed: Reading Bits in Memory Without Accessing Them[J].

[9] Liu L, Luo A, Li G, et al. Jintide®: A Hardware Security Enhanced Server CPU with Xeon® Cores under Runtime Surveillance by an In-Package Dynamically Reconfigurable Processor[C]//2019 IEEE Hot Chips 31 Symposium (HCS). IEEE, 2019: 1-25.