百度安全 有AI更安全 Baidu Security Advanced AI, Stronger Security

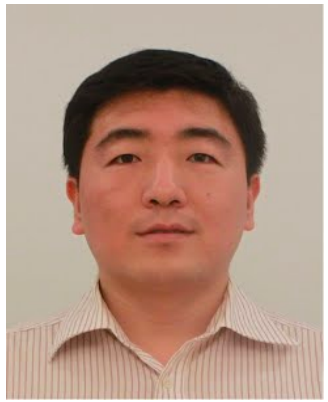
不可能完成的任务
从用户空间窃取内核数据

Yueqiang Cheng, Zhaofeng Chen, Yulong Zhang, Yu Ding, Tao Wei

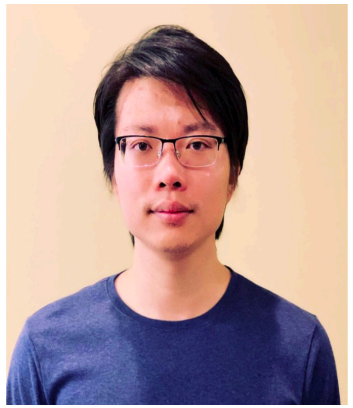
Baidu Security

关于演讲者

Baidu X-Lab



Dr. Yueqiang Cheng



Mr. Zhaofeng Chen



Mr. Yulong Zhang



Dr. Yu Ding



Dr. Tao Wei

我们的安全项目:

MesaLock MesaLink MesaPy MesaTEE MesaArmor

怎样从用户空间读取未授权的内核数据?

这为什么很难?

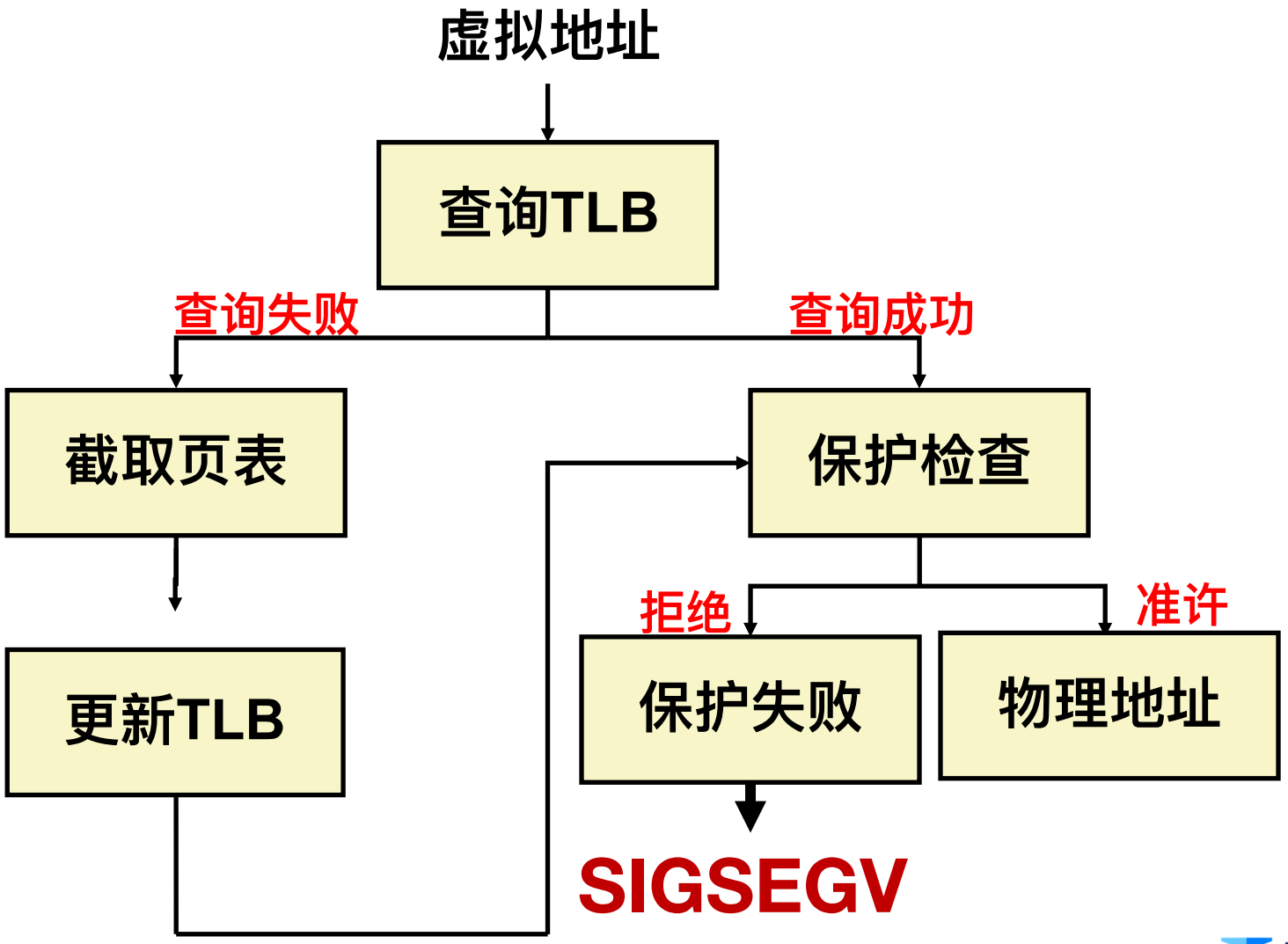
强有力的内核-用户隔离 (KUI)

MMU-页表(Page Table)的加持

假设内核无缺陷:

无内核缺陷来任意的读取内核数据

KUI里的内存访问



权限检查

1: 页表许可

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
Address of page directory[1]																				Ignored							PCD	PWT	Ignored			CR3
Bits 31:22 of address of 4MB page frame										Reserved (must be 0)						Bits 39:32 of address[2]			PAT	Ignored			G	1	D	A	PCD	PWT	U/S	R/W	1	PDE: 4MB page
Address of page table																				Ignored				0	Ign	A	PCD	PWT	U/S	R/W	1	PDE: page table
Ignored																															0	PDE: not present
Address of 4KB page frame																				Ignored			G	PAT	D	A	PCD	PWT	U/S	R/W	1	PTE: 4KB page
Ignored																															0	PTE: not present

2: 控制暂存器, 例如, SMAP in CR4

Image from Intel sdm

百度安全 有AI更安全 Baidu Security Advanced AI, Stronger Security

无计可施?



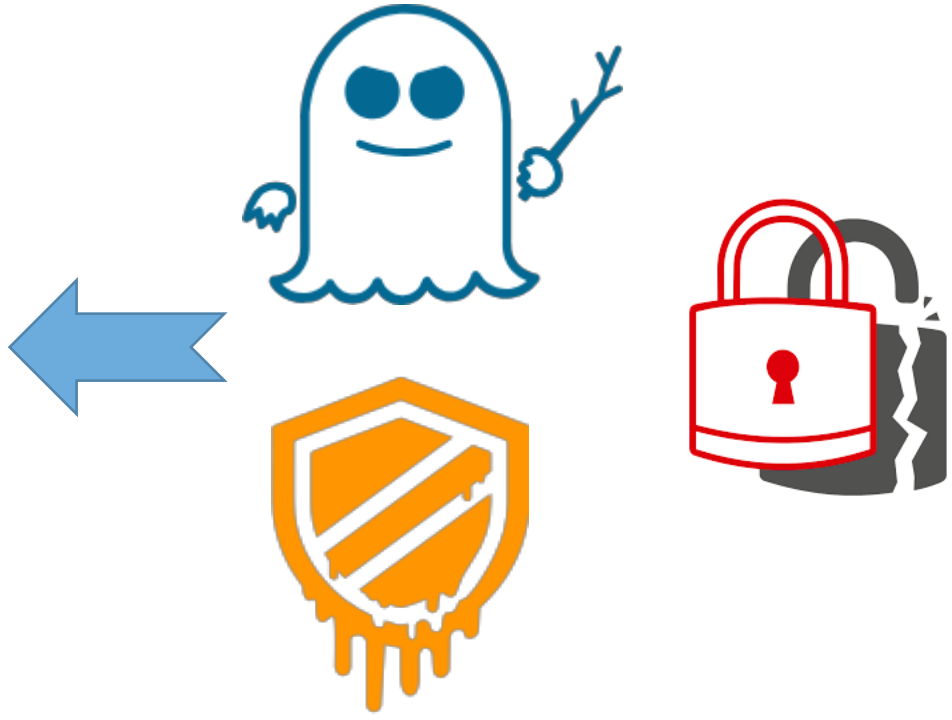
1.非特权应用程序+
2.KUI 权限检查+
3.无缺陷内核

但, 为了获得高性能的CPU …

1.非特权应用程序+

2.权限检查+

3.无缺陷内核





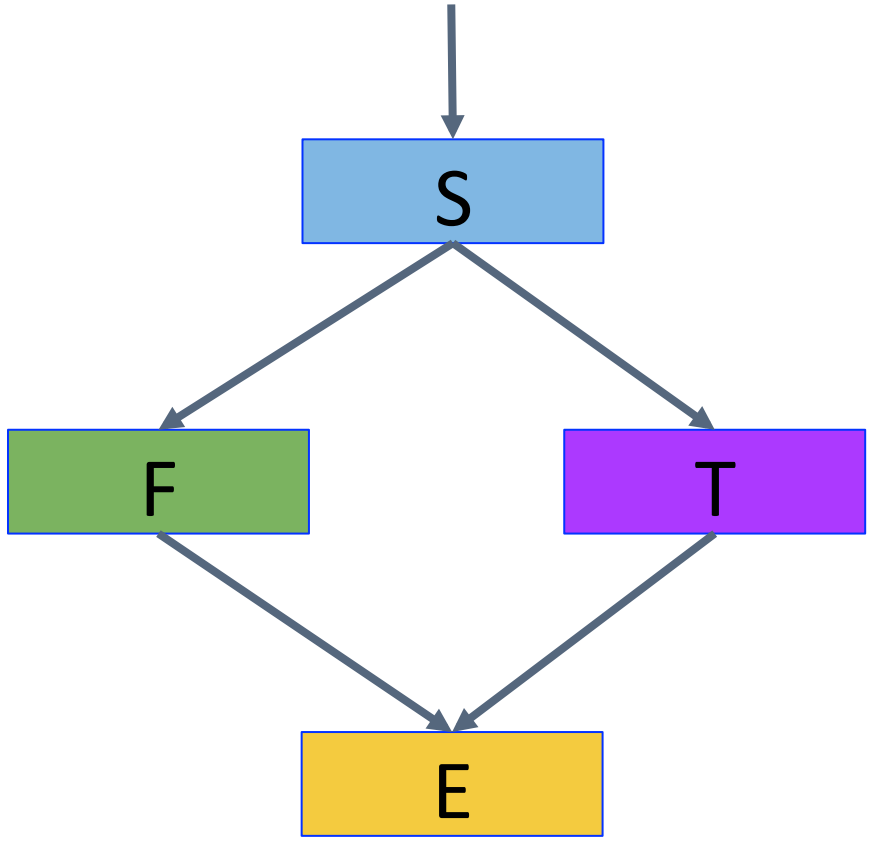




高性能

推测执行+ 无序执行

投机执行



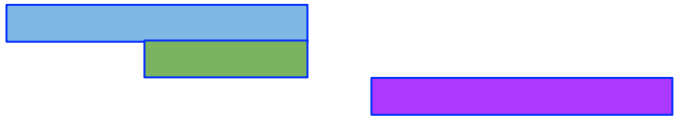
无推测执行



正确预测



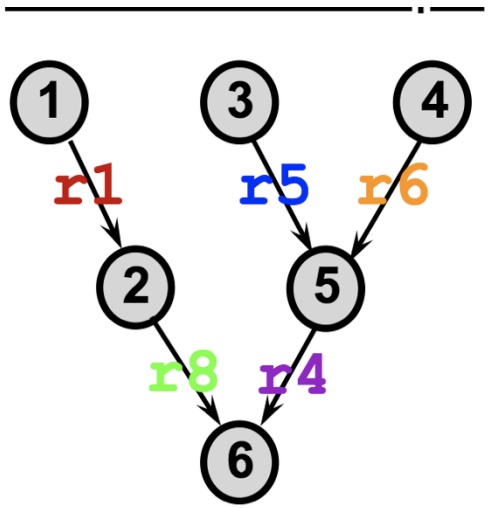
错误预测



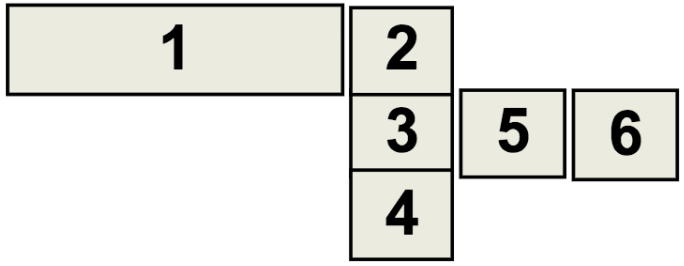
无序执行

例子：

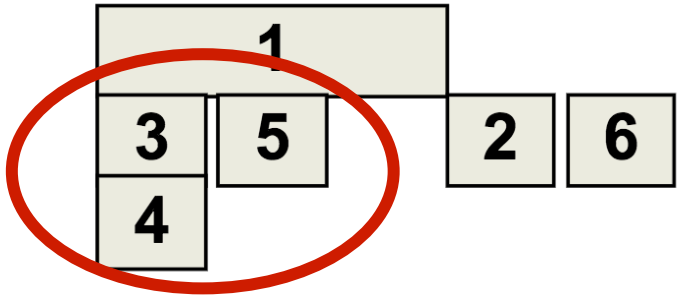
```
(1)  r1 ← r4 / r7
(2)  r8 ← r1 + r2
(3)  r5 ← r5 + 1
(4)  r6 ← r6 - r3
(5)  r4 ← r5 + r6
(6)  r7 ← r8 * r4
```



有序执行：



无序执行：



Images are from Dr. Lihu Rappoport

百度安全 有AI更安全 Baidu Security Advanced AI, Stronger Security

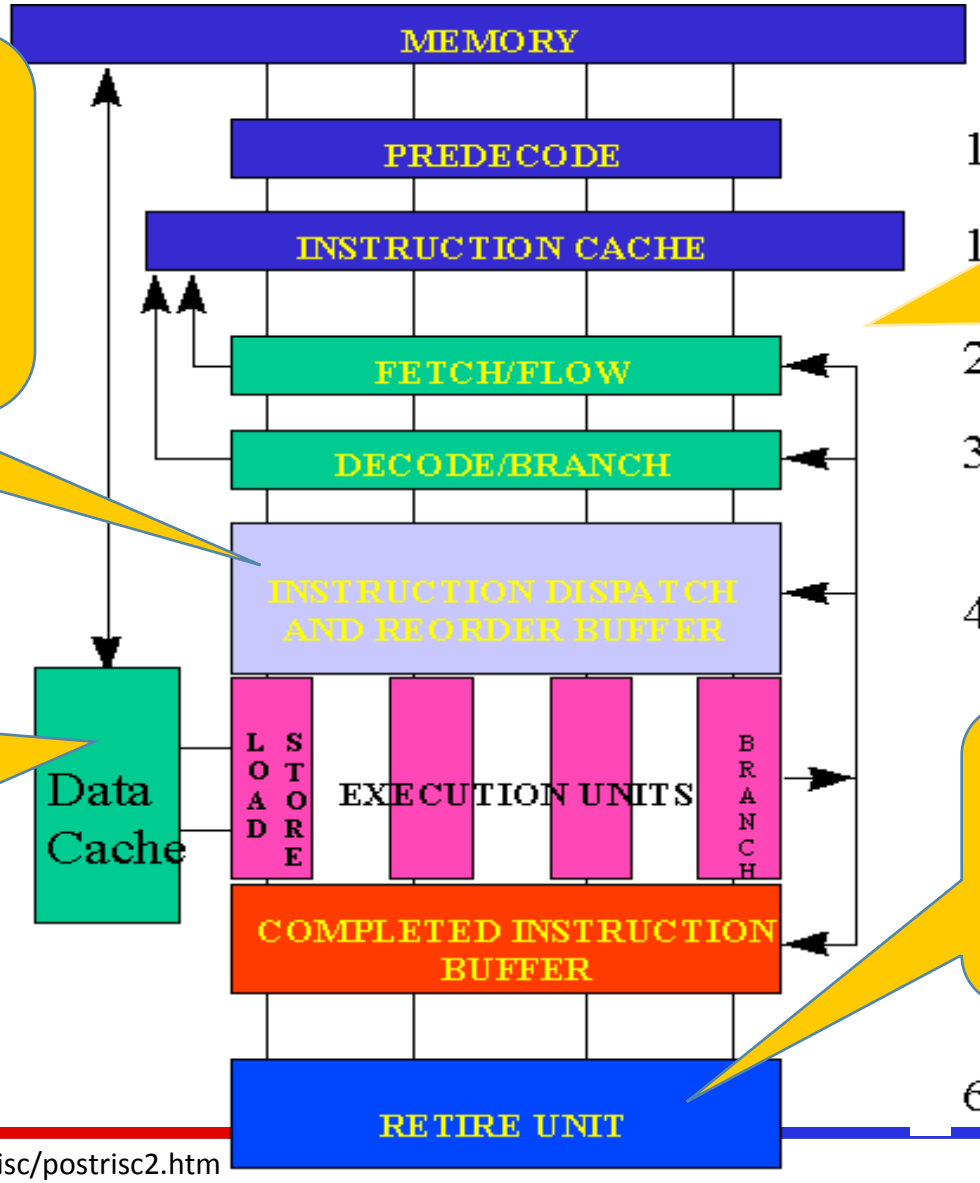
推测执行 + 无序执行

够了么？

不够!!!

延迟的权限检查+ 缓存副作用

执行引擎以无序方式执行



前端分支预测器服务于推测执行

缓存中的副作用仍然存在!!!

权限检查延迟到注销单元

Image from https://www.cse.msu.edu/~enbody/postrisc/postrisc2.htm

百度安全 有AI更安全 Baidu Security Advanced AI, Stronger Security

Meltdown（v3）的工作原理

1. 攻击者选择的，无法访问的内存位置的内容将加载到寄存器中。

```
asm __volatile__ (
    "%=:
    "xorq %%rax, %%rax                    \n"
    "movb (%[addr]), %%al                 \n"
    "shlq $0xc, %%rax                     \n"
    "jz %=b                               \n"
    "movq (%[dest], %%rax, 1), %%rbx      \n"
    :
    : [addr] "r" (addr), [dest] "r" (dest)
    : "%rax", "%rbx");
```

指向目标内核地址

Meltdown（v3）的工作原理

2. 临时指令根据寄存器的秘密内容访问缓存线。

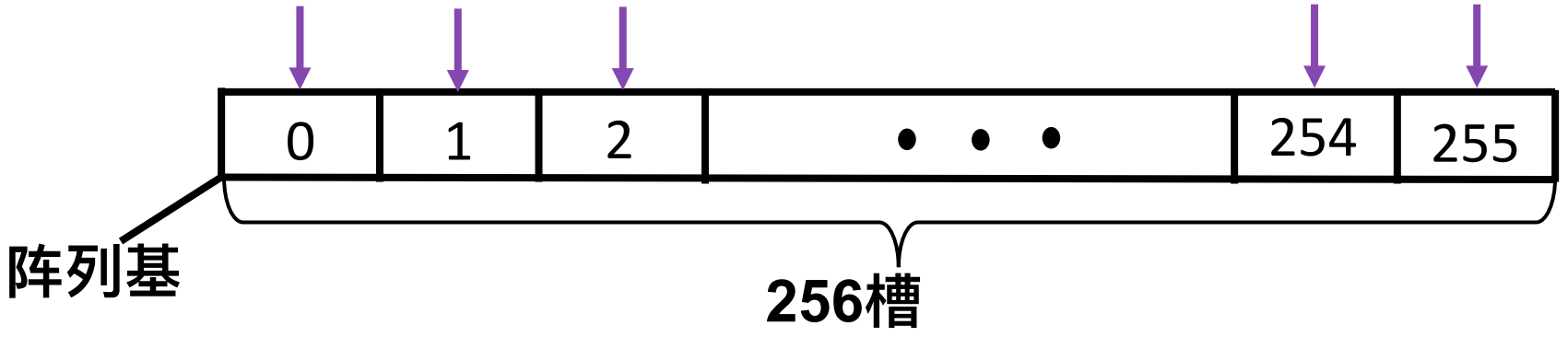
将数据放入缓存

这个数字应该 >= 0x6

```
asm __volatile__ (
    "%=:                                    \n"
    "xorq %%rax, %%rax                      \n"
    "movb (%[addr]), %%al                   \n"
    "shlq $0xc, %%rax                       \n"
    "jz %=b                                 \n"
    "movq (%[dest], %%rax, 1), %%rbx        \n"
    :
    : [addr] "r" (addr), [dest] "r" (dest)
    : "%rax", "%rbx");
```

Meltdown（v3）的工作原理

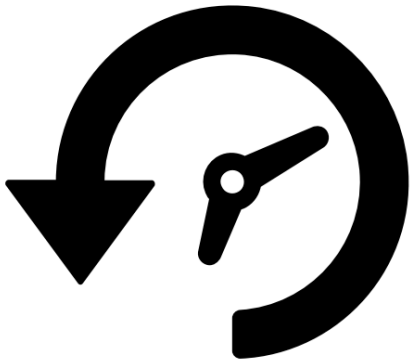
3. 攻击者使用flush+reload来确定访问的缓存线，从而确定存储在所选内存位置的秘密。



所选索引是目标字节的值。

例如，如果所选索引为0x65，则值为‘A’

ForeShadow



把秘密放在一级



取消映射页表条目



Meltdown

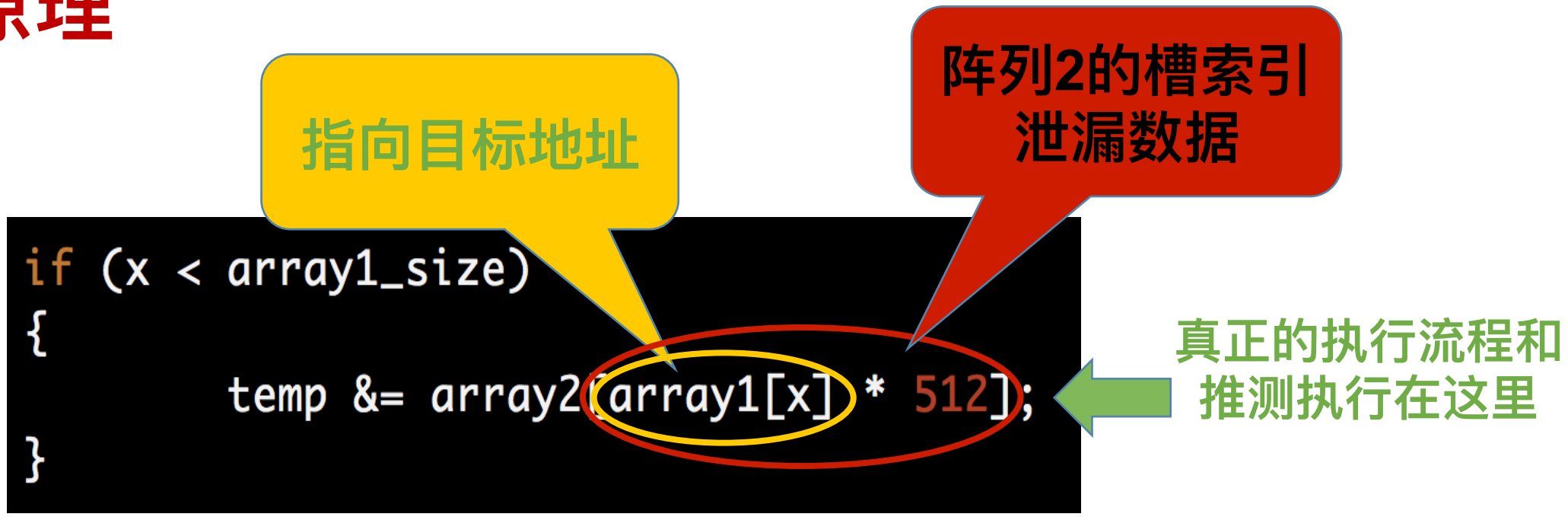


Spectre（v1/v2）怎么样?

Sepctre（v1）的工作原理

1. 在设置阶段，处理器被错误的训练去做出“一个可被利用的错误的推测。”

例如， x < array1_size



2. 处理器推测地将来自目标上下文的指令执行到微体系结构的隐蔽通道中。例如, x > array1_size

Sepctre（v1）的工作原理

执行流程应该在这里

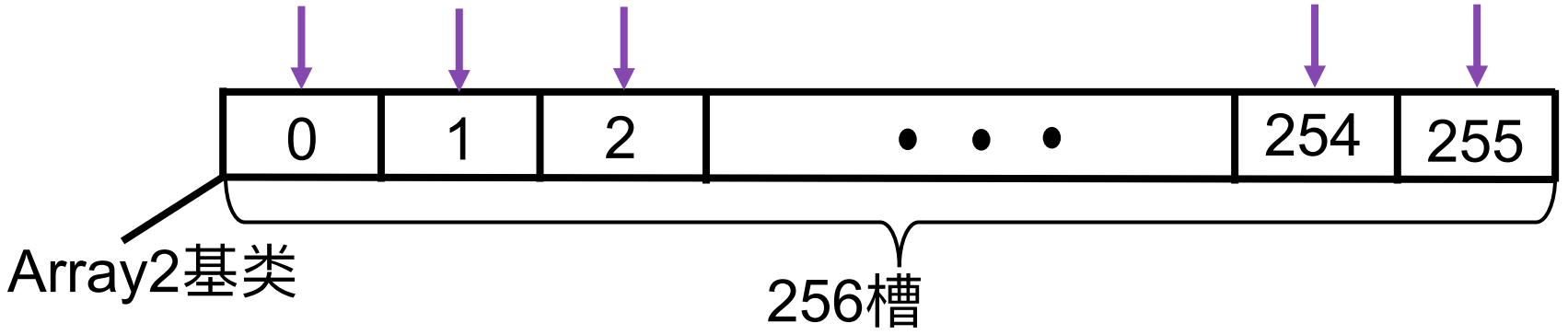
```
if (x < array1_size)
{
        temp &= array2[array1[x] * 512];
}
```

推测执行在这里!

将array2的一个槽加载到缓存中

Sepctre（v1）的工作原理

3: 可以通过定时访问CPU缓存中的内存地址来恢复敏感数据。



所选索引是目标字节的值。例如，如果所选索引是 **0x66**，那对应值就是‘**B**’。

Sepctre如何读取内核数据

- ✓ array1和array2在用户空间中
- ✓ x 由对方控制

array1+x 指向秘密

array2槽索引泄漏内核数据

```
if (x < array1_size)
{
        temp &= array2[array1[x] * 512];
}
```

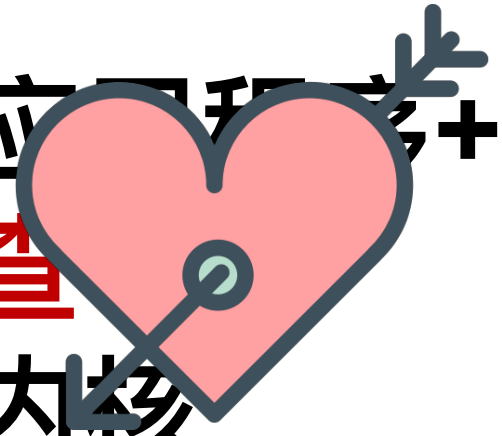
耶！我们现在得到内核数据了



1.非特权应用程序+

2.权限检查

3.无缺陷内核



但是...



KPTI

Meltdown

Spectre (Gadget in User

FAILED



SMA

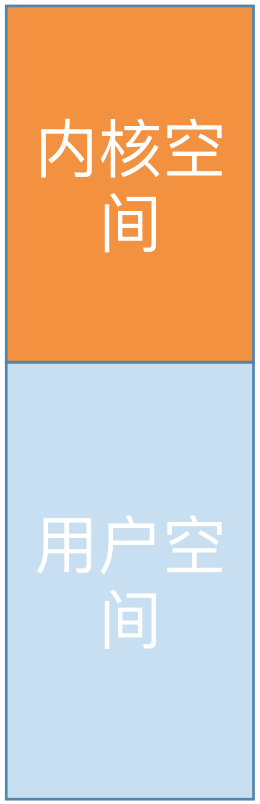
P

Spectre

(Gadget in Kernel Space)

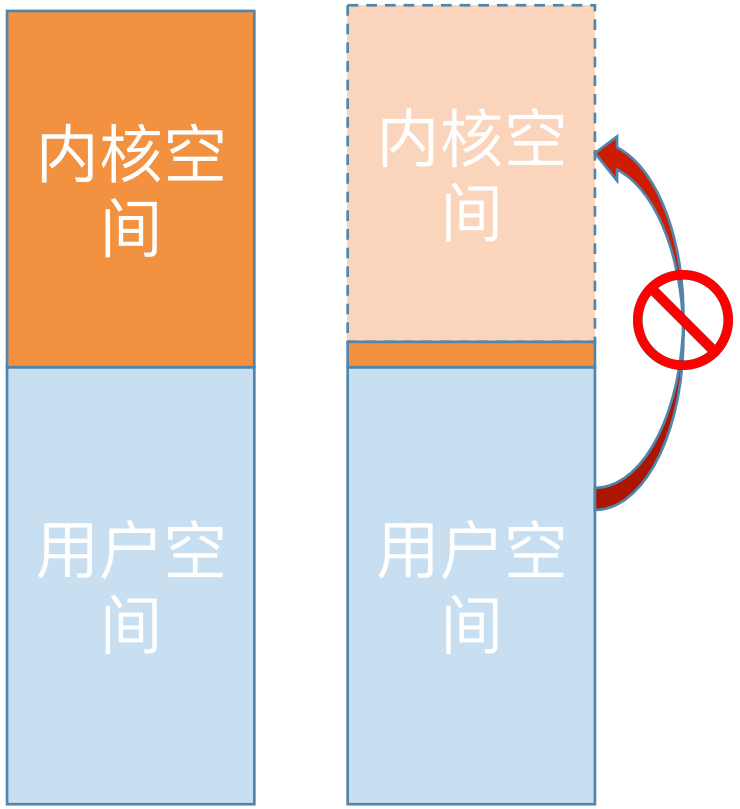
FAILED

KPTI



用户 / 内核模式

KPTI之前



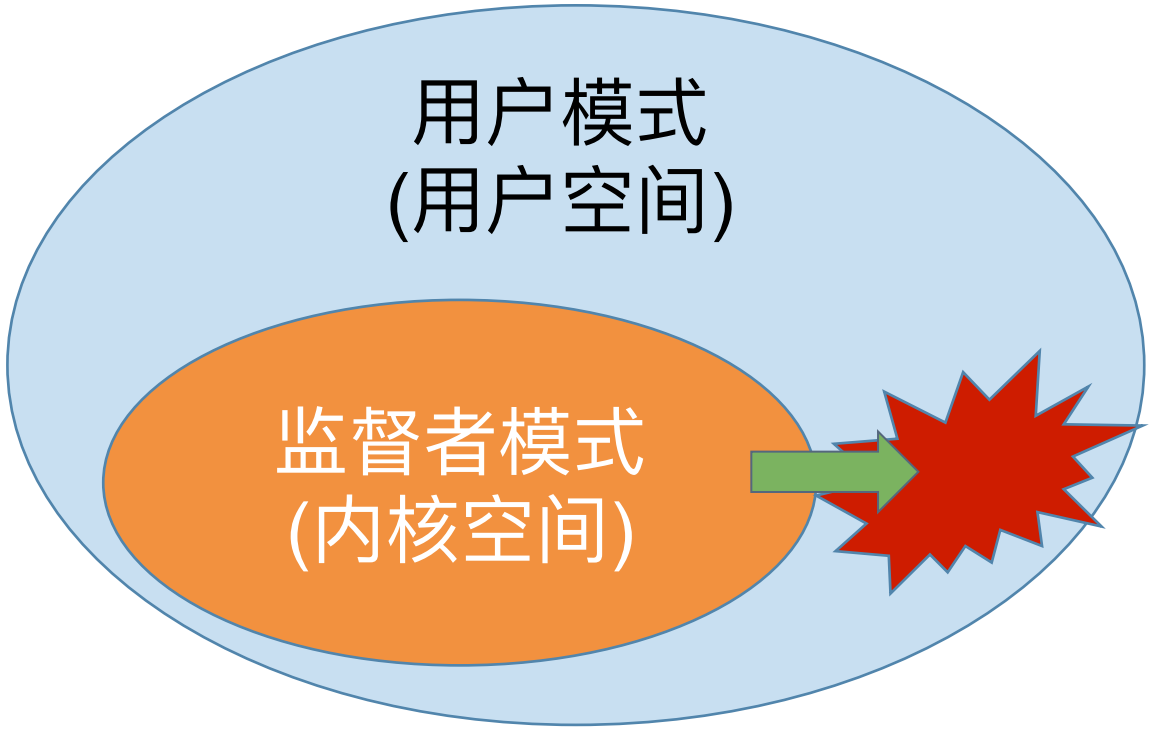
内核模式

用户模式

KPTI之后

PCID有助于提高性能

SMAP



- ✓ 当CR4中的SMAP位被设置时，SMAP将被启用。
- ✓ 通过对EFLAGS.AC标志的设置，可临时禁用SMAP。
- ✓ **SMAP检查早在退役，甚至执行之前就已经完成了。**

即使我们把Spectre组件放进内核空间，SMAP也会阻止它。

攻击和规避总结

技术	窃取内核数据?	规避	规避之后，内核数据泄漏?
Spectre	成功	KPTI + SMAP	失败
Meltdown	成功	KPTI	失败
ForeShadow	成功	KPTI	失败

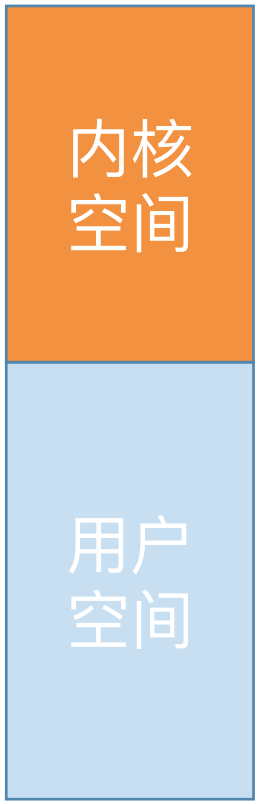
仅用于内核数据泄漏。其它总结不包含在这里。

绝望...



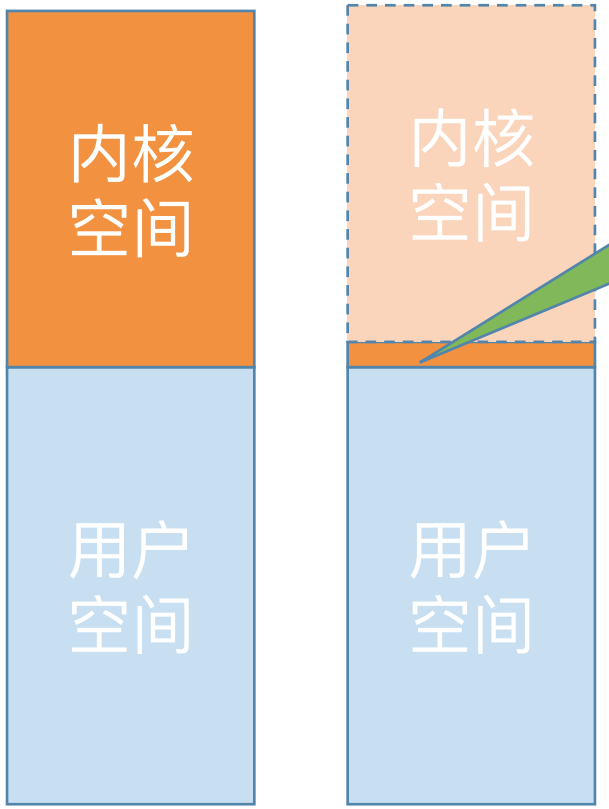
Image from http://nohopefor.us/credits

绝望中的希望



用户 / 内核模式

KPTI之前



内核模式　　用户模式

KPTI之后

这部分不能被消除

共享范围作为泄漏内核数据的**桥**

新型Meltdown v3z

中断 SMAP + KPTI + 用户内核隔离

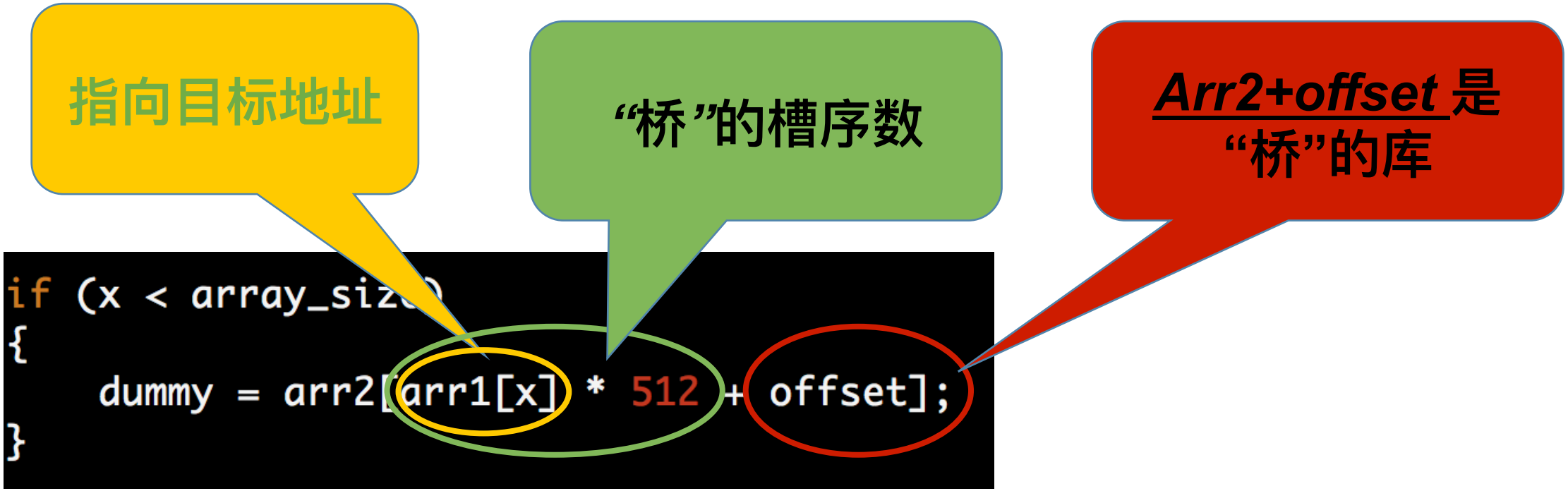
1: 使用**新的组件**在目标内核数据和桥之间建立数据依赖关系(绕过SMAP)

2: 使用可靠的**Meltdown**来探查桥，从而泄漏内核数据 (绕过 KPTI和KUI)



第一步: 触发New Gadget

与Spectre组件类似, 但不完全相同



x and ***offset*** 应由对方控制！！

如何触发新的组件

有很多方法可以触发新的组件：

1: Syscalls
2: /proc and /sys 等接口
3: 中断和异常处理程序
4: eBPF
5: …

如何找到新的组件

源代码扫描

我们在Linux Kernel 4.17.3中使用smatch,

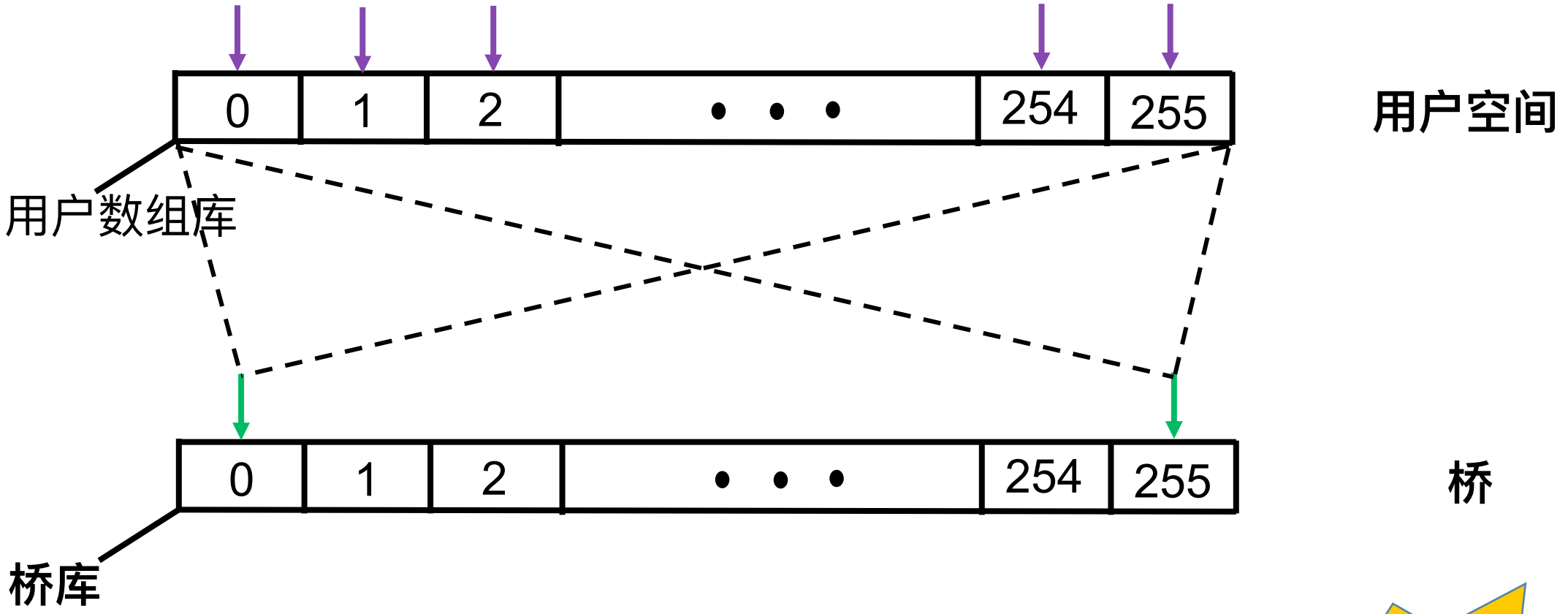
- 默认设置: 36 备选组件
- 全选设置: 166 备选组件

However, there are many restrictions to the gadget in real exploits

- ✓ 偏移量范围
- ✓ 可控调用
- ✓ 缓存噪音
- ✓ …

二进制代码扫描??

第二步: 探查桥

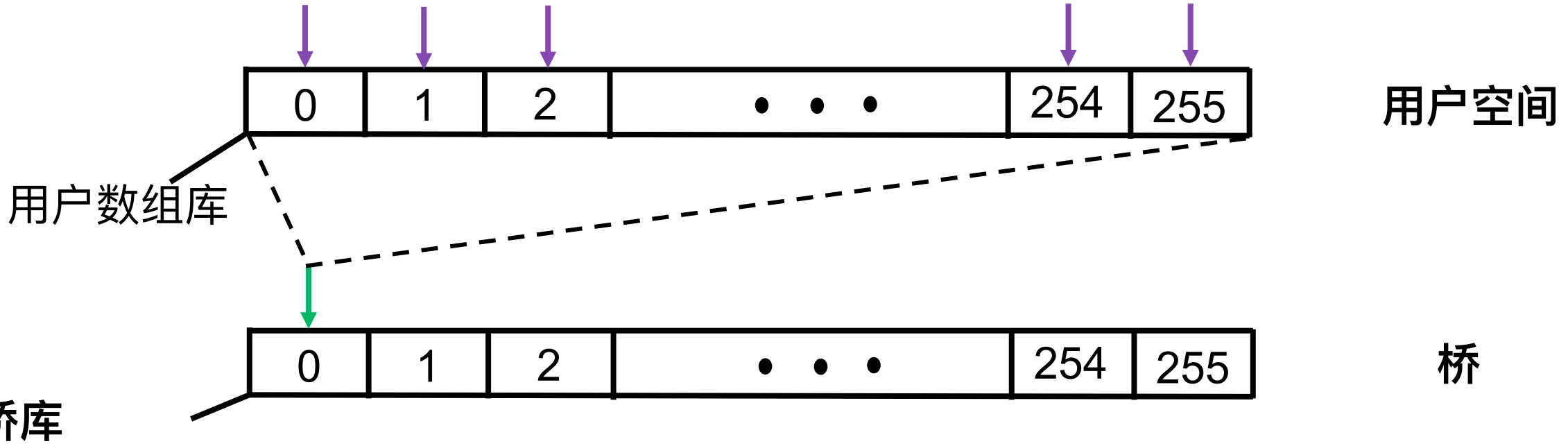


显然，每一轮都有（256*256）个探测
为了使结果可靠，通常我们需要运行多轮次

低效

Baidu Security
Advanced AI , Stronger Security

使其实用/高效



为什么我们需要在Meltdown里探测256次?
如果我们知道桥库槽0的值，我们只需探测一次。

我们可以提前知道这个值吗?

不适用于Meltdown (v3)

Meltdown能够读取内核数据。
但是, 它要求目标数据在CPU L1d缓存中。
如果目标数据**不在**L1d缓存中, **0x00**返回.

我们需要可靠地读取内核数据!

可靠的 Meltdown (V3r)

V3r 共有两步:

第一步：将数据放入L1d缓存

```
if (x < size) {
        data = array[x];
}
```

指向目标地址

内核中的任何地方

第二步：使用v3获取数据

我们使用Intel CPU E3-1280 v6在Linux 4.4.0，和Intel CPU I7-4870HQ在MacOS 10.12.6（16G1036）上进行测试。

信息汇总

离线阶段:

- 使用v3r转储网桥数据，并将其保存到表中

在线阶段:

- 第一步：在目标数据和桥槽之间建立数据依赖关系
- 第二步：探测桥的每个槽

效率:

- 从**几分钟** (在某些情况下甚至是1小时左右) 到只要**几秒钟**只泄漏 一个字节。

演示设置

内核: Linux 4.4.0 with SMAP + KPTI
CPU: Intel CPU E3-1280 v6

在内核空间, 我们有一个
机密消息, 例如, **xlabsecretxlabsecret**,
位置在, 例如, **0xffffffffc0e7e0a0**

探讨对策

软件规避

- ✓ 修补内核以消除所有预期的组件
- ✓ 最小化共享的“桥”区域
- ✓ 随机化共享的“桥”区域
- ✓ 监视基于缓存的侧通道活动

探讨对策

硬件规避

- ✓ 在执行阶段中执行权限检查
- ✓ 修改推测执行和无序执行
- ✓ 使用侧通道抗缓存，例如独占/随机缓存
- ✓ 增加硬件级侧通道检测机制

重点信息

- 在启用KPTI + SMAP + KUI的情况下，Trinational Spectre 和 Meltdown无法窃取内核数据。

- 我们新型的Meltdown variants 能够打破最强的保护(KPTI + SMAP + KUI)。

- 所有现有的内核都需要修补以规避我们的新攻击。



不可能完成的任务
从用户空间窃取内核数据

Yueqiang Cheng

百度安全

Q&A image is from https://i.redd.it/