安全开发者峰会

BULKHEAD：通过分隔化打造内核安全的水密舱

郭迎港 南京大学

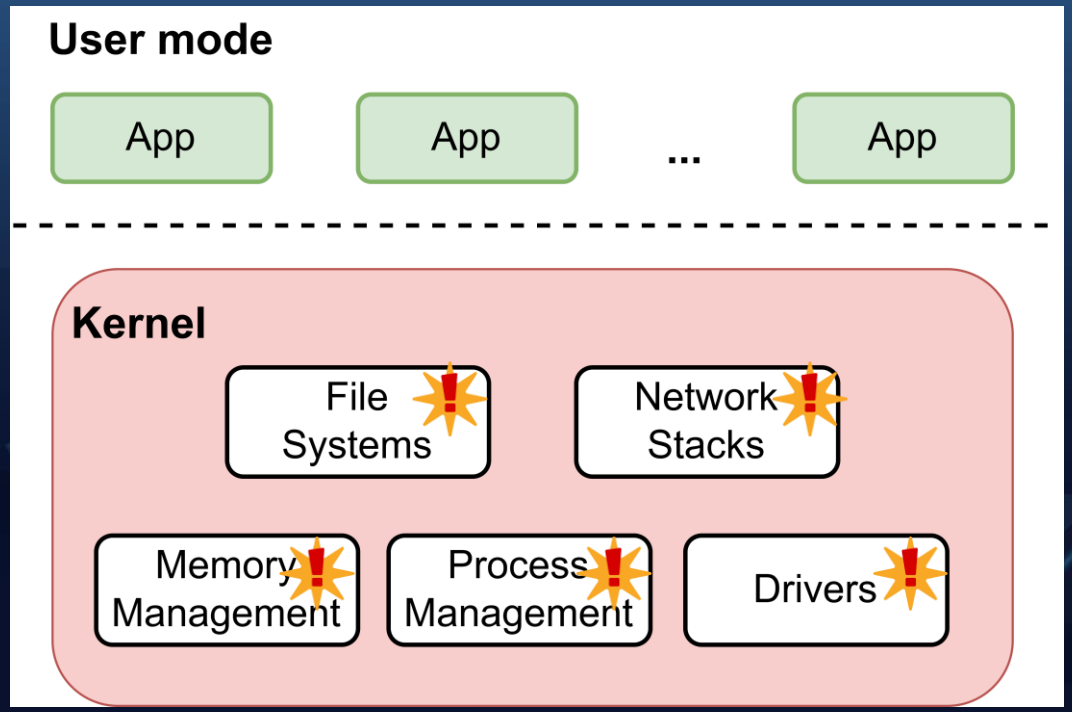
关于我

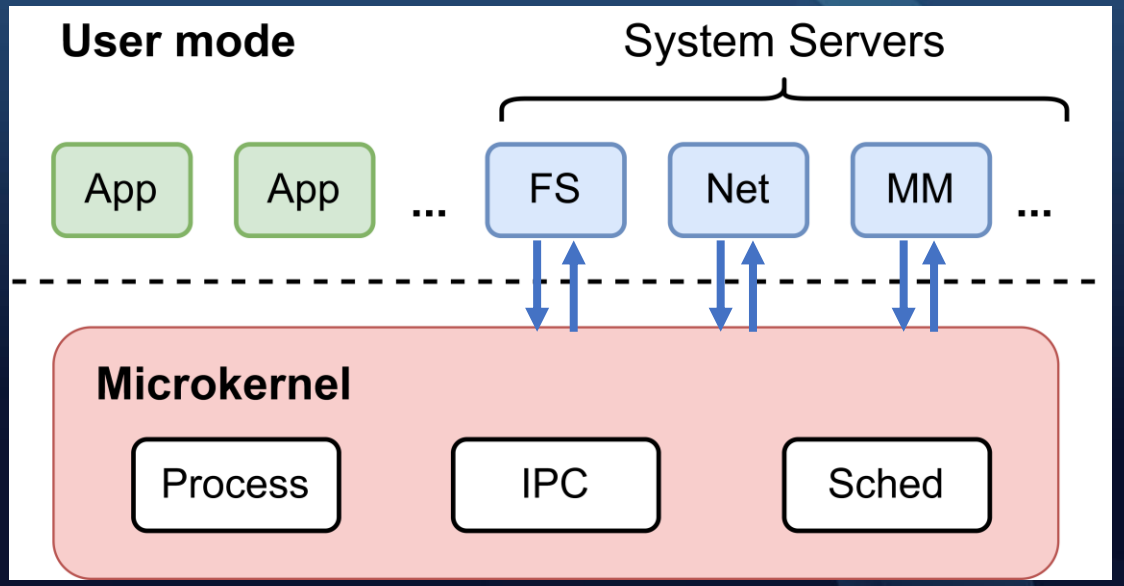
- 南京大学计算机学院博士生，师从曾庆凯教授，入选南大优秀博士生提升计划
- 曾赴美国明尼苏达大学Prof. Kangjie Lu研究组做访问学者
- 研究方向为**操作系统内核安全**，包括特权分离和内核模块分隔化
- 结合硬件特性、程序分析及形式化建模践行最小特权原则，**限制潜在漏洞的影响**
- 相关成果发表于NDSS、ACSAC、ESORICS、软件学报等会议和期刊

目录

宏内核 VS 微内核

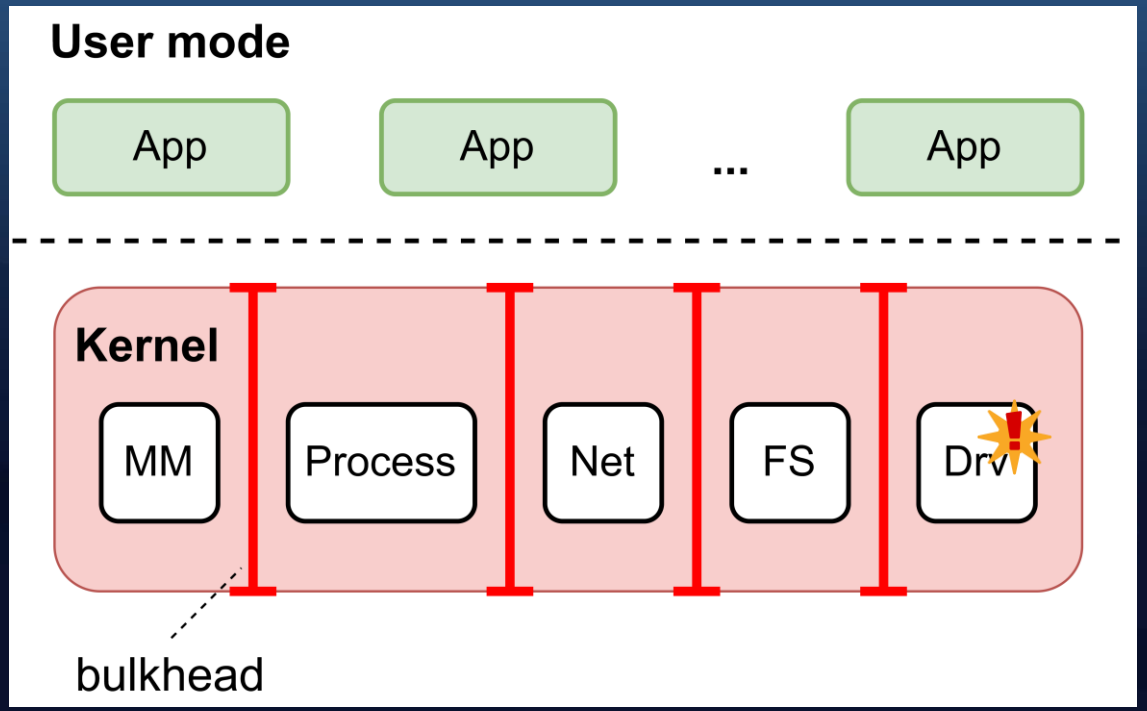
- 宏内核：模块高效交互但缺乏故障隔离
- 微内核：模块安全隔离但IPC开销大

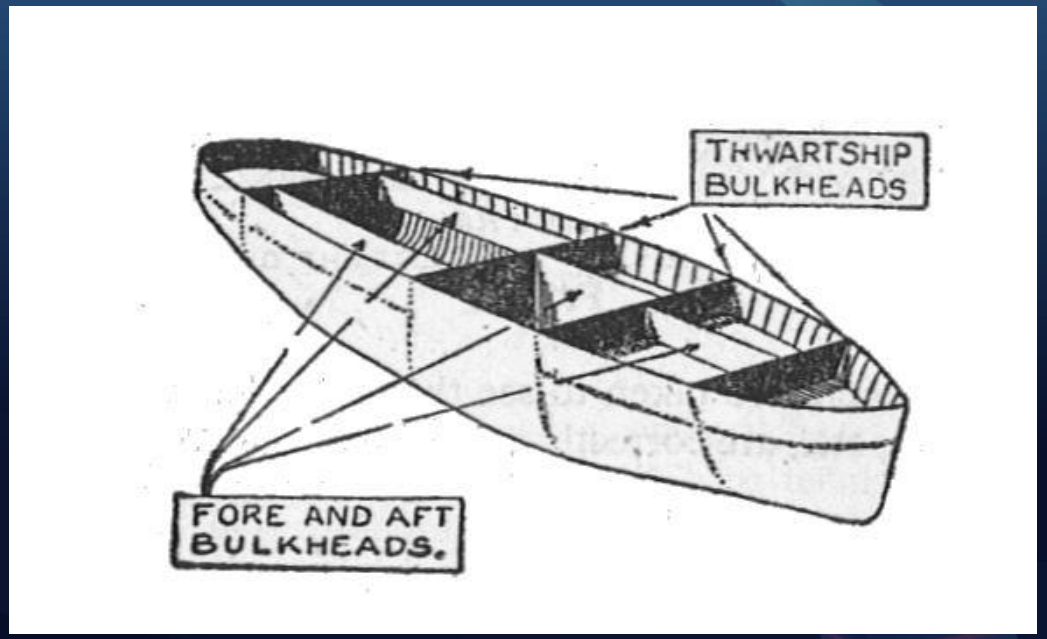




内核分隔化（kernel compartmentalization）

- 将内核模块分隔至互不可信的安全域中以限制潜在漏洞的影响范围





目录

内核漏洞层出不穷

- 消除所有内核漏洞并不现实

近五年Linux kernel CVE数量

** https://www.cvedetails.com/product/47/Linux-Linux-Kernel.html?vendor_id=33*

内核漏洞层出不穷

- 数量最多的内核漏洞类型：内存损坏、溢出、输入验证

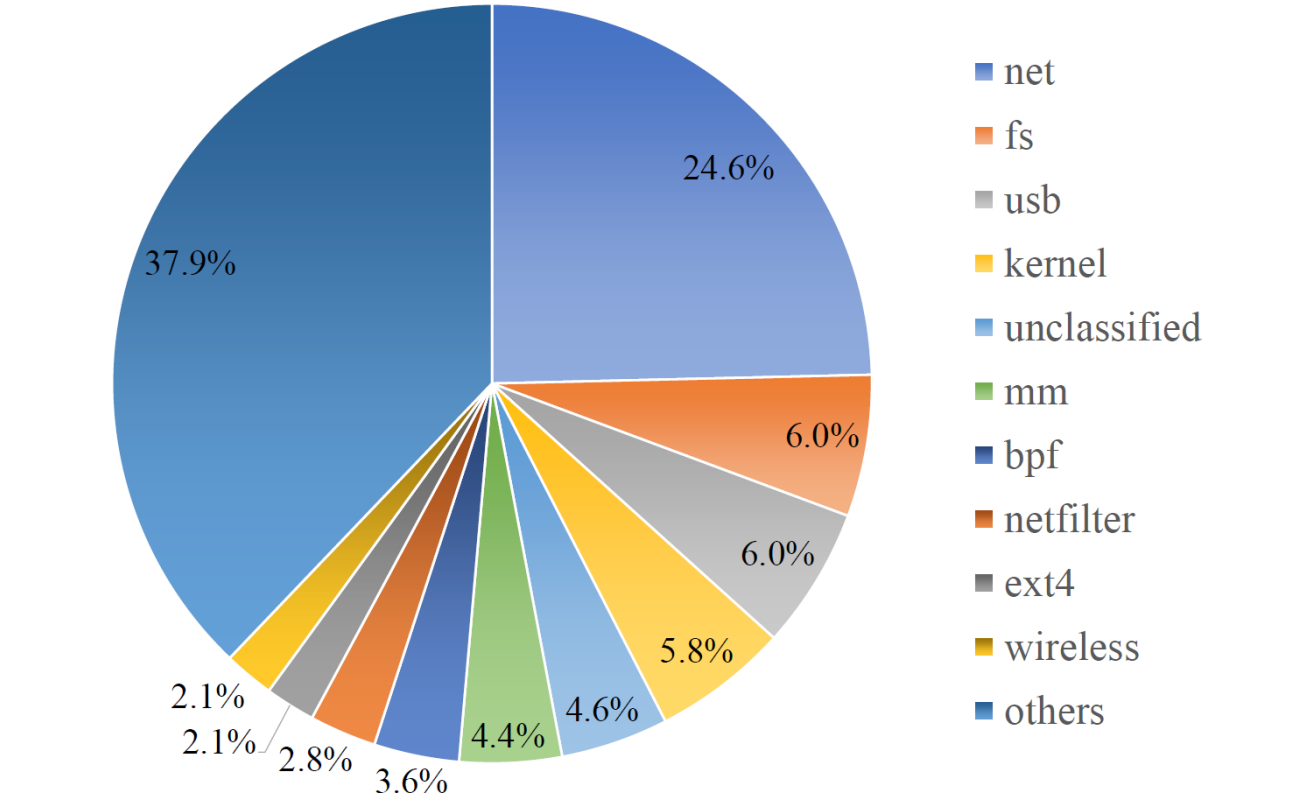
Vulnerabilities by types/categories

Year	Overflow	Memory Corruption	Sql Injection	XSS	Directory Traversal	File Inclusion	CSRF	XXE	SSRF	Open Redirect	Input Validation
2014	18	31	0	0	0	0	0	0	0	0	22
2015	13	17	0	0	0	0	0	0	0	0	5
2016	36	76	0	0	0	0	0	0	0	0	17
2017	62	86	0	0	1	0	0	0	0	0	20
2018	32	70	0	0	0	0	0	0	0	0	11
2019	30	124	0	0	1	0	0	0	0	0	4
2020	10	40	0	0	1	0	0	0	0	0	2
2021	18	54	0	0	1	0	0	0	0	0	7
2022	41	149	0	0	0	0	0	0	0	0	2
2023	19	165	0	0	0	0	0	0	0	0	1
2024	31	649	0	0	0	0	0	0	0	0	0
Total	310	1461			4						91

** https://www.cvedetails.com/product/47/Linux-Linux-Kernel.html?vendor_id=33*

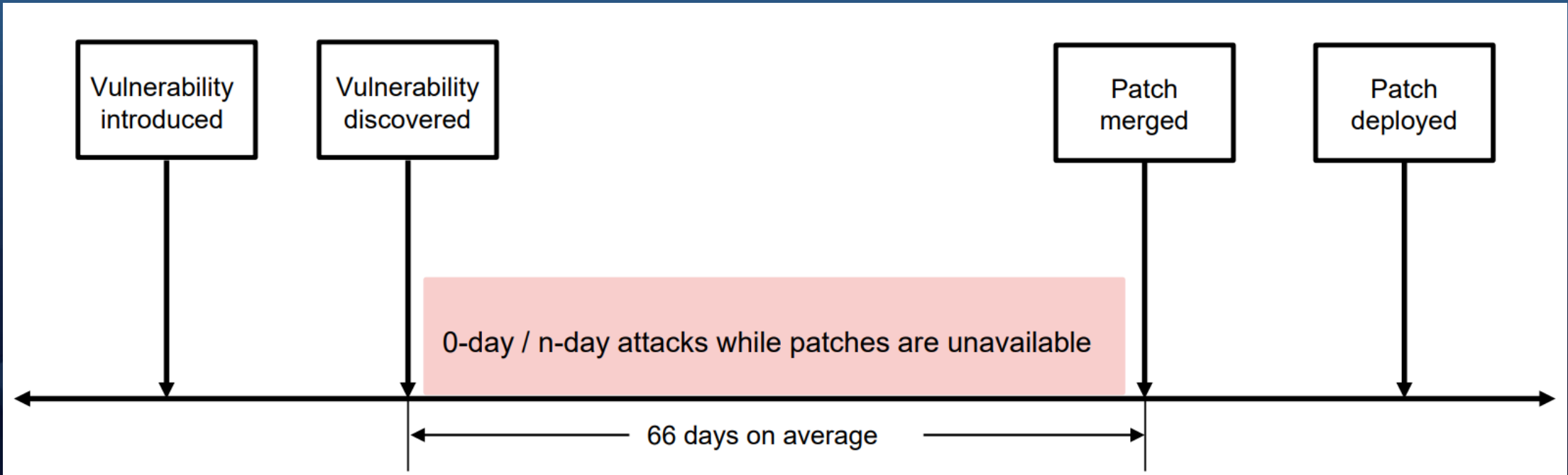
内核漏洞分布广泛

- 无法预测下一个高危漏洞会出现在哪里



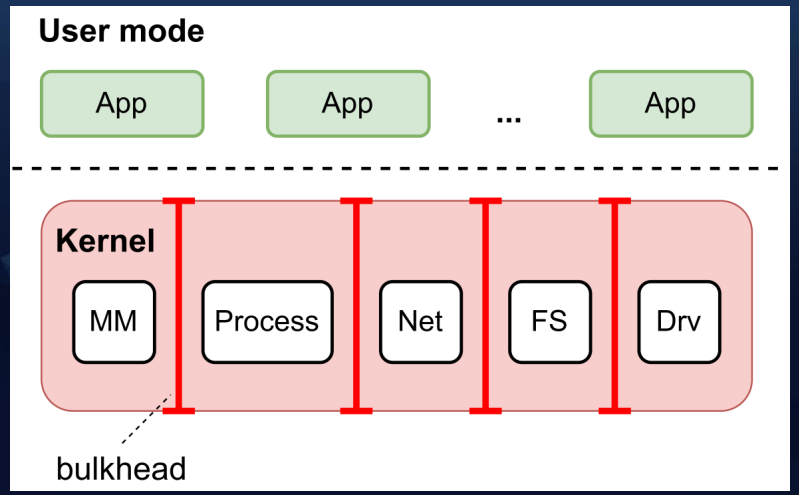
内核漏洞修补困难

- 漏洞修复时间窗口内不应束手待毙



通过分隔化应对内核漏洞

- 层出不穷：消除所有内核漏洞并不现实
- 分布广泛：无法预测下一个高危漏洞会出现在哪里
- 修补困难：漏洞修复时间窗口内不应束手待毙



目录

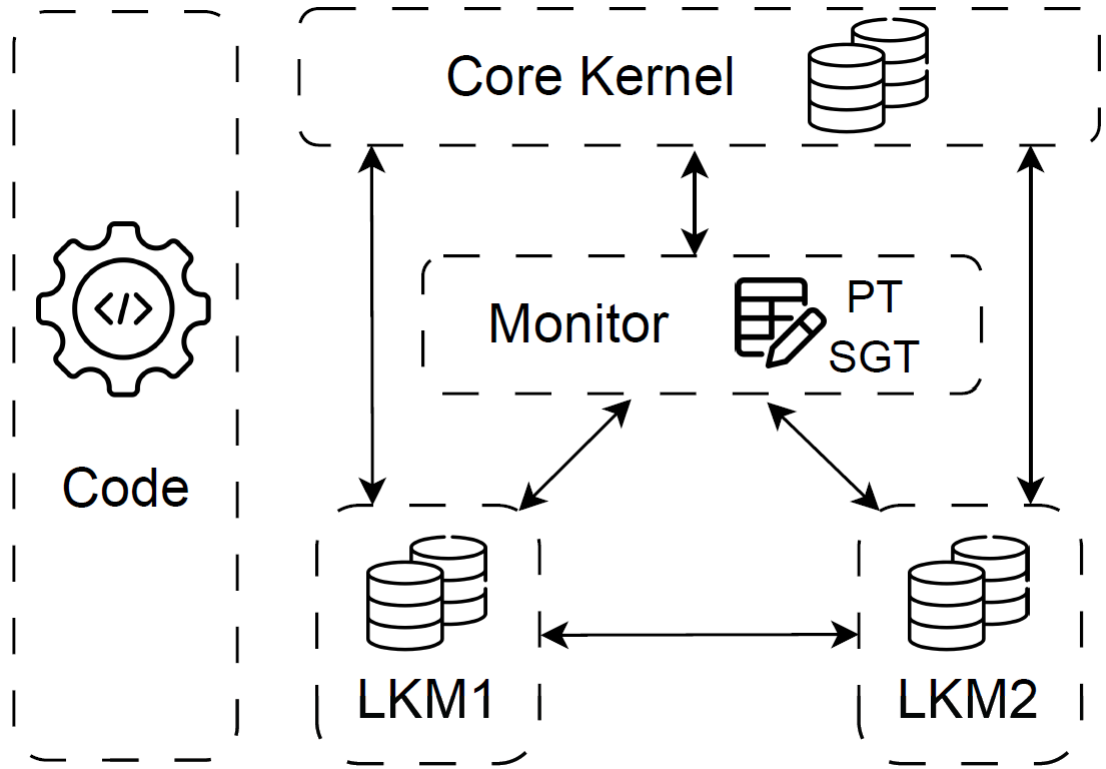
隔离机制对比

- 基于软件插桩检查（SFI）的隔离
  - 检查每条访存指令影响性能
- 基于地址空间的隔离
  - 地址空间切换、数据传输开销大
- 基于虚拟化的隔离
  - 依赖于更高特权层的hypervisor
- 基于硬件机制的隔离
  - 软硬结合，平衡安全与性能

基于Intel PKS的内核分隔化

页表项中4位pkey [59:62]划分内存域

PKRS寄存器控制访问权限（WD不可写，AD不可访问）



PKRS

	…	pkey4 WD	pkey4 AD	pkey3 WD	pkey3 AD	pkey2 WD	pkey2 AD	pkey1 WD	pkey1 AD	pkey0 WD	pkey0 AD
	…	LKM2		LKM1		Monitor		Code		Core	
Core	…	1	0	1	0	1	0	0	1	0	0
Monitor	…	1	0	1	0	0	0	0	1	1	0
LKM1	…	0	1	0	0	1	0	0	1	1	0
LKM2	…	0	0	0	1	1	0	0	1	1	0

系统架构图

In-kernel monitor特权分离

- 页表只能由monitor管理
  - 标有monitor pkey的页表内存池
  - Hook页表分配和更新函数切换至monitor
- 特权寄存器只能由monitor更新
  - Intended指令：替换为switch gate
  - Unintended指令：二进制改写

750f	jne 0xf(%rip)
30c0	xor %al,%al

750f	jne 0xf(%rip)
90	nop
30c0	xor %al,%al

插入nop指令

In-kernel monitor特权分离

- 页表只能由monitor管理
  - 带有monitor pkey的页表内存池
  - Hook页表分配和更新函数切换至monitor
- 特权寄存器只能由monitor更新
  - Intended指令：替换为switch gate
  - Unintended指令：二进制改写

488b440f30	mov 0x30(%rdi,%rcx,1),%rax

52	push %rdx
4889ca	mov %rcx,%rdx
488b441730	mov 0x30(%rdi,%rdx,1),%rax
5a	pop %rdx

寄存器重分配

In-kernel monitor特权分离

- 页表只能由monitor管理
  - 带有monitor pkey的页表内存池
  - Hook页表分配和更新函数切换至monitor
- 特权寄存器只能由monitor更新
  - Intended指令：替换为switch gate
  - Unintended指令：二进制改写

```
41bd0f300000    mov $0x300f,%r13d
```

```
41bd00300000    mov $0x3000,%r13d
4183c50f        add $0xf,%r13d
```

数据调整

安全威胁

- 数据篡改（Data-oriented Attacks）
- 控制流劫持（Control Flow Hijacking）
- 混淆代理攻击（Confused Deputy Attacks）

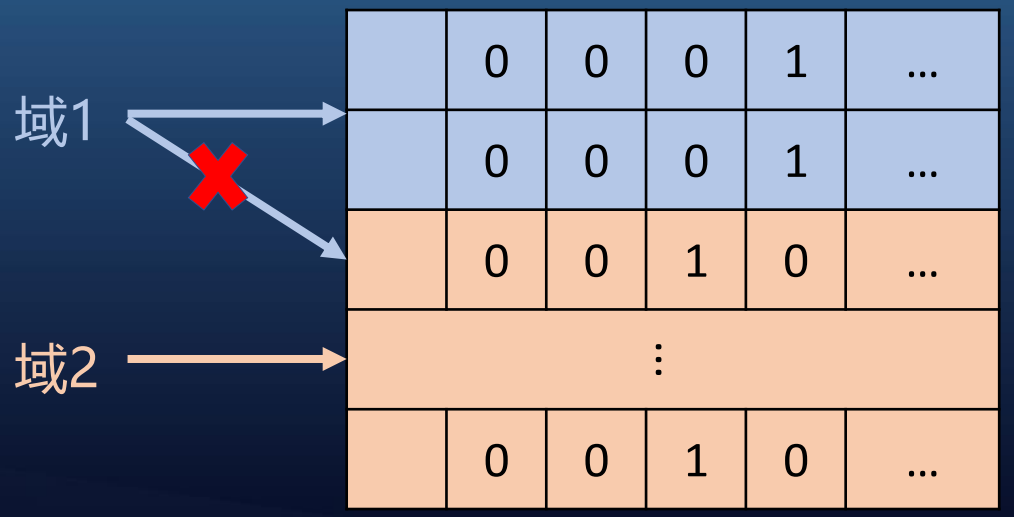
安全威胁-1

- 数据篡改（Data-oriented Attacks）
  - 例如*cred*, *file*, *inode*
- 控制流劫持（Control Flow Hijacking）
- 混淆代理攻击（Confused Deputy Attacks）

安全不变式-1

- **数据完整性**（Data Integrity）
  - 基于内存池的私有堆栈

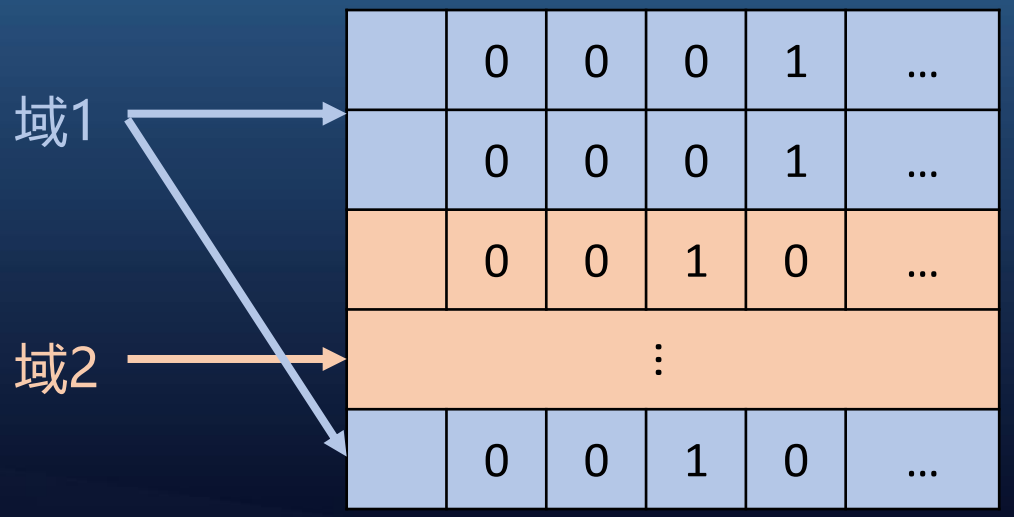
    页表项中的pkey标记了每个内存对象唯一的的所属域。通过hook内存分配函数确保每个域使用标有自身pkey的私有堆/栈。堆/栈漏洞无法篡改其他域的数据。



安全不变式-1

- **数据完整性**（Data Integrity）
  - 零拷贝的数据传输

    对于共享数据，在page fault handler中检查数据合法性，并根据结果由monitor更新数据所属域。

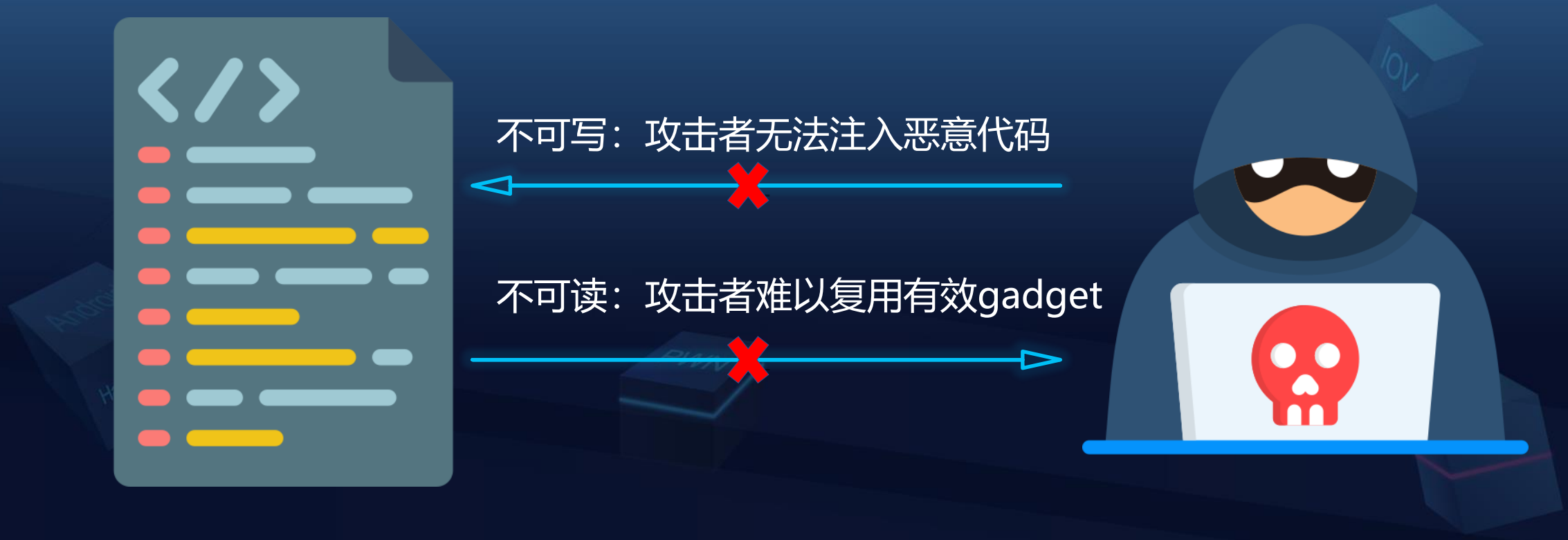


安全威胁-2

- 数据篡改（Data-oriented Attacks）
- 控制流劫持（Control Flow Hijacking）
  - 恶意代码注入
  - 代码复用攻击（如ROP）
- 混淆代理攻击（Confused Deputy Attacks）

安全不变式-2

- **仅执行内存**（eXecute-Only Memory）



安全威胁-3

- 数据篡改（Data-oriented Attacks）
- 控制流劫持（Control Flow Hijacking）
- 混淆代理攻击（Confused Deputy Attacks）
  - 利用接口混淆其他模块代为执行恶意行为

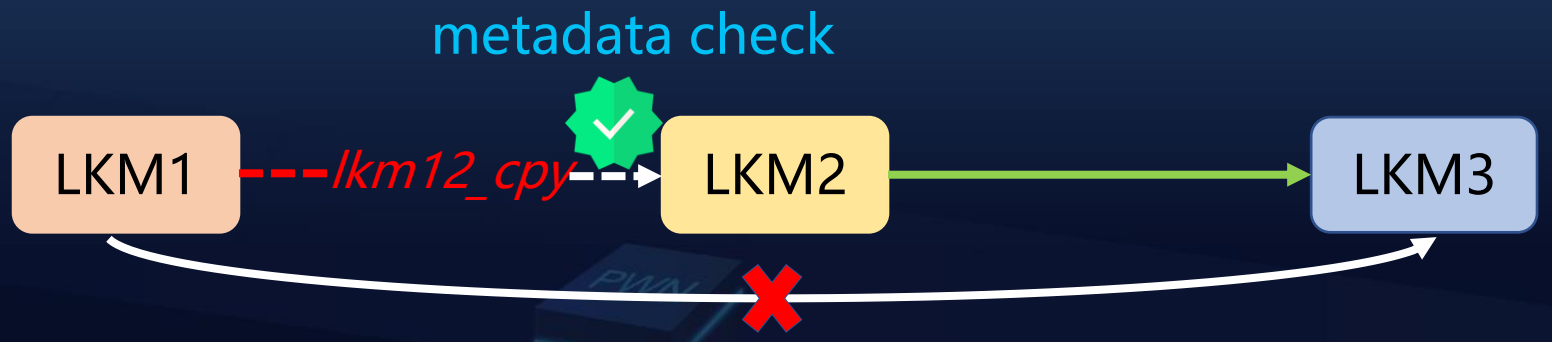
混淆代理攻击示例

- 假设LKM1不能访问LKM3内存，LKM2可以访问LKM3内存
- LKM1可以调用接口*lkm12_cpy(void *mem2, const void *mem1, int size)*

- 通过传入恶意参数，LKM1混淆LKM2代理其篡改LKM3内存

安全不变式-3

- **隔间接口完整性**（Compartment Interface Integrity）
  - 隔间切换只能发生在指定的出入口点
  - 数据交互必须严格遵循安全策略



隔间接口完整性

运行前

- 基于LLVM静态分析的安全策略生成
  - 边界分析、共享状态分析、安全约束分析
- 根据安全策略注册switch gate table (SGT)
- 通过LLVM插桩在接口处插入switch gate

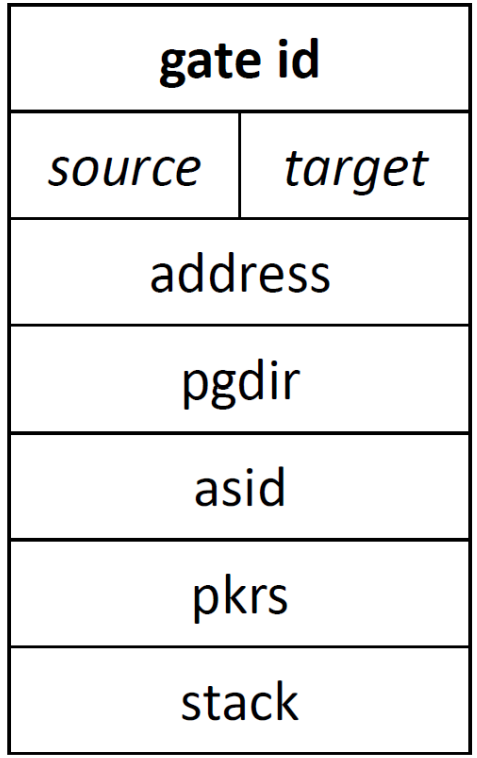
运行时

- switch gate通过gate id查找SGT
- 根据SGT中的metadata验证交互双方信息
- 验证通过时原子性地切换至目标隔间

隔间接口完整性

Switch gate伪代码：原子性、确定性、排他性

```
1  get_metadata(gate_id);
2  verify(source_addr);
3  if (target_pgdir != source_pgdir)
4      load_new_mm_cr3(target_pgdir, target_asid);
5  if (target_pkrs != current_pkrs)
6  loop:
7      write_pkrs(target_pkrs);
8  if (current_pkrs != target_pkrs)
9      goto loop;
10 switch_stack(target_stack);
11 jump(target_addr);
```



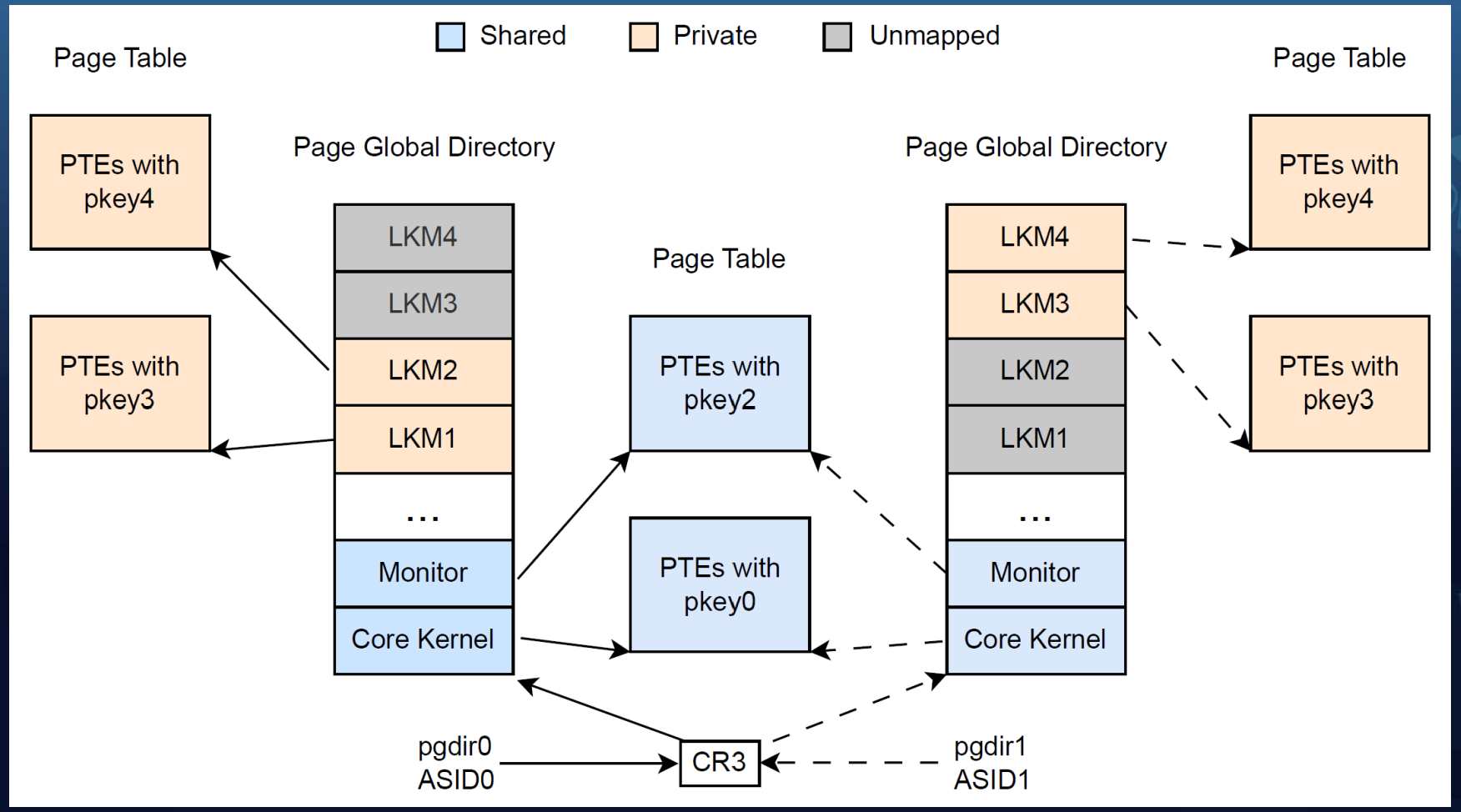
两级分隔化扩展

- 4位pkey只能支持16个不同的分隔域
- 不同地址空间可以复用pkey
- 地址空间切换开销大

Operation	Cost (cycles)
PKRS update by `wrmsr`	185.31
compartment switch	224.31
com-switch without stack switch	203.96
com-switch with address space switch	523.69
syscall null	293.86
hypercall null	2894.06
`vmfunc` (EPT switch)	198.82

两级分隔化扩展

基于PKS的地址空间内隔离 + 局部性感知的地址空间切换



目录

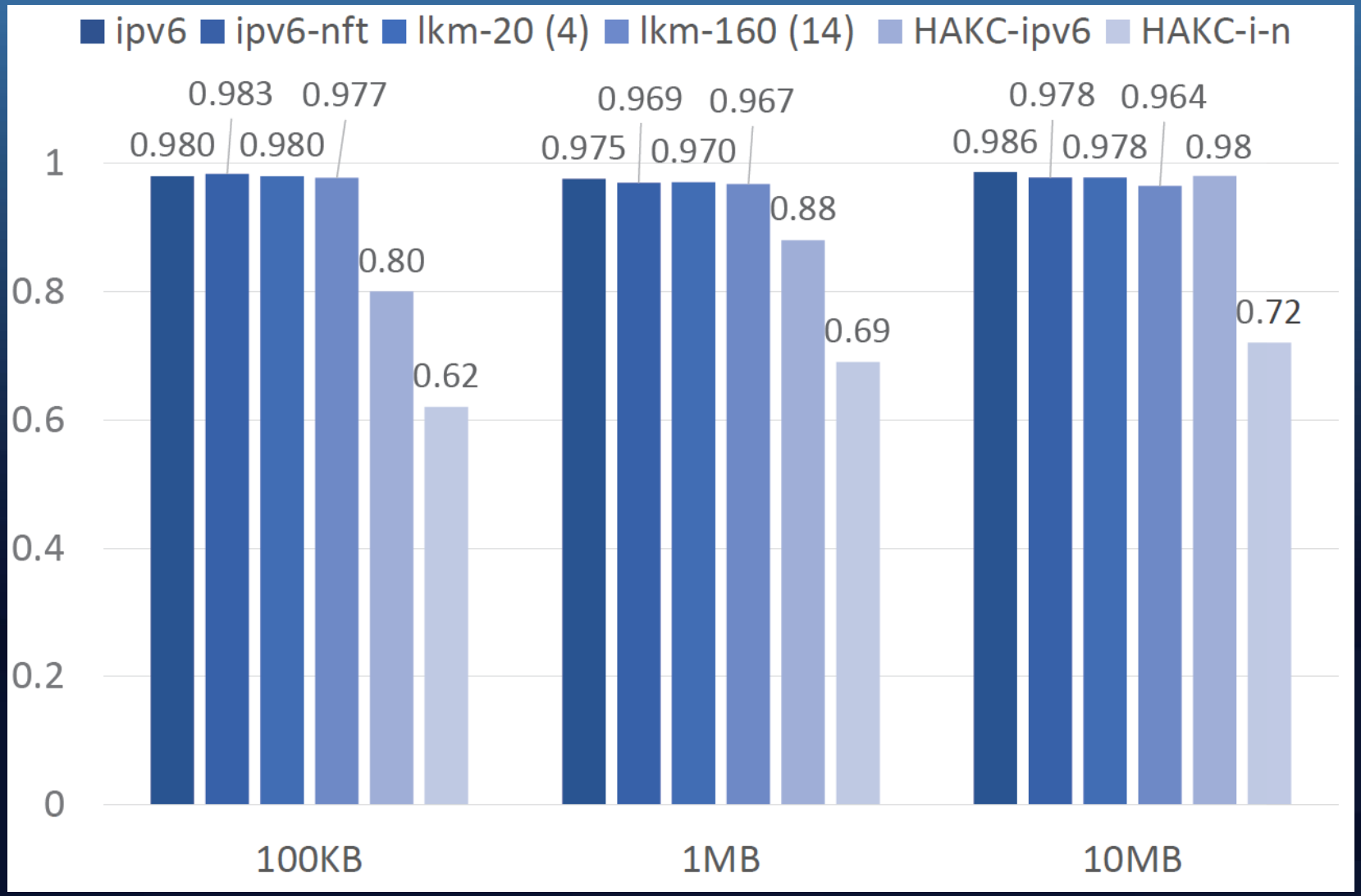
代表性CVE安全分析

CVE ID	Root Cause	Compartment	Countermeasures
2023-4147	use-after-free in net/netfilter/nf_tables_api.c	nf_tables	The private heap prevents the compartment from corrupting other kernel objects.
2022-24122	use-after-free in kernel/ucount.c	core kernel	
2022-27666	heap out-of-bounds write in net/ipv6/esp6.c	esp6	
2022-25636	heap out-of-bounds write in net/netfilter/nf_dup_netdev.c	nf_dup_netdev	
2021-22555	heap out-of-bounds write in net/netfilter/x_tables.c	x_tables	
2018-5703	heap out-of-bounds write in net/ipv6/tcp_ipv6.c	ipv6	
2023-0179	stack buffer overflow in net/netfilter/nft_payload.c	nf_tables	The private stack blocks cross-compartment stack corruption.
2018-13053	integer overflow in kernel/time/alarmtimer.c	core kernel	
2022-1015	improper input validation in net/netfilter/nf_tables_api.c	nf_tables	The monitor-enforced interface checks thwart confused deputy attacks.
2022-0492	missing authorization in kernel/cgroup/cgroup-v1.c	core kernel	
2017-18509	improper input validation in net/ipv6/ip6mr.c	ipv6	

Phoronix测试套件性能开销

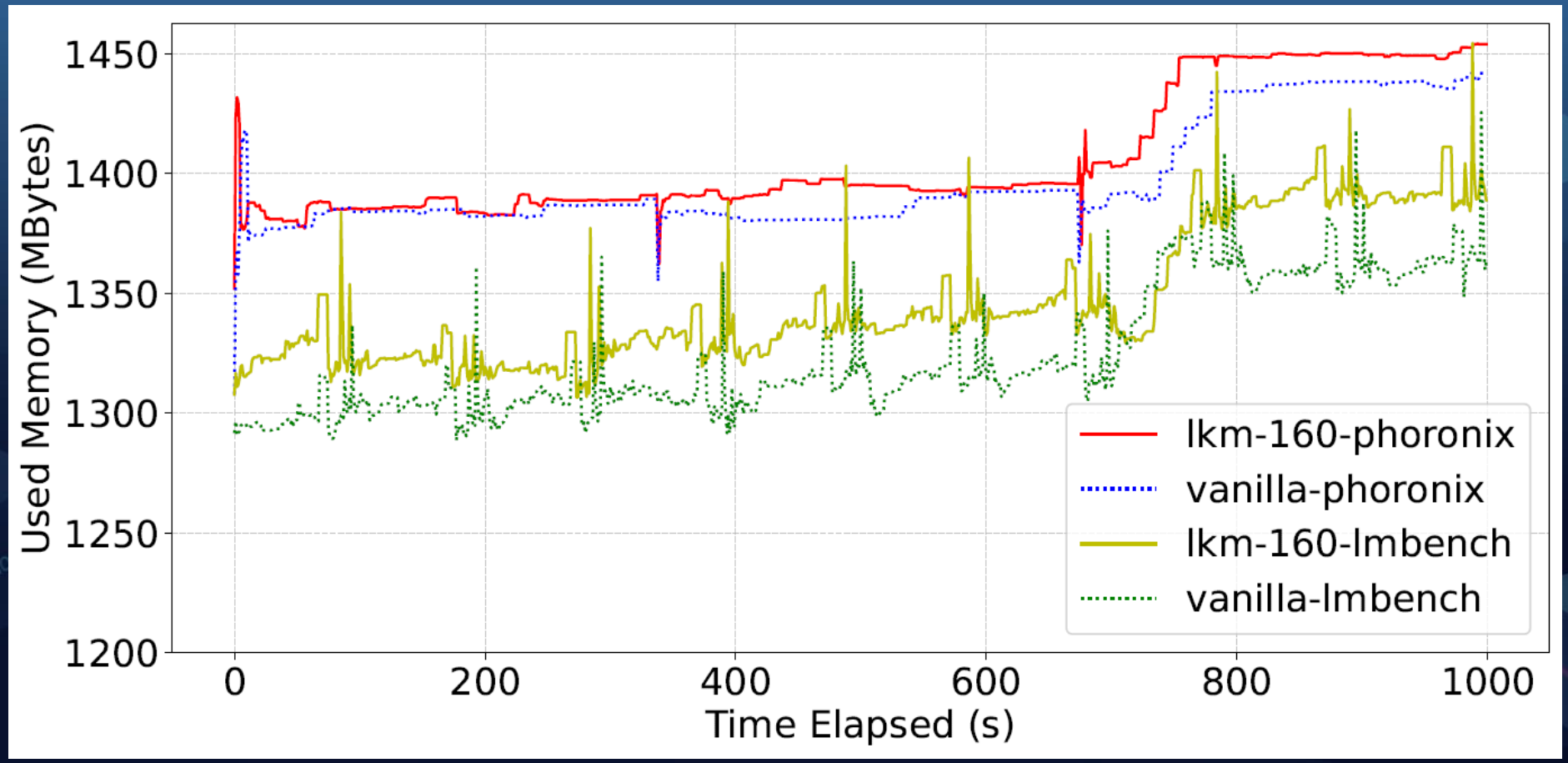
Benchmarks	monitor	ipv6	ipv6-nft	lkm-20	lkm-160
nginx-100	4.88	5.03	6.01	5.70 *(7)*	7.29 *(19)*
nginx-200	4.47	4.55	5.54	5.38 *(7)*	6.54 *(19)*
nginx-500	3.57	3.68	4.40	4.51 *(7)*	5.74 *(19)*
phpbench	-0.24	-0.12	-0.44	-0.28 *(7)*	0.33 *(18)*
pybench	0.35	0.17	0.43	0.52 *(7)*	1.37 *(18)*
povray	0.16	0.57	0.22	0.39 *(7)*	0.2 *(17)*
gnupg	0.10	0.01	0.35	0.08 *(7)*	1.03 *(18)*
dbench-1	0.19	0.20	0.19	0.04 *(7)*	0.47 *(19)*
dbench-48	0.52	1.05	1.73	3.74 *(7)*	5.61 *(19)*
dbench-256	0.22	1.22	2.38	1.64 *(7)*	2.11 *(19)*
postmark	1.84	0.00	1.14	1.14 *(7)*	0.39 *(18)*
sysbench-cpu	-0.05	-0.03	-0.04	-0.01 *(7)*	0.01 *(19)*
sysbench-mem	0.02	0.26	-0.53	0.53 *(7)*	0.69 *(18)*
Average	**1.23**	**1.28**	**1.64**	**1.80** *(7)*	**2.44** *(18.46)*

ApacheBench性能开销



内存开销

LMbench平均内存开销为1.66%，Phoronix平均内存开销为0.63%



目录

分隔化赋能系统安全

- TEE 分隔化：最小化可信计算基（TCB）
- 多语言系统分隔化：legacy module | rust module，unsafe part | safe part
- AI OS 分隔化：LLM / AI Agent sandbox

分隔化赋能系统安全

- 用什么机制作为隔板?
  - Tag-based硬件机制，如Intel PKS，ARM MTE
- 将隔板放置在哪里?
  - 模块边界分析
- 如何放置隔板?
  - 安全高效的switch gate

Thank You!
Q & A