BULKHEAD: Secure, Scalable, and Efficient Kernel Compartmentalization with PKS

Yinggang Guo[1,2], Zicheng Wang[1], **Weiheng Bai[2]**, Qingkai Zeng[1], Kangjie Lu[2]

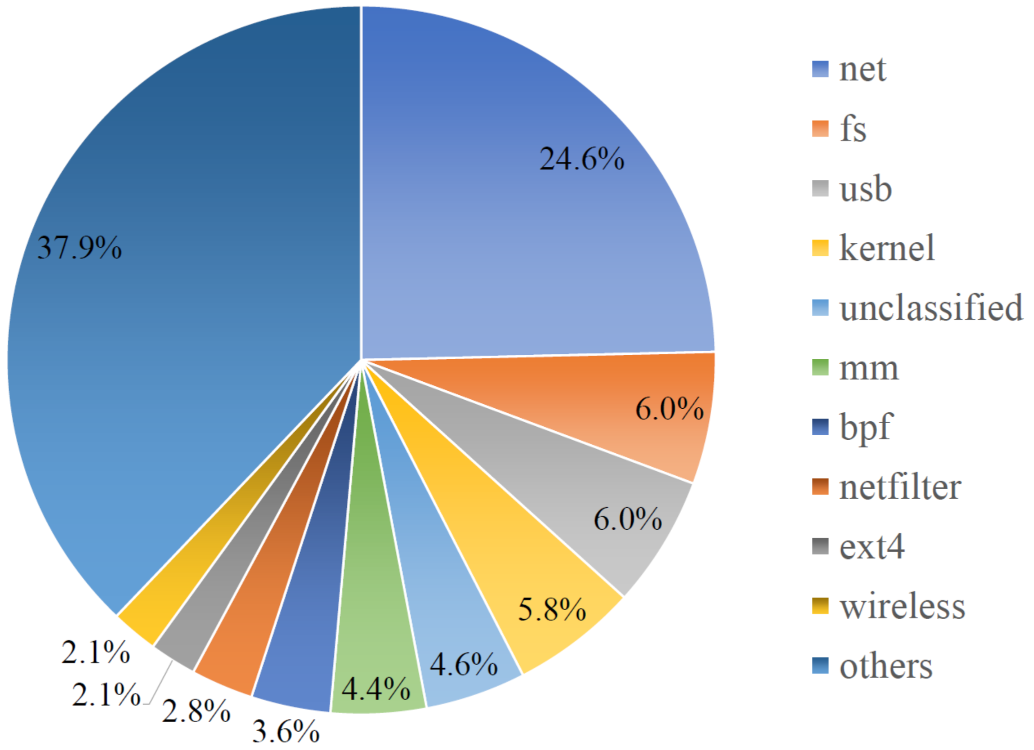
NANJING UNIVERSITY

UNIVERSITY OF MINNESOTA

NDSS SYMPOSIUM

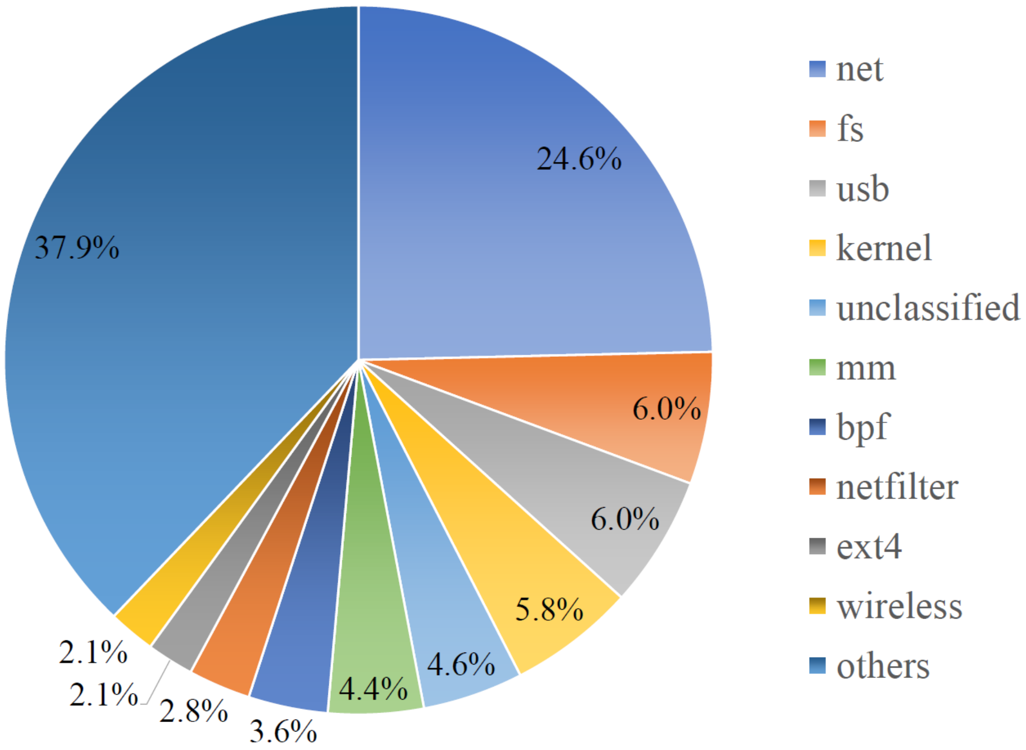
Background

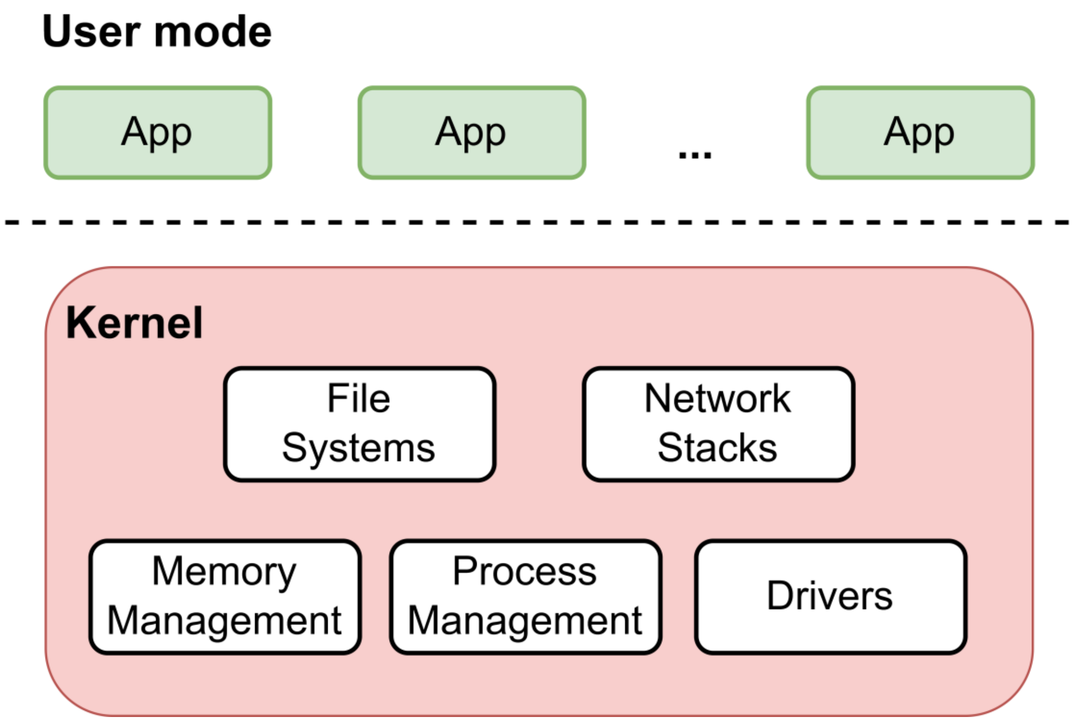
- OS kernel faces a continual influx of vulnerabilities.



Background

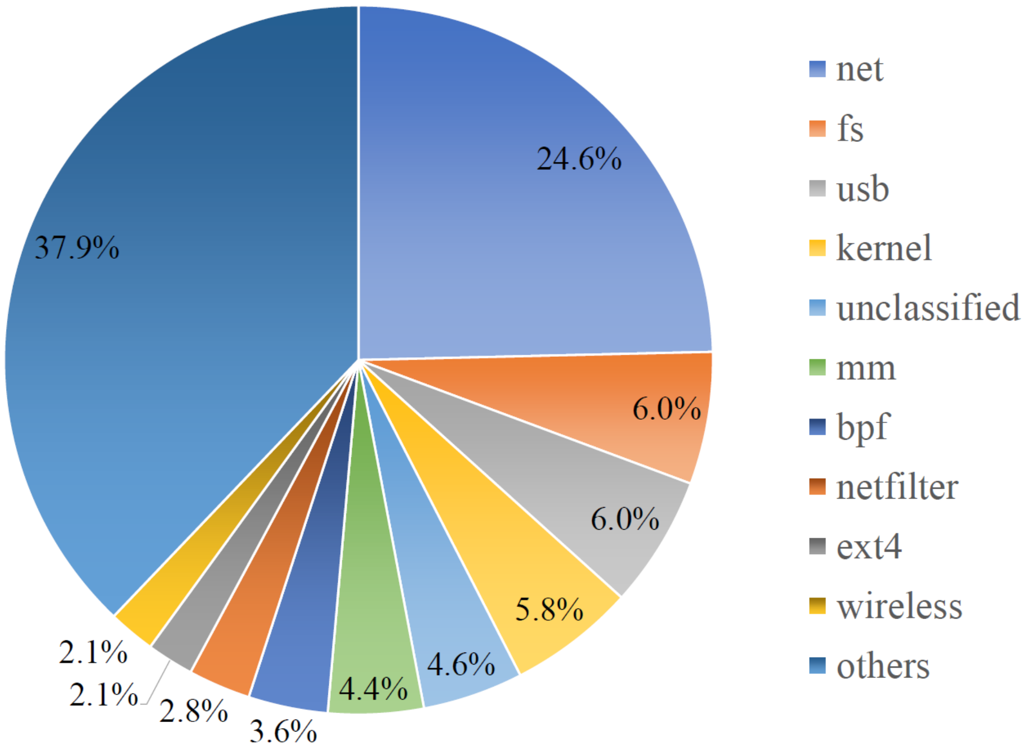
- OS kernel faces a continual influx of vulnerabilities.
- The monolithic architecture shares privileges between modules.

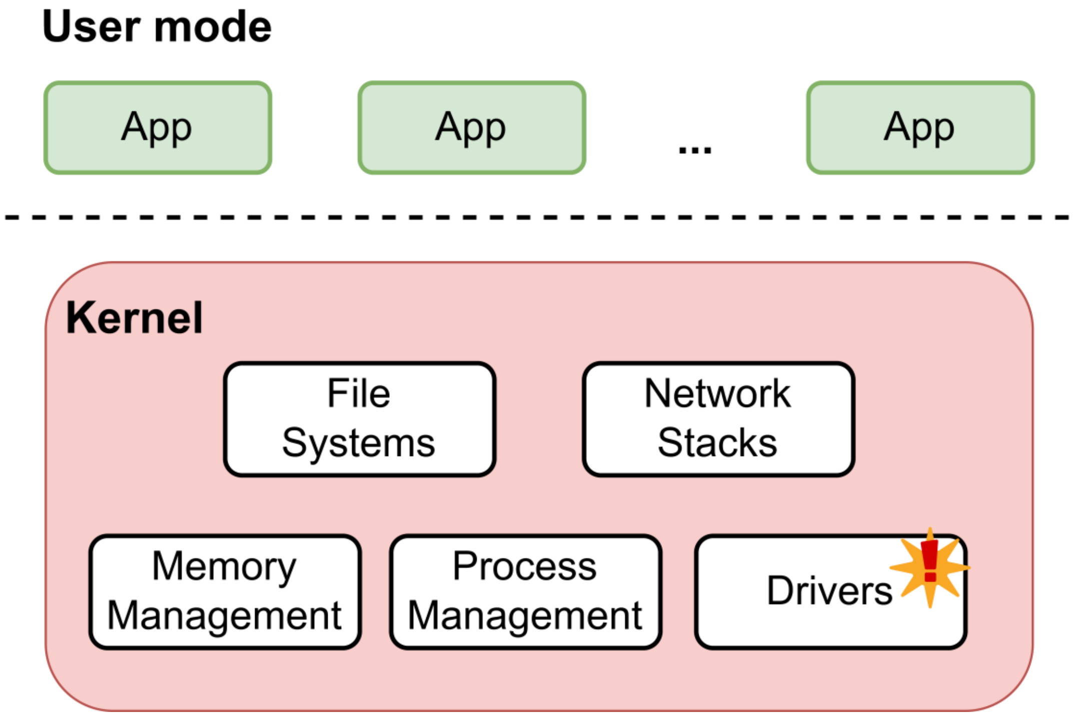




Background

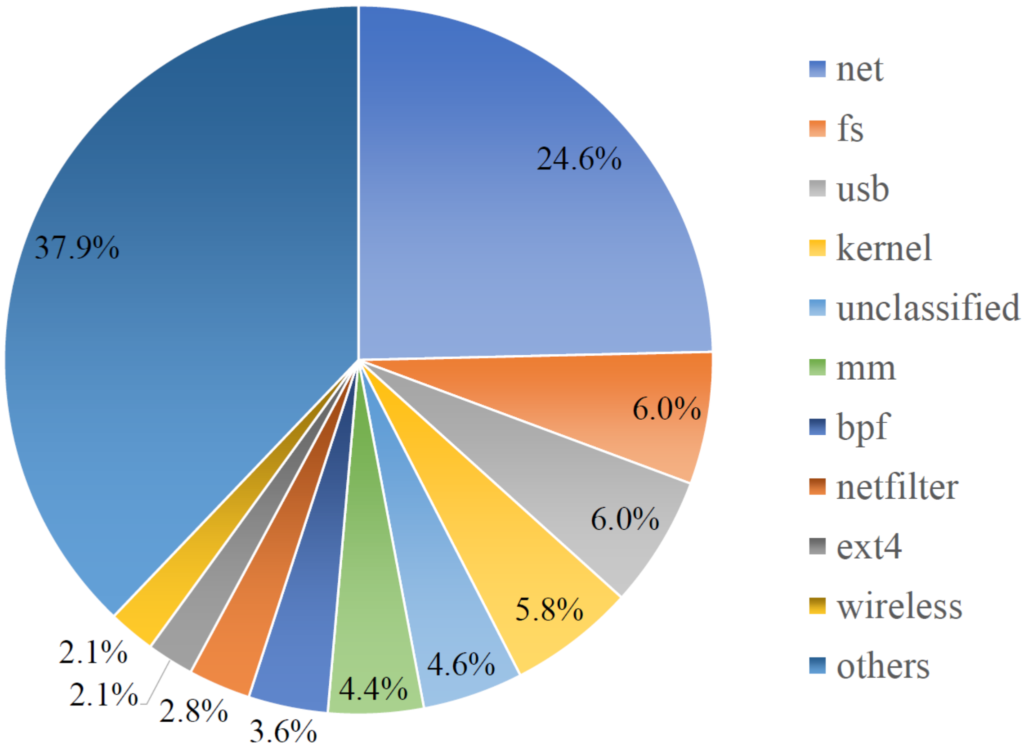
- OS kernel faces a continual influx of vulnerabilities.
- The monolithic architecture shares privileges between modules.

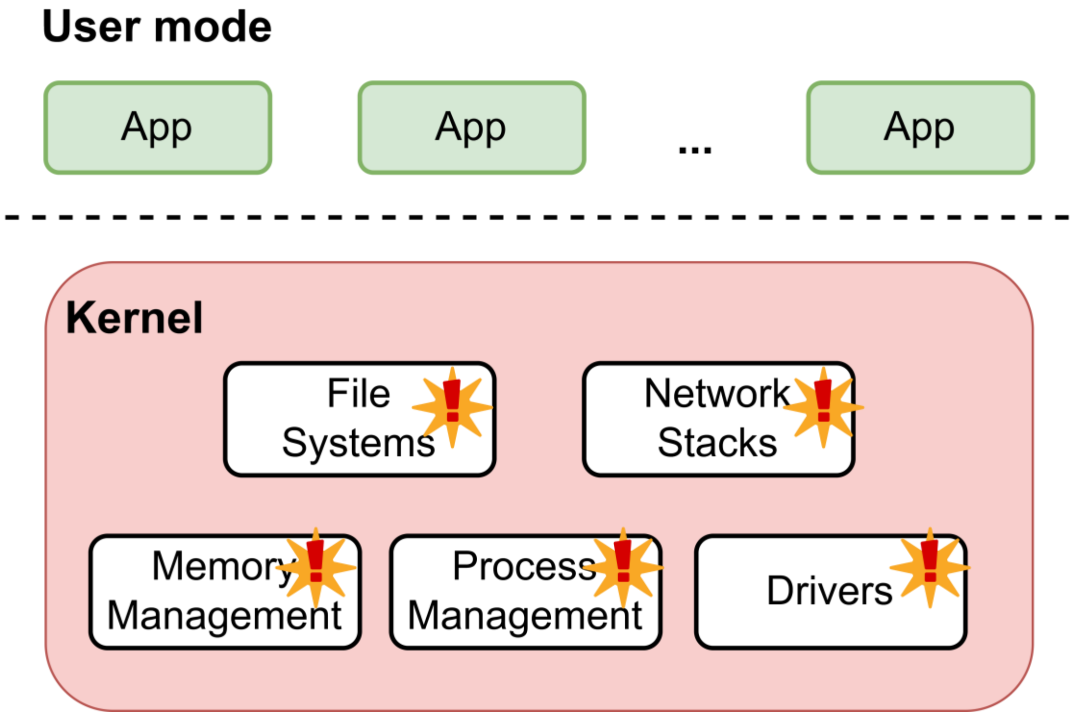




Background

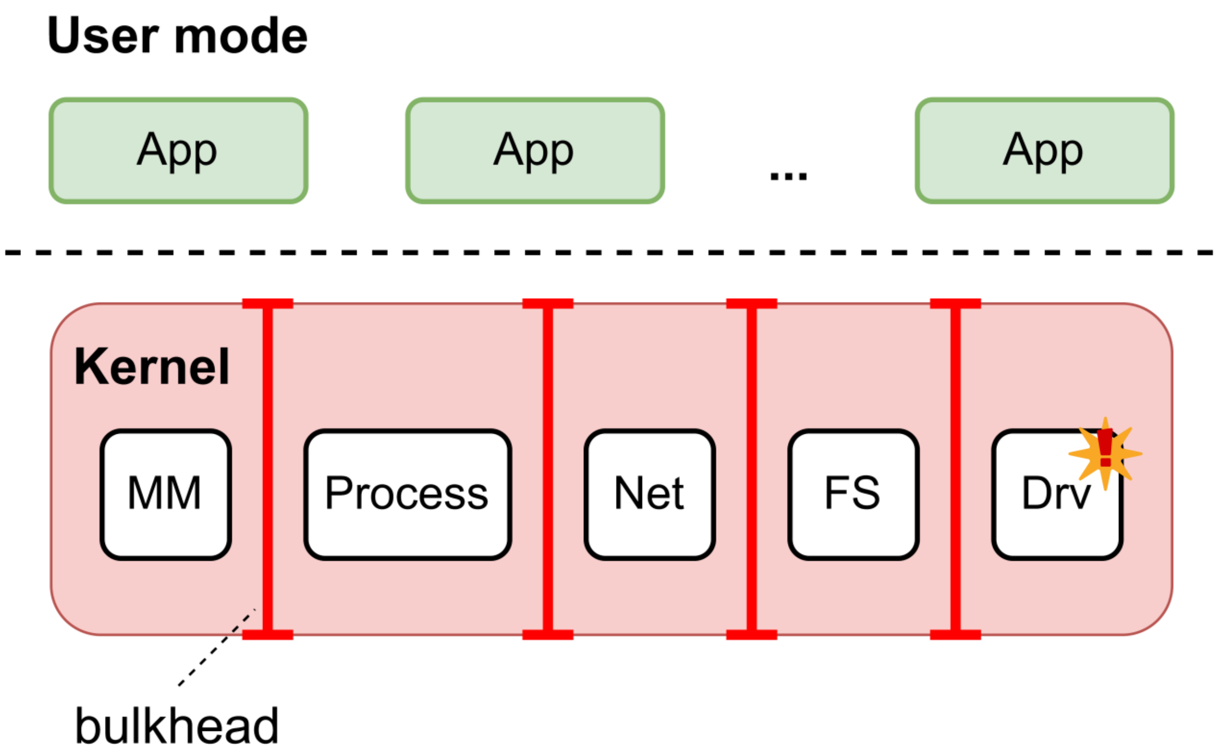
- OS kernel faces a continual influx of vulnerabilities.
- The monolithic architecture shares privileges between modules.

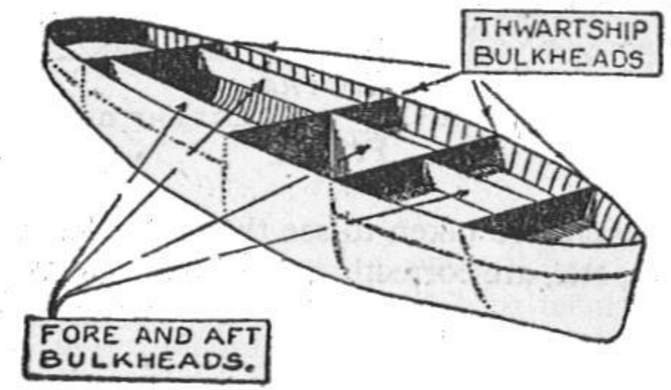




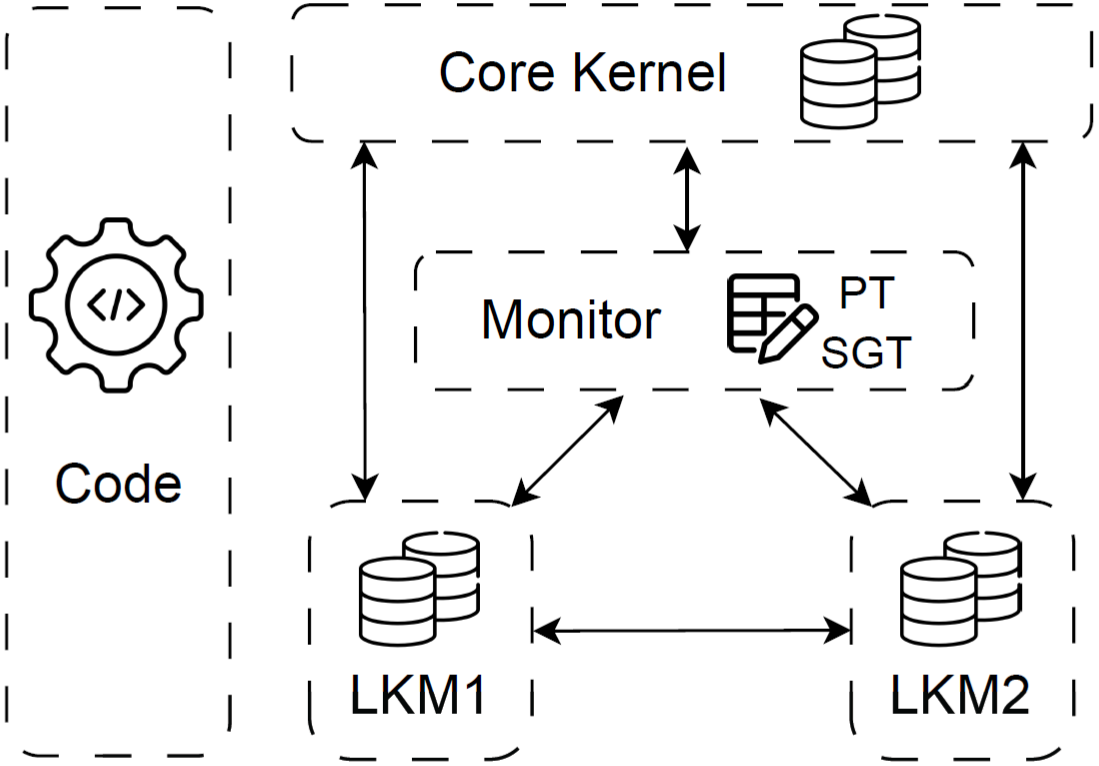
Background

- OS kernel faces a continual influx of vulnerabilities.
- The monolithic architecture shares privileges between modules.
- Kernel compartmentalization is promising to confine the effect of exploitation.





Overview



PKRS

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

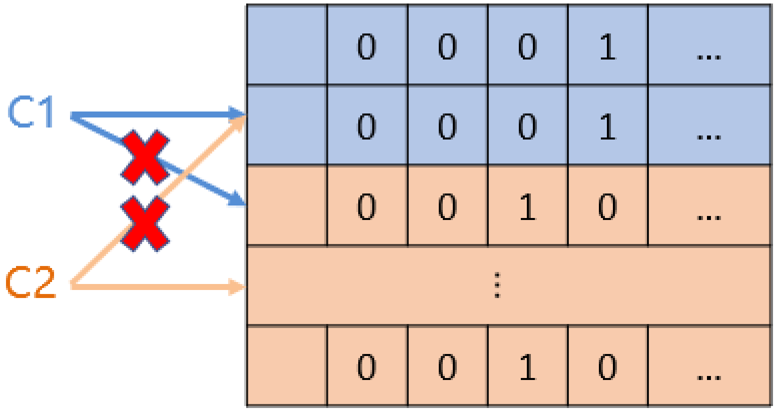
Challenges: mutual untrusted, privileged, numerous and complex compartments

Objectives

	Mechanisms	Security				Scalability	Performance		Compatibility
		bi-directional isolation	data protection	control flow protection	interface protection	domain number	domain switch	data transfer	
seL4 [37]	Microkernel	No	Yes	Yes	No	Unlimited	Low	Low	Heavy redesign
UnderBridge [27]	Microkernel+PKU	No	Yes	Yes	No	16	High	High	Heavy redesign
LXFI [57]	SFI	No	Yes	Yes	Yes	Unlimited	Low	Low	Annotations
LVD [65]	Virtualization	No	Yes	Yes	No	512	High	Low	Nested Virtualization
KSplit [33]	Virtualization	No	Yes	Yes	No	512	High	Low	Nested Virtualization
xMP [71]	Virtualization	No	Yes	No	No	512	High	Low	Nested Virtualization
Nested Kernel [15]	WP bit	No	Yes	Yes	No	2	High	High	x86-64
SKEE [1]	PT switching	No	Yes	Yes	No	2	Medium	Low	ARM
IskiOS [25]	PKU	No	No	Yes	No	8	High	High	SMAP/SMEP
HAKC [58]	MTE+PA	No	Yes	Yes	No	Unlimited	Medium	Medium	ARM
CHERI [93]	New architecture	No	Yes	Yes	Yes	Unlimited	Medium	Medium	New architecture
SecureCells [4]	New architecture	No	Yes	Yes	Yes	Unlimited	High	High	New architecture
DOPE [55]	PKS	No	Yes	No	No	16	High	High	Intel
BULKHEAD	**PKS**	**Yes**	**Yes**	**Yes**	**Yes**	**unlimited**	**High**	**High**	**Intel**

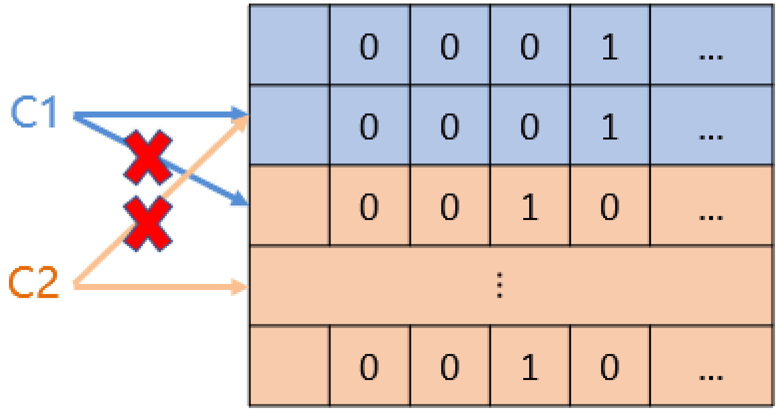
Design

- Security
  - Bi-directional isolation ——> *In-kernel monitor*
    - ✓ Memory isolation



Design

- Security
  - ➢ Bi-directional isolation ——> *In-kernel monitor*
    - ✓ Memory isolation
    - ✓ Instruction deprivation




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

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

(a) nop insertion

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

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

(b) Register reassignment

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

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

(c) Data adjustment

Fig. 5: Some examples of eliminating unintended `wrmsr` (`0x0f30`).

Design

- Security
  - Bi-directional isolation ——> *In-kernel monitor*
    - ✓ Memory isolation
    - ✓ Instruction deprivation
  - Data protection ——> *Data integrity*
    - ✓ Write-protected page tables
    - ✓ Private heap

Design

- Security
  - Bi-directional isolation ——> *In-kernel monitor*
    - ✓ Memory isolation
    - ✓ Instruction deprivation
  - Data protection ——> *Data integrity*
    - ✓ Write-protected page tables
    - ✓ Private heap
  - Control flow protection ——> *Execute-only memory*

Design

- Security
  - Bi-directional isolation ——> *In-kernel monitor*
    - ✓ Memory isolation
    - ✓ Instruction deprivation
  - Data protection ——> *Data integrity*
    - ✓ Write-protected page tables
    - ✓ Private heap
  - Control flow protection ——> *Execute-only memory*
  - Compartment interface protection ——> *Compartment interface integrity*

Compartment Interface Integrity

- Compartment switches must occur at the predefined entry/exit points and pass data according to security policies.

```
get_metadata(gate_id);
verify(source_addr);
if (target_pgdir != source_pgdir)
    load_new_mm_cr3(target_pgdir, target_asid);
if (target_pkrs != current_pkrs)
loop:
    write_pkrs(target_pkrs);
if (current_pkrs != target_pkrs)
    goto loop;
switch_stack(target_stack);
jump(target_addr);
```

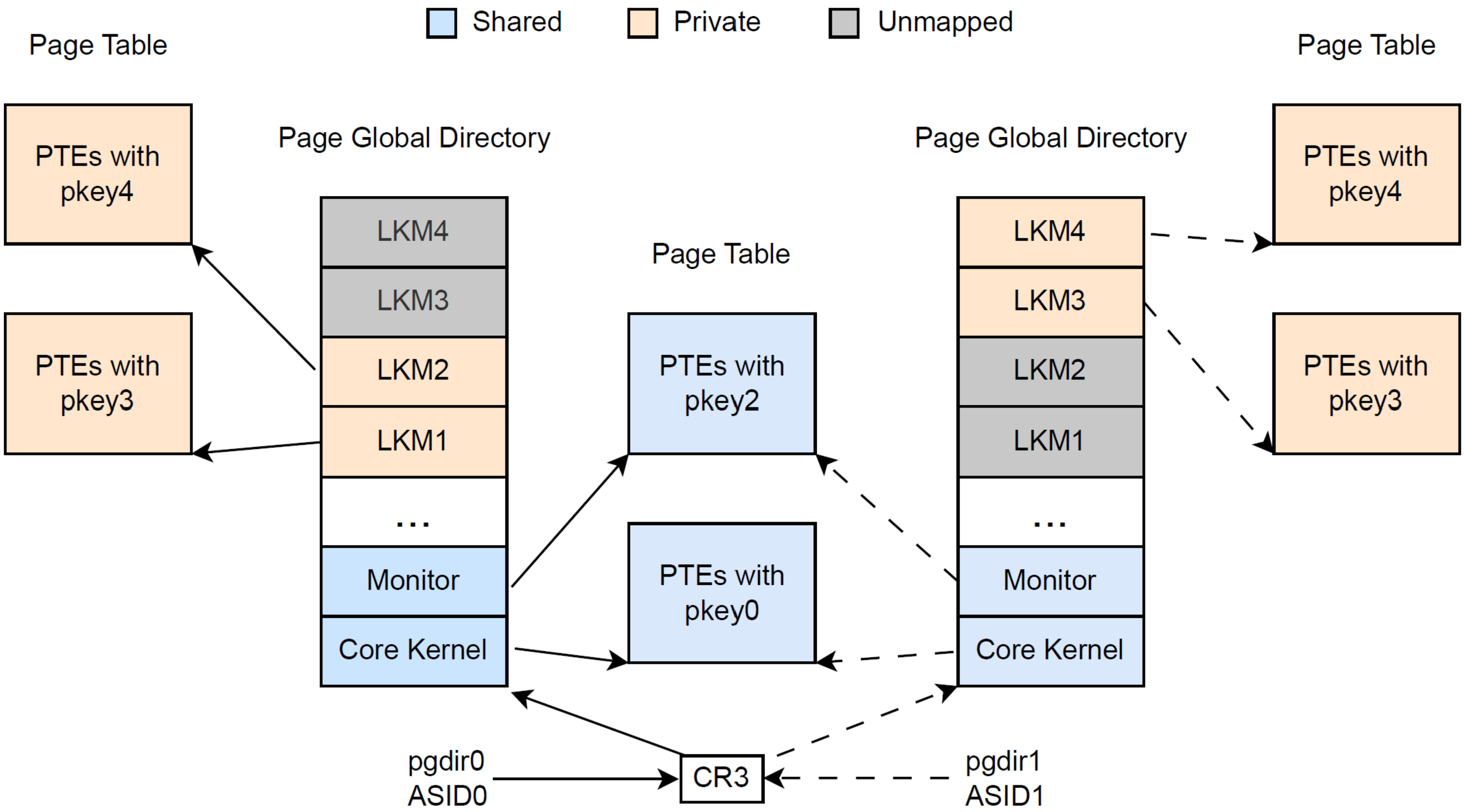
gate id	
source	*target*
address	
pgdir	
asid	
pkrs	
stack	

Design

- Scalability
  - ➢ Support for unlimited compartments ——> *Two-level compartmentalization*
    - ✓ PKS-based intra-address space isolation
    - ✓ locality-aware inter-address space isolation with ASID

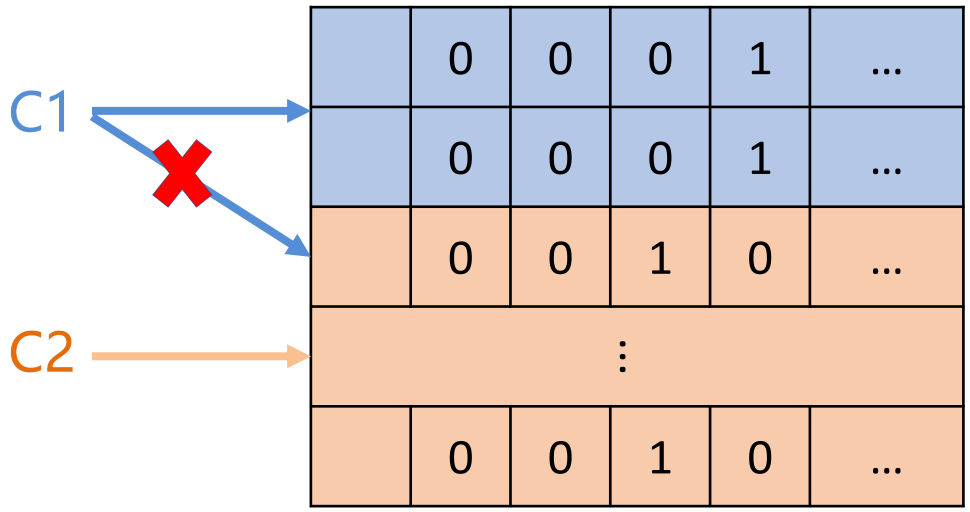
Locality-aware Two-level Compartmentalization

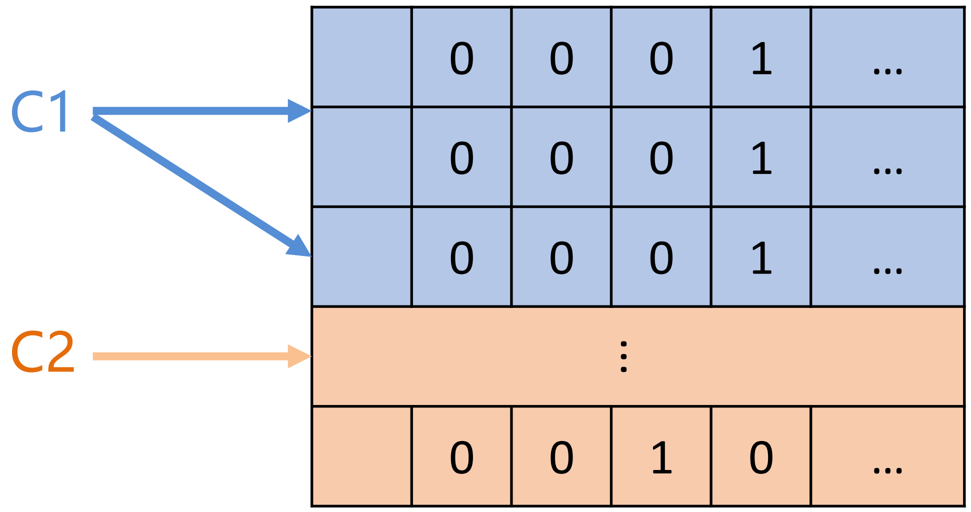
- PKS-based intra-AS isolation + locality-aware AS switching with ASID



Design

- Performance
  - Fast compartment switches ——> *PKRS updates*
  - Zero-copy data transfer ——> *Ownership transfer*





Security Analysis

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	

TABLE II: Representative Linux kernel CVEs, their root causes, the located compartment, and the countermeasures of BULKHEAD.

Performance Evaluation

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)***

TABLE V: BULKHEAD performance overhead (in % over the vanilla kernel) on Phoronix Test Suites. The numbers in parentheses represent the number of compartments traversed for each benchmark.

Performance Evaluation

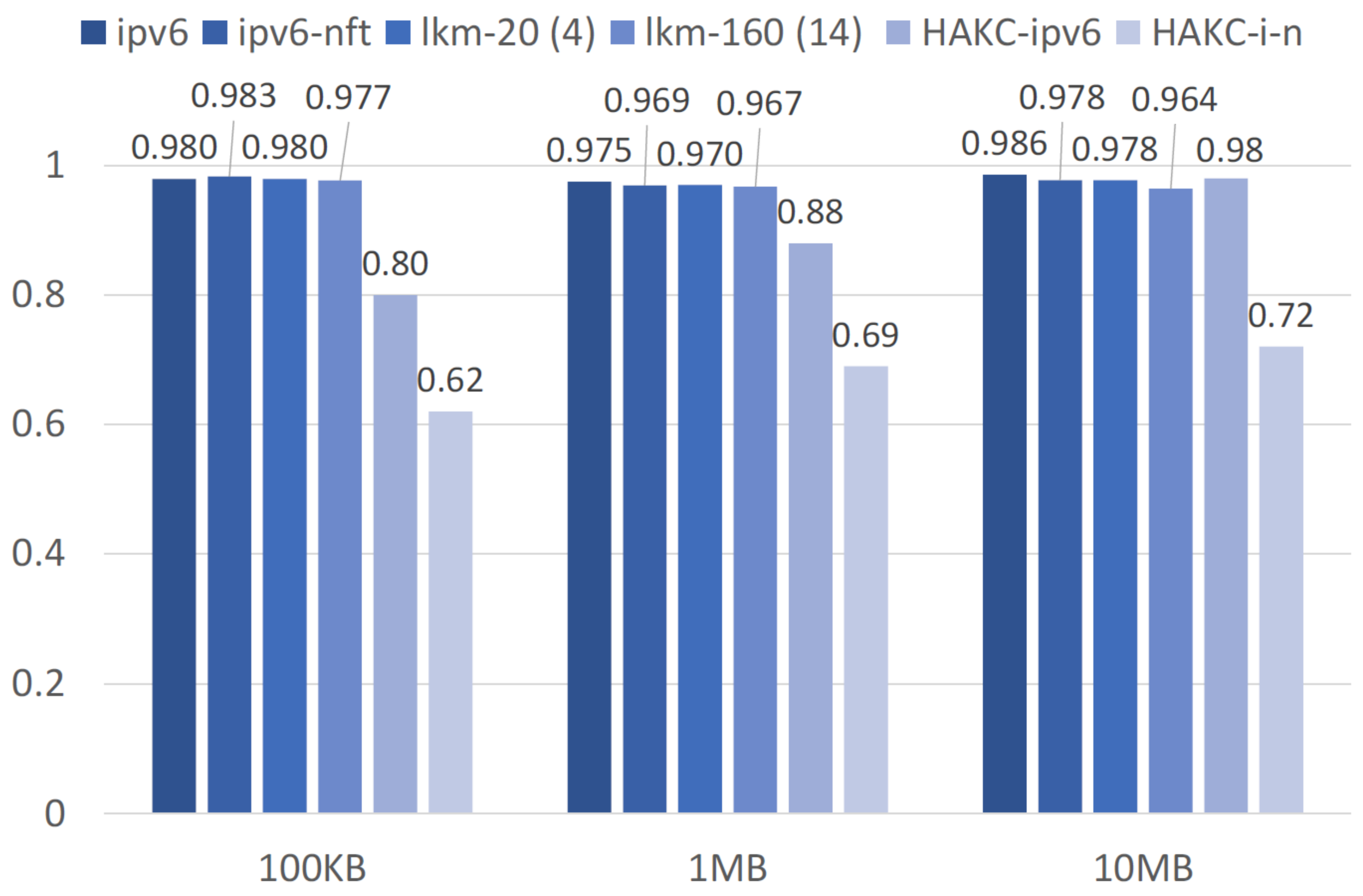


Fig. 9: BULKHEAD performance overhead normalized to the vanilla kernel when transferring various sized payloads on ApacheBench (requests/sec), compared with the overhead of HAKC [58].

Memory Overhead

- On average, the memory overhead is 1.66% for LMbench and 0.63% for Phoronix.

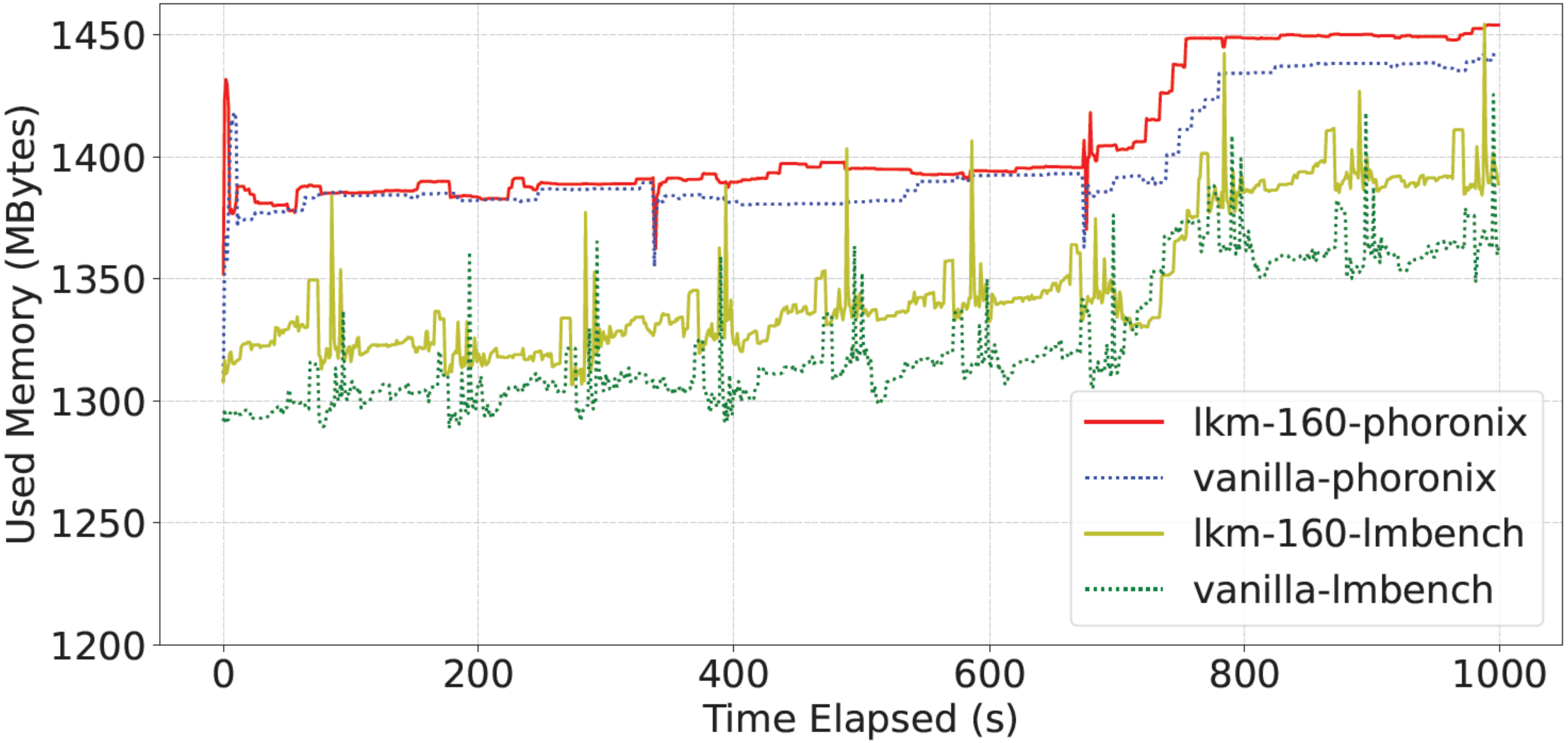


Fig. 10: Memory usage of BULKHEAD when running LMbench and Phoronix with *lkm-160* and the vanilla kernel.

Conclusion

- What to use as the bulkhead ?
    - PKS-based bi-directional isolation
- Where to put the bulkhead ?
    - LLVM-based boundary analysis
- How to set up the bulkhead ?
    - Secure and efficient switch gates
- Compartmentalization for other systems
    - TEE, multi-language systems, LLM systems…

Thank You!

Q & A