Cross-Cache Attacks for the Linux Kernel via PCP Massaging

Claudio Migliorelli[1,2], Andrea Mambretti[1], Alessandro Sorniotti[1], Vittorio Zaccaria[3], Anil Kurmus[1]

[1]IBM Research Europe — Zurich

[2]EPFL

[3]Politecnico di Milano

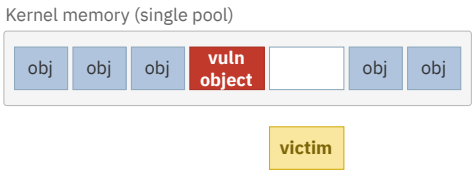
Exploitation in the Linux Kernel

The Goal — what the attacker wants

Kernel memory (single pool)

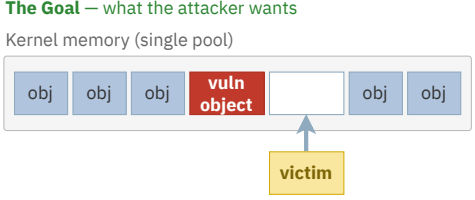
Exploitation in the Linux Kernel

The Goal — what the attacker wants



Exploitation in the Linux Kernel

The Goal — what the attacker wants



Exploitation in the Linux Kernel

The Goal — what the attacker wants

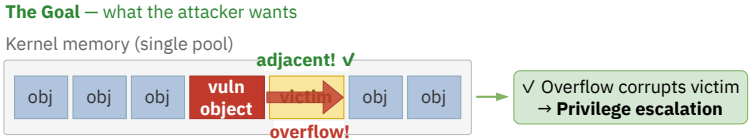
Kernel memory (single pool)

Exploitation in the Linux Kernel

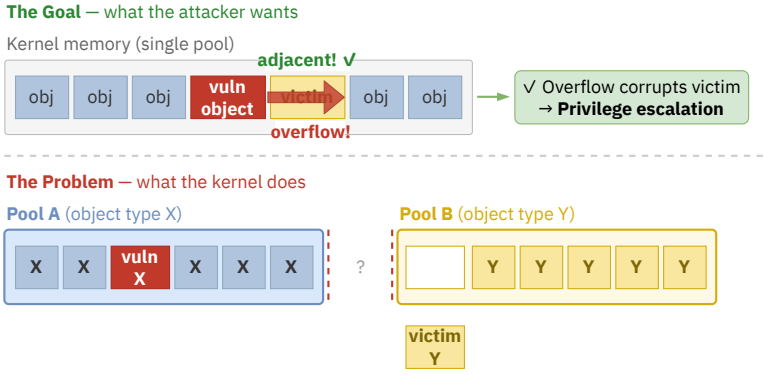
The Goal — what the attacker wants

Exploitation in the Linux Kernel

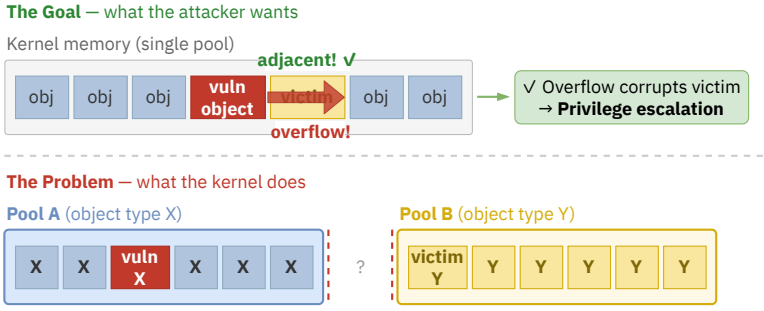
The Goal — what the attacker wants



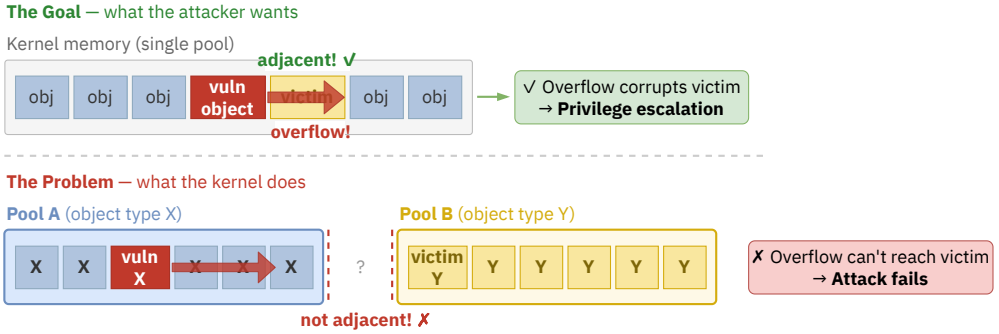
Exploitation in the Linux Kernel



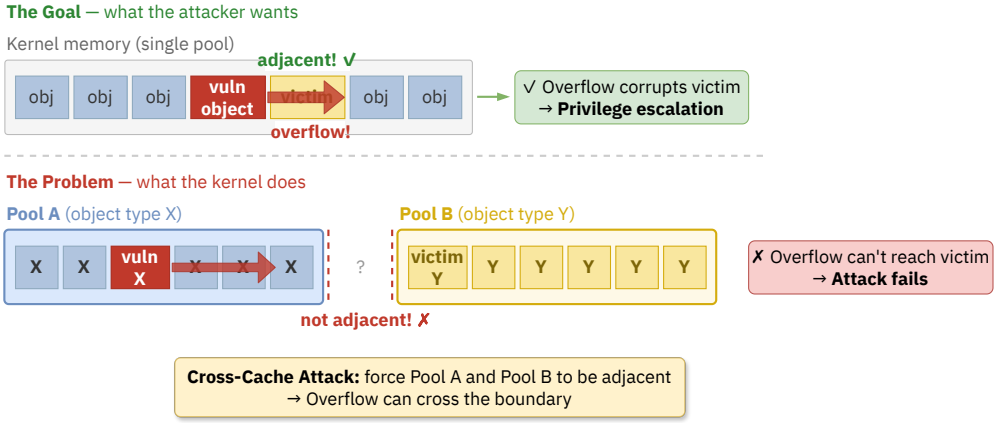
Exploitation in the Linux Kernel



Exploitation in the Linux Kernel



Exploitation in the Linux Kernel



Previous Work

- Timing side-channels on SLUB: **PSPRAY** (USENIX 23)
- Cross-cache attacks for temporal vulnerabilities (page-recycling): **SLUBStick** (USENIX 24), **CROSS-X** (CCS 25)

Previous Work

- Timing side-channels on SLUB: **PSPRAY** (USENIX 23)
- Cross-cache attacks for temporal vulnerabilities (page-recycling): **SLUBStick** (USENIX 24), **CROSS-X** (CCS 25)

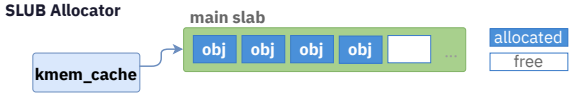
Previous work either focuses on SLUB or page-recycling cross-cache

PCPLost

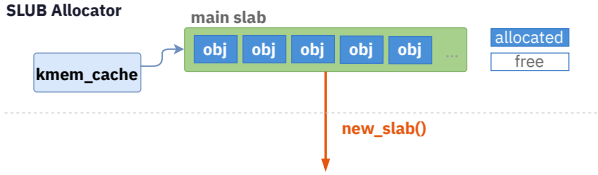
This work:

- PCPLost is a cross-cache memory massaging technique for spatial vulnerabilities (page-adjacency)
- Exploits overlooked interactions between SLUB and Page Frame Allocator
- $\sim \mathbf{90}\%$ reliability in establishing adjacency, even under noise and mitigations

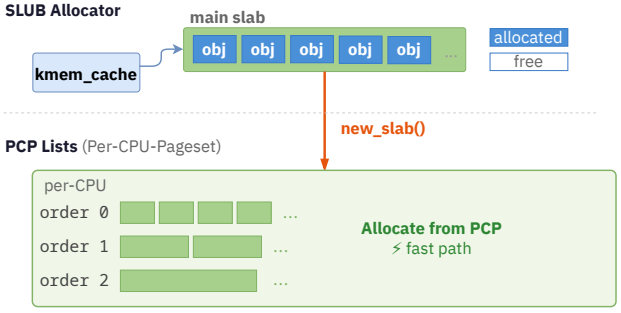
SLUB and Page Frame Allocator (PFA)



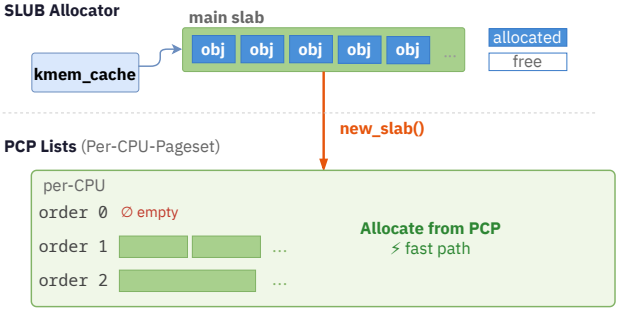
SLUB and Page Frame Allocator (PFA)



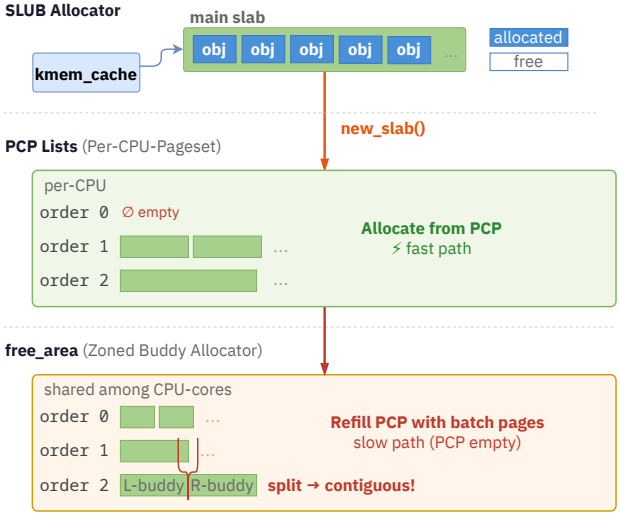
SLUB and Page Frame Allocator (PFA)



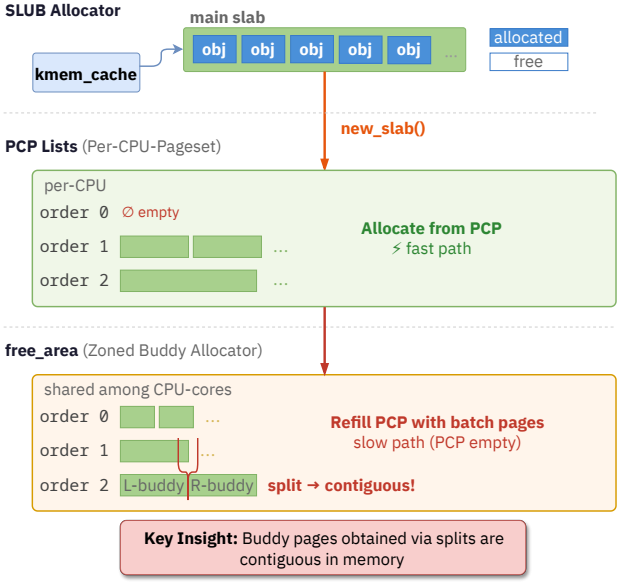
SLUB and Page Frame Allocator (PFA)



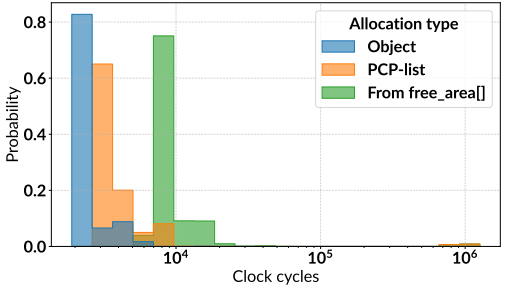
SLUB and Page Frame Allocator (PFA)



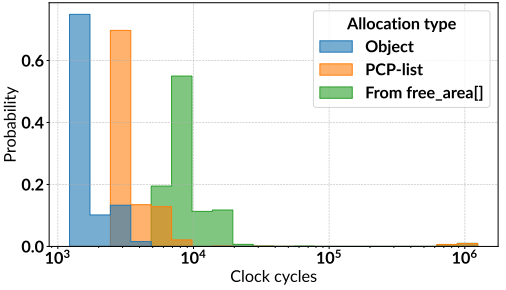
SLUB and Page Frame Allocator (PFA)



Timing side-channel on SLUB/PFA

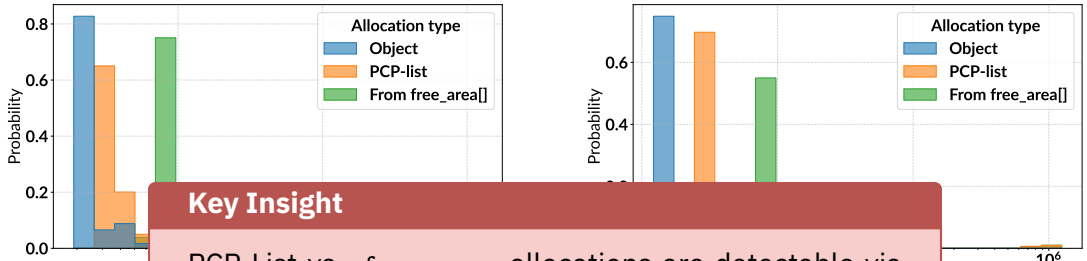


(a) `kmalloc-4k`



(b) `kmalloc-2k`

Timing side-channel on SLUB/PFA



Key Insight

PCP List vs. `free_area` allocations are detectable via timing side-channel

PCPLost attack core steps

1. Repeatedly drain the PCP list for the target order (Probe & Drain)
    - Forces PFA page splits to create contiguous layouts in the PCP list

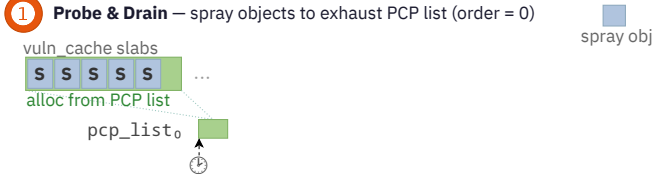
PCPLost attack core steps

1. Repeatedly drain the PCP list for the target order (Probe & Drain)
    - Forces PFA page splits to create contiguous layouts in the PCP list
2. Allocate vulnerable and target cache from the PCP list

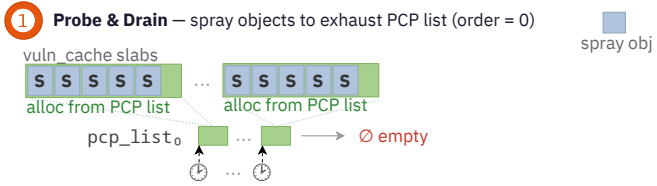
PCPLost attack core steps

1. Repeatedly drain the PCP list for the target order (Probe & Drain)
    - Forces PFA page splits to create contiguous layouts in the PCP list
2. Allocate vulnerable and target cache from the PCP list
3. Trigger OOB write

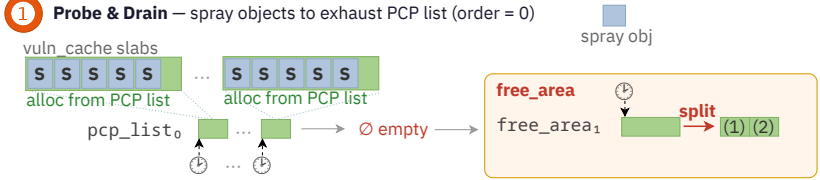
Same-order PCP massaging



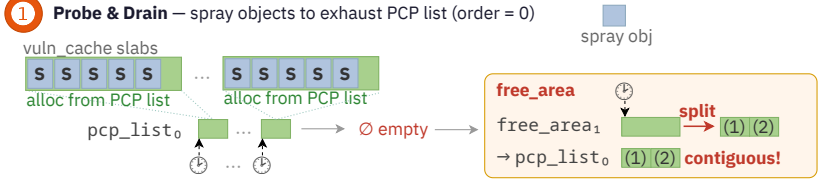
Same-order PCP massaging



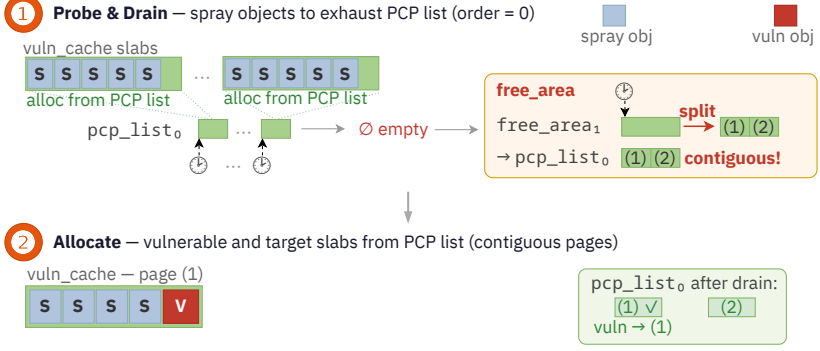
Same-order PCP massaging



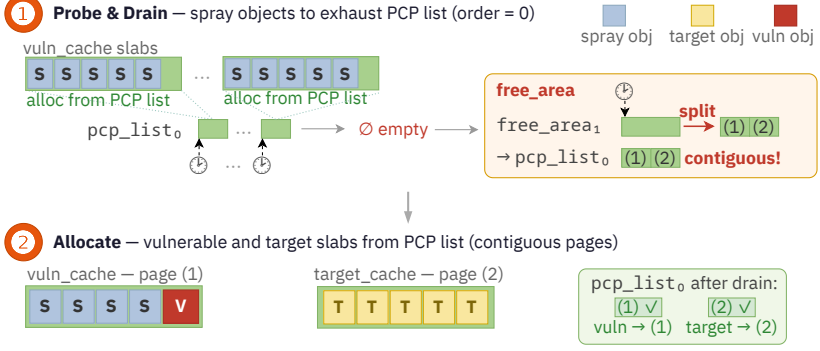
Same-order PCP massaging



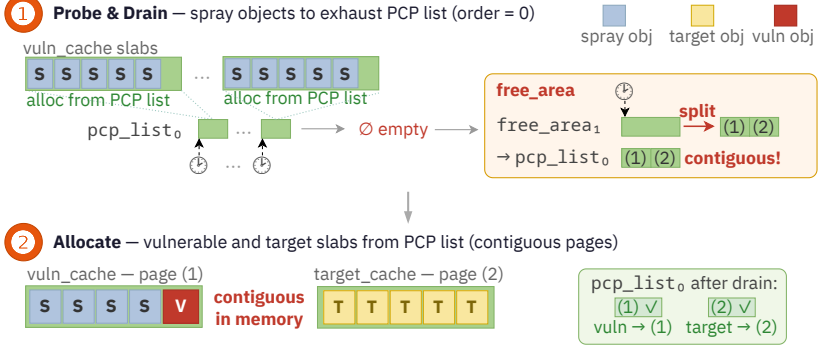
Same-order PCP massaging



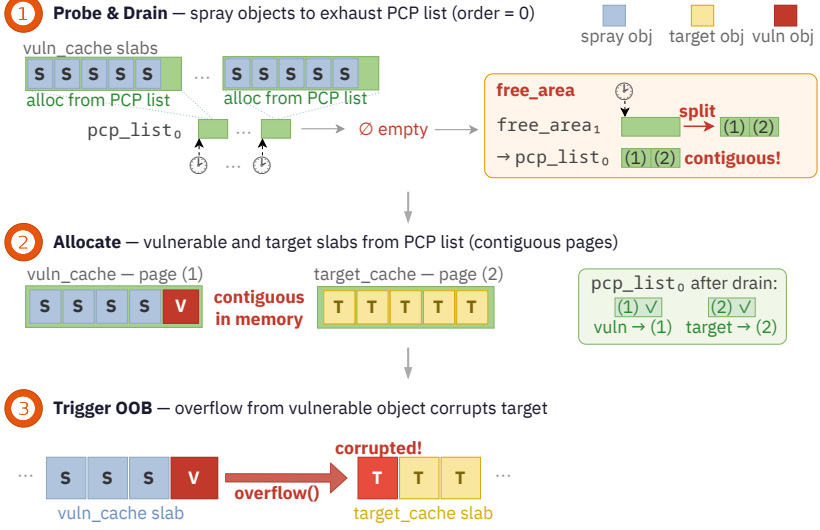
Same-order PCP massaging



Same-order PCP massaging



Same-order PCP massaging



Evaluation

Reliability evaluation

Vuln. cache	Target cache	Idle (%)	Noise (%)
Same-order ($n_v = n_t$)			
`kmalloc-16`	`kmalloc-64`	98.83	97.42
`kmalloc-64`	`kmalloc-128`	96.80	90.94
`kmalloc-128`	`kmalloc-16`	99.84	98.28
Cross-order ($n_v > n_t$)			
`kmalloc-2k`	`kmalloc-16`	94.30	87.58
`kmalloc-4k`	`kmalloc-64`	93.20	92.42
`kmalloc-4k`	`kmalloc-128`	90.31	90.08

With CPU pinning. 800 runs per scenario.

Evaluation

Reliability evaluation

Vuln. cache	Target cache	Idle (%)	Noise (%)
Same-order ($n_v = n_t$)			
`kmalloc-16`	`kmalloc-64`	98.83	97.42
`kmalloc-64`	`kmalloc-128`	96.80	90.94
`kmalloc-128`	`kmalloc-16`	99.84	98.28
Cross-order ($n_v > n_t$)			
`kmalloc-2k`	`kmalloc-16`	94.30	87.58
`kmalloc-4k`	`kmalloc-64`	93.20	92.42
`kmalloc-4k`	`kmalloc-128`	90.31	90.08

With CPU pinning. 800 runs per scenario.

Real-world CVEs

- PCPLost validated on **6 CVEs**
  - **Spatial** bugs
  - **Temporal** bugs via pivoting
- Bypassing mainline mitigations
  `SLAB_FREELIST_RANDOM`
  `SLAB_FREELIST_HARDENED`
- Bypassing experimental `SLAB_VIRTUAL`

Evaluation

Reliability evaluation

Vuln. cache	Target cache	Idle (%)	Noise (%)
Same-order ($n_v = n_t$)			
`kmalloc-16`	`kmalloc-64`	98.83	97.42
`kmalloc-64`	`kmalloc-128`	96.80	90.94
`kmalloc-128`	`kmalloc-16`	99.84	98.28
Cross-order ($n_v > n_t$)			
`kmalloc-2k`	`kmalloc-16`	94.30	87.58
`kmalloc-4k`	`kmalloc-64`	93.20	92.42
`kmalloc-4k`	`kmalloc-128`	90.31	90.08

With CPU pinning. 800 runs per scenario.

Real-world CVEs

- PCPLost validated on **6 CVEs**
  - **Spatial** bugs
  - **Temporal** bugs via pivoting
- Bypassing mainline mitigations
  `SLAB_FREELIST_RANDOM`
  `SLAB_FREELIST_HARDENED`
- Bypassing experimental `SLAB_VIRTUAL`
  Mitigated by `SLAB_VIRTUAL_GP` (with guard pages)

Conclusion

- **PCP list internals** expose a **timing side-channel**

Conclusion

- **PCP list internals** expose a **timing side-channel**
- We use the side-channel to mount PCPLost exploiting **SLUB/PFA interactions**

Conclusion

- **PCP list internals** expose a **timing side-channel**
- We use the side-channel to mount PCPLost exploiting **SLUB/PFA interactions**
- Mainline kernel defenses stop at SLUB
  - **The Page Allocator remains unguarded**
  - Cross-cache security requires mitigations at that level

Conclusion

- **PCP list internals** expose a **timing side-channel**
- We use the side-channel to mount PCPLost exploiting **SLUB/PFA interactions**
- Mainline kernel defenses stop at SLUB
  - **The Page Allocator remains unguarded**
  - Cross-cache security requires mitigations at that level
- We disclosed PCPLost to kernel maintainers

Evaluation with real-world CVEs

CVE	Category	Vulnerable cache	Target cache	Mitgations bypassed
CVE-2024-53141	OOB	`kmalloc-cg-1k`	`kmalloc-cg-2k`	SV,FH
CVE-2021-22555	OOB/UAF	`kmalloc-cg-4k`	`kmalloc-cg-1k`	FR,SV,FH
CVE-2022-0185	OOB	`kmalloc-4k`	`kmalloc-cg-4k`	SV,FH
CVE-2023-0461	UAF	`kmalloc-512`	`kmalloc-cg-1k`	FR,SV,FH
CVE-2021-3715	UAF	`kmalloc-192`	`kmalloc-32`	FR,FH
CVE-2022-27666	OOB	–	`kmalloc-4k`	FR,FH

SLAB_FREELIST_RANDOM bypass