

SLUBStick



Arbitrary Memory Writes through Practical Software Cross-Cache Attacks within the Linux Kernel

Lukas Maar, Stefan Gast, Martin Unterguggenberger, Mathias Oberhuber, and Stefan Mangard

August 15, 2024

Graz University of Technology

Kernel exploit technique **SLUBStick**



Kernel exploit technique **SLUBStick**

- Timing side channel:
 - Make software **cross-cache reuse practical**



Kernel exploit technique **SLUBStick**

- Timing side channel:
 - Make software **cross-cache reuse practical**
- Primitive conversion:
 - Limited heap write to a **UAF write**



Kernel exploit technique **SLUBStick**

- Timing side channel:
 - Make software **cross-cache reuse practical**
- Primitive conversion:
 - Limited heap write to a **UAF write**
- Page-table manipulation:
 - Obtain an **arbitrary physical r/w primitive**



Kernel exploit technique **SLUBStick**

- Timing side channel:
 - ⚡ Make software **cross-cache reuse practical**
- Primitive conversion:
 - ⚡ Limited heap write to a **UAF write**
- Page-table manipulation:
 - ⚡ Obtain an **arbitrary physical r/w primitive**
- Implement 9 PoC exploits



Kernel exploit technique **SLUBStick**

- Timing side channel:
 - 🛠️ Make software **cross-cache reuse practical**
- Primitive conversion:
 - 🛠️ Limited heap write to a **UAF write**
- Page-table manipulation:
 - 🛠️ Obtain an **arbitrary physical r/w primitive**
- Implement 9 PoC exploits
- Opensource our artifacts on:
 - 🌐 <https://github.com/IAIK/SLUBStick>





Lukas Maar

PhD candidate @ Graz University of Technology

Improve system security

 <https://lukasmaar.github.io>

 lukas.maar@iaik.tugraz.at

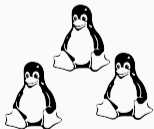
Background

Kernel

Applications

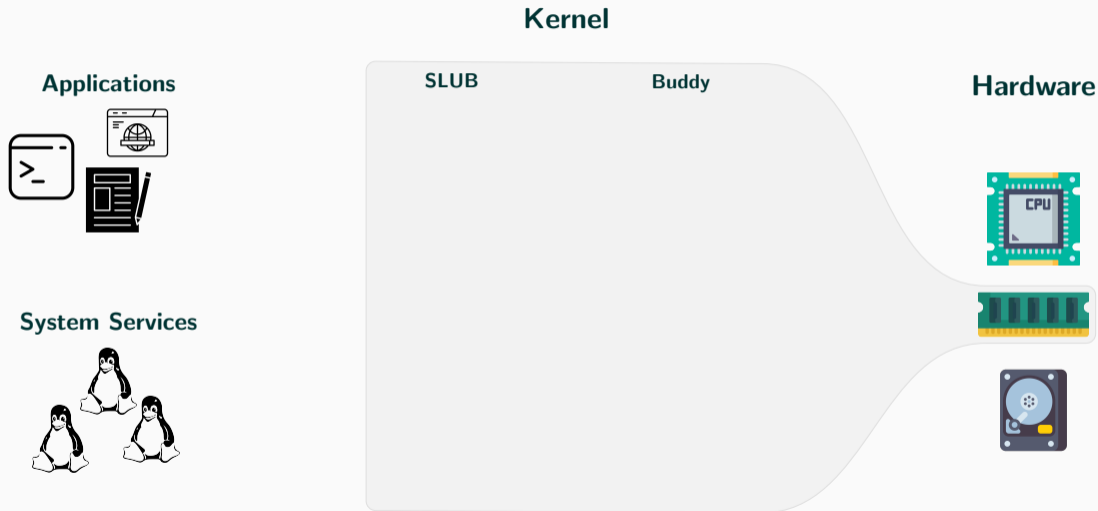


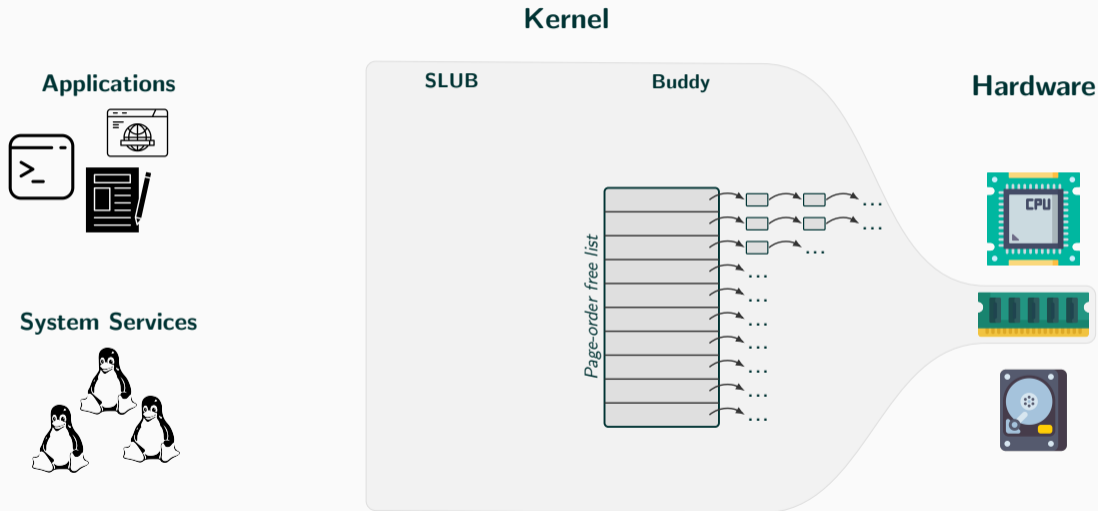
System Services



Hardware





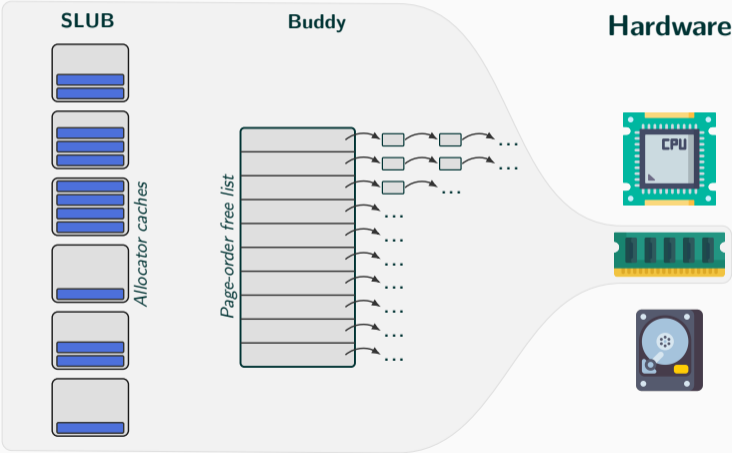
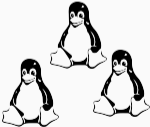


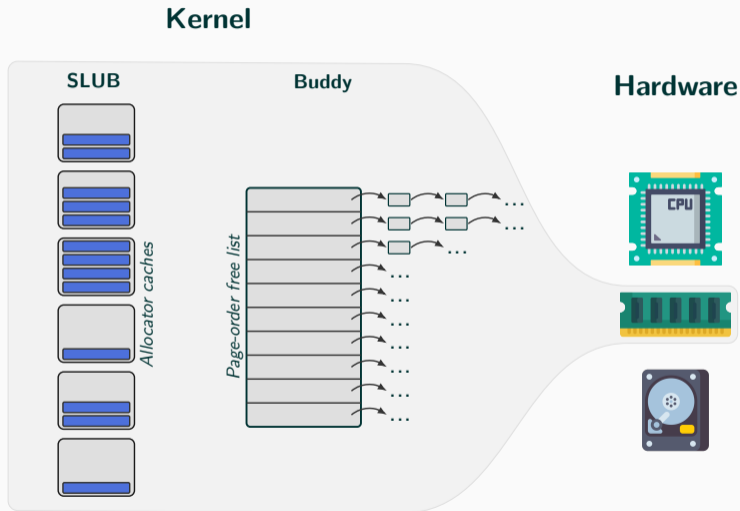
Kernel

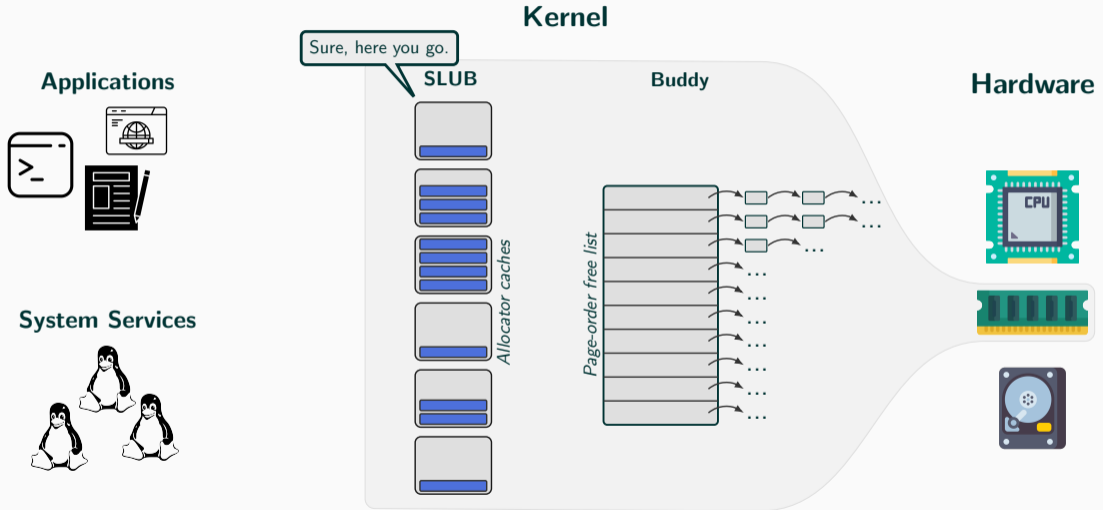
Applications

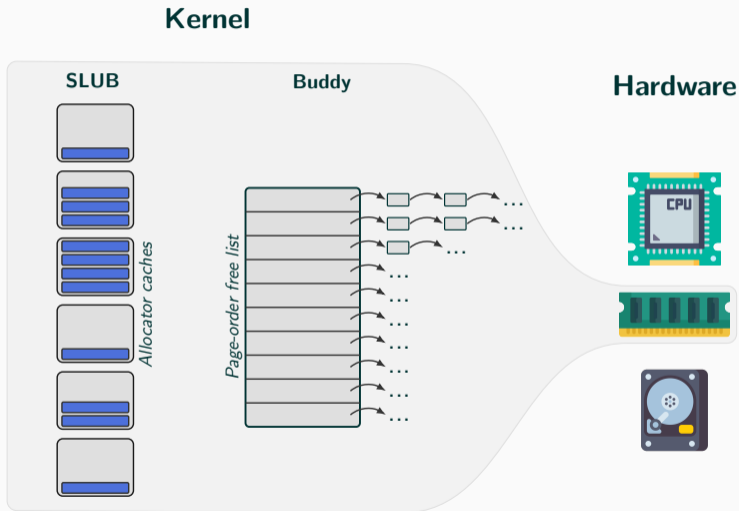
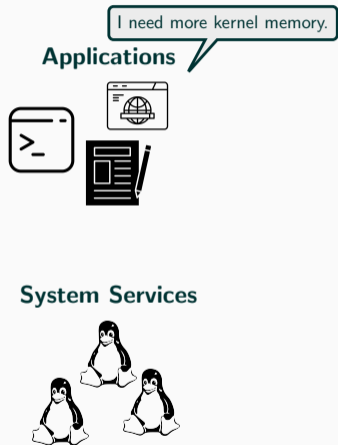


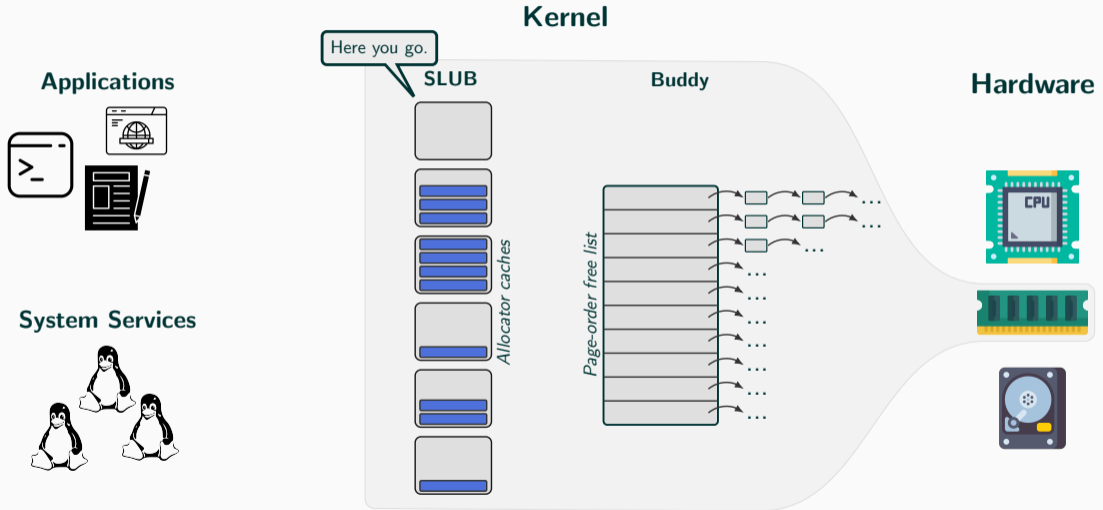
System Services

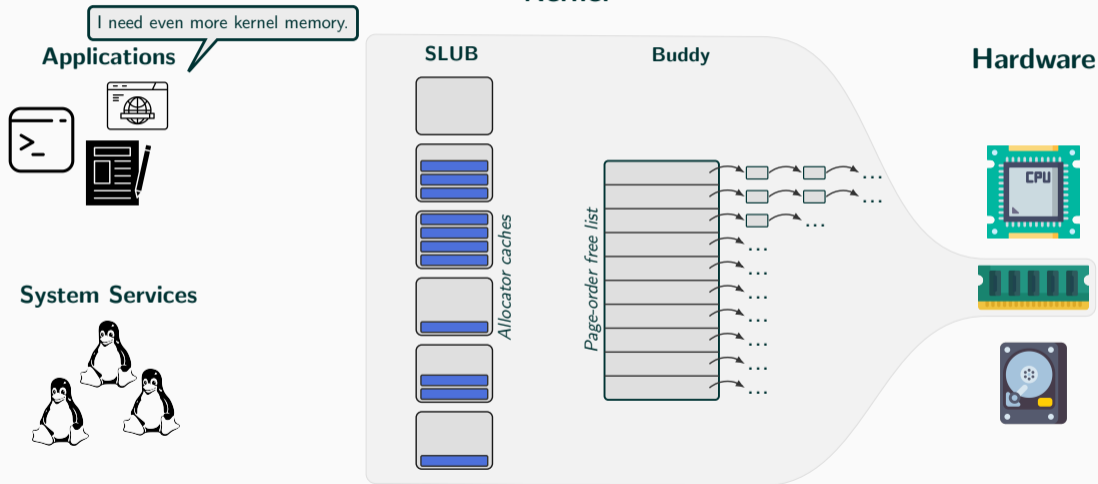


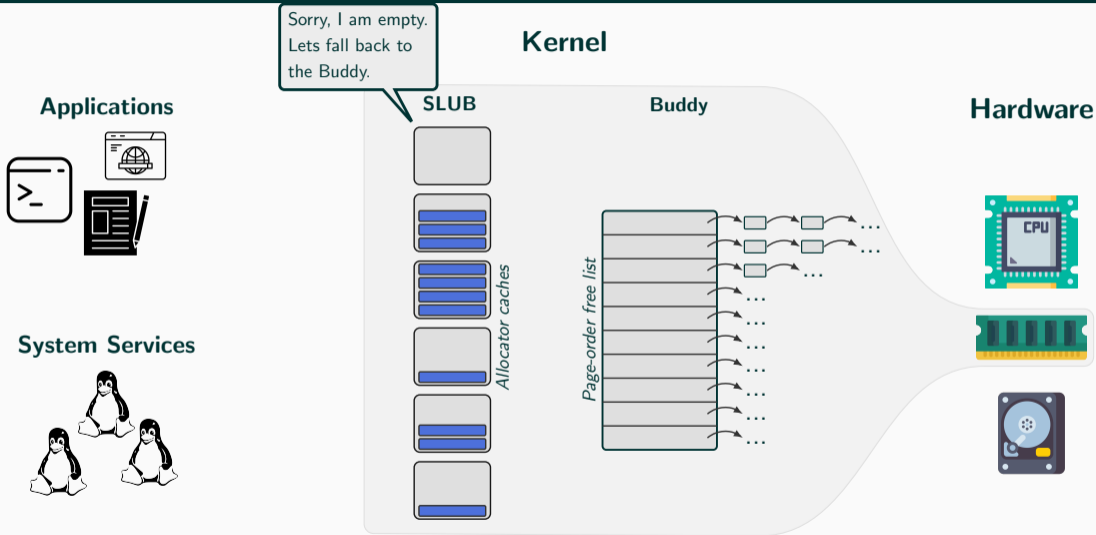


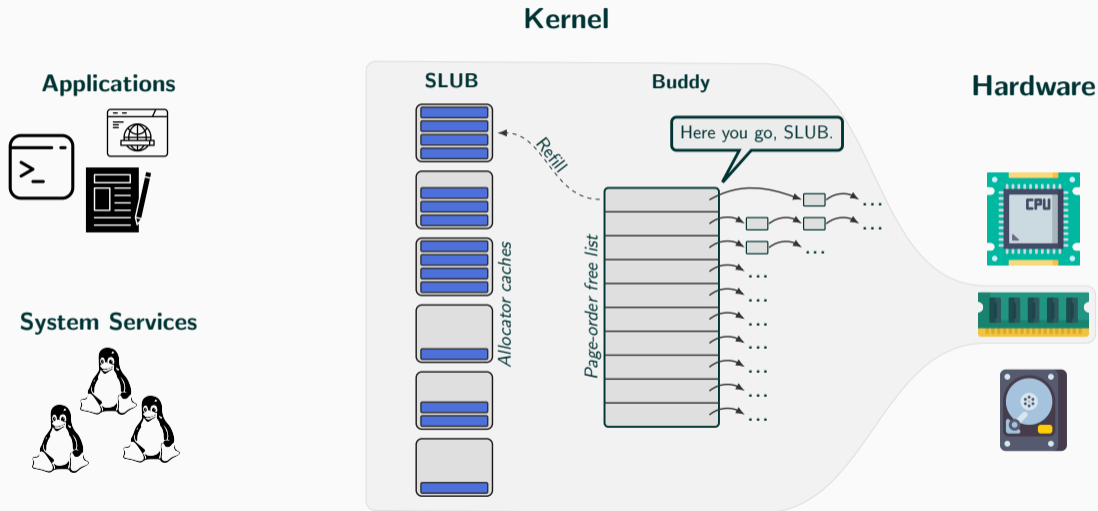


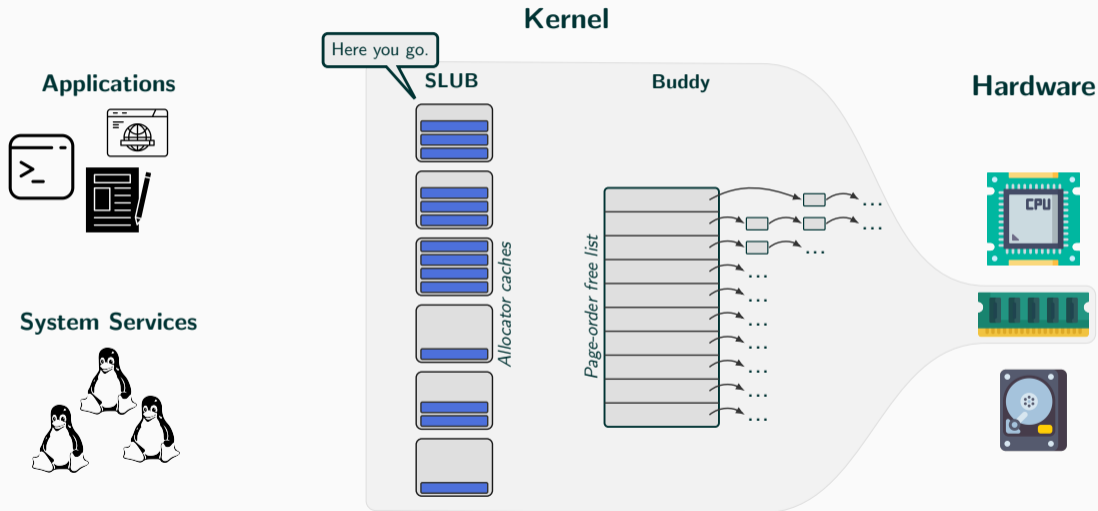


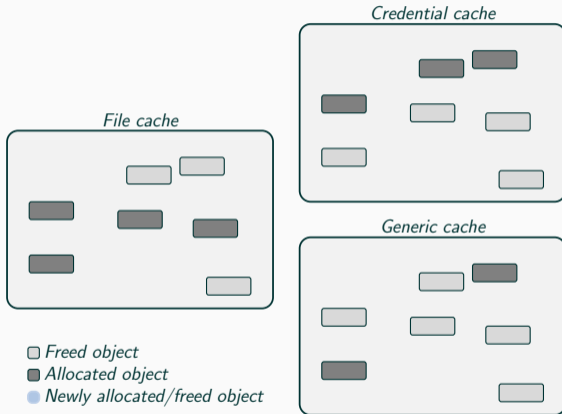


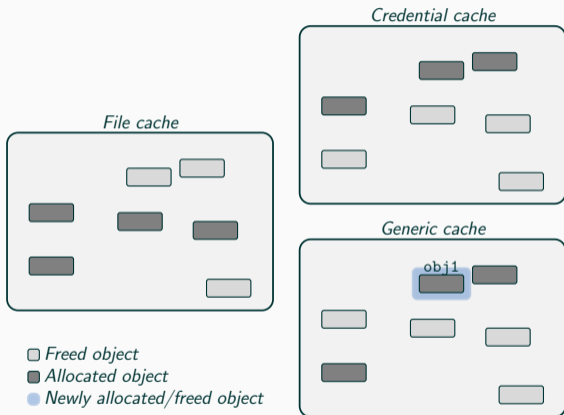




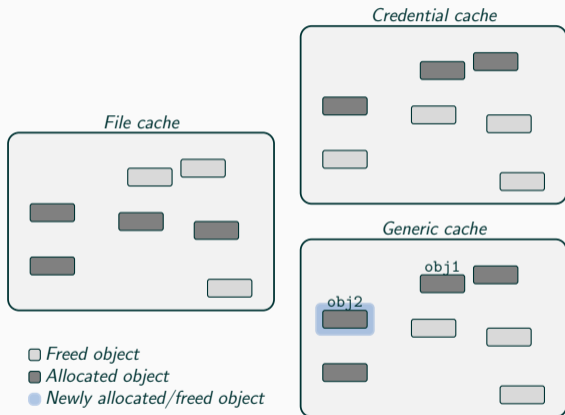




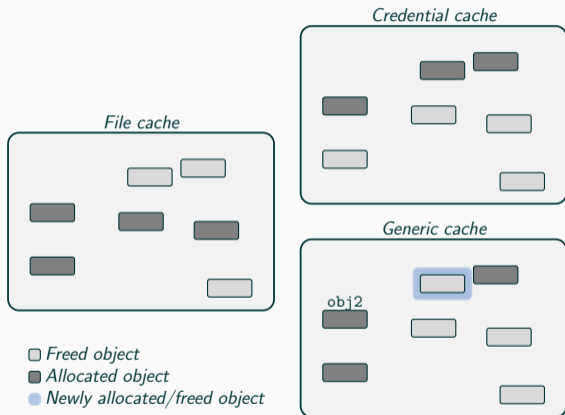




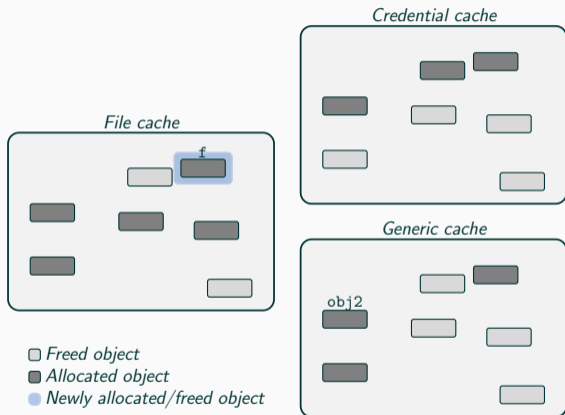
```
/* alloc from generic cache */  
void *obj1 = kmalloc();
```



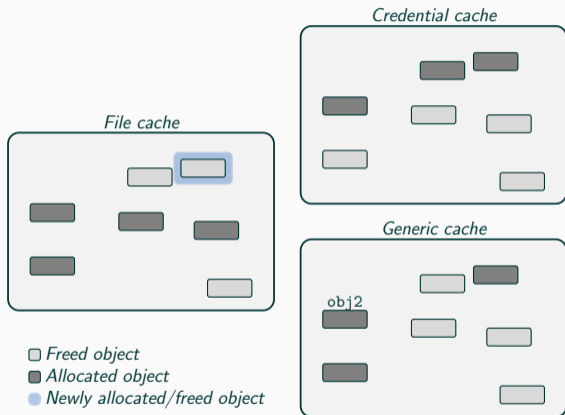
```
/* alloc from generic cache */  
void *obj1 = kmalloc();  
void *obj2 = kmalloc();
```

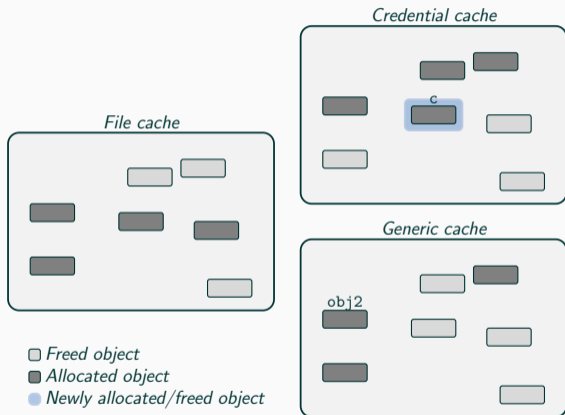
```
/* alloc from generic cache */  
void *obj1 = kmalloc();  
void *obj2 = kmalloc();  
kfree(obj1);
```



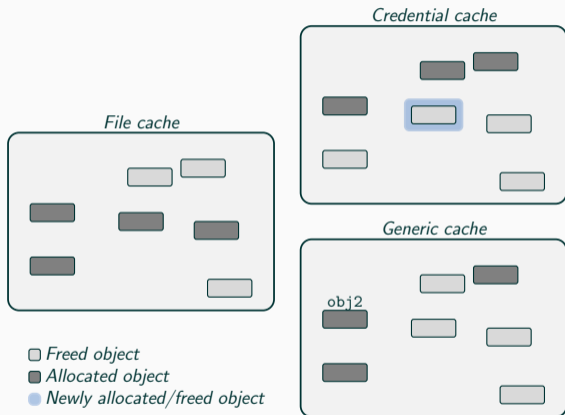
```
/* alloc from generic cache */  
void *obj1 = kmalloc();  
void *obj2 = kmalloc();  
kfree(obj1);  
...  
/* alloc from file cache */  
struct file *f = kmem_cache_alloc(filp_cachep);
```



```
/* alloc from generic cache */  
void *obj1 = kmalloc();  
void *obj2 = kmalloc();  
kfree(obj1);  
...  
/* alloc from file cache */  
struct file *f = kmem_cache_alloc(filp_cachep);  
kfree(f);
```



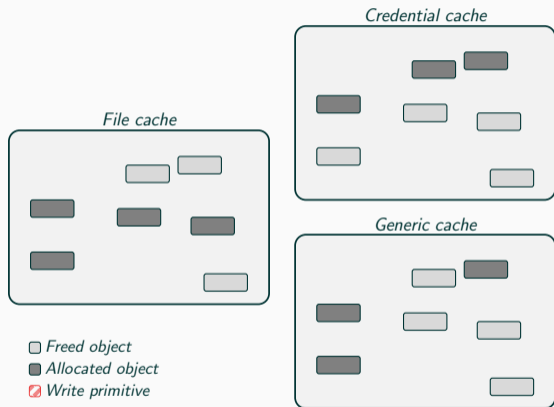
```
/* alloc from generic cache */  
void *obj1 = kmalloc();  
void *obj2 = kmalloc();  
kfree(obj1);  
...  
/* alloc from file cache */  
struct file *f = kmem_cache_alloc(filp_cachep);  
kfree(f);  
...  
/* alloc from credential cache */  
struct cred *c = kmem_cache_alloc(cred_jar);
```

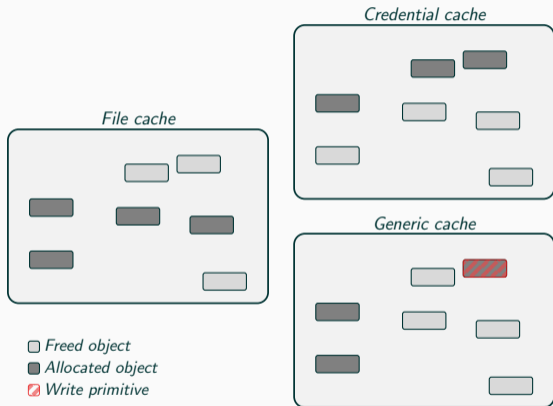


```
/* alloc from generic cache */  
void *obj1 = kmalloc();  
void *obj2 = kmalloc();  
kfree(obj1);  
...  
/* alloc from file cache */  
struct file *f = kmem_cache_alloc(filp_cachep);  
kfree(f);  
...  
/* alloc from credential cache */  
struct cred *c = kmem_cache_alloc(cred_jar);  
kfree(c);
```

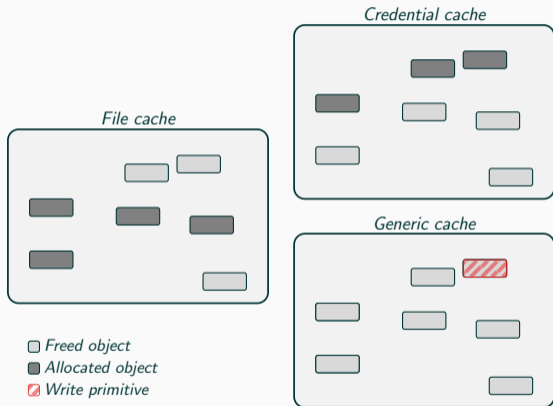


Cross-Cache Reuse Attack [Xu+15; Lin21; WZ24]

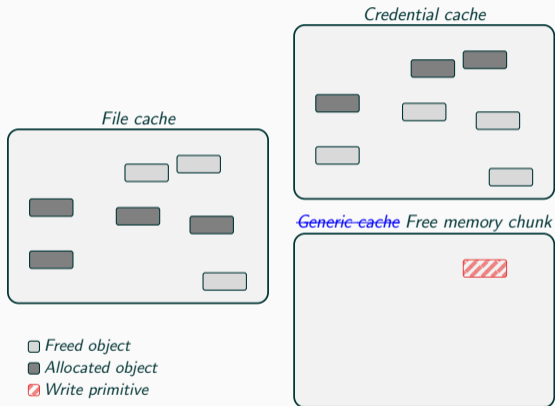




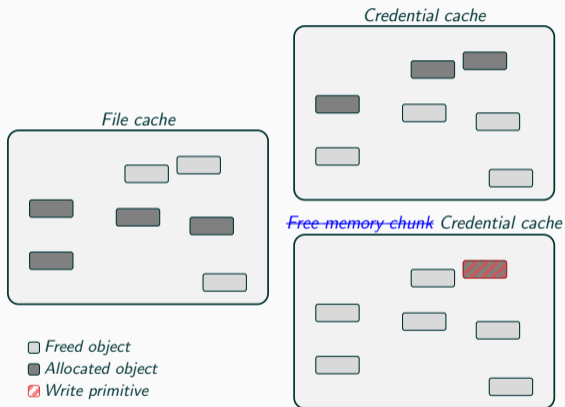
- Exploit vulnerability
 - Obtain a write primitive



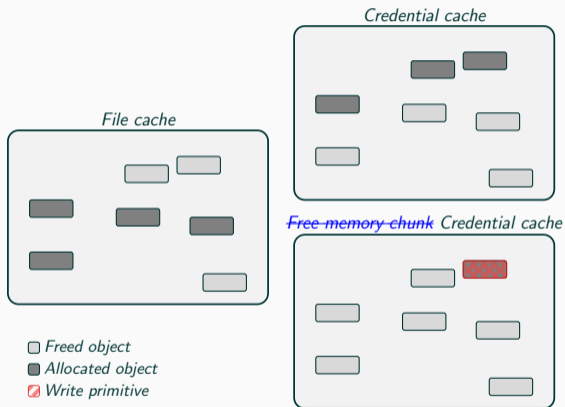
- Exploit vulnerability
 - Obtain a write primitive
- Free all generic objects



- Exploit vulnerability
 - Obtain a write primitive
- Free all generic objects
- Its memory chunk is recycled



- Exploit vulnerability
 - Obtain a write primitive
- Free all generic objects
- Its memory chunk is recycled
- Reclaim as sensitive object
 - As struct cred



- Exploit vulnerability
 - ☛ Obtain a write primitive
- Free all generic objects
- Its memory chunk is recycled
- Reclaim as sensitive object
 - 📄 As struct cred
- Trigger **write primitive**
 - 👤 Overwrite sensitive data





Dedicated Cache

- Dirty PageTable [Wu23]
- CVE-2022-2982 [Ap22]
- kCTF exploit [h0m23]
- CVE-2022-42703 [Set22]
- NO-CVE [jav23]
- CVE-2020-29660 [Hor21]

- pid_cachep
- filp_cachep
- anon_vma_cachep



Stabilization Objects

📄 Dirty PageTable [Wu23]

🗨️ Dirty Page [Lin+23]

Dedicated Cache

📄, 📄 Dirty PageTable [Wu23]

📄 CVE-2022-2982 [Ap22]

📄 kCTF exploit [h0m23]

👤 CVE-2022-42703 [Set22]

📄 NO-CVE [jav23]

📄 CVE-2020-29660 [Hor21]

📄 seq_file

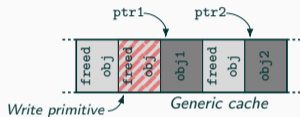
🗨️ io_vec

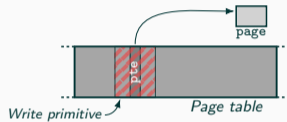
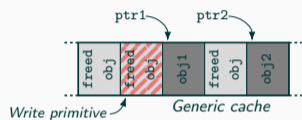
📄 pid_cachep

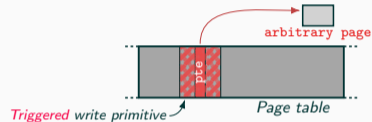
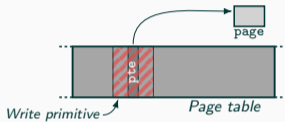
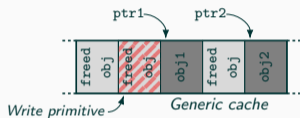
📄 filp_cachep

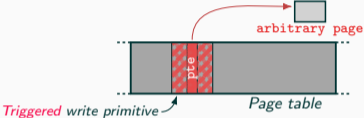
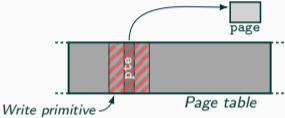
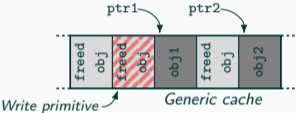
👤 anon_vma_cachep

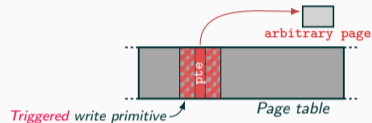
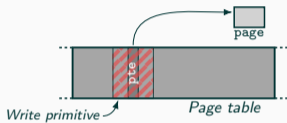
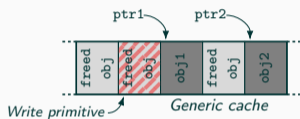
SLUBStick



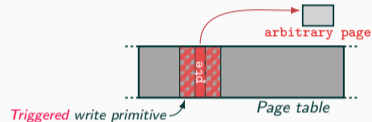
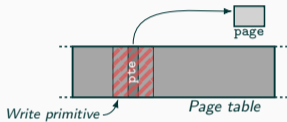
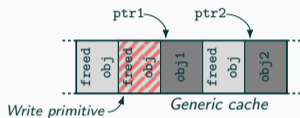






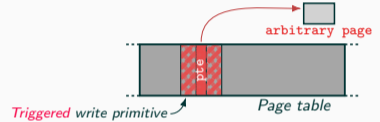
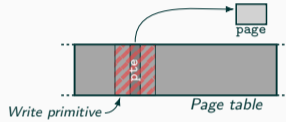
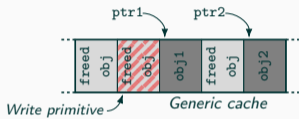







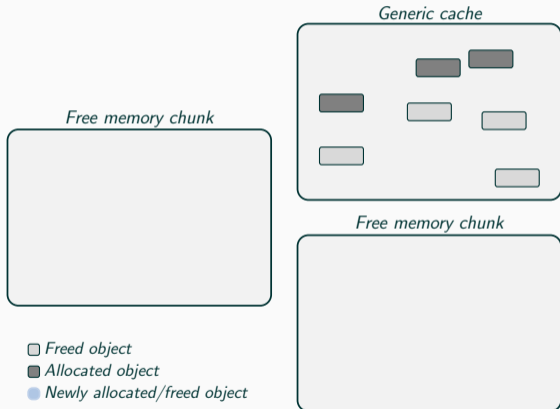
C1 Cross-cache attacks on generic caches are unreliable.

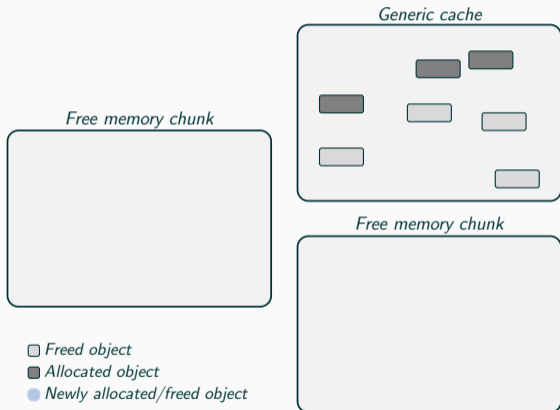


-  C1 Cross-cache attacks on generic caches are unreliable.
-  C2 Most heap vulnerabilities only grant weak write primitive.



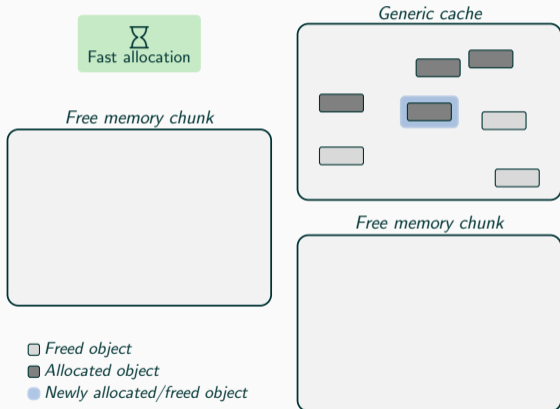
-  C1 Cross-cache attacks on generic caches are unreliable.
-  C2 Most heap vulnerabilities only grant weak write primitive.
-  C3 From page manipulation to an **arbitrary r/w primitive**.





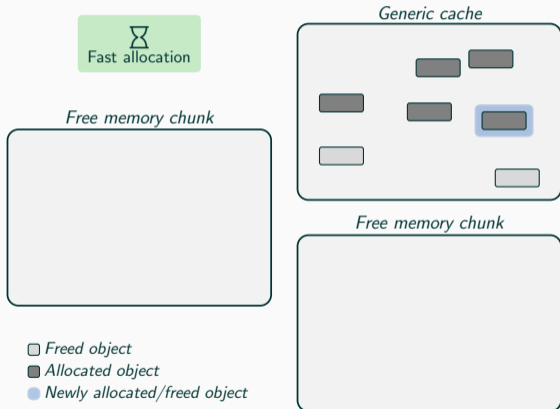
- Measure timing of allocations

👥 Group according to timing ⌚/⌚



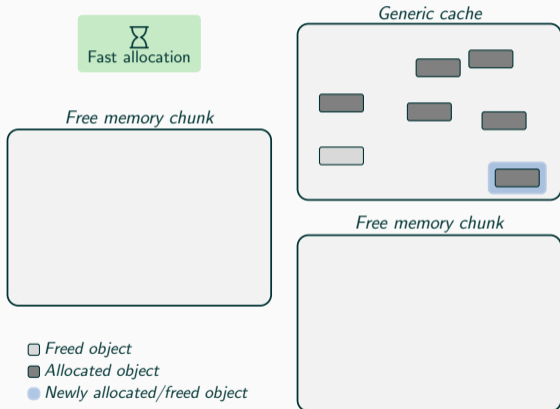
- Measure timing of allocations

 Group according to timing 



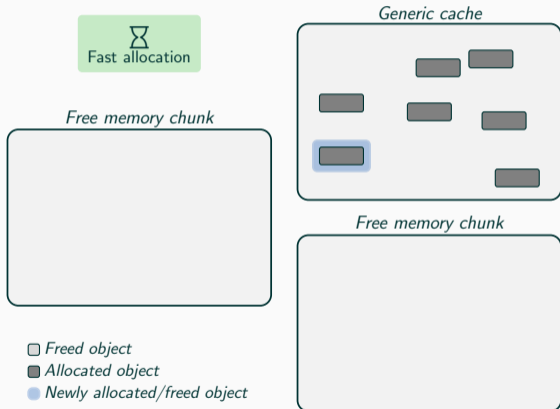
- Measure timing of allocations

 Group according to timing 



- Measure timing of allocations

 Group according to timing 



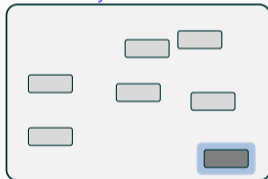
- Measure timing of allocations

 Group according to timing 



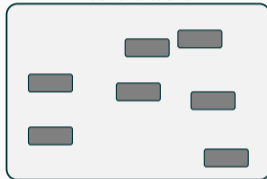
 Slow allocation

~~Free memory chunk~~ Generic cache

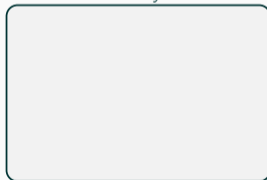


- Freed object
- Allocated object
- Newly allocated/freed object

Generic cache



Free memory chunk



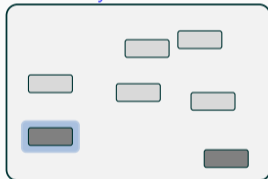
- Measure timing of allocations

 Group according to timing 



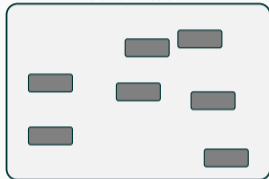
 Fast allocation

~~Free memory chunk~~ Generic cache

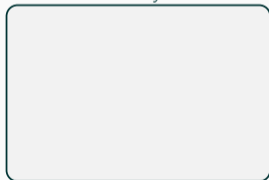


- Freed object
- Allocated object
- Newly allocated/freed object

Generic cache



Free memory chunk



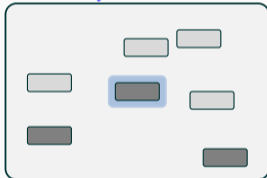
- Measure timing of allocations

 Group according to timing 



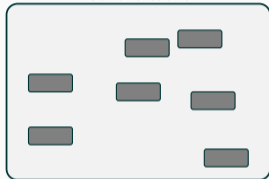
 Fast allocation

~~Free memory chunk~~ Generic cache

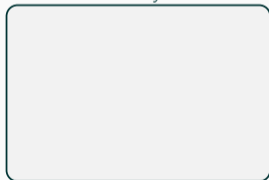


- Freed object
- Allocated object
- Newly allocated/freed object

Generic cache



Free memory chunk



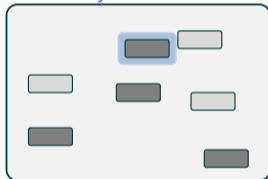
- Measure timing of allocations

 Group according to timing 



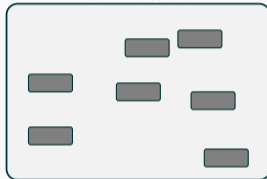

Fast allocation

~~Free memory chunk~~ Generic cache

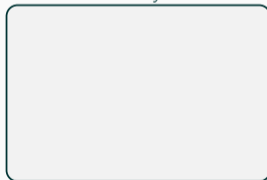


- Freed object
- Allocated object
- Newly allocated/freed object

Generic cache

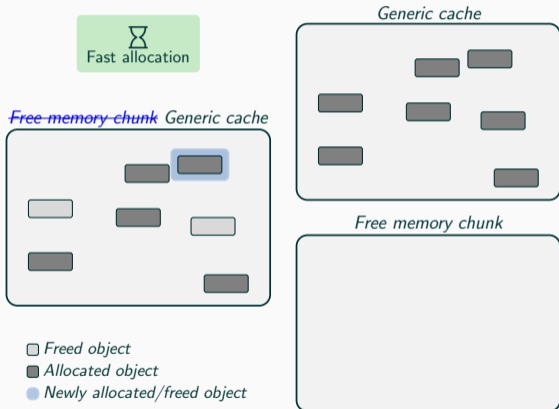


Free memory chunk



- Measure timing of allocations

 Group according to timing 



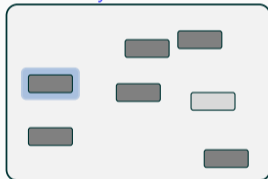
- Measure timing of allocations

 Group according to timing 



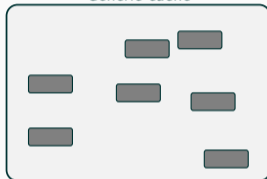
 Fast allocation

~~Free memory chunk~~ Generic cache

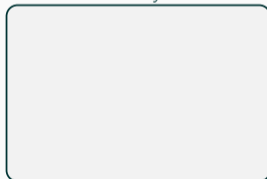


- Freed object
- Allocated object
- Newly allocated/freed object

Generic cache



Free memory chunk



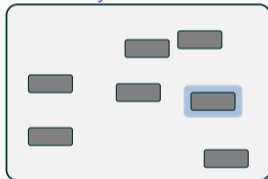
- Measure timing of allocations

 Group according to timing 



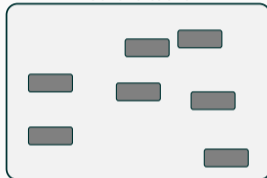
 Fast allocation

~~Free memory chunk~~ Generic cache

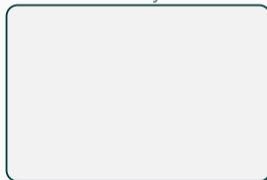


- Freed object
- Allocated object
- Newly allocated/freed object

Generic cache



Free memory chunk

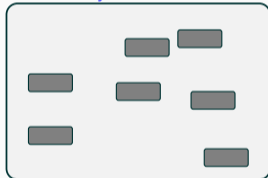


- Measure timing of allocations

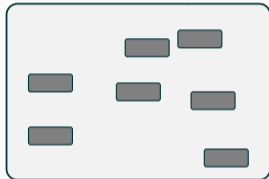
 Group according to timing 



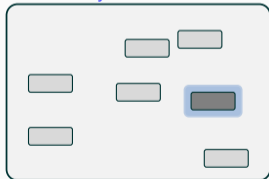
~~Free memory chunk~~ Generic cache



Generic cache



~~Free memory chunk~~ Generic cache



- Freed object
- Allocated object
- Newly allocated/freed object

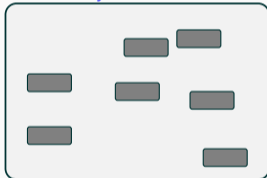
- Measure timing of allocations

👤 Group according to timing ⌚/⌚

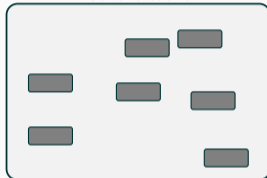


 Fast allocation

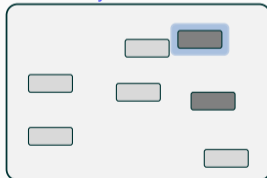
~~Free memory chunk~~ Generic cache






Generic cache



~~Free memory chunk~~ Generic cache



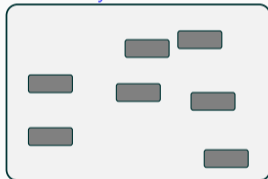
-  Freed object
-  Allocated object
-  Newly allocated/freed object

- Measure timing of allocations

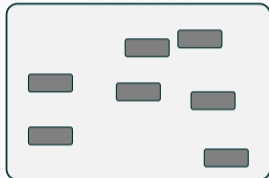
 Group according to timing  



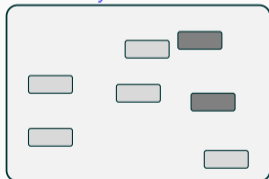
Free memory chunk Generic cache



Generic cache

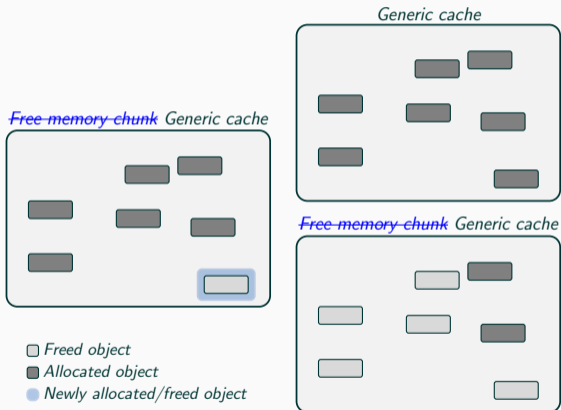


Free memory chunk Generic cache

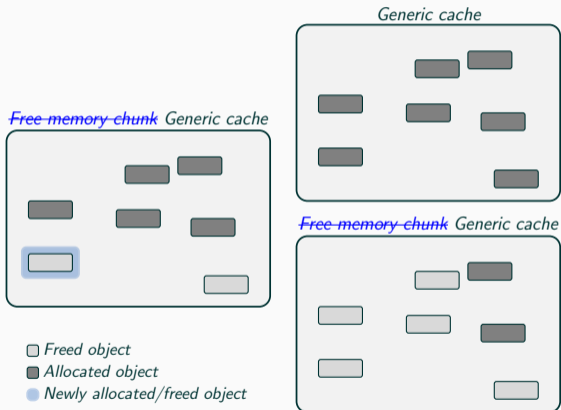


- Freed object
- Allocated object
- Newly allocated/freed object

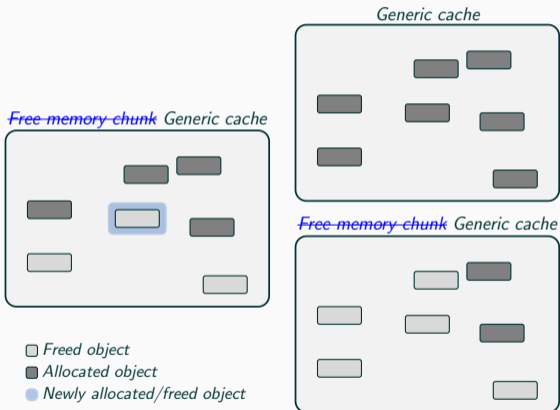
- Measure timing of allocations
 - Group according to timing ⏱/⏱
- Free all objects of left memory chunk



- Measure timing of allocations
 - Group according to timing ⏱/⏱
- Free all objects of left memory chunk



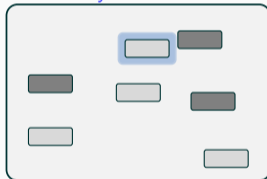
- Measure timing of allocations
 - Group according to timing ⏱/⏱
- Free all objects of left memory chunk



- Measure timing of allocations
 - Group according to timing ⏱/⏱
- Free all objects of left memory chunk

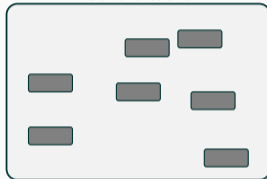


Free memory chunk Generic cache

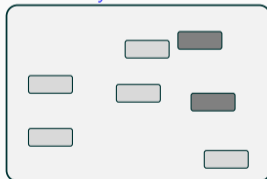


- Freed object
- Allocated object
- Newly allocated/freed object

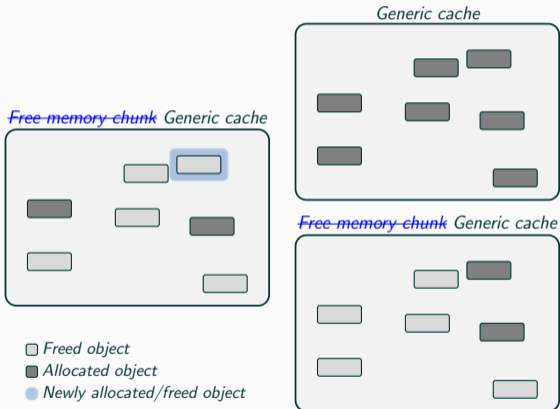
Generic cache



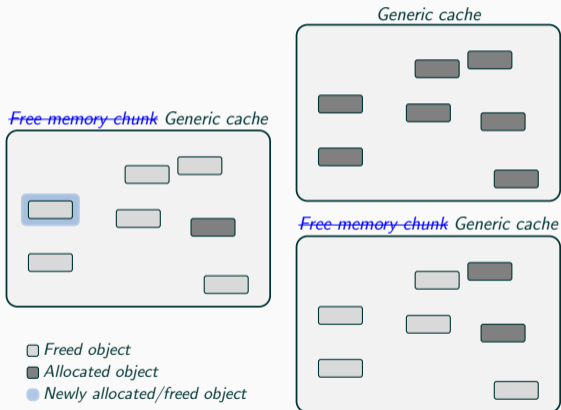
Free memory chunk Generic cache



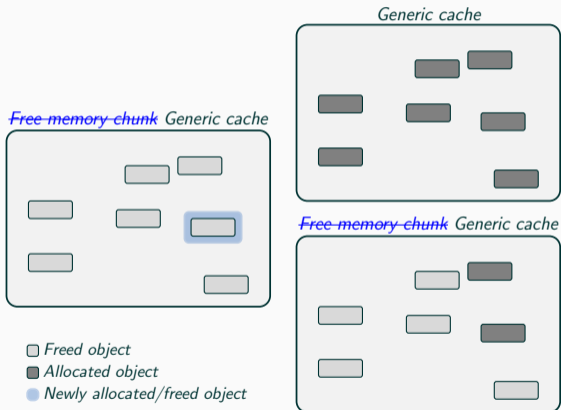
- Measure timing of allocations
 - Group according to timing ⏱/⏱
- Free all objects of left memory chunk



- Measure timing of allocations
 - Group according to timing ⏱/⏱
- Free all objects of left memory chunk



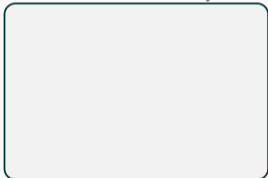
- Measure timing of allocations
 - Group according to timing ⌚/⌚
- Free all objects of left memory chunk



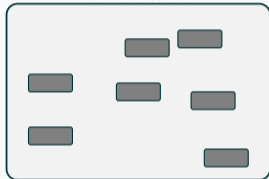
- Measure timing of allocations
 - Group according to timing ⌚/⌚
- Free all objects of left memory chunk



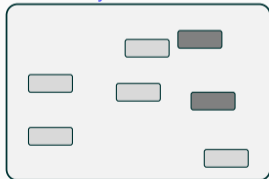
Generic-cache Free memory chunk



Generic cache

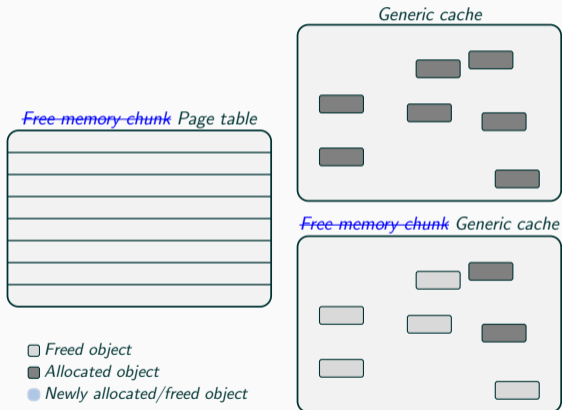


Free memory chunk *Generic cache*



- Freed object
- Allocated object
- Newly allocated/freed object

- Measure timing of allocations
 - Group according to timing ⏳/⏳
- Free all objects of left memory chunk
- The memory chunk is recycled



- Measure timing of allocations
 - Group according to timing ⏱/⏱
- Free all objects of left memory chunk
- The memory chunk is recycled
- Reclaim memory chunk
 - As a **page table**



Generic Cache	#Pages	Success Rate		
		Idle	No CPU pinning	External noise
		%	%	%
kmalloc-8	1	99.9 ± 0.1	99.9 ± 0.1	99.6 ± 0.7
kmalloc-16	1	99.4 ± 0.6	98.9 ± 1.2	99.9 ± 0.4
kmalloc-32	1	99.4 ± 0.9	99.7 ± 0.5	99.9 ± 0.3
kmalloc-64	1	99.2 ± 1.3	99.2 ± 0.9	81.0 ± 6.4
kmalloc-96	1	99.9 ± 0.4	99.9 ± 0.1	99.8 ± 0.6
kmalloc-128	1	99.9 ± 0.4	99.8 ± 0.5	99.9 ± 0.3
kmalloc-192	1	99.9 ± 0.4	99.8 ± 0.4	99.3 ± 1.2
kmalloc-256	1	99.9 ± 0.3	99.9 ± 0.3	99.7 ± 0.7
kmalloc-512	2	90.2 ± 5.4	87.2 ± 3.1	65.2 ± 2.8
kmalloc-1024	4	88.1 ± 7.2	79.5 ± 3.3	70.3 ± 8.1
kmalloc-2048	8	83.1 ± 9.2	70.5 ± 16	57.8 ± 5.7
kmalloc-4096	8	82.1 ± 3.4	73.3 ± 19	53.8 ± 10



Thread A

Thread B



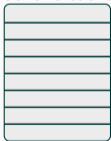


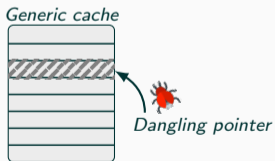
Thread A

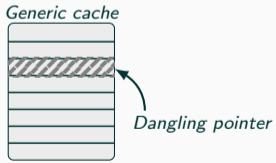
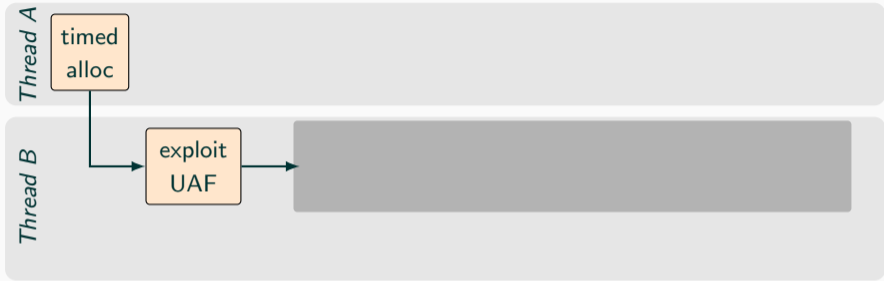
timed
alloc

Thread B

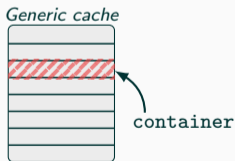
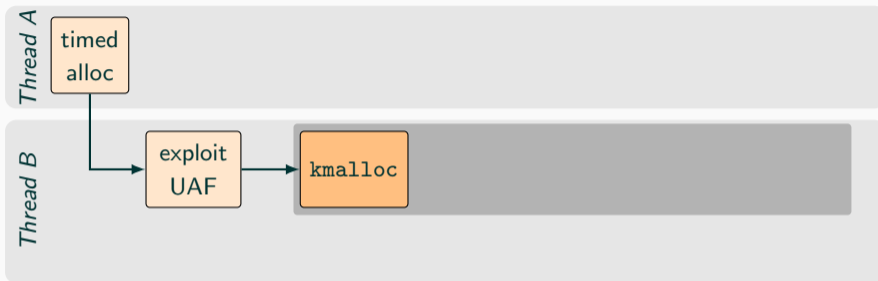
Generic cache



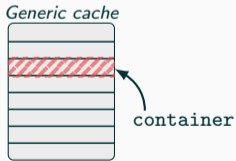
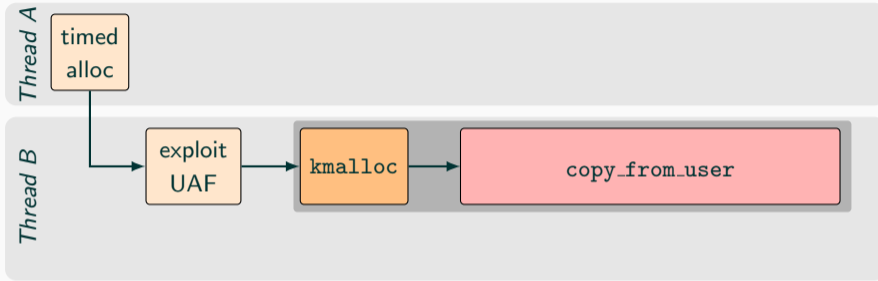




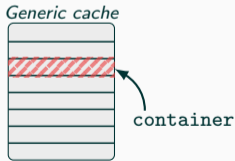
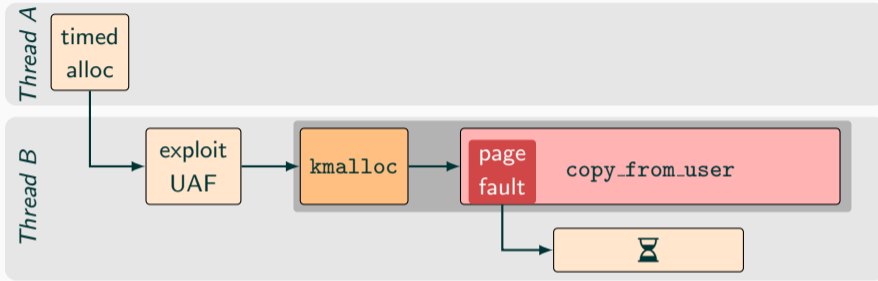
```
i32 replace_user_tlv(u32 *buf, u64 size) {  
  
}  
}
```



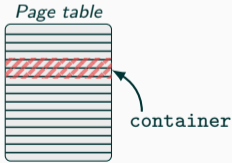
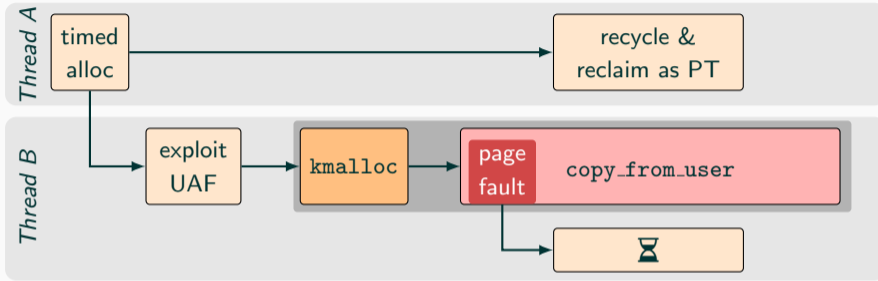
```
i32 replace_user_tlv(u32 *buf, u64 size) {  
    /* allocate object */  
    u32 *container = kmalloc(size);  
}
```



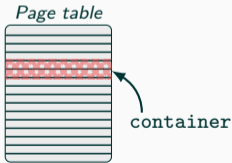
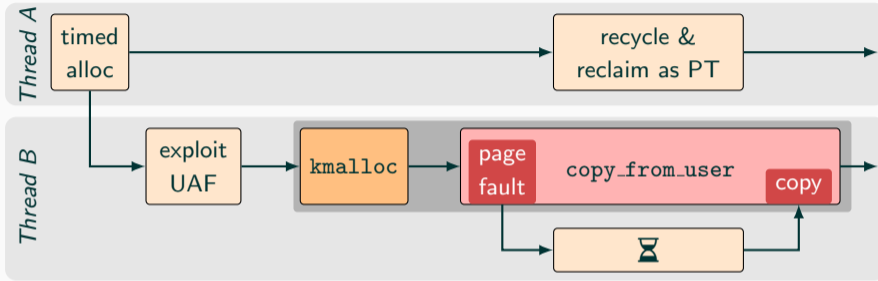
```
i32 replace_user_tlv(u32 *buf, u64 size) {  
    /* allocate object */  
    u32 *container = kmalloc(size);  
    /* copy data from user */  
    copy_from_user(container, buf, size);  
}
```



```
i32 replace_user_tlv(u32 *buf, u64 size) {
    /* allocate object */
    u32 *container = kmalloc(size);
    /* copy data from user */
    copy_from_user(container, buf, size);
}
```

```
i32 replace_user_tlv(u32 *buf, u64 size) {  
    /* allocate object */  
    u32 *container = kmalloc(size);  
    /* copy data from user */  
    copy_from_user(container, buf, size);  
}
```

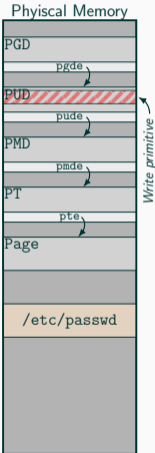


```
i32 replace_user_tlv(u32 *buf, u64 size) {  
    /* allocate object */  
    u32 *container = kmalloc(size);  
    /* copy data from user */  
    copy_from_user(container, buf, size);  
}
```

Use the UAF Write for Page-Table Manipulation



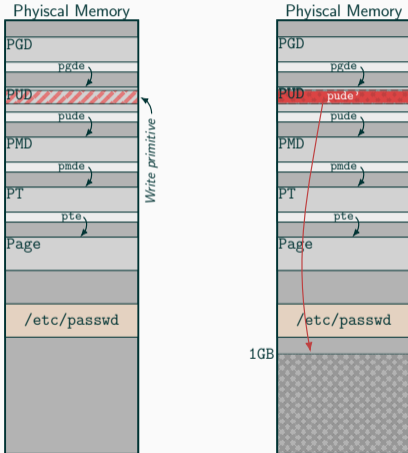
- page
- target page
- table entry
- ▨ overwritable area
- corrupted memory
- corrupted table entry
- lowest 1GB
- reference
- corrupted reference



Use the UAF Write for Page-Table Manipulation



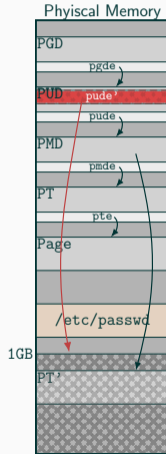
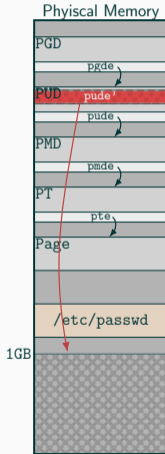
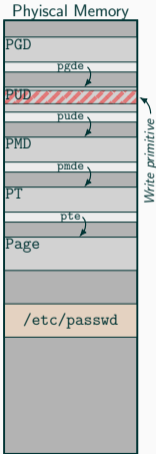
- page
- target page
- table entry
- ▨ overwritable area
- corrupted memory
- corrupted table entry
- ▨ lowest 1GB
- reference
- corrupted reference



Use the UAF Write for Page-Table Manipulation



- page
- target page
- table entry
- ▨ overwritable area
- corrupted memory
- corrupted table entry
- ▨ lowest 1GB
- reference
- corrupted reference

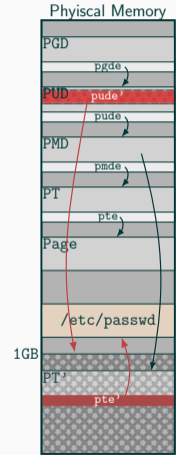
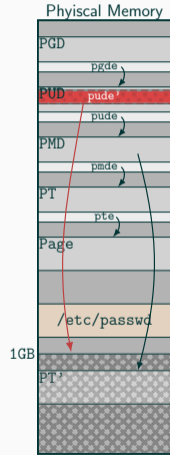
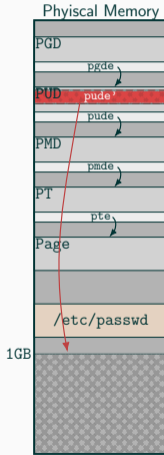
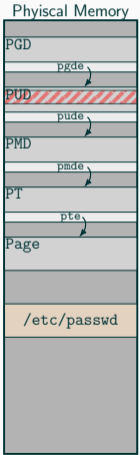


Use the UAF Write for Page-Table Manipulation



- page □ target page □ table entry
- ▨ overwritable area ▩ corrupted memory
- corrupted table entry ▩ lowest 1GB
- reference → corrupted reference

 **C3**
Obtain arb r/w





CVE	Capability	Cache
CVE-2023-21400	DF	kmalloc-32
CVE-2023-3609	UAF	kmalloc-96
CVE-2022-32250	UAF	kmalloc-64
CVE-2022-29582	UAF	filep_cache
CVE-2022-27666	OOB	kmalloc-4096
CVE-2022-2588	DF	kmalloc-192
CVE-2022-0995	OOB	kmalloc-96
CVE-2021-4157	OOB	kmalloc-64
CVE-2021-3492	DF	kmalloc-4096



<https://lukasmaar.github.io>

- **Timing side channel:**
 - ☞ Make software cross-cache attacks practical
- **Primitive conversion:**
 - ☞ Limited heap write to UAF write
- **Uses the UAF write for page-table manipulation:**
 - ☞ Obtain an **arbitrary physical r/w primitive**
- **Implemented 9 PoC exploits:**
 - 🐧 For Linux kernel v6.2

References

- [Ap22] Awarau and pql. CVE-2022-29582 An io_uring vulnerability. 2022. URL: <https://ruia-ruia.github.io/2022/08/05/CVE-2022-29582-io-uring/>.
- [h0m23] h0mbre. Escaping the Google kCTF Container with a Data-Only Exploit. 2023. URL: https://h0mbre.github.io/kCTF_Data_Only_Exploit/#.
- [Hor21] J. Horn. How a simple Linux kernel memory corruption bug can lead to complete system compromise. 2021. URL: <https://googleprojectzero.blogspot.com/2021/10/how-simple-linux-kernel-memory.html>.
- [jav23] javierprtd. No CVE for this bug which has never been in the official kernel. 2023. URL: <https://soez.github.io/posts/no-cve-for-this.-It-has-never-been-in-the-official-kernel/>.

- [Lin+23] Z. Lin, X. Xing, Z. Chen, and K. Li. Bad io_uring: A New Era of Rooting for Android. 2023. URL: https://i.blackhat.com/BH-US-23/Presentations/US-23-Lin-bad_io_uring.pdf.
- [Lin21] Z. Lin. How AUTOSLAB Changes the Memory Unsafety Game. 2021. URL: https://grsecurity.net/how_autoslab_changes_the_memory_unsafety_game.
- [Set22] Seth Jenkins. Exploiting CVE-2022-42703 - Bringing back the stack attack. 2022. URL: <https://googleprojectzero.blogspot.com/2022/12/exploiting-CVE-2022-42703-bringing-back-the-stack-attack.html>.
- [Wu23] N. Wu. Dirty Pagetable: A Novel Exploitation Technique To Rule Linux Kernel. 2023. URL: https://yanglingxi1993.github.io/dirty_pagetable/dirty_pagetable.html.
- [WZ24] L. Wu and Q. Zhang. Game of Cross Cache: Let's win it in a more effective way! 2024. URL: <https://i.blackhat.com/Asia-24/Presentations/Asia-24-Wu-Game-of-Cross-Cache.pdf>.

- [Xu+15] W. Xu, J. Li, J. Shu, W. Yang, T. Xie, Y. Zhang, and D. Gu. From collision to exploitation: Unleashing use-after-free vulnerabilities in linux kernel. In: CCS. 2015.