SLUBStick:

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

Lukas Maar — *Graz University of Technology*
Stefan Gast — *Graz University of Technology*
Martin Unterguggenberger — *Graz University of Technology*
Mathias Oberhuber — *Graz University of Technology*
Stefan Mangard — *Graz University of Technology*

ARTIFACT EVALUATED usenix ASSOCIATION AVAILABLE
ARTIFACT EVALUATED usenix ASSOCIATION FUNCTIONAL
ARTIFACT EVALUATED usenix ASSOCIATION REPRODUCED

Introduction

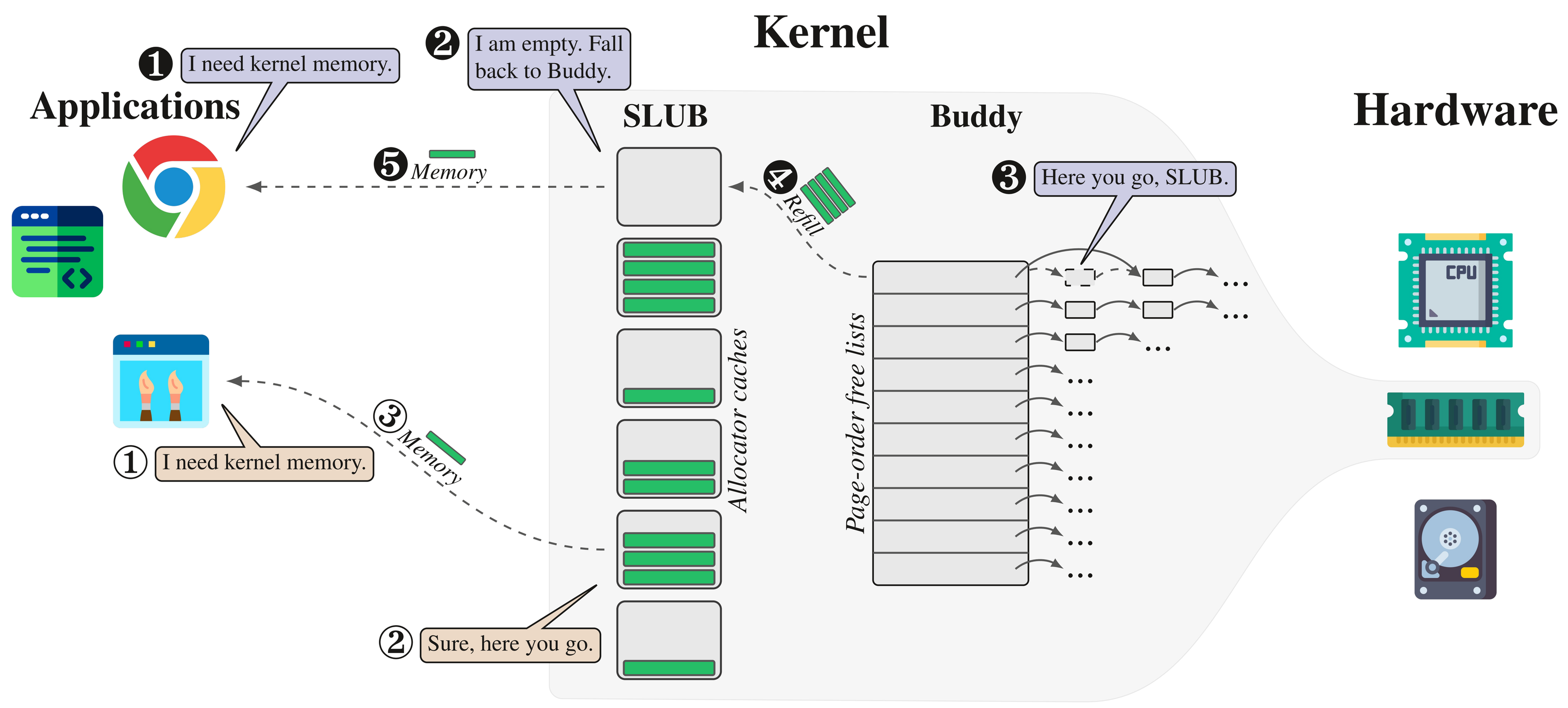
In recent years, the number of vulnerabilities as well as defenses in the Linux kernel has increased significantly. This results in a situation where many kernel vulnerabilities exist, while their exploitation is difficult.

We present a new kernel exploit technique, SLUBStick, which allows bad actors to fully compromise Linux systems with state-of-the-art kernel defenses enabled. We show the practicality of SLUBStick by implementing 9 exploits and compromising Ubuntu 22.04 LTS 9 times.

Background

Kernel Memory Management. Two allocators:

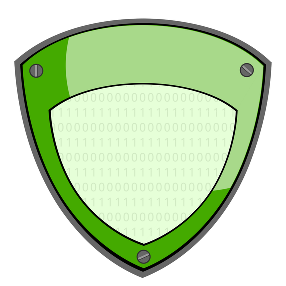
- Buddy allocator: splits the entire memory space into page-order memory chunks and stores them in *page-order free lists*.
- SLUB allocator: uses chunks from Buddy and stores free memory slots for object allocation in *allocator caches*.



Object Allocation. Applications use the SLUB allocator caches:

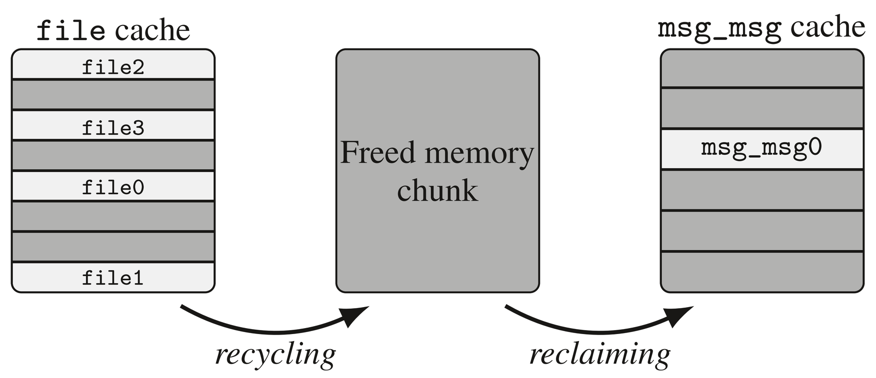
- Fast path: on a memory allocation (①), the allocator cache has free memory slots (②) and returns one slot (③).
- Slow path: on a memory allocation (❶), the allocator cache has no free memory slots (❷), so it resorts to Buddy (❸) and refills the memory slots (❹), returning one to the application (❺).

Heap Segregation. Linux uses different allocator caches for different security contexts, so vulnerable and security-critical objects never share the same cache. Hence, a UAF write to a vulnerable object cannot be directly exploited to overwrite security-critical objects.

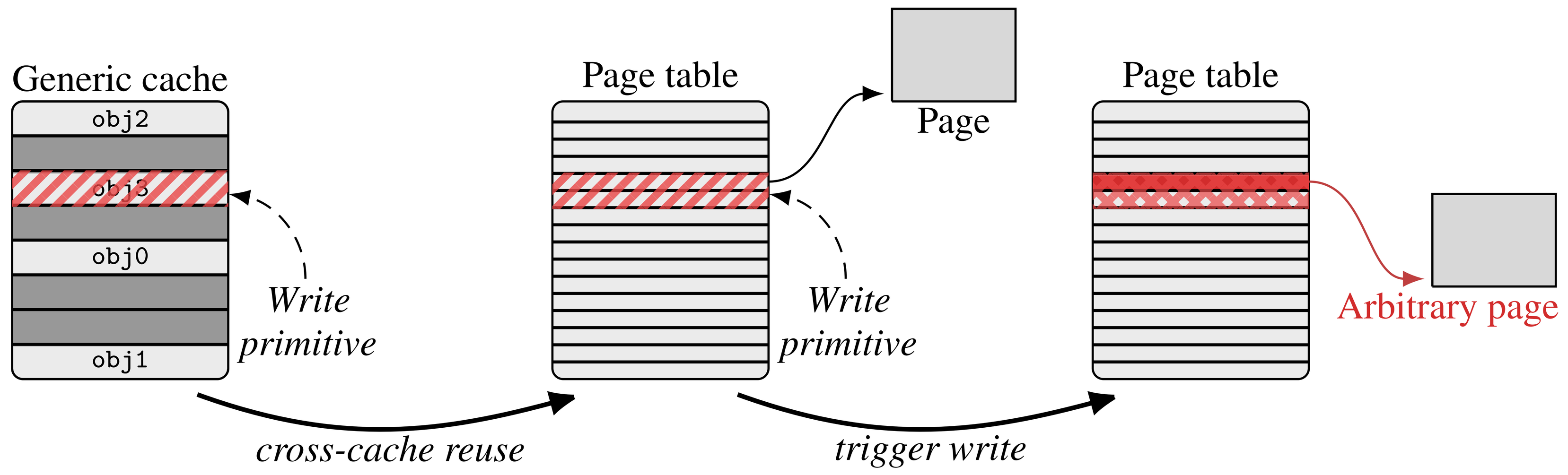


Cross-Cache Reuse. A bad actor exploits Buddy's memory reuse. They free all memory chunk slots from an allocator cache (e.g., `file`), causing to *recycle* this chunk. They then *reclaim* the chunk for security-critical objects (e.g., `msg_msg`).

This cross-cache reuse is mostly **unreliable and impractical**, with a success rate of 40%, where unsuccessful attempts may crash.



High-Level Overview



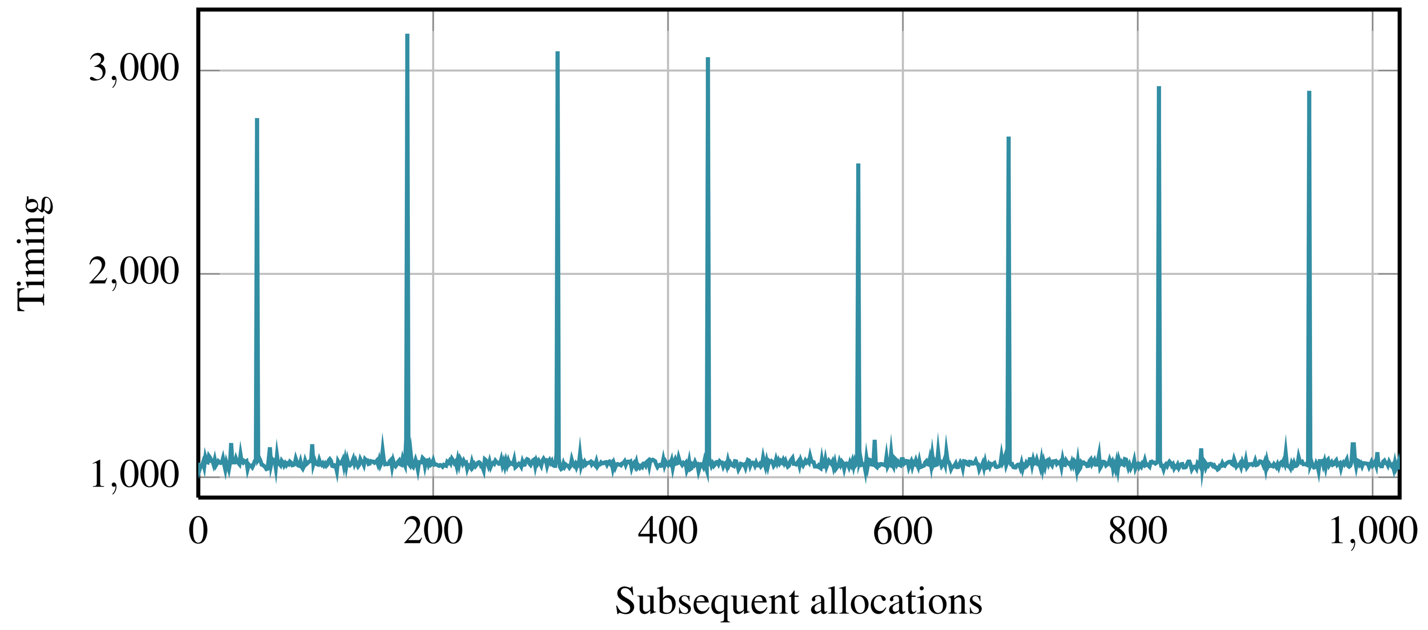
SLUBStick exploits a kernel heap vulnerability to obtain a *write primitive* for a vulnerable object at a given time. It then performs a *cross-cache reuse*, where the write primitive refers to a page table. Finally, it *triggers the write* to corrupt a page-table entry, granting its user address with arbitrary read/write access to the underlying physical page.

Technical Challenges. SLUBStick overcomes three challenges:

- **C1** Cross-cache reuse attacks on generic caches are unreliable.
- **C2** Most kernel heap vulnerabilities only grant weak write primitives.
- **C3** From page-table manipulation to an arbitrary read/write.

Timing Side Channel on the SLUB Allocator C1

SLUBStick **makes cross-cache reuse reliable and practical** by performing a timing side channel on SLUB. It measures the syscall timings of allocations and distinguishes between fast (①-③) from slow (❶-❺) paths, allowing objects to be grouped based on their chunk.



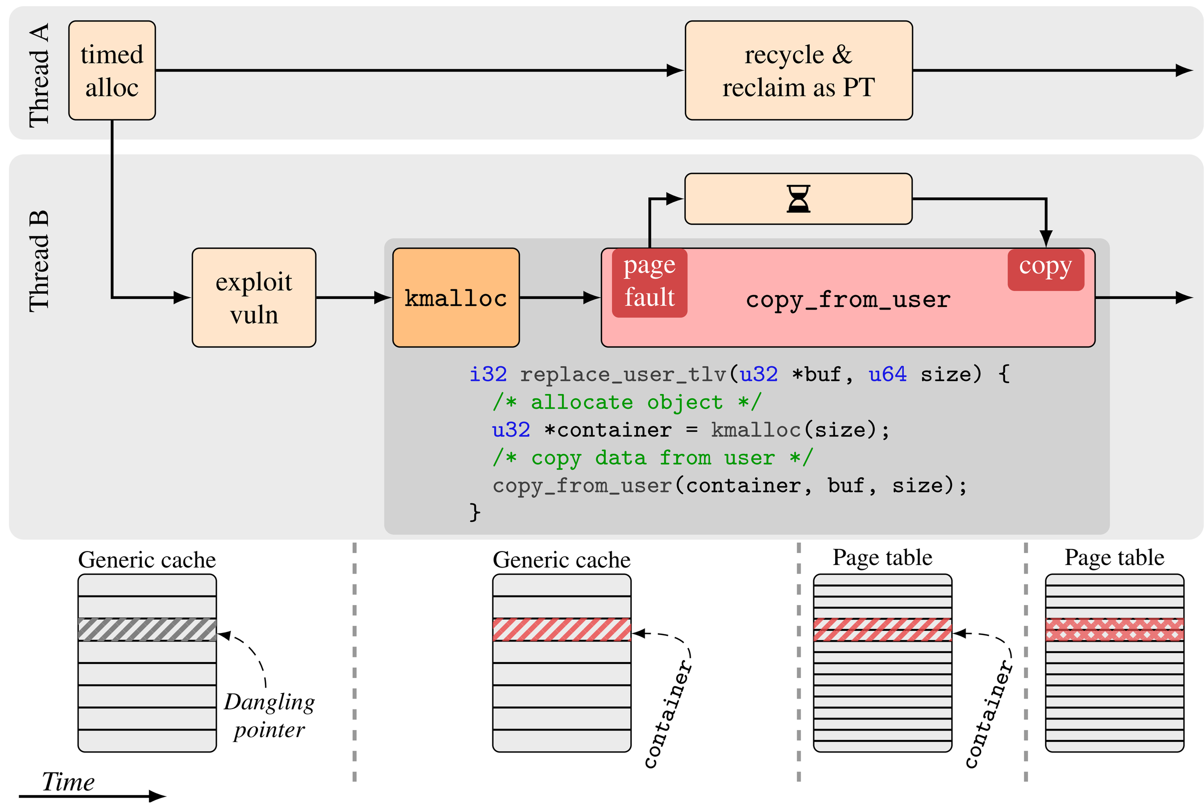
- Fast path (①-③): ~1,100 time stamps.
- Slow path (❶-❺): >2,500 time stamps.

Group allocated objects based on their chunk and free all grouped objects for cross-cache reuse.

- Ubuntu 22.04 LTS
- Linux kernel v6.2.
- Multiple generic caches.
- Tested on idle and noisy systems.
- **Success rates well above 40%.**

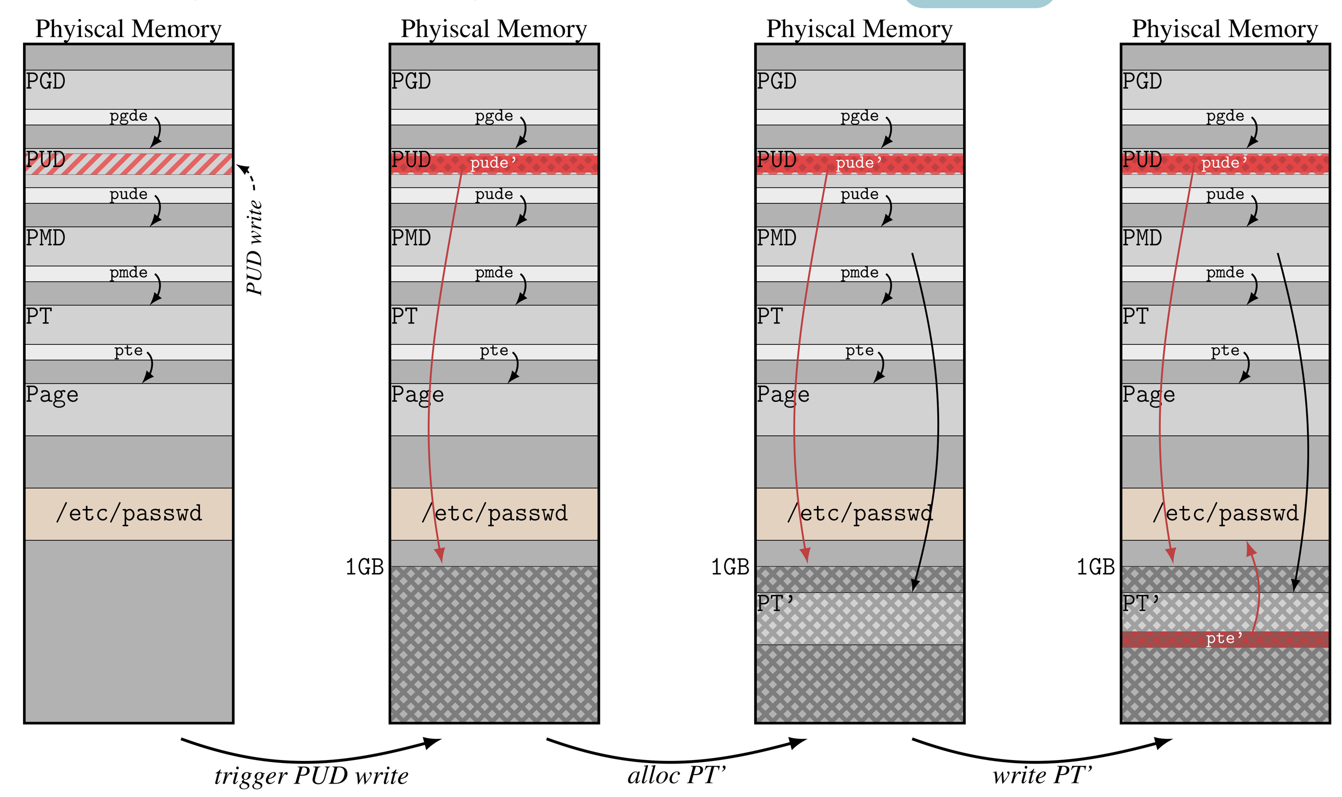
Generic Cache	#Pages	Success Rate		
		Idle	No CPU pinning	External noise
		%	%	%
`kmalloc-8`	1	99.9 ± 0.1	99.9 ± 0.1	99.6 ± 0.7
`kmalloc-16`	1	99.4 ± 0.6	98.9 ± 1.2	99.9 ± 0.4
`kmalloc-32`	1	99.4 ± 0.9	99.7 ± 0.5	99.9 ± 0.3
`kmalloc-64`	1	99.2 ± 1.3	99.2 ± 0.9	81.0 ± 6.4
`kmalloc-96`	1	99.9 ± 0.4	99.9 ± 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

Pivoting Kernel Heap Vulnerabilities C2



SLUBStick exploits a **heap vulnerability for a page-table manipulation**, by first creating a *dangling pointer*. It then reclaims the pointer's memory for `container`, where writing via `copy_from_user` causes a slow page fault. SLUBStick recycles the cache's page and reclaims it as a page table, where copying then overwrites page-table entries.

Arbitrary Memory Read/Write C3



SLUBStick converts a **single-shot page-table manipulation to an arbitrary physical read/write**: It *triggers the PUD write* so that the user address with pude' refers to the first physical GB. It then *allocates PT'* and *overwrites* a PT' entry with pte'. The user address with pte' now refers to an arbitrary physical location, allowing the arbitrary physical read/write.

Conclusion

Timing side channel:
- Makes software cross-cache reuses practical.

Primitive conversions:
- Limited heap write to page-table manipulation.
- Single-shot page-table manipulation to an arbitrary physical read/write primitive.

Implemented 9 POC exploits.