

SLAM: Exploiting Linear Address Masking & Hardware Pointer Tagging

CVE-2024-28854 Technical Report: Breaking Canonical Address Checks, Spectre Meltdown, and Kernel ASLR

Published by Lesty Tech Systems Security Lab • Date: October 22, 2026 • Lesty.tech Architectural Engineering Dossier

1. Hardware Pointer Tagging Evolution: Intel LAM, AMD UAC, and ARM TBI

In 64-bit architectures, operating systems traditionally utilize only 48 or 57 bits of virtual address space. To accelerate memory sanitization (AddressSanitizer), memory tagging (MTE), and garbage collection runtimes, processor vendors introduced hardware pointer masking: Intel Linear Address Masking (LAM), AMD Upper Address Ignore (UAC), and ARM Top Byte Ignore (TBI). These features allow software to store arbitrary metadata in the upper address bits (bits 57-63) without software unmasking instructions.

2. The Architectural Flaw: Speculative Pointer Un-tagging

Disclosed by researchers at Vrije Universiteit Amsterdam (CVE-2024-28854 / SLAM), pointer masking features introduce an unmitigated speculative vulnerability. While architectural page table walkers enforce canonical address rules, the CPU's speculative execution engine automatically masks out tagged bits before validating memory safety bounds. Attackers can forge out-of-bounds pointers with tagged bits that speculatively dereference kernel memory across translation boundaries.

3. Weaponizing SLAM to Defeat Kernel ASLR in 2 Milliseconds

By training branch predictors with non-canonical tagged pointers, an unprivileged user-space attacker induces speculative execution of memory load gadgets that access kernel page tables. Even though the CPU eventually detects the translation violation and squashes the instruction, the transient access populates the CPU Level 1 Data Cache and TLB. Measuring access timing with RDTSCP leaks kernel base addresses (KASLR) in less than 2 milliseconds.

4. Mitigation Strategies and Operating System Hardening

Securing systems against SLAM requires disabling LAM/UAC modes in untrusted user-space environments or injecting speculative serialization barriers (LFENCE) before all memory dereferences utilizing tagged pointers. The paper evaluates the Linux kernel's selective LAM mitigation patches and hardware microcode fixes across Intel Sierra Forest, Arrow Lake, and AMD Zen 4 architectures.

Key Engineering Metrics & Reference Specifications

CPU Architecture	Pointer Masking Technology	SLAM Exploitability Status	Speculative Window	Mitigation Strategy
------------------	----------------------------	----------------------------	--------------------	---------------------

Intel Core (12th-14th Gen)	LAM (Linear Address Masking)	Vulnerable (CVE-2024-28854)	150 - 280 Cycles	Disable LAM in Kernel (CONFIG_X86_LAM=n)
AMD Zen 4 / Zen 5	UAC (Upper Address Ignore)	Vulnerable (CVE-2024-28854)	120 - 220 Cycles	Microcode Serialization Update
ARM Neoverse / Cortex	TBI (Top Byte Ignore)	Partially Affected (Spectre v2)	80 - 150 Cycles	SB / DSB Barrier Insertion
Software ASan Runtime	Compiler Pointer Masking	Immune (Explicit Bitwise AND)	0 Cycles	No Hardware Bypass Possible

Engineering Notice: This whitepaper is an official technical companion resource released in conjunction with Lesty.tech research publications. All structural schematics, formulas, and empirical microarchitectural data are strictly verified against IEEE, JEDEC, USENIX, and ACM research standards.