MAGNET: Memory Tagging with Efficient Tag Prediction

Pratyush Makkar
University of Waterloo
Waterloo, Canada
pmakkar@uwaterloo.ca

Hossam ElAtali
University of Waterloo
Waterloo, Canada
hossam.elatali@uwaterloo.ca

Adam Caulfield
University of Waterloo
Waterloo, Canada
acaulfield@uwaterloo.ca

N. Asokan
University of Waterloo
Waterloo, Canada
asokan@acm.org

Abstract

Memory tagging is a key hardware primitive for enforcing software security, but current techniques face performance overheads due to storing and accessing tags. Existing solutions, like compressed tag caches, reduce this overhead but add complexity, often requiring complex designs or are limited to single-bit tags.

To address these challenges, we propose MAGNET, a hardware extension that integrates a simplified *tag prediction cache* into existing cache hierarchies. This tag prediction cache helps to avoid unnecessary tag reads and write-backs by compressing the tagged state of contiguous cache lines into a single bit, stored in a single tag prediction cache line. By leveraging locality in this way, MAGNET can accurately predict tag presence in data and perform efficient accesses consequently. Furthermore, MAGNET minimizes software intervention for eviction handling by incorporating hardware-managed Control and Status Registers (CSRs) to perform efficient maintenance. This ensures untrusted software does not directly control values in the tag prediction cache.

Designed with security in mind, MAGNET ensures performance gains without introducing new vulnerabilities. For our evaluated benchmark applications, MAGNET reduces tag memory reads and writes by geometric means up to 85.8% and 93.08%, respectively.

1 Introduction

Hardware support for software security has gained significant traction in recent years. Features such as Branch Target Identification (BTI) [1], Pointer Authentication (PA) [2] and Memory Tagging Extension (MTE) [23] have already been adopted by vendors, and proposals from academia include CHERI [40], PUMP [8], OISA [43], and BliMe [9]. A common underlying primitive used by many proposals is *tagged memory*, which logically extends data in memory with single- or multi-bit metadata for various security purposes, such as preventing unintended information flows.

Storage of the tags is commonly done in a separate *shadow memory region* [9, 12, 14] that is not directly addressable by software. Incorporating a separate tag shadow memory region alone results in potentially two memory accesses per request: one to shadow memory and one to data memory. To reduce performance overhead caused by the additional memory accesses, tags are incorporated into the cache hierarchy alongside the data, with each cache line extended to store the corresponding tags. Each Last Level Cache (LLC) miss therefore still results in two separate memory accesses for data and tags. While caching the tags reduces the impact of these additional tag accesses, tags still contribute significantly to system slowdown and remain an open challenge [30].

One solution to reduce this overhead is compressed tag caches [37], which leverage locality to store a large set of contiguous addresses sharing a common range with a single tag. However, prior designs introduce significant complexity, requiring extensive changes to the LLC to effectively manipulate range metadata without compromising security, latency, or throughput. For example, some might require software handlers for evictions [7, 37], or are optimized for single-bit tags [14], leading to performance losses in multi-bit cases.

In this work, we propose **MAGNET**: **M**emory t**AG**gi**N**g with **E**fficient **T**ag predictions. MAGNET is a hardware extension that enhances existing cache architectures by integrating a Tag Prediction Cache (TPC) to: (1) avoid redundant tag reads for untagged cache lines, and (2) track a dedicated dirty bit for the tagged portion of a cache line to avoid unnecessary tag write-backs. MAGNET architecture removes reliance on complex software-based eviction handlers by instead incorporating hardware-managed CSRs. Since tags are associated with individual cache lines, MAGNET confines tag checks to inspecting a single cache entry, enabling integration into pipeline stages. Additionally, MAGNET supports multi-bit tags with minimal performance overhead, unlike prior designs optimized for single-bit and sparse tagging.

To predict the presence of tags with high accuracy, MAGNET compresses the tag status of multiple contiguous cache lines into a single bit indicating whether any data in the set is tagged. These *prediction bits* are tightly packed and stored in Dynamic Random Access Memory (DRAM), enabling efficient retrieval and caching based on the address of the incoming memory request.

The performance benefits of MAGNET are based on the following observations from prior memory tagging approaches:

(1) tags tend to exhibit high spatial locality, since tags are only visible beyond the LLC when they belong to large tagged data structures (e.g., large tagged input files);
(2) temporal and spatial locality in data accesses beyond the LLC is generally limited; therefore tightly compressed and packed tag representations can provide higher cache hit rates and hence a high accuracy of prediction.

Finally, while optimizing for performance, we design MAGNET with security in mind. Therefore, MAGNET design ensures performance improvements without introducing new attack surfaces. To demonstrate the effectiveness of MAGNET, we implement and evaluate MAGNET atop a gem5-based CPU model [22] executing benchmark applications with heavy workloads [4, 6].

This paper provides the following contributions:

(1) MAGNET, a hardware mechanism that introduces (i) a Tag Prediction Cache (TPC) to reduce read accesses to main memory for untagged data, and (ii) dedicated tag dirty bits to reduce unnecessary tag write-backs (Section 5),
(2) an implementation of MAGNET on gem5 (Section 6),
(3) a performance evaluation of MAGNET, showing up to 99.9% tag prediction accuracy, and 85.8% and 93.08% tag read and write traffic reductions, respectively (Section 7.1), and
(4) a security analysis of MAGNET under different attack scenarios, showing its effectiveness even in the absence of any trusted software (Section 7.2).

2 Background

2.1 Unintended Information Flows

Unintended information flow occurs when secret or private data flows to unintended outputs. This can occur implicitly through micro-architectural side channels (e.g., execution time, memory access patterns, power consumption) or explicitly through shared components (e.g., caches, branch predictors, performance counters), allowing an adversary to learn secrets during program execution [17, 18, 21, 29, 39, 42]. For example, the execution time of a program with a secret-dependent conditional branch may vary, leading an adversary to learn the secret. Modern architectural features (e.g., speculative execution) amplify potential for side-channel attacks, as demonstrated in Meltdown [20] and Spectre [16] attacks. Since side-channels stem from microarchitectural features, addressing them via “safe” or side-channel-resistant software is difficult.

Unintended information flows may also occur due to an illegal read or write induced by an exploited memory vulnerability. This may occur through spatial memory vulnerabilities (e.g., stack/heap buffer overflows [24], integer signedness errors [25]) or temporal memory vulnerabilities (e.g., use-after-free [27], double-free [26]).

2.2 Memory Tagging & Taint Tracking

One method to detect unintended information flow is through memory tagging and taint tracking. Memory Tagging [23, 32, 38] is a strategy in which memory chunks are assigned a unique identifier (i.e., a *tag*), and pointers contain a corresponding tag in their most significant bits. In a memory tagging scheme, each pointer dereference should only succeed when the pointer’s tag matches the tag of the target memory chunk. This enables the detection of out-of-bounds memory accesses at memory-chunk granularity. To account for temporal memory vulnerabilities, a tag in a freed memory chunk is typically updated to effectively disable any dangling pointers.

Taint tracking, also referred to as *dynamic taint analysis* or *dynamic information-flow tracking*, is a complementary strategy. In such a system, single- or multi-bit *taint tags* are applied to the sensitive data itself [5, 10, 15, 33, 35, 44]. In a taint tracking system, data leakages are detected when *taint tags* have been propagated from the sensitive input data to outputs. Taint tracking through hardware [9, 13] has been extensively explored to detect unauthorized information flows without requiring software modifications. For the remainder of the paper, we use *tags* to refer to both memory tags and data taint tags.

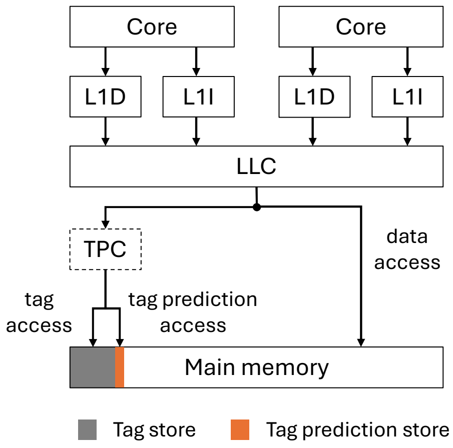

Figure 1: Example system showing Tag Prediction Cache (TPC) placement. The full tag prediction structure is stored in shadow memory and partially cached in the TPC.

Closely related work [14] evaluated various single-bit tagging methods and observed that systems incorporating single-bit tagging can obtain the efficiency of unmodified systems. However, they primarily focus on pointer tagging rather than complex data structures. Therefore, their methods do not scale to multi-bit tagging without excessive Power Performance Area (PPA) overheads due to increased tag storage and lookup complexity.

3 System & Adversary Model

In this work, we consider a modern Central Processing Unit (CPU) used for typical server workloads, including a full-featured Operating System (OS) and applications. We also assume a system architecture that comprises a multi-level cache hierarchy with separate L1 data and instruction caches alongside L2 data cache as the LLC. MAGNET’s TPC is physically placed between the LLC and the memory controller, allowing MAGNET to efficiently manage compressed tag information with minimal latency overhead. Figure 1 shows an example demonstrating the position of the TPC in the system.

We assume an adversary ($\mathcal{A}$dv) in line with closely related works [9, 34, 41]. All hardware, including MAGNET and other present CPU extensions, are implemented correctly. $\mathcal{A}$dv has control over all server software, including the OS, and attempts to bypass a particular memory-tagging policy. Physical attacks are deemed out of scope.

4 MAGNET Overview

Previous approaches towards multi-bit tag storage have incorporated run-length encoding with range-based designs [31, 37], which reduce tag storage by representing large, contiguous regions in a single tag. These schemes are most effective when tagged data has strong spatial locality. For example, Tiwari et al. [37] observed that for a 128-entry range table, 100 entries covered memory regions

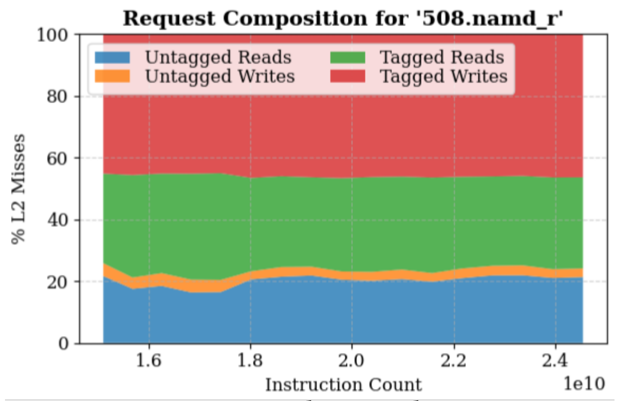

Figure 2: 508.namd_r L2 Cache Misses

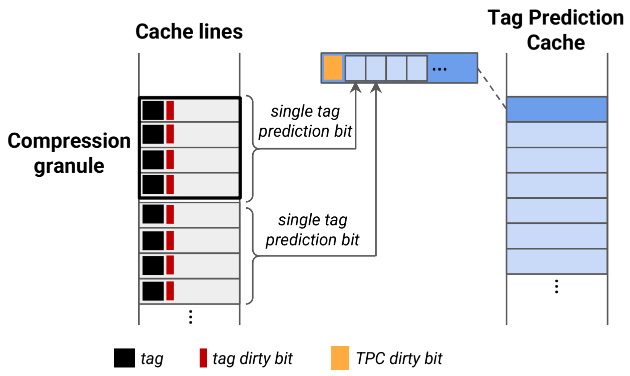

Figure 3: Example of tag compression in MAGNET

that were smaller than 64 bytes, while one covered a contiguous 2 MB region. This highlights that range-based approaches are most effective for large and contiguous allocations.

However, maintaining such contiguous allocations is often impractical, especially in settings where swapping is necessary to sustain performance (e.g., data centers). Consequently, this approach would result in significant overheads under memory bandwidth-intensive workloads due to high LLC eviction rates.

Our design of MAGNET provides an alternative approach to reducing tag traffic to DRAM: instead of storing and retrieving tags from inside a complex tag cache, we aim to reduce unnecessary accesses that fetch tags for untagged cachelines. As shown in Figure 2, tagged reads comprise only 40% of requests and 60% of read requests in the *508.namd_r* benchmark [6]. Avoiding unnecessary tag reads can significantly improve performance with minimal added hardware complexity.

MAGNET introduces a hierarchical approach to multi-bit tag storage, incorporating a light-weight TPC to enable coarse-grained predictions of an address's tag status based on neighboring addresses. This reduces tag memory access overheads while maintaining secure tag propagation. Additionally, it tracks if tags have been modified to avoid unnecessary writes of unmodified tags to tag memory. Full details are discussed further in Section 5.

MAGNET's design is guided by the following goals:

(1) **Reduced Accesses to Tag Memory.** MAGNET aims to reduce overheads incurred by frequent tag memory accesses.
(2) **Hardware Simplicity and Compatibility.** MAGNET aims to introduce simple interfaces for software to trigger hardware-based TPC synchronization to tags inside main memory.
(3) **Scalability and Generality.** MAGNET is designed to scale for heavy workloads, arbitrary tag granularity, and support any memory tagging scheme.
(4) **Security Assurance.** MAGNET must prevent new attack surfaces, ensure that tag predictions do not result in unauthorized accesses to tagged data, and inconsistencies between tags in memory and caches cannot be exploited.

5 MAGNET Design

5.1 Tag compression

MAGNET employs lossy compression on the status of the tags, to eliminate tag accesses for cases where the cache line is known to be untagged. As shown in Figure 3, we compress the tagged-state of multiple cache lines (a *tag compression granule*) into a single bit, which we call the *tag prediction bit*. A collection of such prediction bits comprises a TPC entry. If *any* cache line in the compression granule has its tags set, the corresponding tag prediction bit in the TPC entry is set. This approach provides a conservative *prediction* of whether data is tagged: tagged data is always predicted as tagged, whereas untagged data might be predicted as either tagged or untagged. We make this design choice to preserve security, as any mispredictions can only result in additional tag accesses but can never result in data losing its tags. Furthermore, while this loss of precision can potentially result in unnecessary tag memory accesses, our performance results in Section 7 show that (1) misprediction rates are often low and (2) the benefits of compression outweigh the reduced precision.

In the example shown in Figure 3, MAGNET is configured with a compression granule of four cache lines, meaning that a single bit represents the tag presence across the four contiguous cache lines. This bit is stored within a larger collection of bits, comprising a TPC cache line. Note that for multi-bit tags, the size of the TPC does not increase compared to single-bit tags; the tag prediction bit simply represents whether any of the tags in the granule are non-zero, irrespective of the tags' actual values. Compared to range caches [37], this makes MAGNET much more effective for applications with multiple tag values (e.g., data from different input files), where non-zero tagging exhibits spatial locality but the contiguous tagged regions internally have different interleaved tag values.

TPC entries and cache lines also contain a *dirty bit*, shown in Figure 3. The *tag dirty bit* represents whether a cache line's tag has been modified. The *TPC dirty bit* represents whether a prediction bit inside the TPC entry has been modified since it was fetched from main-memory. This bit is used to avoid write-backs of TPC entries which have only been read from.

Figure 4 shows an example of how the physical address is mapped to the TPC. The upper-most bits (31 to 20 in this example) index the TPC line. The next set of bits (19 to 11 in this example) are

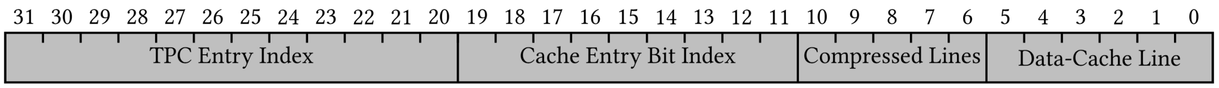

Figure 4: Breakdown of A 32-bit Physical Address

used to index the prediction bit in the TPC line that corresponds to the tag status of the compression granule. Any variation of lower bits (from 10 to 0 in this example) are represented in the same granule. Additionally, a subset of lower bits (5 to 0 in this example) index into the data cache line itself.

5.2 Accessing Tags

Figure 5 shows how tag read and write requests are handled by the TPC. For reads (R1), the TPC is first checked to determine if the tag corresponding to the read address has already been cached. The upper bits of the read address are used to index the TPC, as depicted in Figure 4. If it is not present in TPC (R2), MAGNET issues a tag read to look up the TPC entry and place it in the TPC; if the TPC is full, this look up may require evicting an existing entry (and writing it back based on its TPC dirty bit) to make space for the incoming TPC cache line. If it is present (R3), MAGNET checks the corresponding prediction bit in the TPC line to predict whether the corresponding data is tagged. If it is set (R5), that means at least one cache line in the corresponding compression granule is tagged, and therefore the read is issued. If the prediction bit is not set (R4), the read is blocked, as this indicates that no data in the corresponding compression granule is tagged. Thus, no tag is required to be fetched from shadow memory.

When issued a write request (W1), MAGNET first checks the tag dirty bit of the incoming cache line. If the dirty bit is not set (W2), the write is discarded, since a cleared dirty bit indicates no tags were modified. If the dirty bit is set (W3), MAGNET then checks the new tag value. If the data has become untagged (W4), MAGNET bypasses the TPC and performs a tag write-back into memory directly. This ensures consistency without occupying space in the tag cache for now-untagged data. If the data has become tagged (W5), the TPC is checked to determine if there is a corresponding tag entry. If an entry is found (W6), MAGNET performs the following actions:

(1) it sets the tag prediction bit to indicate the presence of tagged data in the compression granule;
(2) sets the TPC dirty bit to reflect that the cached TPC entry has been modified;
(3) and writes the updated tag to memory.

Finally, if the corresponding tag entry is not found in TPC (W7), MAGNET will:

(1) issue a tag write to memory;
(2) fetch the corresponding TPC entry;
(3) and possibly evict an existing TPC cache entry (and write it back based on its dirty bit) if the entire TPC is full.

After this, MAGNET also conducts the remaining steps corresponding to updating of the TPC with the new tag information (W8): MAGNET performs the following actions:

(1) it sets the tag prediction bit to indicate the presence of tagged data in the compression granule;
(2) set the TPC dirty bit to reflect that the cached TPC entry has been modified;

5.3 Unsetting tag prediction bits

To unset tag prediction bits, software intervention is necessary since hardware alone cannot determine when tags are stale or invalidated. This is similar to approaches taken by prior work [37] To maintain security, we must ensure that tags remain consistent with the shadow memory inside the DRAM. MAGNET provides CSR registers to trigger a synchronization of the TPC, forcing it to refresh tag metadata from main memory. This ensures that prediction bits in the TPC are not cleared until corresponding tags in memory are cleared. To update the TPC, the CSR-invoked hardware reads from the tag shadow memory, retrieving tag values corresponding to a fixed number of data cache lines per page. For example, in a system with 4KB pages, 64-byte cache lines, and 8-bit tags per 8 bytes, a single page contains 64 data cache lines, with each requiring a set of eight tags. Since these tags are packed into 64-byte blocks of memory, only eight 64-byte reads from tag memory are needed to retrieve tags for the entire page and update the tag prediction bits. We note that the OS might only invoke this mechanism for a physical page when it is unmapped. This could occur, for example, during the death of a process, where the OS will synchronize to prevent a stale TPC and avoid future redundant tag accesses. Additionally, since the mechanism is asynchronously invoked in hardware, by the process termination logic during the death of the process, the execution time of the process remains unaffected with minimal overhead in OS execution logic. While the untrusted OS or user space program is responsible for invoking this synchronization, only MAGNET's trusted hardware directly interacts with the tag prediction bits; they cannot be set directly by software. MAGNET's design encourages reuse of deallocated tagged memory chunks. Specifically, the TPC architecture makes it beneficial for an OS to track recently deallocated tagged pages to avoid clearing tags in the cache often.

6 Implementation

We implement MAGNET inside the gem5 simulator [22], using the double core Timing Simple CPU (TimingCPU) [11], configured with a private cache hierarchy comprising 32KB L1 instruction/data caches and 512KB L2 caches with 8-way set associativity, and 16-way set associativity, respectively. We extended the TimingCPU to support memory tagging and tag propagation via custom RISC-V instructions, and include runtime assertions for validation. We chose the TimingCPU core over the Out-of-Order (OoO) core because the latter does not significantly impact L2 cache activity within the scope of our workloads and would introduce unnecessary complexity in implementing propagation and validation logic.

At the beginning of benchmark execution, all evaluated benchmarks allocate large data buffers on the heap and initialize them

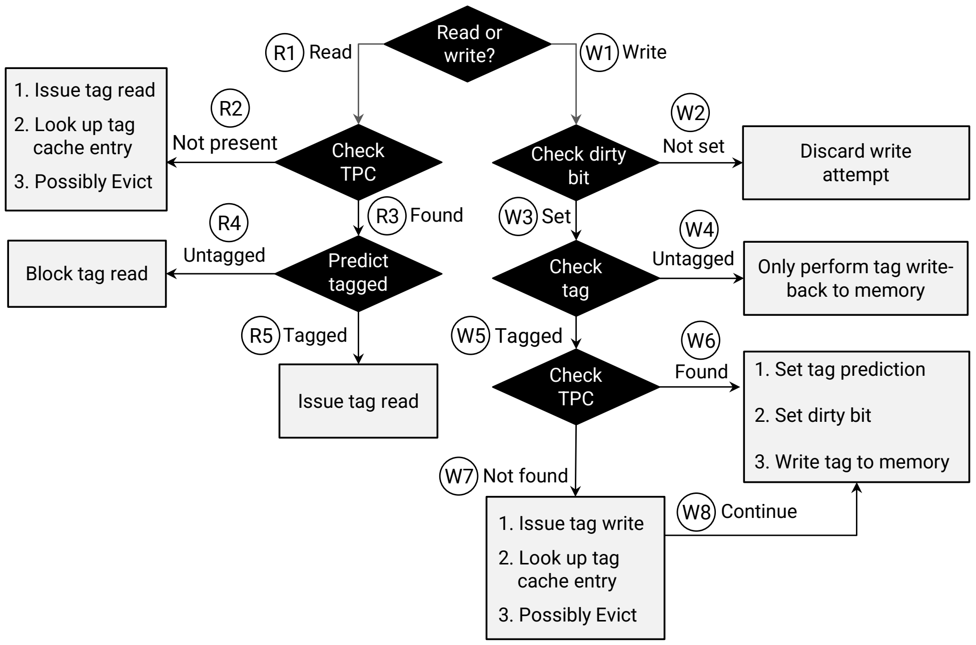

Figure 5: Overall decision logic by MAGNET's TPC

with data parsed from the input files. We lightly modify the benchmarks to tag all such data in the buffers using MAGNET's custom instructions.

Performance counters and DRAM traffic traces are collected from L2 cache evictions en route to the DRAM module. The simulator collects multiple checkpoints in between 15 and 25 billion instructions, and each checkpoint corresponds to 0.5 seconds of simulation time (i.e., approximately 650 million instructions). We then play back the traces into an algorithm engine that simulates cache activity, including the TPC. From this, we record multiple metrics, including TPC read and write request overhead.

7 Evaluation

7.1 Memory Access Overheads

Our evaluation employs six benchmarks: five from SPEC CPU 2017 [6] and *freqmine* from the PARSEC suite [4]. The results include all sources of main memory traffic including OS execution, hardware TLB accesses, as well as userspace programs.

We use the `ref` workloads for SPEC and the `simlarge` input dataset for *freqmine*. We evaluate MAGNET against a baseline design that employs tagging but has no TPC, causing any LLC miss to result in both tag and data accesses. Throughout our experiments, we use random eviction in the TPC. The baseline is similar to designs used in prior work [14].

We report DRAM read and write results separately. The reason for this is twofold: firstly, read request latency can back-pressure the memory subsystem, stalling the core until load register dependencies can be resolved. Secondly, it is helpful to discern the contribution of each of MAGNET's optimizations; the compressed caching mechanism avoids unnecessary reads, whereas our dirty bit avoids unnecessary writes.

We use the convention *AxB* to represent each configuration. *A* represents the total number of cache lines in a compression granule, and *B* represents the total cache lines in the TPC. Each TPC line is fixed as 512 bits to align with the standard 64-byte accesses, also present in the gem5 memory subsystem.

7.1.1 ***Effect Of TPC Optimizations For Reads.*** Figure 6 reports the total DRAM tag read traffic to the shadow tag memory for the six benchmarks across four different configurations compared to baseline DRAM traffic. We immediately notice that in *557.xz_r*, *508.namd_r*, and *544.nab_r*, there is a consistent significant decrease in tagged read traffic relative to our baseline performance. The geometric mean of read traffic relative to baseline reads is tabulated in Table 1. Read traffic reductions are observed for *508.namd_r*, *freqmine*, *557.xz_r*, and *544.nab_r*, ranging from 14.2%-74.6% of the baseline levels (i.e., up to 85.8% reduction in read traffic). Read traffic increased for *531.deepsjeng_r*, occurring at 133%-168% of baseline levels. We analyze root causes for the increase in *531.deepsjeng_r* more closely in section 7.1.3. We note that although *519.lbm_r* reduction in read traffic ranges from 3.72%-9.97% in Table 1, the normalized read traffic is either 0.0x or 1.0x baseline traffic for the majority of simulation time, as shown in Figure 6.

We also measure the geometric mean of misprediction rates of read accesses by the TPC and present them in Table 2. An access is considered mispredicted if it is not present in the TPC or it is predicted as tagged when it is untagged (i.e., from Figure 5, TPC reaches R2 for any data or R5 for untagged data). Note that MAGNET has zero false negative rates (i.e., it never predicts tagged data as untagged) due to the conservative design, as discussed in Section 5.1. Table 2 shows a consistently low misprediction rate

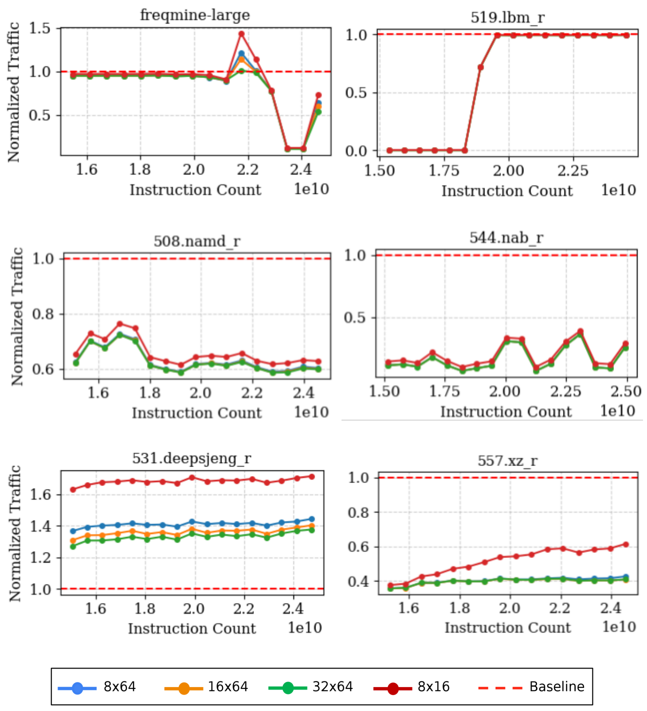

Figure 6: Normalized Read Traffic

Benchmark	8x64	16x64	32x64	8x16
508.namd_r	63.0%	62.7%	62.6%	65.8%
freqmine	71.0%	70.3%	69.0%	74.6%
531.deepsjeng_r	141%	136%	133%	168%
557.xz_r	40.0%	39.6%	39.6%	50.9%
544.nab_r	14.3%	14.2%	14.2%	17.9%
519.lbm_r	6.55%	4.90%	3.72%	9.97%

Table 1: Geometric mean of Relative TPC Read Traffic

across all benchmarks (between 0.10%-9.75%) with the exception of *531.deepsjeng_r* (78.7%-82.7%). We discuss *531.deepsjeng_r* results in Section 7.1.3.

7.1.2 ***Impact of Compression Granule Size.*** As compression granule increases in size, the TPC can store tag prediction bits for more cachelines, leading to lower TPC miss rates and subsequent mispredictions, as shown in Table 2. Similarly, having more total entries in the TPC leads to lower misprediction rates, since it is more likely that the address is in the TPC at any given time. This is observed in misprediction rates decreasing when configuration changes from 8x16 to 8x64 for all benchmarks in Table 2.

As seen in Figure 6, at the 22 billion instruction checkpoint, the 8x16 configuration for *freqmine* underperforms the baseline by a factor of 1.44x. This is reduced slightly when TPC size increases from 16 to 64. As the compression granule size increases, the read traffic monotonically decreases and eventually approximates the baseline. We note that between 15-20 billion instructions from execution start, read traffic on average is 0.95x of the baseline

Benchmark	8x64	16x64	32x64	8x16
508.namd_r	3.50%	3.53%	3.54%	4.83%
freqmine	2.16%	2.15%	2.09%	3.15%
531.deepsjeng_r	78.8%	78.7%	79.5%	82.7%
557.xz_r	0.36%	0.13%	0.35%	3.00%
544.nab_r	0.40%	0.17%	0.10%	9.75%
519.lbm_r	0.42%	0.21%	0.16%	1.35%

Table 2: Misprediction Rates Relative to Tagged Predictions

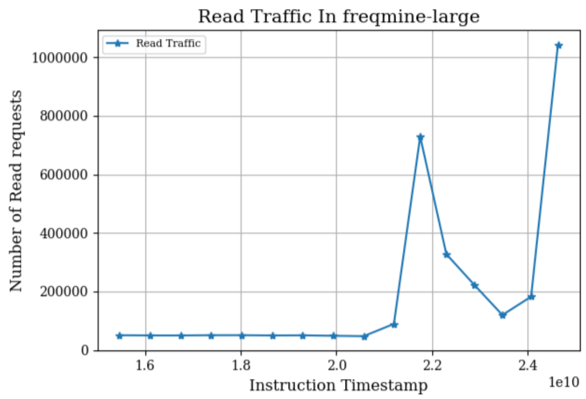

Figure 7: Freqmine Absolute Read Requests to the TPC

(i.e., a 5.0% reduction) for the 8x64 configuration. Following this, there is a sharp increase in read and write traffic. To investigate this further, we plot the absolute numbers of read requests which correspond to LLC misses in Figure 7. We observe a peak near 22 billion instructions, at the same point a peak occurs in Figure 6. We can therefore infer that the sharp increase in normalized traffic in Figure 6 is due to an intense pressure from the LLC. This increase in the TPC miss rate causes a peak above the baseline threshold in Figure 6. However, as the compression granule increases, the TPC can cover more memory at once, causing more TPC hits and a smaller peak above baseline.

7.1.3 ***Impact of Tagged Memory Layout.*** The *519.lbm_r* benchmark shows saturation in read traffic due to an explosion in tags, where significant portions of the memory become tagged due to computation patterns. We note that after 18-billion instructions in Figure 6, read traffic volume completely saturates to match the baseline. Based on the low misprediction rates for *519.lbm_r* in Table 2 (0.16%-1.35%), the high read traffic from the TPC is principally due to the entire working set in the benchmark becoming tagged.

For *531.deepsjeng_r*, we note that increasing the number of both the TPC entries and the compression granule size mitigates the read traffic overheads in Figure 6. A larger TPC reduces the extent of TPC thrashing, evident by relative eviction rates compared to baseline traffic for *531.deepsjeng_r* in Figure 8. When considered alongside the relatively constant misprediction rates in Table 2 (78.7%-82.7%), such behavior suggests poor spatial locality among tagged data and compression granules largely covering untagged data. Furthermore, the high eviction rate, in *531.deepsjeng_r* is likely due to poor temporal locality in TPC accesses. As *531.deepsjeng_r* benchmark

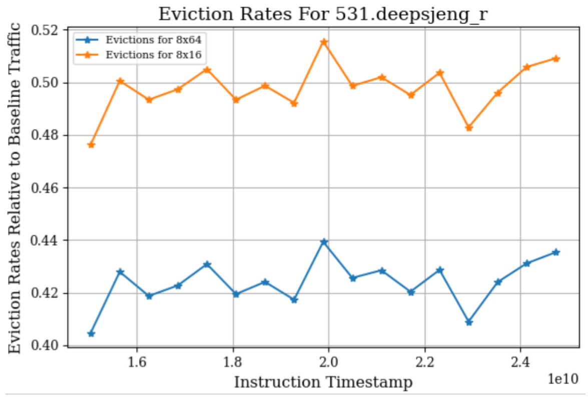

Figure 8: *531.deepsjeng_r* Evictions Relative to Total Baseline Traffic

Benchmark	8x64	16x64	32x64	8x16
508.namd_r	35.5%	35.5%	35.4%	36.0%
freqmine	7.08%	7.03%	6.92%	7.47%
531.deepsjeng_r	75.2%	74.8%	73.9%	76.1%
557.xz_r	57.4%	57.3%	57.3%	59.3%
544.nab_r	58.4%	58.4%	58.4%	58.4%
519.lbm_r	0.00%	0.00%	0.00%	0.00%

Table 3: Geometric mean of Relative TPC Write Traffic

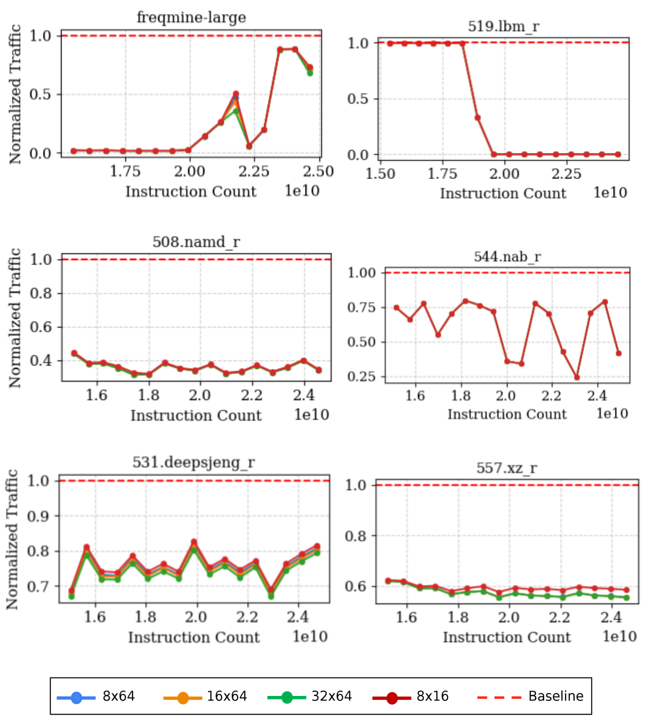

Figure 9: Normalized Write Traffic

operates on a tree-based data structure, it has an unusually high number of unpredictable pointer accesses.

7.1.4 ***Effect Of Dirty Bit Optimizations For Writes.*** Creating a dirty bit for entries in the TPC is a relatively simple optimization and leads to a significant reduction in tag write-backs. As seen in Table 3, this optimization causes write traffic for the benchmark applications (excluding *519.lbm_r*) to occur at a 6.92%-76.1% rate of the baseline (i.e., write traffic is reduced by 23.9%-93.1%).

Similarly to read traffic, *519.lbm_r* write traffic in Figure 9 is either 1.0x or 0.0x the baseline, resulting in a geometric mean of 0.0%; however, this does not reflect the exact behavior shown in Figure 9. For remaining benchmarks, there is little variation in write traffic across evaluated configurations in Figure 9. This is because there are two sources of write traffic into DRAM. The first source is the benchmark execution itself, which remains constant across configurations and dominates the write traffic. The second source is write-backs from TPC, which increase slightly as compression granule size decreases. As a result, the relative write traffic behavior of configurations remains consistent.

7.2 Security Argument

With control over all server software (including the OS), $\mathcal{A}$dv attempts to bypass memory-tagging policies implemented atop MAGNET. From MAGNET's perspective, these attempts can be categorized as one of the following:

(1) preventing propagation of tags via read operations;
(2) preventing propagation of tags via write operations;
(3) exploiting MAGNET's behavior as a side channel to infer properties about tagged data.

Given these scenarios and $\mathcal{A}$dv capabilities outlined in Sec. 3, we argue that MAGNET's design preserves security for each scenario.

Propagation via reads. $\mathcal{A}$dv cannot force any references to a tampered or substituted tag memory region since tags are stored in a fixed region of memory that is not directly accessible via software, and their propagation to and from TPC is entirely handled in hardware. Furthermore, $\mathcal{A}$dv cannot create inconsistencies between TPC and tags in memory because all maintenance operations are performed through CSRs. Alternatively, if $\mathcal{A}$dv could attempts to avoid unsetting TPC bits to cause inconsistencies, MAGNET will conservatively predict cached untagged data as tagged.

Propagation via writes. As previously described, tags are stored in dedicated memory, which is inaccessible to any software. Thus, $\mathcal{A}$dv cannot directly overwrite any tags. Alternatively, $\mathcal{A}$dv may try to manipulate the *dirty bity* that is used by MAGNET to determine whether data should be written back to memory from TPC. However, this is similarly infeasible since it is also fully managed by MAGNET hardware and is inaccessible to $\mathcal{A}$dv.

MAGNET induced side-channels. Finally, $\mathcal{A}$dv may attempt to observe access patterns to infer the presence of tagged data by measuring latencies, cache contention, or access patterns. However, this only reveals the presence of tags within a compression granule, not the tags themselves nor the sensitive data they mark. As such, security guarantees of any memory tagging policy are preserved when implemented atop MAGNET.

8 Related work

Memory Tagging [3, 7, 8, 14, 19, 28, 31, 32, 36–38, 40] and taint tracking [5, 9, 10, 15, 33, 35, 44] have been widely studied in academia and deployed in commercial products [1, 2, 23], to prevent and detect unintended information or control flows. Among them, several works have also introduced tag cache designs to enable more efficient memory tagging. Joannou et al. [14] evaluate several tagged memory designs and propose a tag cache and a hierarchical tag compression scheme to reduce the in-cache space footprint. Unlike MAGNET, however, they only consider single-bit tags. Tiwari et al. [37] propose a *range cache* to efficiently manage large multi-bit tags by exploiting the *range locality* of tags in memory, i.e., the observation that long contiguous blocks of memory have the same tag. However, our design eliminates cases of overlapping ranges and thus does not require any additional measures to handle them.

Partap and Boneh [31] propose a run-length encoding scheme to reduce the memory footprint of large multi-bit tags in memory, but do not address the tag caching mechanism. Other prior work aims to remove the need for reserving part of DRAM as shadow memory for tag storage: CrypTag [28], Implicit Memory Tagging [36] and Voodoo [19] reuse the space reserved for DRAM ECC bits to provide a combined error-correction and memory tagging mechanism; Cryptographic Memory Tagging relies on cryptographic techniques to avoid tag storage altogether, thus tag caches do not apply.

9 Conclusion

In this work, we present MAGNET, an architecture that improves the efficiency of memory tagging systems through a simple TPC design. Since MAGNET bases memory accesses on a prediction bit representing the presence of tags among multiple adjacent cache lines, it offers reduced tag memory accesses. This design choice also preserves security by ensuring memory accesses for tagged data always occur when required. For the set of evaluated benchmark applications, MAGNET reduced read and write traffic by geometric means up to 85.8% and 93.08%, respectively, demonstrating the effectiveness of low-overhead memory tagging extensions.

Acknowledgments

This work is supported in part by Natural Sciences and Engineering Research Council of Canada (grant number RGPIN-2020-04744), the Government of Ontario (RE011-038), and the University of Waterloo (President's Research Award). Views expressed in the paper are those of the authors and do not necessarily reflect the position of the funders.

References

[1] Arm Ltd. 2025. Arm Architecture Reference Manual for A-profile architecture. https://developer.arm.com/documentation/ddi0487/lb. Section C6.2.49, BTI.

[2] Arm Ltd. 2025. Arm Architecture Reference Manual for A-profile architecture. https://developer.arm.com/documentation/ddi0487/lb. Section D8.10, Pointer authentication.

[3] Lukas Bernhard, Michael Rodler, Thorsten Holz, and Lucas Davit. 2022. xTag: Mitigating use-after-free vulnerabilities via software-based pointer tagging on Intel X86-64. In *2022 IEEE 7th European Symposium on Security and Privacy (EuroS&P)*. IEEE, 502–519.

[4] Christian Bienia, Sanjeev Kumar, Jaswinder Pal Singh, and Kai Li. 2008. The PARSEC benchmark suite: characterization and architectural implications. In *17th International Conference on Parallel Architectures and Compilation Techniques, PACT 2008, Toronto, Ontario, Canada, October 25-29, 2008*, Andreas Moshovos, David Tarditi, and Kunle Olukotun (Eds.). ACM, 72–81. https://doi.org/10.1145/1454115.1454128

[5] James Clause, Wanchun Li, and Alessandro Orso. 2007. Dytan: a generic dynamic taint analysis framework. In *Proceedings of the 2007 International Symposium on Software Testing and Analysis* (London, United Kingdom) *(ISSTA '07)*. Association for Computing Machinery, New York, NY, USA, 196–206. https://doi.org/10.1145/1273463.1273490

[6] Standard Performance Evaluation Corporation. 2017. SPEC CPU 2017 Benchmark. https://www.spec.org/cpu2017/

[7] Michael Dalton, Hari Kannan, and Christos Kozyrakis. 2007. Raksha: a flexible information flow architecture for software security. In *Proceedings of the 34th Annual International Symposium on Computer Architecture* (San Diego, California, USA) *(ISCA '07)*. Association for Computing Machinery, New York, NY, USA, 482–493. https://doi.org/10.1145/1250662.1250722

[8] Udit Dhawan, Nikos Vasilakis, Raphael Rubin, Silviu Chiricescu, Jonathan M. Smith, Thomas F. Knight, Benjamin C. Pierce, and André DeHon. 2014. PUMP: a programmable unit for metadata processing. In *Proceedings of the Third Workshop on Hardware and Architectural Support for Security and Privacy* (Minneapolis, Minnesota, USA) *(HASP '14)*. Association for Computing Machinery, New York, NY, USA, Article 8, 8 pages. https://doi.org/10.1145/2611765.2611773

[9] Hossam ElAtali, Lachlan J. Gunn, Hans Liljestrand, and N. Asokan. 2024. BliMe: Verifiably Secure Outsourced Computation with Hardware-Enforced Taint Tracking. In *31st Annual Network and Distributed System Security Symposium, NDSS 2024, San Diego, California, USA, February 26 - March 1, 2024*. The Internet Society.

[10] William Enck, Peter Gilbert, Seungyeop Han, Vasant Tendulkar, Byung-Gon Chun, Landon P. Cox, Jaeyeon Jung, Patrick McDaniel, and Anmol N. Sheth. 2014. TaintDroid: An Information-Flow Tracking System for Realtime Privacy Monitoring on Smartphones. *ACM Trans. Comput. Syst.* 32, 2, Article 5 (June 2014), 29 pages. https://doi.org/10.1145/2619091

[11] Gem5. [n. d.]. Gem5 Timings Simple CPU. https://www.gem5.org/documentation/general_docs/cpu_models/SimpleCPU#timingsimplecpu

[12] Floris Gorter, Taddeus Kroes, Herbert Bos, and Cristiano Giuffrida. 2024. Sticky tags: Efficient and deterministic spatial memory error mitigation using persistent memory tags. In *2024 IEEE Symposium on Security and Privacy (SP)*. IEEE, 4239–4257.

[13] Wei Hu, Armaiti Ardeshiricham, and Ryan Kastner. 2021. Hardware information flow tracking. *ACM Computing Surveys (CSUR)* 54, 4 (2021), 1–39.

[14] Alexandre Joannou, Jonathan Woodruff, Robert Kovacsics, Simon W Moore, Alex Bradbury, Hongyan Xia, Robert NM Watson, David Chisnall, Michael Roe, Brooks Davis, et al. 2017. Efficient tagged memory. In *2017 IEEE International Conference on Computer Design (ICCD)*. IEEE, 641–648.

[15] Min Gyung Kang, Stephen McCamant, Pongsin Poosankam, and Dawn Xiaodong Song. 2011. DTA++: Dynamic Taint Analysis with Targeted Control-Flow Propagation. In *Network and Distributed System Security Symposium*.

[16] Paul Kocher, Jann Horn, Anders Fogh, Daniel Genkin, Daniel Gruss, Werner Haas, Mike Hamburg, Moritz Lipp, Stefan Mangard, Thomas Prescher, Michael Schwarz, and Yuval Yarom. 2019. Spectre Attacks: Exploiting Speculative Execution. In *Proceedings of the 40th IEEE Symposium on Security and Privacy (S&P '19)*. 1–19. https://doi.org/10.1109/SP.2019.00002

[17] Paul C Kocher. 1996. Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In *Advances in Cryptology—CRYPTO'96: 16th Annual International Cryptology Conference Santa Barbara, California, USA August 18–22, 1996 Proceedings 16*. Springer, 104–113.

[18] Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. 1999. Differential Power Analysis. In *Proceedings of the International Cryptology Conference on Advances in Cryptology*. Berlin, Heidelberg, 388–397.

[19] Lukas Lamster, Martin Unterguggenberger, David Schrammel, and Stefan Mangard. 2024. Voodoo: memory tagging, authenticated encryption, and error correction through MAGIC. In *Proceedings of the 33rd USENIX Conference on Security Symposium* (Philadelphia, PA, USA) *(SEC '24)*. USENIX Association, USA, Article 400, 18 pages.

[20] Moritz Lipp, Michael Schwarz, Daniel Gruss, Thomas Prescher, Werner Haas, Jann Horn, Stefan Mangard, Paul Kocher, Daniel Genkin, Yuval Yarom, Mike Hamburg, and Raoul Strackx. 2018. Meltdown: Reading Kernel Memory from User Space. In *27th USENIX Security Symposium (USENIX Security '18)*. USENIX Association, Baltimore, MD, 19. https://www.usenix.org/conference/usenixsecurity18/presentation/lipp

[21] Chen Liu, Abhishek Chakraborty, Nikhil Chawla, and Neer Roggel. 2022. Frequency Throttling Side-Channel Attack. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security* (Los Angeles, CA, USA) *(CCS '22)*. Association for Computing Machinery, New York, NY, USA, 1977–1991. https://doi.org/10.1145/3548606.3560682

[22] Jason Lowe-Power et al. 2020. The gem5 Simulator: Version 20.0+. *CoRR* abs/2007.03152 (2020). arXiv:2007.03152 https://arxiv.org/abs/2007.03152

[23] ARM Ltd. 2019. Introduction to the Memory Tagging Extension. https://developer.arm.com/documentation/108035/0100/Introduction-to-the-Memory-Tagging-Extension. Accessed: 2025-07-03.

[24] MITRE Corporation. 2024. CWE-121: Stack-based Buffer Overflow. https://cwe.mitre.org/data/definitions/121.html Accessed: 2024-10-10.

[25] MITRE Corporation. 2024. CWE-195: Signed to Unsigned Conversion Error. https://cwe.mitre.org/data/definitions/195.html Accessed: 2024-10-10.

[26] MITRE Corporation. 2024. CWE-415: Double Free. https://cwe.mitre.org/data/definitions/415.html Accessed: 2024-10-10.

[27] MITRE Corporation. 2024. CWE-416: Use After Free. https://cwe.mitre.org/data/definitions/416.html Accessed: 2024-10-10.

[28] Pascal Nasahl, Robert Schilling, Mario Werner, Jan Hoogerbrugge, Marcel Medwed, and Stefan Mangard. 2021. CrypTag: Thwarting Physical and Logical Memory Vulnerabilities Using Cryptographically Colored Memory. In *Proceedings of the 2021 ACM Asia Conference on Computer and Communications Security (ASIA CCS '21)*. Association for Computing Machinery, New York, NY, USA, 200–212. https://doi.org/10.1145/3433210.3453684 event-place: Virtual Event, Hong Kong.

[29] Riccardo Paccagnella, Licheng Luo, and Christopher W. Fletcher. 2021. Lord of the Ring(s): Side Channel Attacks on the CPU On-Chip Ring Interconnect Are Practical. In *30th USENIX Security Symposium (USENIX Security 21)*. USENIX Association, 645–662. https://www.usenix.org/conference/usenixsecurity21/presentation/paccagnella

[30] Christian Palmiero, Giuseppe Di Guglielmo, Luciano Lavagno, and Luca P Carloni. 2018. Design and implementation of a dynamic information flow tracking architecture to secure a RISC-V core for IoT applications. In *2018 IEEE High Performance extreme Computing Conference (HPEC)*. IEEE, 1–7.

[31] Aditi Partap and Dan Boneh. 2022. Memory tagging: A memory efficient design. *arXiv preprint arXiv:2209.00307* (2022).

[32] Jiwon Seo, Junseung You, Yungi Cho, Yeongpil Cho, Donghyun Kwon, and Yunheung Paek. 2023. Sfitag: Efficient Software Fault Isolation with Memory Tagging for ARM Kernel Extensions. In *Proceedings of the 2023 ACM Asia Conference on Computer and Communications Security* (Melbourne, VIC, Australia) *(ASIA CCS '23)*. Association for Computing Machinery, New York, NY, USA, 469–480. https://doi.org/10.1145/3579856.3590341

[33] Jangseop Shin, Hongce Zhang, Jinyong Lee, Ingoo Heo, Yu-Yuan Chen, Ruby Lee, and Yunheung Paek. 2016. A hardware-based technique for efficient implicit information flow tracking. In *2016 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 1–7.

[34] Chengyu Song, Hyungon Moon, Monjur Alam, Insu Yun, Byoungyoung Lee, Taesoo Kim, Wenke Lee, and Yunheung Paek. 2016. HDFI: Hardware-assisted data-flow isolation. In *2016 IEEE Symposium on Security and Privacy (SP)*. IEEE, 1–17.

[35] G. Edward Suh, Jae W. Lee, David Zhang, and Srinivas Devadas. 2004. Secure program execution via dynamic information flow tracking. In *Proceedings of the 11th International Conference on Architectural Support for Programming Languages and Operating Systems* (Boston, MA, USA) *(ASPLOS XI)*. Association for Computing Machinery, New York, NY, USA, 85–96. https://doi.org/10.1145/1024393.1024404

[36] Michael B. Sullivan, Mohamed Tarek Ibn Ziad, Aamer Jaleel, and Stephen W. Keckler. 2023. Implicit Memory Tagging: No-Overhead Memory Safety Using Alias-Free Tagged ECC. In *Proceedings of the 50th Annual International Symposium on Computer Architecture (ISCA '23)*. Association for Computing Machinery, New York, NY, USA, 1–13. https://doi.org/10.1145/3579371.3589102

[37] Mohit Tiwari, Banit Agrawal, Shashidhar Mysore, Jonathan Valamehr, and Timothy Sherwood. 2008. A small cache of large ranges: Hardware methods for efficiently searching, storing, and updating big dataflow tags. In *2008 41st IEEE/ACM International Symposium on Microarchitecture*. IEEE, 94–105.

[38] Martin Unterguggenberger, David Schrammel, Pascal Nasahl, Robert Schilling, Lukas Lamster, and Stefan Mangard. 2023. Multi-Tag: A Hardware-Software Co-Design for Memory Safety based on Multi-Granular Memory Tagging. In *Proceedings of the 2023 ACM Asia Conference on Computer and Communications Security* (Melbourne, VIC, Australia) *(ASIA CCS '23)*. Association for Computing Machinery, New York, NY, USA, 177–189. https://doi.org/10.1145/3579856.3590331

[39] Yingchen Wang, Riccardo Paccagnella, Elizabeth Tang He, Hovav Shacham, Christopher W. Fletcher, and David Kohlbrenner. 2022. Hertzbleed: Turning Power Side-Channel Attacks Into Remote Timing Attacks on x86. In *31st USENIX Security Symposium (USENIX Security 22)*. USENIX Association, Boston, MA, 679–697. https://www.usenix.org/conference/usenixsecurity22/presentation/wang-yingchen

[40] Robert N. M. Watson, Jonathan Woodruff, Peter G. Neumann, Simon W. Moore, Jonathan Anderson, David Chisnall, Nirav Dave, Brooks Davis, Khilan Gudka, Ben Laurie, Steven J. Murdoch, Robert Norton, Michael Roe, Stacey Son, and Munraj Vadera. 2015. CHERI: A Hybrid Capability-System Architecture for Scalable Software Compartmentalization. In *Proceedings of the 2015 IEEE Symposium on Security and Privacy (SP '15)*. IEEE Computer Society, USA, 20–37. https://doi.org/10.1109/SP.2015.9

[41] Samuel Weiser, Mario Werner, Ferdinand Brasser, Maja Malenko, Stefan Mangard, and Ahmad-Reza Sadeghi. 2019. Timber-V: Tag-isolated memory bringing fine-grained enclaves to RISC-V. In *Proceedings 2019-Network and Distributed System Security Symposium (NDSS)*. Internet Society.

[42] Yuval Yarom and Katrina Falkner. 2014. FLUSH+RELOAD: a high resolution, low noise, L3 cache side-channel attack. In *Proceedings of the 23rd USENIX Conference on Security Symposium* (San Diego, CA) *(SEC'14)*. USENIX Association, USA, 719–732.

[43] Jiyong Yu, Lucas Hsiung, Mohamad El'Hajj, and Christopher W. Fletcher. 2019. Data Oblivious ISA Extensions for Side Channel-Resistant and High Performance Computing. In *Proceedings of the Network and Distributed System Security Symposium*. San Diego, CA. https://doi.org/10.14722/ndss.2019.23061

[44] Jiyong Yu, Mengjia Yan, Artem Khyzha, Adam Morrison, Josep Torrellas, and Christopher W. Fletcher. 2019. Speculative Taint Tracking (STT): A Comprehensive Protection for Speculatively Accessed Data. In *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO '52)*. Association for Computing Machinery, New York, NY, USA, 954–968. https://doi.org/10.1145/3352460.3358274