When Good Protections Go Bad: Exploiting Anti-DoS Measures to Accelerate Rowhammer Attacks

Misiker Tadesse Aga
misiker@umich.edu

Zelalem Birhanu Aweke
zaweke@umich.edu

Todd Austin
austin@umich.edu

University of Michigan

***Abstract*—The rowhammer vulnerability, where repeated accesses to a DRAM row can speed the discharge of neighboring bits, has emerged as a significant security concern in the computing industry. To address the problem, computer and software vendors have: *i)* doubled DRAM refresh rates, *ii)* restricted access to virtual-to-physical page mappings, and *iii)* disabled access to cache-flush operations in sandboxed environments. While recent efforts have shown how to overcome each of these protections individually, machines today are protected from rowhammer attacks if they employ all three of these protections simultaneously. In this paper, we demonstrate the first rowhammer attack that overcomes all three of these protections when used in tandem. Our attack is a virtual-memory based cache-flush free attack that is sufficiently fast to rowhammer with double rate refresh. The most astonishing aspect of our attack is that it is enabled by the recently introduced Cache Allocation Technology, a mechanism designed in part to protect virtual machines from inter-VM denial-of-service attacks. The subtext of this paper asks the question: "Is there any hope for system security, when the protections for one attack enable yet another?" We claim that the solution to this conundrum lies in the approach taken to protecting systems. Adopting a subtractive approach to secure systems, in contrast to additive measures, could go a long way toward building provably secure systems.**

I. Introduction

In recent years, hardware security concerns that have grown as more potential dangerous vulnerabilities have been demonstrated. These vulnerabilities range from side-channel attacks [16] to cache-based attacks [3] to virtual memory-based attacks [8]. Perhaps the greatest concern in recent years has been given to the rowhammer attack. This attack works by repeatedly accessing a row of DRAM within a single refresh cycle, which due to electrical coupling in densely packed DRAM speeds the discharge of the adjacent rows. Early demonstrations of this attack found that it was possible to flip bits for a variety of DRAM modules across a wide range of systems.

Shortly after the first demonstration of bit-flips with rowhammer attacks, researchers at Google Project Zero demonstrated two techniques to weaponize the attack [1]. The first demonstration had a program rowhammer itself to allow the execution of illegal instructions inside the Google Native Client sandbox (NaCl). Their second weaponization approach was perpetrated on the Linux OS, where a user program filled physical DRAM with page tables by repeatedly mmap'ing a file into its address space. The attack then rowhammers kernel physical DRAM with the intent of pointing one of the many created page table entries to another page table page, which has the effect of mapping a page table into the program's user address space. Once this occurs the program can manipulate its own address space mapping from user code.

Given the potential risks posed by rowhammer attacks, the computer industry has moved deliberately toward protections against these attacks. Specifically, three measures have seen broad use in commercial systems. These protections are *i)* double rate DRAM refresh, *ii)* restricted access to virtual memory mappings, and *iii)* restricted access to cache-flush operations in sandboxed environments.

Double-rate refesh reduces DRAM refresh periods from 64ms to 32ms, which has the effect of increasing refresh overheads, but at the same time, it cuts in half the time allowed to implement a rowhammer attack. This change alone disabled most early attacks. Additionally, most operating systems quickly disabled default user-level access to the virtual memory page mapping (*e.g.*, /proc/pid/pagemap in Linux). Since DRAM rows in modern DRAMs are much larger than virtual memory pages, the victim and aggressor DRAM rows will be in different virtual memory pages, and without access to the pagemap the attack will not know which virtual memory pages to access. Finally, some systems (*e.g.*, Google's Native Client) have disabled user-level access to the x86 CLFLUSH cache-flush instruction, which is critical to many attacks since it permits fast high-locality access to DRAM.

Since the first demonstration of rowhammer attacks, there has been a steady erosion in the effectiveness of these protection mechanisms. Very early, researchers discovered the double-sided rowhammer attack [15], which repeatedly accesses the rows preceding and following a victim row. This approach significantly reduces time-to-first bit flip, and these attacks (which use CLFLUSH) could flip bits with up to quadrate refresh [6], [11]. Additionally, clever search techniques emerged that could discover which virtual memory pages reside in adjacent DRAM rows, however, these techniques were only computationally feasible for single-sided rowhammer attacks [15]. Finally, recent efforts have demonstrated CLFLUSH-free rowhammer attacks, which cleverly manipulate the cache state to force frequent misses to DRAM despite memory caches [6], [7].

While attackers have been successful at overcoming these individual protections, it is interesting (and important) to note that systems that employ *all three of these protections simultaneously are indeed today safe from rowhammer attacks*. Double-sided rowhammer attacks, while able to overcome fast DRAM refresh rates, are computationally infeasible to implement without knowledge of the virtual memory page mapping. Similarly, CLFLUSH-free rowhammer attacks, which overcome restricted access to the CLFLUSH instruction, are unable to successfully flip bits with double-rate DRAM refresh. This limitation stems from the high-associativity of modern last-level caches, which often have 12 or more ways, thus requiring many accesses to initiate a directed miss to DRAM.

In this paper, we demonstrate, for the first time to our knowledge, a single-sided CLFLUSH-free rowhammer attack

978-1-5386-3929-0/17/$31.00

that works in the presence of all three previously mentioned protection measures. This attack is sufficiently fast to successfully work with double-rate refresh. In addition, the attack is single-sided, thus it is computationally feasible to blindly search for adjacent DRAM rows in a system with a protected pagemap. Finally, the attack's implementation does not use the CLFLUSH instruction, but rather it manipulates the Least Recently Used (LRU) state of the last-level cache to speed up accesses to DRAM rows.

Our approach to speeding up single-sided CLFLUSH-free rowhammer attacks is to use (or more accurately, abuse) Intel's Cache Allocation Technology (CAT) [9] to reduce the number of active ways in the last-level cache, which significantly reduces the number of accesses required to initiate a miss to the DRAM. Intel's CAT extension is a system-level capability that limits the number of ways available to an application in the last-level cache. The extension was added to recent Intel Xeon processors, in part, to address the problem of cache-based denial-of-service (DoS) attacks [9], where an aggressor virtual machine trashes the last-level cache, thereby severely impacting the performance of other virtual machines in the system. By restricting the amount of cache accessible to the offending virtual machine via the CAT extension, other virtual machines can still be guaranteed good execution performance.

To demonstrate the power of this attack, we implemented it on an Intel Xeon D-1541 CPU connected to DDR4 DRAM. We find that our accelerated rowhammer attack is able to function with double refresh, while having no access to the CLFLUSH instruction, and without needing explicit knowledge of the virtual page mapping. Our attack induces bit flips within a double-rate refresh of 32ms as long as the number of cache ways is restricted to 4 or less by the CAT feature. Specifically, in this paper we make the following contributions:

- We demonstrate the first single-sided CLFLUSH-free attack that is capable of working with double-rate DRAM refresh protections. This attack cannot be stopped by any combination of widely deployed rowhammer protections.
- We illustrate the potential dangers of additive security protections, by using Intel's anti-DoS Cache Allocation Technology to accelerate rowhammer attacks. Because designers cannot fully anticipate how a protection measure might be used (or abused), their addition may introduce risk into the system. To counter this deficiency, we highlight the recently proposed subtractive security approach [4], where security measures remove critical attack functionality from a system.

The remainder of this paper is organized as follows. Section II gives more background on rowhammmer attacks and their mitigations. Section III introduces our single-sided CLFLUSH-free rowhammer attack, and Section IV presents performance analyses of the attack and suggests potential protections for this more aggressive rowhammer attack. Section V presents related works. Finally, Section VI concludes the paper.

II. Rowhammer Attacks and Mitigations

Recent studies have shown that repeatedly accessing a row in DRAM induces bit flips on adjacent rows due to the electrical coupling between rows [11]. To achieve this effect, the attacker must make hundreds of thousands of accesses to the same DRAM row within a single refresh period.

To overcome this vulnerability, many mitigation techniques have been proposed. Table I lists the rowhammer protections that have been widely deployed by industry to date and the documented rowhammmer attacks. The table shows whether or not a particular protection stops a given type of attack. The "CAT Assisted" attack is the one presented in this paper. The widely deployed protections for rowhammer attacks are *i)* doubling the DRAM refresh rate, *ii)* restricted access to virtual-to-physical page mappings, and *iii)* disabling user-level access to the CLFLUSH instruction. Each of these techniques serve to make rowhammer attacks more difficult to implement. Specifically, faster refresh rate reduces the time that attackers have to discharge neighboring DRAM cells, while not having access to page maps significantly complicates attacks because it is not possible to definitively know which virtual addresses map to adjacent DRAM rows. Finally, restricting the use of the CLFLUSH instruction sends repeated accesses to a virtual address to the cache, instead of the DRAM.

TABLE I: Deployed Rowhammmer Protections Vs. Attacks: This table lists vertically the three rowhammer protections widely deployed today. The table then lists horizontally four rowhammer attacks (the initial single-sided attack, plus 3 second-generation attacks), with the table entries indicating if the attack overcomes a specific protection (✓) or if it is defeated (×). The "CAT Assisted" attack is detailed in this paper. Clearly, advanced attacks are able to overcome protections in isolation, but only our approach is able to overcome all of the protections in tandem.

Protections \ Attacks	CLFLUSH based		CLFLUSH	CAT
	Single	Double	Free	Assisted
Double Refresh Rate	x	✓	x	✓
Restricted page map	✓	x	x	✓
Disabled CLFLUSH	x	x	✓	✓

Double-Sided Rowhammer Attacks: With DRAM refresh rates doubled (from every 64ms to every 32ms), an attacker has half the time to effect a discharge of the DRAM bits of adjacent rows. Initial attacks were single-sided, in that one DRAM row next to a victim DRAM row was repeatedly accessed. But quickly it was discovered that rowhammer attacks could be made much more efficient by hammering both rows adjacent to a victim row, that is, the row preceeding the victim row and the row following (physically in the DRAM). The technique is widely known as a *double-sided rowhammer* attack [15]. For reasons that are still openly debated, a double-sided rowhammmer attack requires typically less than half the accesses to flip a bit, compared to a single-sided attack. Double-sided attacks have been shown to overcome double *and quadruple* (16ms) refresh rates [6].

While less efficient, single-sided rowhammer attacks remain quite dangerous, since the attacker need not have explicit knowledge of the virtual address space mapping. With single-sided rowhammering, using a simple search an attacker can perform an attack on every physical page of a system until the intended bit-flip is experienced [15]. For example in a 4GB DRAM with 4KB pages, the search would require up to 2^{20} trials to find the correct pair of pages for a successful single-sided rowhammer attack. In contrast, a double-sided attack would require as many as 2^{40} trials, since two adjacent pages must be discovered. If the attack in this paper were double-sided, discovery of the address mapping necessary to implement a rowhammmer attack (without access to the page map) would require a search time of up to approximately 293 days! Since it is single-sided, the same search only requires approximately 27 seconds of computation.

CLFLUSH-Free Rowhammer Attacks: Successful rowhammer

attacks require many accesses to the same DRAM row, thus, if these accesses were naively sent to the memory system their locality would result in them being serviced by the system's caches. As such, attackers flush the caches after accessing DRAM, for example, with the CLFLUSH instruction on x86-based processors. To counter this attack, some platforms have restricted access to cache flush operations, *e.g.*, Google's Native Client environment. Subsequent work has shown that rowhammer without cache flushes is possible by cleverly manipulating the cache eviction policy of the processor. Aweke *et. al.* [6] and Gruss *et. al.* [7] demonstrated that by making repeated access to set of conflicting addresses it is possible to perform a CLFLUSH-free rowhammer. Due to the high associativity of modern last-level caches (*e.g.*, the system attacked in this paper has a 12-way last-level cache), it takes many memory accesses to force a miss to DRAM. As such, CLFLUSH-free rowhammer attacks have not been able to successfully rowhammer systems that utilize double refresh.

As shown in Table I, previously disclosed attacks have overcome the three protections that have been put in place in industry. However, when used in tandem, we are unaware of any existing attack that could successfully rowhammer a fully protected machine. In this work, we successfully rowhammer DDR4 memory despite the use of all three protections working in tandem. Using CAT to limit last-level cache associativity, which permits an application's access to the last-level cache to be restricted to a limited number of ways, our single-sided CLFLUSH-free rowhammer attack is able to overcome all three currently deployed industry protections.

III. Accelerating DDR4 Rowhammer

Recent work [12] has shown DDR4 is still vulnerable to rowhammer attacks. Our major insight is that CAT-based DoS defenses limit the associativity of the LCC, and hence reduce the number of memory accesses required by a CLFLUSH-free rowhammer attack. We implemented our DDR4-based rowhammer attack on an Intel Xeon D-1541 Broadwell processor. We performed both single and double sided CLFLUSH-free and CLFLUSH-based attacks. A key challenge in implementing the double-sided rowhammer attacks is determining the physical address to *i)* cache slice mapping and *ii)* DRAM row mapping. Broadwell's DDR4 address mapping is, to our knowledge, neither publicly available nor already reverse-engineered. The mapping used is illustrated in Figure 1, and the following sections detail how we discovered this mapping.

A. Reverse Engineering the Broadwell Address Mapping

Broadwell cache address mappings: To devise access patterns that evict each other, we must understand the geometry of the last-level cache (LLC). Modern chip multi-processors divide the LLC into slices, with the number of slices typically equal to the number of physical cores, to increase the bandwidth to the LLC. The slice selection is done by a linear combination of bits from the physical address, designed in a way to reduce the conflicts between simultaneous accesses. In addition to cache slice selection, the physical address also specifies the cache set index and block offset.

The Xeon D-1541 processor has a 12Mbyte 12-way set associative cache with 8 physical cores, and 8 cache slices, with each slice containing 1.5MB of cache storage. The 1.5MB cache slice is divided into 12 ways, each way having 128KB of storage in 2048 cache sets. We generated sets of potentially conflicting random physical address pairs and did timing-based side-channel analysis to find out the bits used in cache set and cache slice selection. This same techniques is used to find conflicting addresses for our single-sided CLFLUSH-free attack. Our experiments show that physical address bits 6 to 16 are used for cache set selection. The slice bits on the other hand are an XOR-hash of the most significant bits of the physical address. We did an exhaustive search over the collected conflicting address pairs to find unique combination of hashes used for cache slice selection and found the XOR combination in Figure 1.

Broadwell DDR4 DRAM mapping: To ensure that the accesses we issue result in a rowhammer attack, the accesses must map to the same DRAM bank and adjacent DRAM rows. We achieved this using the integrated memory controller's performance counters. The controller supports DRAM event counting, such that it can be configured to count the number of accesses to a specific DRAM channel and bank. Using the performance counters, we determine the physical address mapping into the DRAM banks and channels.

Finally, we found the row bits by analyzing the latency difference between row buffer hits and row buffer misses. Figure 1 shows the address mapping of Intel Broadwell processor.

B. Cache Allocation Technology

CAT [9], which was introduced in Intel's Broadwell architecture, is a feature that enables allocation of a portion of LLC to a logical processor so that cache misses by that processor will not evict data from other portions of the shared LLC. This is achieved by associating a Class of Service (COS) with a logical processor and then configuring its Cache Bit Mask (CBM) to only allow access to specific ways of the LLC.

Intel documentation indicates that CAT technology was added to *i)* achieve a reduced and predictable latency in highly shared virtual machine environments, and *ii)* provide a

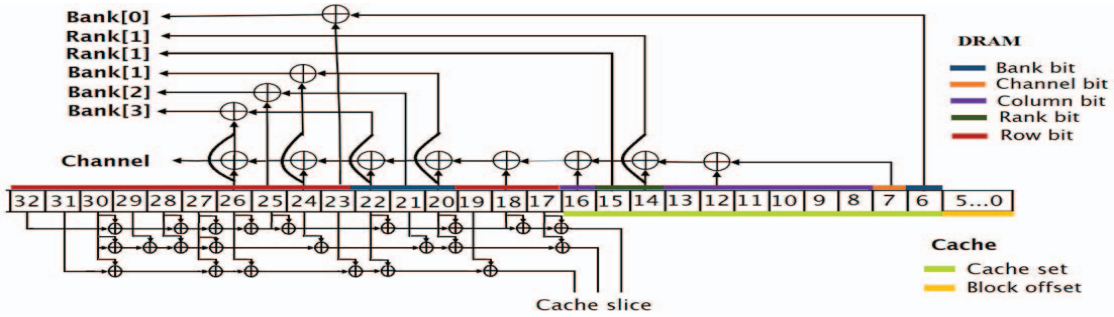

Fig. 1: Broadwell DDR4 address mapping: This figures presents the address bits (or combination) used for bank, channel, column, rank and row selection in Broadwell CPUs as well as the last-level cache set and slice selection bits of the physical address.

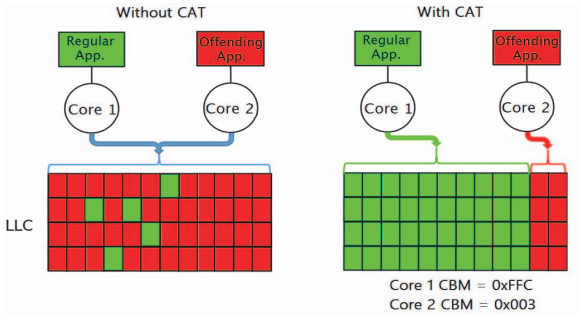

Fig. 2: LLC with and without CAT: In the left, LLC has an unconstrained noisy aggressor application. In the right, the CAT configuration has limited the aggressor to the last 2-ways of the LLC.

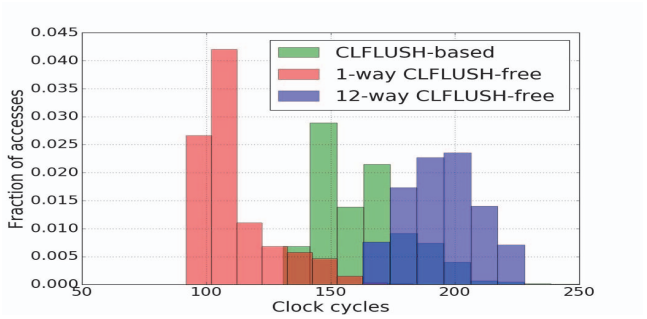

Fig. 3: Eviction Latency: CLFLUSH-based, 1-way and 12-way enabled: This figure demonstrates the distribution of eviction latency of conflicting accesses with a 12-way (blue), CAT assisted 1-way enabled (red) and using CLFLUSH instructions (green).

strong security measure for isolating and safeguarding virtual machines from malicious and noisy neighbors [9]. As shown in Figure 2, without CAT technology the "offending app" is able to starve the "regular app" of cache resources, by simply aggressively filling the LLC with its own data. With CAT enabled, however, it is possible to restrict the "offending app" to only a few ways of the cache, thereby, allowing the "regular app" unfettered access to the LLC. In this work, we show that CAT's anti-DoS protections can be used to significantly accelerate CLFLUSH-free rowhammer attacks.

C. Abusing the CAT

CAT [9] provides software-programmable control over the amount of cache space that can be consumed by a given thread, process, or VM. The cache allocations can be updated at runtime, allowing cloud service providers, for example, to change VM cache allocations based on quality of service expectations of clients. Intel in collaboration with AppFormix has demonstrated how to contain noisy VM neighbors in cloud environments using CAT [13]. Their DoS protection detects[1] VM's with greedy use of the LLC that are squeezing out the working set of other VMs, after which, CAT is used to restrict the greedy VM to a few ways of the LLC.

Threat Model: For our attack we assume the attacker has access to a high-resolution timer. This is readily available on most architectures (*e.g.*, *rdtsc* instruction on Intel x86 architecture). We also assume the attacker VM is collocated with the victim VM on a CAT enabled server with DoS attack protection (*e.g.*, AppFormix Cloud Orchestration Solution [13]). However, we do not assume the attacker has access to privileged operations such as configuration of CAT registers or access to the *pagemap* interface.

Limiting LLC Allocation: The first step of our attack involves abusing CAT technology to reduce the number of ways assigned for the offending application. There are a variety of ways to do this including *i)* purchasing cloud services with a low-quality of service which would assign our VM only a small fraction of the last-level cache, or *ii)* by first initiating a denial-of-service attack on another VM by thrashing the last-level cache, which would result in an eventual reduction of last-level cache occupancy. For the purpose of experiments in this work we assume the first case, thus we simply set the number of ways accessible in the last-level cache to a value smaller than the maximum (*i.e.*, less than 12 ways).

Once CAT restricts our attack's access to the LLC, it has effectively reduced the number of accesses (hence the time) required to produce conflicting misses for a rowhammer attack. To demonstrate how CAT can be used to reduce the cache eviction latency, we show an access latency distribution in Figure 3 for experiments involving one million DRAM accesses generated by a CLFLUSH-free attack on the baseline LLC (12-way) and a CAT restricted 1-way LLC. The figure also shows latency for a CLFLUSH-based attack on the baseline 12-way LLC. The 1-way cache is much more efficient in forcing misses, cutting the average time in more than half. Surprisingly, it is more efficient on average to generate conflict set misses out of a CAT-restricted 1-way LLC than to flush the rowhammer aggressor addresses out of the LLC with CLFLUSH. Our CAT-assisted CLFLUSH-free attack had an average DRAM activation latency of 112 cycles vs. an average latency of 163 cycles for the CLFLUSH-based attack.

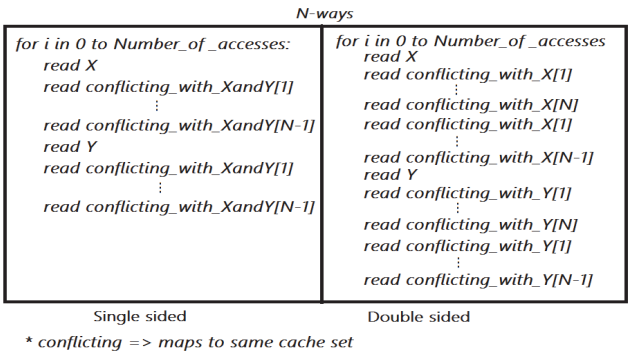

Fig. 4: CLFLUSH-free rowhammer attacks: Single-sided (Left): X is aggressor and Y maps to the same cache set and DRAM bank. Double-sided(Right): X and Y are aggressors of the victim row.

Performing CAT-assisted CLFLUSH-free rowhammer attacks: We studied the cache eviction algorithm of our CAT enabled processor and found it to follow a LRU replacement strategy. Consequently, it requires $N+1$ accesses to evict a cache line from an N-way inclusive LLC.[2] Figure 4 shows the pseudo-code for a CLFLUSH-free algorithm where N can be reduced

[1]Intel Resource Director Technology frameworks such as Cache Monitoring Technology (CMT) can be used to detect "noisy neighbors" by monitoring the cache resource utilization of VMs in multi-tenant environments.

[2]With cache inclusivity property (which all CPUs we encountered possessed), we can concentrate solely on evictions in the LLC without concerning ourselves with higher levels of the cache hierarchy. The inclusion property will force any eviction in the last-level cache to also evict the data from all higher-level caches.

TABLE II: Min. time to induce bit flips for 4 DRAMs: The table shows the minimum time required to induce rowhammer bit flips for commodity DRAMs. We saw bit flips in 3 of the 4 cases.

DIMM	Capacity	Min. Time
Vendor A	4GB	21ms
Vendor A	8GB	45ms
Vendor B	8GB	-
Vendor C	8GB	59ms

to 1-way using the techniques described above. For single-sided CLFLUSH-free rowhammering we pick conflicting addresses (X, Y, $conflicting\_with\_X and Y[1...N-1]$), which can be found using timing-based side-channel analysis, where X and Y are two aggressors for single-sided rowhammering—same bank different row. With the LRU policy, accesses $read\ conflicting\_with\_X and Y[1]$...$read\ conflicting\_with\_X and Y[N-1]$ are cache hits. The access $read\ Y$, on the other hand, evicts X from the LLC and clears the row buffer. Using this technique we were able to perform a single-sided CLFLUSH-free rowhammer attack on a 1-way configured processor with only two accesses. Since bank selection bits are also used for slice selection, it is impossible to make aggressors of double-sided rowhammer attack evict each other from LLC. Instead we use N conflicting addresses for each aggressor ($conflicting\_with\_X[1...N]$ and $conflicting\_with\_Y[1...N]$). The algorithm is designed in such a way that the accesses ($read\ X$, $read\ conflicting\_with\_X[N]$) and ($read\ Y$, $read\ conflicting\_with\_Y[N]$) evict each other; and accesses $read\ conflicting\_with\_X[1]$ ... $read\ conflicting\_with\_X[N-1]$ and $read\ conflicting\_with\_Y[1]$ ... $read\ conflicting\_with\_Y[N-1]$ are cache hits.

IV. Results and Discussion

In this section we show the results of a series of experiments we performed to demonstrate the inherent vulnerability posed by CAT-assisted rowhammering. We did our experiments on a server-grade Xeon D-1541 based machine with a 12MByte 12-way last-level cache, with 8 physical cores and supporting a DDR4-based memory system.

Our first experiment shows the minimum time required to induce a bit flip on commodity DDR4 DRAMs of different capacity and manufacturers. For each experiment, we used double-sided CLFLUSH-based rowhammering (the easy-to-stop but most efficient attack) to find vulnerable regions of the DDR4 DRAMs. Next, to measure the minimum time to induce bit flips we reduced the number of accesses per attack trial until no bit flips were possible. Table II shows the variation in the minimum time to induce bit flips among four DRAM models from three manufacturers.

As shown in Table II, the degree of vulnerability for the DDR4-based DRAM varies significantly. In fact, we could not rowhammer Vendor B's DRAMs with any form of attack. Perhaps this vendor has eliminated cross-talk issues or added a rowhammer protection mechanism internal to the DRAM. Vendor C's DRAMs could be rowhammered, but not with double refresh. Vendor A's DRAMs, however, in the 4GB DIMM size could be rowhammered with double refresh. It is certainly interesting to note that there is some measure of protection offered simply by purchasing rowhammer resistant DRAMs. For the rest of the experiments we used the most vulnerable DRAM, *i.e.*, Vendor A's 4GB DIMM.

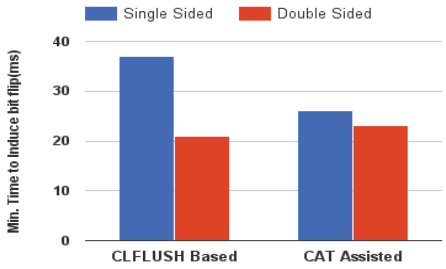

Fig. 5: CLFLUSH based Vs. CAT assisted rowhammering: CLFLUSH based (left) refers to single and double sided rowhammering using the CLFLUSH instruction to evict the cache line. On the other hand, CAT assisted (right) evicts cache lines by issuing conflicting LLC accesses on a 1-way configured LLC.

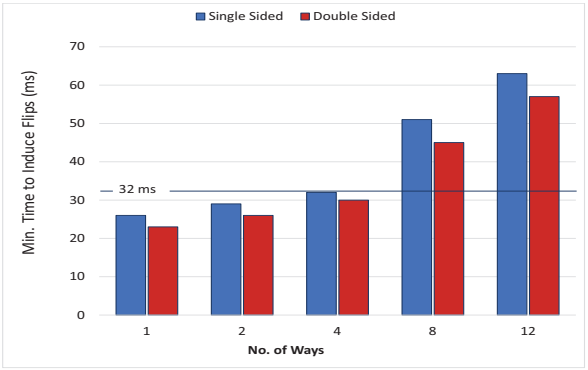

Fig. 6: Effect of number of ways on minimum time to induce bit flip: Configuring the CLOS of the offending processor to have 1-way, 2-way, 4-way, 8-way and 12-way, we show the effect of associativity on the minimum time to induce bit flips.

In a second set of experiments, we compared the time it takes to induce bit flips with our CAT-assisted (1-way) CLFLUSH-free rowhammering vs. the very efficient (but easy to stop) previous approach of double-sided CLFLUSH-based rowhammering. The results of the experiment are shown in Figure 5. The most efficient attack is the CLFLUSH-based attack, which can induce bit-flips in only 21ms. However, it should be noted that this attack can be easily stopped with restricted access to pagemaps and/or CLFLUSH. Our approach, which is resistant to all current protections, is nearly as efficient as the CLFLUSH-based attack. Our single-sided CLFLUSH-free CAT-assisted attack can flip a bit in just 26ms.

To demonstrate the risk of CAT-restricted LLC access, we performed both single-sided and double-sided CLFLUSH-free rowhammer attacks, while varying the number of ways assigned to the offending processor. The results in Figure 6 show that an offending processor can induce bit flips within a double-rate refresh of 32ms as long as the number of ways in the last-level cache is 4 or less. It is interesting to note that the high associativity offered by server-class machines today does offer some level of protection against CLFLUSH-free rowhammer attacks. Protections, of course, that are negated with the introduction of CAT extensions.

Additive vs. Subtractive Security Protections: The results of this paper begs the following question: *Is there any hope for*

system security, when the protections for one attack enable yet another? At first glance, one might think the answer is "no". One only need consider that nearly all protections added to systems (NX-bits, ASLR, read-only code, etc.) are eventually bypassed by attackers. The failing of these protections are that they are additive in nature, thus, to ensure that they are complete and not potentially abused, the designer must prove: *For all <programs, inputs>, there does not exist an unchecked vulnerability*. Clearly, for all but the simplest systems this non-existance proof is wildly intractable.

A new approach, however, is emerging in the form of subtractive security [5], where mechanisms are removed from a system to disable security vulnerabilities. A recent example of this approach is the control-data isolation (CDI) [5] technique to stop control-flow injection, where all forms of indirect jumps (*e.g.*, returns and indirect calls) are removed from the system. With this approach the designer need only prove: *There exists a useful system, such that it lacks the mechanisms used for attacks*. Such a proof is a constructive one, requiring only a single working system to complete the proof. In addition, the removal of mechanisms cannot create unexpected vulnerabilities, even for other security attacks as in the case of this paper, because no additional capability is introduced into the system.

While there are additive security measures that might protect a system from the attack detailed in this paper (*e.g.*, PARA [11] or ANVIL [6]), we are as yet unaware of any subtractive techniques to provide protections against rowhammer attacks.

V. Related Works

The rowhammer attack was introduced to the research community by Kim, et. al [11] in 2014. This revelation was quickly followed by various exploits, including the working privilege escalation exploits presented in Google's Project Zero, where bit flips granted kernel level privileges to a user level process on an x86 architecture [1]. Another major exploit by the same group was using rowhammer attacks to induce bit flips on access permission bits in page table entries to gain write access to read-only pages.

Different rowhammer mitigation techniques have been proposed, including doubling the refresh rate of the system DRAM, which has high overhead and is not a sufficient solution, and disallowing CLFLUSH instructions as is the case with Google's Native Client (NaCl) x86 sandbox. In fact, doubling refresh rate has been bypassed by double-sided rowhammer attacks [15]. But double-sided attacks, on the other hand, require user-level access to page maps. A CLFLUSH-free attack has been demonstrated by Aweke [6] and Gruss [7], by issuing sets of conflicting cache accesses to perform repeated DRAM row accesses, thereby circumventing the CLFLUSH instruction. Qiao et. al [14] proposed a new CLFLUSH-free rowhammer attack. However, their approach requires access to x86 non-temporal instructions.

A number of hardware-based mitigation techniques such as TRR [2] [10], pTRR and PARA have been proposed. DRAMs supporting TRR track the number of accesses to a row within a predefined period of time and activate its neighbors if the number of accesses exceed some threshold. Even though the details have not yet been released, Intel announced that server grade Xeon-class machines will have pTRR support. PARA, on the other hand, relies on activating the neighboring rows of the accessed row with a low probability, in order to refresh the neighboring rows in the case of repeated localized accesses.

VI. Conclusion

In this work, we demonstrate that, by performing CAT-assisted rowhammering, it is possible to bypass the widely deployed rowhammer protections, even when they are used in tandem. Our single-sided CLFLUSH-free rowhammer attack is able to rowhammer DDR4 DRAMs with double refresh ($32ms$), restricted pagemaps and disabled cache-flush instructions. We accomplish this goal by utilizing CAT on an Intel Xeon processor to restrict the LLC to 4-ways or less, which significantly speeds up our CLFLUSH-free attack. We close by noting that a subtractive security approach rather than an additive one has good potential in the future to keep good protections from going bad.

VII. Acknowledgements

The authors would like to thank the reviewers for the constructive suggestions that helped to improve the paper. This work was supported in part by C-FAR, one of the six STARnet centers, sponsored by MARCO and DARPA.

References

[1] Exploiting the DRAM Rowhammer Bug to Gain Kernel Privileges. http://googleprojectzero.blogspot.com/2015/03/exploiting-dram-rowhammer-bug-to-gain.html. Accessed: 2016-03-11.

[2] JEDEC Solid State Technology Association . Low Power Double Data Rate 4 (LPDDR4), 2015.

[3] S. Alarifi and S. D. Wolthusen. Robust coordination of cloud-internal denial of service attacks. In *Cloud and Green Computing (CGC), 2013 Third International Conference on*, pages 135–142, Sept 2013.

[4] William Arthur, Ben Mehne, Reetuparna Das, and Todd Austin. Getting in Control of Your Control Flow with Control-data Isolation. In *Proceedings of the 13th Annual IEEE/ACM International Symposium on Code Generation and Optimization*, CGO '15, pages 79–90, Washington, DC, USA, 2015. IEEE Computer Society.

[5] William P. Arthur. *Control-Flow Security*. dissertation, University of Michigan, 2016.

[6] Zelalem Birhanu Aweke, Salessawi Ferede Yitbarek, Rui Qiao, Reetuparna Das, Matthew Hicks, Yossi Oren, and Todd Austin. ANVIL: Software-Based Protection Against Next-Generation Rowhammer Attacks. In *Proceedings of the Twenty-First International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 743–755. ACM, 2016.

[7] Daniel Gruss, Clémentine Maurice, and Stefan Mangard. Rowhammer. js: A Remote Software-Induced Fault Attack in Javascript. *arXiv preprint arXiv:1507.06955*, 2015.

[8] R. Guanciale, H. Nemati, C. Baumann, and M. Dam. Cache Storage Channels: Alias-Driven Attacks and Verified Countermeasures. In *2016 IEEE Symposium on Security and Privacy (SP)*, pages 38–55, May 2016.

[9] Intel Inc. Intel open network platform release 2.1 application note on resource director technology, document revision 1.0, march 2016.

[10] Micron Inc. DDR4 SDRAM MT40A2G4, MT40A1G8, MT40A512M16 Data sheet. 2015.

[11] Yoongu Kim, Ross Daly, Jeremie Kim, Chris Fallin, Ji Hye Lee, Donghyuk Lee, Chris Wilkerson, Konrad Lai, and Onur Mutlu. Flipping bits in memory without accessing them: An experimental study of DRAM disturbance errors. In *ACM SIGARCH Computer Architecture News*, volume 42, pages 361–372. IEEE Press, 2014.

[12] M Lanteigne. How rowhammer could be used to exploit weaknesses in computer hardware, 2016.

[13] Khang N. Proof Points for Cache Allocation Technology in the Intel Xeon Processor E5 v4 Family, February 2016. [Online; posted September 11-2016].

[14] Rui Qiao and Mark Seaborn. A new approach for rowhammer attacks. In *Hardware Oriented Security and Trust (HOST), 2016 IEEE International Symposium on*, pages 161–166. IEEE, 2016.

[15] Mark Seaborn and Thomas Dullien. Exploiting the DRAM rowhammer bug to gain kernel privileges. In *Black Hat conference, https://www.blackhat.com/docs/us-15/materials/us-15-Seaborn-Exploiting-The-DRAM-Rowhammer-Bug-To-Gain-Kernel-Privileges.pdf*, 2015.

[16] Yuval Yarom and Katrina Falkner. FLUSH+RELOAD: A High Resolution, Low Noise, L3 Cache Side-Channel Attack. In *23rd USENIX Security Symposium (USENIX Security 14)*, pages 719–732, San Diego, CA, August 2014. USENIX Association.