# Rowhammering Storage Devices

Tao Zhang
The University of North Carolina at Chapel Hill
zhtao@cs.unc.edu

Boris Pismenny
Technion – Israel Institute of Technology
borispi@cs.technion.ac.il

Donald E. Porter
The University of North Carolina at Chapel Hill
porter@cs.unc.edu

Dan Tsafrir
Technion – Israel Institute of Technology &
VMware Research
dan@cs.technion.ac.il

Aviad Zuck
aviad.zuck@gmail.com

## ABSTRACT

Peripheral devices like SSDs are growing more complex, to the point they are effectively small computers themselves. Our position is that this trend creates a new kind of attack vector, where untrusted software could use peripherals strictly as intended to accomplish unintended goals. To exemplify, we set out to rowhammer the DRAM component of a simplified SSD firmware, issuing regular I/O requests that manage to flip bits in a way that triggers sensitive information leakage. We conclude that such attacks might soon be feasible, and we argue that systems need principled approaches for securing peripherals against them.

**ACM Reference Format:**
Tao Zhang, Boris Pismenny, Donald E. Porter, Dan Tsafrir, and Aviad Zuck. 2021. Rowhammering Storage Devices. In *13th ACM Workshop on Hot Topics in Storage and File Systems(HotStorage '21), July 27–28, 2021, Virtual, USA*. ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3465332.3470871

## 1 INTRODUCTION

A single computer system is increasingly composed of multiple embedded systems, which are smaller, but still resemble full systems themselves. In the case of SSDs, which serve as the focus of this study, even a commodity drive is typically equipped with hundreds of MBs of DRAM and a multicore ARM chip running nontrivial firmware [2, 6, 24, 58]. This complexity is driven by factors such as the ever-increasing throughput of the peripherals and richer offloading capabilities. Alas, increased complexity implies a greater security risk. A well-known example highlighting this risk, and our tendency to ignore it, is the fact that many were surprised to learn that Intel's Management Engine (ME) was running a full Minix OS capable of accessing the local hardware without the end-user's knowledge [18]; ME exploits shortly followed [15, 39]. Another example is the Thunderstrike boot kit that leverages compromised thunderbolt accessories to subvert the UEFI boot firmware [22].

We observe the possibility of constructing a new kind of attack against a peripheral: exclusively using it as "intended" while exploiting the mere fact that it is a full system in order to accomplish unlawful goals. In particular, we try to attack a Flash Translation Layer (FTL) by using its SSD via unprivileged software, supposedly as it was meant to be used: for reading and writing. Our proposal exploits the fact that SSDs are sophisticated peripherals and, as such, include DRAM that might be susceptible to rowhammering [26]. Our attack triggers standard NVMe commands with the goal of generating fast enough reads to: (1) flip bits and corrupt FTL data in SSD-internal DRAM; (2) in a manner that possibly exfiltrates sensitive information or even gains administrative control over the system.

To demonstrate the feasibility of this class of attacks, our work-in-progress study simplifies the target system: we experiment with an emulated FTL rather than an actual one. Instead of presenting a complete attack, which we do not have yet, we explain at each step (i) which pieces of the attack that we envision are missing, (ii) the probability of success where possible, and (iii) the remaining obstacles for a motivated attacker to overcome. Despite our simplified setup, our attack does manage to flip real DRAM bits as described, and our initial results indicate that we are at the cusp of unprivileged FTL rowhammering being feasible.


*HotStorage '21, July 27–28,2021, Virtual, USA*

ACM ISBN 978-1-4503-8550-3/21/07…$15.00
https://doi.org/10.1145/3465332.3470871

Our study leaves us with two questions. Firstly, we wonder if additional "large" system attacks, analogous to rowhammering, are applicable to peripherals. Secondly, and more importantly, we wonder if there exists some principled way to ensure end-to-end security isolation in a system that is composed of "smaller" systems—the peripheral devices. In light of a trend toward self-multiplexing devices, using technologies such as SRIOV [7, 14, 16], wherein the OS does not mediate the data path for performance reasons, we expect an increase in the difficulty and risks of failing to harden devices against direct access by untrusted software [4, 46, 57].

# 2 WHY SSDS ARE ROWHAMMER-ABLE

## 2.1 SSDs and FTLs

Vendors deliver higher-capacity, more capable SSDs, which can serve millions of I/O requests per second [9, 55, 56]. Flash memories lack support for in-place writes and perform accesses in large units due to physical limitations of flash cell technology. For this reason, among others, SSDs and other flash memory devices typically include an indirection layer—the FTL— to map logical block addresses (LBAs) to physical block addresses (PBAs). The FTL is usually implemented in software on top of an embedded system within the SSD. Similarly to host systems, such embedded systems are themselves commonly equipped with high-frequency multicore CPUs to regulate flash chips' operations, and FTLs use on-board DRAM modules for storing metadata and data including logical-to-physical mapping tables, caching frequently accessed data, and incoming writes.

## 2.2 DRAM and Rowhammering

DRAM realizes high throughput by operating hardware units in parallel. Modern DRAM modules are composed of multiple chips operating in tandem. Chips are further composed of multiple banks, which in turn contain multiple two-dimensional arrays of DRAM cells. A line of cells accessed as a unit in each array constitutes a row, which corresponds to a memory address. DRAM modules must refresh data periodically (e.g., every 64ms) to ensure retention.

Kim et al. were the first to demonstrate the possibility of a "rowhammer attack", where intentional repeated accesses to a DRAM row introduce uncorrectable errors to cells in adjacent rows if the accesses are carefully scheduled between the chip's refresh interval [26]. Google's Project Zero subsequently proposed a privilege escalation exploit on systems with susceptible DRAM modules [45]. Multiple follow-up studies have since then demonstrated various ways to induce rowhammering exploits [13, 17, 19, 20, 40, 42, 49, 50]. Kim et al. provide a recent, detailed overview of the mechanics, history, and state-of-the-art of rowhammering attacks and mitigations [25, 36].

Nethammer and Throwhammer first explored the feasibility of launching a rowhammer attack remotely [31, 48]. Instead of rowhammering via direct memory accesses, they issue network requests to a remote host quickly enough to, in turn, trigger rapid memory accesses inside the kernel network driver code or memcached internal data structures, which then eventually causes bitflips in the server's main memory. We face a similar situation, where direct access to victim DRAM is not allowed; what is different in this position paper is the focus on bitflips within the peripheral device's internal DRAM, rather than of main memory of the host system.

Interestingly, several studies recently demonstrated how to deliberately induce uncorrectable errors to flash cells [8, 28] and other types of storage [36]; the attack we consider in this study is different in that it targets the system embedded in the peripheral rather than the storage media.

## 2.3 The Risk

It is easy to overlook rowhammering vulnerabilities in peripherals, since there is a level of indirection and physical separation between their on-board DRAM chip and malicious attackers. In particular, rowhammering attacks require direct access to victim DRAM modules, which is usually not feasible in peripherals. Rather, unprivileged attackers are usually constrained to running host-level, and often user-level, code; they cannot run code on the peripheral.

We contend, however, that the ever-increasing performance of modern SSDs make their memories vulnerable to rowhammering nevertheless. Specifically, state-of-the-art rowhammering attacks on modern DRAM modules require as few as ~50K row accesses per a 64ms refresh interval [17], i.e., ~780K accesses per second. Consequently, NVMe interfaces easily allow sufficiently high 4KiB-based I/O rates necessary for a successful rowhammering attack.

We further note that the architecture of modern SSDs includes several key design choices that make the attacks more likely to succeed, discussed next.

First, SSD capacity is proportional to its internal DRAM size, e.g., 1 GiB of SSD capacity requires 1 MiB of DRAM [6]. Modern SSDs (including consumer-level) already support capacities of up to several Terabytes and thus utilize Gigabytes of on-board DRAM (Middleboxes and SmartNICs similarly include Gigabytes of DRAM as well [10, 34]). In addition to increasing the DRAM size, vendors are concerned with keeping costs low and power-efficiency high.

The problem is that the risk of rowhammering worsens when either increasing DRAM size (by making it denser), or when reducing its power consumption [19, 26]. Indeed,

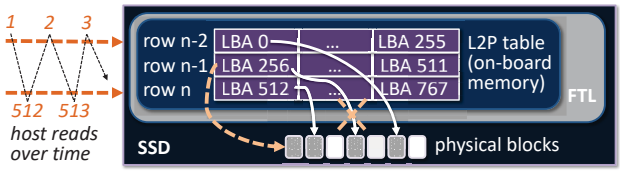


**Figure 1:** *A simple example of a two-sided FTL rowhammering attack. The onboard memory stores the L2P table. After an initial sequential write setup, a read workload accesses L2P table entries in the first and third rows (n-2 and n, called aggressors). This flips bits in the middle, victim row (n-1), redirecting LBA 256 to a different PBA.*

rowhammering mitigation techniques tend to sacrifice power-efficiency and performance [36], making them unlikely to be used in the peripheral settings we consider.

SSD internals are typically unknown and unpublished. But in our experience, which is based on reverse engineering one modern SSD from a popular vendor, the internal DRAM is not cached. We speculate that the FTL's CPU does not have caches to lower costs, or that it disables caches to simplify concurrency, perhaps because the performance benefit of caching is marginal. Regardless of the reason, no caching makes the DRAM more prone to rowhammering [26, 31, 43, 50, 51], as caches reduce DRAM access frequency.

In total, this section shows that sufficient bandwidth to launch a rowhammering attack against SSD-internal DRAM is either present already in some devices, or will be soon.

## 3 FTL ROWHAMMERING

This section shows how an unprivileged attacker can use an SSD as intended and still rowhammer device-side memory. We first present an overview of the attack primitives (§3.1), and then how FTL rowhammering can lead to data corruption, information leak, and privilege escalation (§3.2).

**Threat model.** We assume attackers have access to an unprivileged user process with high-speed read/write access to an SSD whose DRAM modules are vulnerable to rowhammering. The SSD is shared with other users (e.g., root), and the specific SSD model details are known to the attacker.

Attackers with direct access to unmapped/trimmed blocks may accelerate access rates by avoiding the overheads of additional, slower, accesses to flash. Such access may require elevated privileges, such as in VMs sharing an SSD (see §4).

### 3.1 Attack Primitive

In this attack, we use rowhammering to flip a bit in the logical to physical (L2P) table. Flipping a bit in this table can effectively overwrite the mapping of a victim logical block to a different physical address.

Our proposed attack requires an I/O workload on the order of millions of requests per second. At the firmware level, these IOs translate to repeated accesses to aggressor rows that are adjacent in memory to a victim row.

Existing interfaces available to unprivileged users, including (O_DIRECT) combined with high-performance asynchronous interfaces, such as Linux AIO or io_uring, can realize 1.5M IOPS on the latest PCIe 4.0 NVMe SSDs [1]. Upcoming PCIe 5.0 NVMe SSDs are expected to reach over 2M IOPS [5].

The attack is illustrated in Figure 1. First, the attacker prepares the L2P table by writing data to contiguous LBAs; the goal is for the SSD firmware to then allocate physical pages and corresponding L2P table entries in two aggressor rows (n-2 and n). The attacker then identifies the aggressor rows using a combination of prior device DRAM structure knowledge and trial and error. For simplicity, we depict a row as storing 256 LBAs; in practice, rows are much larger.

Next, the attacker issues a carefully orchestrated read workload (italic text and dashed lines in Figure 1) that induces rowhammering. Our attack workload repeatedly issues a read request sequence that alternates between addresses whose L2P table entries reside in the two aggressor rows. The result is a series of repeated, frequent, and alternating row activations by the firmware, effectively inducing a double-sided rowhammering attack on the target row. In our demonstration, we used a double-sided row hammer [45], although a one-location [19] variant can be simpler to implement on a device with sufficient throughput.

Finally, the translation in the victim row (n-1) is corrupted such that it points to a different physical location.

### 3.2 Attack Scenarios

The FTL Rowhammering vulnerability leads to several security sensitive outcomes: (1) data corruption, (2) information leak, and (3) privilege escalation.

**Data corruption.** The most straightforward outcome of the attack is causing random data corruption. The corruption may lead to more severe damage if the corruption happens on critical file system metadata or other SSD-internal metadata, rendering the file system unmountable or bricking the device.

**Information leak.** If the attacker can remap an LBA in a file under the attacker's control to the PBA hosting a victim's file block, the attacker can read that block, bypassing file system access controls. For instance, an attacker may get a redirection to a file block containing another user's SSH private key. This can potentially also lead to a privilege escalation if credentials are leaked. This redirection does not provide attackers with the ability to directly write victim LBAs, as flash writes are copy-on-write (§2.1). Although most bitflips will not point to sensitive PBAs, the attacker can repeat this process until successful.

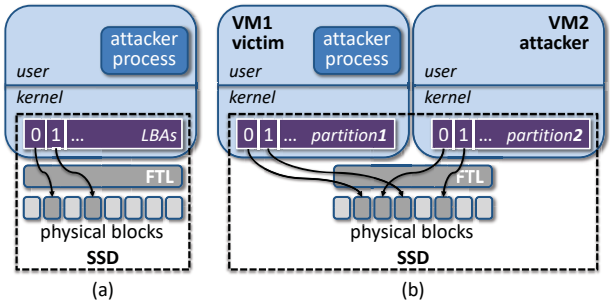


**Figure 2:** *On our existing testbed, we need a helper attacker VM to reach a high-enough access rate to make rowhammering possible (b); in the future, we foresee that such assistance will be unneeded (a).*

| *year* | *refs* | *type* | *rate (K access/s)* |
|---|---|---|---|
| 2014 | [26] | DDR3 | 2200 |
| | | DDR3 | 2500 |
| | | DDR3 | 4400 |
| 2016 | [20, 49] | DDR3 | 672 |
| | | LPDDR3 | 4000 |
| 2018 | [31, 48] | DDR3 | 9400 |
| | | DDR4 | 6140 |
| 2020 | [17, 25] | DDR4 | 800 |
| | | DDR3 (old) | 4800 |
| | | DDR3 (new) | 750 |
| | | DDR4 (old) | 547 |
| | | DDR4 (new) | 313 |
| | | LPDDR4 (old) | 1400 |
| | | LPDDR4 (new) | 150 |

**Table 1:** *Reported minimal access rate to trigger bitflips.*

**Privilege escalation.** Attacker bitflips that redirect the victim's LBAs to attacker PBAs will grant attackers a *write-something-somewhere* primitive: both the location and the contents of the victim data are not known in advance. This vulnerability is the hardest to exploit.

Before flipping any bits, the attacker needs to blindly spray the disk with polyglot blocks [21], i.e., blocks that are valid as executable code, file data, and file metadata. Replacing a victim LBA in a sensitive file with a polyglot block can result in a privilege escalation. For example, rewriting a binary executable that has `setuid` permission (e.g. `sudo`) can result in executing malicious code as root.

## 4 CLOUD CASE STUDY

This section demonstrates how, using the SSD only as intended, to turn an FTL bitflip into a privileged information leak in a VM hosted on cloud server over a shared SSD, and potentially escalate privilege using the Ext4 file system. Various cloud providers advertise over 2 million IOPS storage performance provided to VMs [11, 38]. For a proof-of-concept, we emulate an SSD in main memory and select an older system with DRAM comparable to what is in modern SSDs. There are a number of prerequisite complexities in reverse engineering an SSD that are time-consuming, orthogonal to the primary point, but needed to build an end-to-end attack. We leave the complexities for future work and ignore them for now.

### 4.1 Prototype Setup

We set up the testbed for our proof-of-concept attack as shown in Figure 2 (b). This setup is representative of a multi-tenant cloud server. We place the victim in a VM, including an unprivileged attacker process, which has non-root user privileges to create, delete, read, and write files but no direct access to the underlying storage (e.g., VMware's Hatchway [52]). And a second, attacker-controlled VM is co-located on the same server, sharing the same SSD with the victim VM. In a typical cloud hosting service, the attacker has privileged direct access to the SSD inside their own VM, via hardware multiplexing techniques like SRIOV [44] or namespaces [35]. Each VM's storage space is a partition of the shared SSD, treated as a block device with its own logical address space. In each VM, therefore, a block address is only valid within its partition. However, the underlying FTL and its mapping table are shared across partitions.

The SSD in the testbed is emulated using Intel's Storage Performance Development Kit (SPDK) [23], which uses a memory-backed block device (ramdisk). The SPDK FTL library, like most flash-based storage devices [29], stores a large L2P table in memory as a linear array. Our proposed attack works on other L2P table layouts, such as a hash table [6, 37], provided the attacker can learn the structure offline. Notably, a linear layout is *more challenging* for a two-sided rowhammering attack than a hash map, as it is more challenging to place an aggressor on each side of the victim row. The SPDK FTL library also uses the emulating host machine's cached memory for storing the L2P table. In order to further mimic the behavior of real-life SSDs (§2.2), we modified the SPDK FTL library to perform cache invalidation on every access to L2P entries.

We set up a 1 GiB emulated SSD on a machine with Intel Core i7-2600 CPU and 16 GiB DDR3 DRAM modules (4×4 GiB Samsung DIMMs, organized as 2 channels × 2 DIMMs × 2 ranks × 8 banks × $2^{15}$ rows) known to be vulnerable to rowhammer attacks. The emulation environment doesn't support ECC (Error Correction Code) or TRR (Target Row Refresh). The L2P table size for our SSD is 1MiB [6]. As a comparison, Samsung PM1733 enterprise SSD is equipped with up to 16 GiB on-board DDR4 memory (ECC and TRR support status unknown) [44].

Rowhammering requires a minimal access rate to aggressor rows, which varies with factors such as DDR generation and memory controller configuration. As shown in Table 1, common minimal rates on DDR3 range from 2 million to 9 million accesses per second, although a bitflip has been

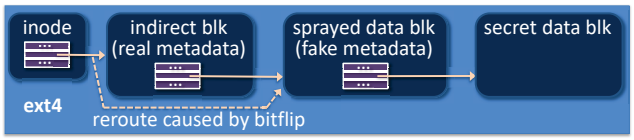


**Figure 3:** *Example of an exploit on Ext4 indirect block.*

observed at rates as low as 700K per second [17, 20, 26, 31, 48, 49]. The smaller technology node in newer DRAM modules makes them even more vulnerable to disturbance errors [25].

Because our L2P table is small relative to system memory in our testbed, we place the table in a physical memory region which we have confirmed is vulnerable to a rowhammer attack. Our testbed DRAM shows bitflips from direct accesses at a rate of 3M per second; because SPDK adds other accesses, we must issue SPDK-level accesses at a higher rate (about 7M/s). To emulate this, we manually amplified each L2P row activation (5 hammers per I/O request) in SPDK. Note that with DRAM modules that are more vulnerable to rowhammer attack, the rate amplification can be reduced or even dropped completely.

We choose the setup in Figure 2 (b) because our main system is relatively slow, so that direct access from user space is not sufficiently fast for the attack. Given a system that provides fast enough unprivileged direct access to the SSD, the attacker VM can be dropped and a simpler setup, as shown in Figure 2 (a), can be used to launch the attack.

## 4.2 Attacking the Ext4 File System

For concreteness, we attack the ext4 [33] file system. By default, ext4 inodes index file blocks using an extent tree. To prevent metadata corruptions, the extent tree is protected by CRC-32C checksum. However, for backward compatibility with previous versions, ext4 also has an optional direct/indirect block addressing mechanism used to map in-file blocks to filesystem blocks. Critically, indirect blocks are not verified against any checksum. Users may also select the direct/indirect block mechanism on files they have write access to.

In a nutshell, our attack redirects an FTL mapping entry from a victim inode to a victim's indirect block to an attacker-provided indirect block. The attacker's indirect block points to LBAs containing privileged content on the victim VM. A successful attack will modify an unprivileged file, owned by the attacker process in the victim VM, to point to the contents of a privileged file.

Our attack follows these steps:

**Filesystem spraying stage.** The attacker process inside the victim VM first sprays the victim filesystem with files configured to use indirect blocks. Each file includes a single indirect block pointing to a lone data block. The attacker creates each file with a hole of 12 blocks (to avoid storing direct data blocks) and then stores a single data block mapped using an indirect block. The data blocks in turn contain a *maliciously formed indirect block* pointing at target LBAs of potentially privileged content (Figure 3).

This spraying is needed to increase the probability of a successful attack. The locations of bitflips at the L2P table are unpredictable, so the more malicious indirect blocks on the disk, the higher the probability of success.

To further increase the possibility of a successful exploit, the attacker's VM sprays its own partition with blocks that contain similar malicious indirect blocks.

**Hammering stage.** The attacker VM launches a double-sided rowhammering attack on the L2P table. We assume that the attacker can map out potential aggressor and victim rows in a given SSD model offline; the row-level adjacency should be consistent among instances of the same model [40]. The attacker must also identify which set of rows are actually rowhammerable (the attacker could randomly pick rows to rowhammer, but the success rate may be unacceptably low); rowhammerability is determined primarily by variation in the manufacturing process and must be tested online and on the specific device.

The remaining challenge, then is getting a victim row between two aggressor rows, when the L2P table is a simple physical partition. We can run a single-sided rowhammering on the *boundary area of attacker and victim partition*, but single-sided attacks flip fewer bits in practice.

Fortunately, modern memory controllers also use a mapping function to spread DRAM accesses across different hardware units [12, 40, 47, 53, 54]. By reverse engineering or reading documentation, we can also identify a contiguous run of three rows (vulnerable to a double-sided rowhammer) that do not have monotonically increasing physical addresses. In our example system, we were able to identify 32 sets of three vulnerable rows that could potentially place the victim row in a separate memory partition from the aggressors. We note that 32 sets of vulnerable rows is on the lower end; other DRAM mapping functions or L2P structures (e.g., hash tables) could generate many more vulnerable pairs.

**Scan for bitflip.** After a certain period (e.g., 5 minutes) of hammering, the attacker process in the victim VM iterates over files created in the spraying stage to detect content modifications due to bitflips in the L2P table (see Figure 3). A successful bitflip causes an unprivileged file's inode to point at a maliciously formed indirect block. The attacker can then dump potentially-privileged content and repeat the process as necessary by editing the malicious indirect block to map other LBAs. If no bitflips are detected the attacker can re-spray the system with new files, forcing the FTL to

re-shuffle all address mappings to reside in new memory rows.

By repeating these steps enough times, the attacker can eventually dump the content of the *entire victim partition even as an unprivileged user*. The resulting content can also be used for privilege escalation, e.g., by reading the private key file of an administrator user.

The time needed to flip single bit and control a victim indirect block can vary widely. On our testbed this took about two hours, which is longer than expected in practice because SPDK limits file spraying to 5% of the victim partition due to technical issues in the FTL library.

### 4.3 Probability of Success

We estimate the probability that a given bitflip will be useful to the attacker following one cycle of the attack described in §4.2. We assume the following parameters: $\boldsymbol{LB}$ and $\boldsymbol{PB}$ represent the total number of logical and physical addresses of the SSD, respectively; the number of blocks related to the victim and attacker partitions are $\boldsymbol{C_v}$ and $\boldsymbol{C_a}$, respectively (where $C_v$+$C_a \leq LB$); the overall number of blocks related to sprayed files that attacker can create inside the victim and attacker partitions is $\boldsymbol{F_v}$ and $\boldsymbol{F_a}$, respectively. Then the number of sprayed indirect blocks is $F_v/2$, and total number of malicious data block on the device is $F_a + F_v/2$.

The probability that a bitflip happens on an LBA belonging to a sprayed victim partition indirect block is: $\frac{F_v/2}{C_v}$. The probability that the bitflipped L2P entry is redirected to a malicious block is: $\frac{F_v/2+F_a}{PB}$. Consequently, the combined probability rate of getting a useful bitflip is $\frac{F_v/2}{C_v} \cdot \frac{F_v/2+F_a}{PB} = \frac{F_v(F_v+2F_a)}{4C_v \cdot PB}$.

To illustrate, if the attacker and victim partitions equally share the SSD (i.e., $C_a = C_v = PB/2 = LB/2$). Conservatively assuming the attacker user can only fill 25% of victim partition (i.e., $F_v = 1/4C_v$), and 100% of attacker partition (i.e., $F_a = C_a$), the resulting success rate is 7% for a *single* attack cycle. Simply repeating the attack cycle for 10 times brings the chances of success to more than 50%.

## 5 MITIGATIONS

A number of proposed techniques can protect DRAM against rowhammering [3, 25, 36, 50]. Some methods, such as strengthening ECC, may also protect against FTL rowhammering. Others may not be applicable. For example, increasing DRAM refresh rate reduces the window of vulnerability, but is considered prohibitively power-hungry even in host systems.

As discussed above, SSDs could enable caches on the internal CPUs. Although there are already attacks that make use of cache eviction policies and successfully trigger bitflips in DRAM [13, 20], these attacks are not directly applicable to the memory accesses in SSD FTLs. We speculate that, with more details about FTL memory access behavior, an attack could bypass the FTL-side cache and disturb FTL memory.

One can mitigate vulnerabilities in the SSD itself. Physically isolating memory and flash hardware units across partition boundaries may protect against attacks on shared SSDs (see §4), but potentially increases manufacturing costs. Rate-limiting user IOs below the rowhammering access rate can also remove this potential attack, but it is at odds with the overall performance goals of NVMe. One could also randomize the FTL-internal structures, thwarting the assumption that the attacker could gain this knowledge offline; this is most easily accomplished with a hashed L2P table that uses a device-specific key. Finally, block data integrity [41] and encryption [32] algorithms protect data integrity and confidentiality from misdirected writes by relying on the block's LBA to digest and encrypt block data.

Alternatively, one can mitigate vulnerabilities in software by encrypting data using per-tenant keys to protect data confidentiality, or by enforcing extent tree addressing to exclude indirect file data block overwrites. The checksum protection on the extent tree should make it much more difficult to exploit, but the attacker can still induce data corruptions as described in §3.2.

## 6 CONCLUSIONS

This position paper demonstrates a new kind of attack on a peripheral, using only user-level requests as intended. Although an end-to-end attack is not yet demonstrated, we believe the remaining work will yield to effort. We are left with an open question about whether there is a more principled solution, and what other high-profile attacks, such as Spectre or Meltdown [27, 30], may work on these peripherals.

## ACKNOWLEDGMENTS

We thank the anonymous reviewers and our shepherd, Peter Desnoyers, for their insightful comments on earlier drafts of the work. We also thank Philipp Gühring, Carlo Meijer, and Eyal Ronen for their support in earlier iterations of this project. This research was supported in part by a grant from the United States-Israel Binational Science Foundation (BSF), Jerusalem, Israel, grant # 2017702; the United States National Science Foundation (NSF) grant #CNS-1816263, VMware, the Technion Hiroshi Fujiwara cyber security research center, and the Israel cyber directorate.

## REFERENCES

[1] Adam Armstrong. KIOXIA CM6 PCIe 4.0 SSD Review. https://www.storagereview.com/review/kioxia-cm6-pcie-4-0-ssd-review, 2020. Accessed: Jun 2021.

[2] Nitin Agrawal, Vijayan Prabhakaran, Ted Wobber, John D. Davis, Mark Manasse, and Rina Panigrahy. Design tradeoffs for SSD performance. In *USENIX Annual Technical Conference (ATC)*, pages 57–70, 2008. https://www.usenix.org/legacy/events/usenix08/tech/full_papers/agrawal/agrawal.pdf.

[3] 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 *ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, pages 743–755, 2016. https://doi.org/10.1145/2872362.2872390.

[4] Adam Bates, Benjamin Mood, Joe Pletcher, Hannah Pruse, Masoud Valafar, and Kevin Butler. On detecting co-resident cloud instances using network flow watermarking techniques. *International Journal of Information Security*, 13(2):171–189, 2014. https://doi.org/10.1007/s10207-013-0210-0.

[5] Billy Tallis. Marvell announces first pcie 5.0 nvme ssd controllers: Up to 14 gb/s. https://www.anandtech.com/show/16703/marvell-announces-first-pcie-50-nvme-ssd-controllers, 2021. Accessed: Jun 2021.

[6] Andrew Birrell, Michael Isard, Chuck Thacker, and Ted Wobber. A design for high-performance flash disks. *ACM SIGOPS Operating Systems Review*, 41(2):88–93, 2007. https://doi.org/10.1145/1243418.1243429.

[7] Edouard Bugnion, Jason Nieh, and Dan Tsafrir. *Hardware and Software Support for Virtualization*. Morgan & Claypool Publishers, 2017. https://doi.org/10.2200/S00754ED1V01Y201701CAC038.

[8] Yu Cai, Saugata Ghose, Yixin Luo, Ken Mai, Onur Mutlu, and Erich F Haratsch. Vulnerabilities in MLC NAND flash memory programming: Experimental analysis, exploits, and mitigation techniques. In *IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pages 49–60, 2017. https://doi.org/10.1109/HPCA.2017.61.

[9] Wonil Choi, Jie Zhang, Shuwen Gao, Jaesoo Lee, Myoungsoo Jung, and Mahmut Kandemir. An in-depth study of next generation interface for emerging non-volatile memories. In *5th Non-Volatile Memory Systems and Applications Symposium (NVMSA)*, pages 1–6, 2016. https://doi.org/10.1109/NVMSA.2016.7547177.

[10] Cisco. Cisco ASR 1000 Series Router Specifications. https://www.cisco.com/c/en/us/td/docs/routers/asr1000/install/guide/asr1routers/asr-1000-series-hig/asr-hig-spfy.pdf, 2008. Accessed: Dec 2020.

[11] Google Cloud. Block storage performance. https://cloud.google.com/compute/docs/disks/performance, 2021. Accessed: Apr 2021.

[12] Lucian Cojocar, Jeremie Kim, Minesh Patel, Lillian Tsai, Stefan Saroiu, Alec Wolman, and Onur Mutlu. Are we susceptible to rowhammer? an end-to-end methodology for cloud providers. In *IEEE Symposium on Security and Privacy (S&P)*. IEEE, May 2020. https://doi.org/10.1109/SP40000.2020.00085.

[13] Finn de Ridder, Pietro Frigo, Emanuele Vannacci, Herbert Bos, Cristiano Giuffrida, and Kaveh Razavi. SMASH: Synchronized Many-sided Rowhammer Attacks From JavaScript. In *USENIX Sec*, August 2021.

[14] Haggai Eran, Lior Zeno, Maroun Tork, Gabi Malka, and Mark Silberstein. NICA: An infrastructure for inline acceleration of network applications. In *USENIX Annual Technical Conference (ATC)*, pages 345–362, 2019. https://www.usenix.org/conference/atc19/presentation/eran.

[15] Mark Ermolov and Maxim Goryachy. How to hack a turned-off computer, or running unsigned code inintel management engine. BlackHat, https://papers.put.as/papers/firmware/2017/eu-17-Goryachy-How-To-Hack-A-Turned-Off-Computer-Or-Running-Unsigned-Code-In-Intel-Management-Engine.pdf, 2017. Accessed: Jan 2021.

[16] Daniel Firestone, Andrew Putnam, Sambhrama Mundkur, Derek Chiou, Alireza Dabagh, Mike Andrewartha, Hari Angepat, Vivek Bhanu, Adrian Caulfield, Eric Chung, Harish Kumar Chandrappa, Somesh Chaturmohta, Matt Humphrey, Jack Lavier, Norman Lam, Fengfen Liu, Kalin Ovtcharov, Jitu Padhye, Gautham Popuri, Shachar Raindel, Tejas Sapre, Mark Shaw, Gabriel Silva, Madhan Sivakumar, Nisheeth Srivastava, Anshuman Verma, Qasim Zuhair, Deepak Bansal, Doug Burger, Kushagra Vaid, David A. Maltz, and Albert Greenberg. Azure accelerated networking: Smartnics in the public cloud. In *USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, pages 51–66, 2018. https://www.usenix.org/conference/nsdi18/presentation/firestone.

[17] Pietro Frigo, Emanuele Vannacc, Hasan Hassan, Victor Van Der Veen, Onur Mutlu, Cristiano Giuffrida, Herbert Bos, and Kaveh Razavi. TRRespass: Exploiting the many sides of target row refresh. In *IEEE Symposium on Security and Privacy (S&P)*, pages 747–762, 2020. https://doi.org/10.1109/SP40000.2020.00090.

[18] Matthew Garrett. Intel's remote AMT vulnerablity. https://mjg59.dreamwidth.org/48429.html, 2017. Accessed: Jan 2021.

[19] Daniel Gruss, Moritz Lipp, Michael Schwarz, Daniel Genkin, Jonas Juffinger, Sioli O'Connell, Wolfgang Schoechl, and Yuval Yarom. Another flip in the wall of rowhammer defenses. In *IEEE Symposium on Security and Privacy (S&P)*, pages 245–261, 2018. https://doi.org/10.1109/SP.2018.00031.

[20] Daniel Gruss, Clémentine Maurice, and Stefan Mangard. Rowhammer.js: A remote software-induced fault attack in javascript. In *International conference on detection of intrusions and malware, and vulnerability assessment*, pages 300–321. Springer, 2016. https://doi.org/10.1007/978-3-319-40667-1_15.

[21] H.L.J.Laloge. Polyglot database. https://github.com/Polydet/polyglot-database, 2018.

[22] Trammell Hudson and Larry Rudolph. Thunderstrike: EFI firmware bootkits for Apple MacBooks. In *ACM International Systems and Storage Conference (SYSTOR)*, pages 1–10, 2015. https://doi.org/10.1145/2757667.2757673.

[23] Intel. Storage Performance Development Kit (SPDK). https://spdk.io, 2015. Accessed: Jan 2021.

[24] Hyukjoong Kim, Dongkun Shin, Yun Ho Jeong, and Kyung Ho Kim. SHRD: Improving spatial locality in flash storage accesses by sequentializing in host and randomizing in device. In *USENIX Conference on File and Storage Technologies (FAST)*, pages 271–284, 2017. https://www.usenix.org/conference/fast17/technical-sessions/presentation/kim.

[25] Jeremie S. Kim, Minesh Patel, A. Giray Yaglikci, Hasan Hassan, Roknoddin Azizi, Lois Orosa, and Onur Mutlu. Revisiting rowhammer: An experimental analysis of modern dram devices and mitigation techniques. In *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*, pages 638–651, 2020. https://doi.org/10.1109/ISCA45697.2020.00059.

[26] 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 International Symposium on Computer Architecture (ISCA)*, pages 361–372, 2014. https://doi.org/10.1145/2678373.2665726.

[27] Paul Kocher, Jann Horn, Anders Fogh, Daniel Genkin, Daniel Gruss, Werner Haas, Mike Hamburg, Moritz Lipp, Stefan Mangard, Thomas Prescher, et al. Spectre attacks: Exploiting speculative execution. In *IEEE Symposium on Security and Privacy (S&P)*, pages 1–19, 2019. https://doi.org/10.1109/SP.2019.00002.

[28] Anil Kurmus, Nikolas Ioannou, Matthias Neugschwandtner, Nikolaos Papandreou, and Thomas Parnell. From random block corruption to privilege escalation: A filesystem attack vector for rowhammer-like attacks. In *USENIX Workshop on Offensive Technologies (WOOT)*, 2017. https://www.usenix.org/conference/woot17/workshop-program/presentation/kurmus.

[29] Kim Kwonyoup and Lee Seungjoon. A new hope: The one last chance to save your ssd data. *Black Hat USA*, 2020, 2020. https://i.blackhat.com/eu-20/Wednesday/eu-20-Lee-A-New-Hope-The-One-Last-Chance-to-Save-Your-SSD-Data.pdf.

[30] Moritz Lipp, Michael Schwarz, Daniel Gruss, Thomas Prescher, Werner Haas, Anders Fogh, Jann Horn, Stefan Mangard, Paul Kocher, Daniel Genkin, Yuval Yarom, and Mike Hamburg. Meltdown: Reading kernel memory from user space. In *USENIX Security Symposium*, pages 973–990, 2018. https://www.usenix.org/conference/usenixsecurity18/presentation/lipp.

[31] Moritz Lipp, Michael Schwarz, Lukas Raab, Lukas Lamster, Misiker Tadesse Aga, Clémentine Maurice, and Daniel Gruss. Nethammer: Inducing rowhammer faults through network requests. In *IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*, pages 710–719, 2020. https://doi.org/10.1109/EuroSPW51379.2020.00102.

[32] Luther Martin. XTS: A mode of AES for encrypting hard disks. *IEEE Security & Privacy*, 8(3):68–69, 2010. https://doi.org/10.1109/MSP.2010.111.

[33] Avantika Mathur, Mingming Cao, Suparna Bhattacharya, Andreas Dilger, Alex Tomas, and Laurent Vivier. The new ext4 filesystem: current status and future plans. In *Proceedings of the Linux symposium*, volume 2, pages 21–33, 2007. https://www.kernel.org/doc/ols/2007/ols2007v2-pages-21-34.pdf.

[34] Mellanox. Bluefield SmartNIC for Ethernet. https://www.mellanox.com/sites/default/files/related-docs/prod_adapter_cards/PB_BlueField_Smart_NIC.pdf, 2019. Accessed: Dec 2020.

[35] Micron. Micron 9300 NVMe SSD. https://media-www.micron.com/-/media/client/global/documents/products/product-flyer/9300_ssd_product_brief.pdf?la=en&rev=b6908d03082d4fd7b022a2f40d1b731e, 2020. Accessed: Dec 2020.

[36] Onur Mutlu and Jeremie S. Kim. Rowhammer: A retrospective, 2019. http://arxiv.org/abs/1904.09724. Accessed: Dec 2020.

[37] Fan Ni, Chunyi Liu, Yang Wang, Chengzhong Xu, Xiao Zhang, and Song Jiang. A hash-based space-efficient page-level ftl for large-capacity ssds. In *2017 International Conference on Networking, Architecture, and Storage (NAS)*, pages 1–6, 2017. https://doi.org/10.1109/NAS.2017.8026838.

[38] Oracle. Oracle cloud infrastructure–cloud storage. https://www.oracle.com/cloud/storage/, 2021. Accessed: Apr 2021.

[39] ID Pankov, AS Konoplev, and A Yu Chernov. Analysis of the security of uefi bios embedded software in modern intel-based computers. *Automatic Control and Computer Sciences*, 53(8):865–869, 2019. https://doi.org/10.3103/S0146411619080224.

[40] Peter Pessl, Daniel Gruss, Clémentine Maurice, Michael Schwarz, and Stefan Mangard. DRAMA: Exploiting DRAM addressing for cross-cpu attacks. In *25th USENIX Security Symposium (USENIX Security 16)*, pages 565–581, 2016. https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/pessl.

[41] Martin K Petersen. T10 data integrity feature (logical block guarding). https://www.usenix.org/legacy/event/lsf07/tech/petersen.pdf, 2007. Accessed: Dec 2020.

[42] Salman Qazi, Yoongu Kim, Nicolas Boichat, Eric Shiu, and Mattias Nissler. Introducing half-double: New hammering technique for dram rowhammer bug. https://security.googleblog.com/2021/05/introducing-half-double-new-hammering.html, May 2021.

[43] Rui Qiao and Mark Seaborn. A new approach for rowhammer attacks. In *IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, pages 161–166, 2016. https://doi.org/10.1109/HST.2016.7495576.

[44] Samsung. Samsung PM1733 NVMe SSD. https://samsungsemiconductor-us.com/labs/pdfs/PM1733_U2_Product_Brief.pdf, 2020. Accessed: Dec 2020.

[45] Mark Seaborn and Thomas Dullien. Exploiting the DRAM rowhammer bug to gain kernel privileges. http://googleprojectzero.blogspot.com.tr/2015/03/exploiting-dram-rowhammer-bug-to-gain.html, 2015. Accessed: Jan 2021.

[46] Igor Smolyar, Muli Ben-Yehuda, and Dan Tsafrir. Securing self-virtualizing Ethernet devices. In *USENIX Security Symposium*, pages 335–350, 2015. https://www.usenix.org/conference/usenixsecurity15/technical-sessions/presentation/smolyar.

[47] Andrei Tatar, Cristiano Giuffrida, Herbert Bos, and Kaveh Razavi. Defeating software mitigations against rowhammer: a surgical precision hammer. In *International Symposium on Research in Attacks, Intrusions, and Defenses*, pages 47–66, 2018. https://doi.org/10.1007/978-3-030-00470-5_3.

[48] Andrei Tatar, Radhesh Krishnan Konoth, Elias Athanasopoulos, Cristiano Giuffrida, Herbert Bos, and Kaveh Razavi. Throwhammer: Rowhammer attacks over the network and defenses. In *USENIX Annual Technical Conference (ATC)*, pages 213–226, 2018. https://www.usenix.org/conference/atc18/presentation/tatar.

[49] Victor van der Veen, Yanick Fratantonio, Martina Lindorfer, Daniel Gruss, Clementine Maurice, Giovanni Vigna, Herbert Bos, Kaveh Razavi, and Cristiano Giuffrida. Drammer: Deterministic rowhammer attacks on mobile platforms. In *ACM Conference on Computer and Communications Security (CCS)*, pages 1675–1689, 2016. https://doi.org/10.1145/2976749.2978406.

[50] Victor van der Veen, Martina Lindorfer, Yanick Fratantonio, Harikrishnan Padmanabha Pillai, Giovanni Vigna, Christopher Kruegel, Herbert Bos, and Kaveh Razavi. GuardION: Practical mitigation of DMA-based rowhammer attacks on ARM. In *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment (AHES)*, pages 92–113, 2018. https://doi.org/10.1007/978-3-319-93411-2_5.

[51] Pepe Vila, Boris Köpf, and José F Morales. Theory and practice of finding eviction sets. In *IEEE Symposium on Security and Privacy (S&P)*, pages 39–54, 2019. https://doi.org/10.1109/SP.2019.00042.

[52] VMware. Project Hatchway: Persistent Storage for Cloud-Native Applications. https://blogs.vmware.com/cloudnative/2017/09/06/project-hatchway-persistent-storage-cloud-native-applications/, 2017. Accessed: Jan 2021.

[53] Minghua Wang, Zhi Zhang, Yueqiang Cheng, and Surya Nepal. Dramdig: A knowledge-assisted tool to uncover dram address mapping. In *ACM/IEEE Design Automation Conference (DAC)*, pages 1–6, 2020. https://doi.org/10.1109/DAC18072.2020.9218599.

[54] Yuan Xiao, Xiaokuan Zhang, Yinqian Zhang, and Radu Teodorescu. One bit flips, one cloud flops: Cross-vm row hammer attacks and privilege escalation. In *USENIX Security Symposium*, pages 19–35, 2016. https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/xiao.

[55] Jie Zhang, Miryeong Kwon, Michael Swift, and Myoungsoo Jung. Manycore-based scalable ssd architecture towards one and more million IOPS. In *Annual Non-Volatile Memories Workshop (NVMW)*, 2021. http://nvmw.ucsd.edu/nvmw2021-program/nvmw2021-data/nvmw2021-final27.pdf.

[56] Tao Zhang, Aviad Zuck, Donald E. Porter, and Dan Tsafrir. Apps can quickly destroy your mobile's flash: why they don't, and how to keep it that way. In *ACM International Conference on Mobile Systems, Applications, and Services (MobiSys)*, pages 207–221, 2019. https://doi.org/10.1145/3307334.3326108.

[57] Zhe Zhou, Zhou Li, and Kehuan Zhang. All your VMs are disconnected: Attacking hardware virtualized network. In *ACM Conference on Data and Application Security and Privacy (COADSPY)*, 2017. https://doi.org/10.1145/3029806.3029810.

[58] Aviad Zuck, Philipp Gühring, Tao Zhang, Donald E Porter, and Dan Tsafrir. Why and how to increase ssd performance transparency. In *USENIX Workshop on Hot Topics in Operating Systems (HOTOS)*, 2019. https://doi.org/10.1145/3317550.3321430.