The RowHammer Problem and Other Issues We May Face as Memory Becomes Denser

Onur Mutlu
ETH Zürich
onur.mutlu@inf.ethz.ch
https://people.inf.ethz.ch/omutlu

***Abstract*—As memory scales down to smaller technology nodes, new failure mechanisms emerge that threaten its correct operation. If such failure mechanisms are not anticipated and corrected, they can not only degrade system reliability and availability but also, perhaps even more importantly, open up security vulnerabilities: a malicious attacker can exploit the exposed failure mechanism to take over the entire system. As such, new failure mechanisms in memory can become practical and significant threats to system security.**

In this work, we discuss the RowHammer problem in DRAM, which is a prime (and perhaps the first) example of how a circuit-level failure mechanism in DRAM can cause a practical and widespread system security vulnerability. RowHammer, as it is popularly referred to, is the phenomenon that repeatedly accessing a row in a modern DRAM chip causes bit flips in physically-adjacent rows at consistently predictable bit locations. It is caused by a hardware failure mechanism called DRAM disturbance errors, which is a manifestation of circuit-level cell-to-cell interference in a scaled memory technology. Researchers from Google Project Zero recently demonstrated that this hardware failure mechanism can be effectively exploited by user-level programs to gain kernel privileges on real systems. Several other recent works demonstrated other practical attacks exploiting RowHammer. These include remote takeover of a server vulnerable to RowHammer, takeover of a victim virtual machine by another virtual machine running on the same system, and takeover of a mobile device by a malicious user-level application that requires no permissions.

We analyze the root causes of the RowHammer problem and examine various solutions. We also discuss what other vulnerabilities may be lurking in DRAM and other types of memories, e.g., NAND flash memory or Phase Change Memory, that can potentially threaten the foundations of secure systems, as the memory technologies scale to higher densities. We conclude by describing and advocating a principled approach to memory reliability and security research that can enable us to better anticipate and prevent such vulnerabilities.

I. Introduction

Memory is a key component of all modern computing systems, often determining the overall performance, energy efficiency, and reliability characteristics of the entire system. The push for increasing the density of modern memory technologies via technology scaling, which has resulted in higher capacity (i.e., density) memory and storage at lower cost, has enabled large leaps in the performance of modern computers [77]. This positive trend is clearly visible in especially the dominant main memory and solid-state storage technologies of today, i.e., DRAM [62, 28] and NAND flash memory [16], respectively. Unfortunately, the same push has also greatly decreased the reliability of modern memory technologies, due to the increasingly smaller memory cell size and increasingly smaller amount of charge that is maintainable in the cell, which makes the memory cell much more vulnerable to various failure mechanisms and noise and interference sources, both in DRAM [69, 53, 46, 45] and NAND flash [16, 20, 19, 22, 24, 17, 18, 23, 21, 72].

In this work, and the associated invited special session talk, we discuss the effects of reduced memory reliability on system security. As memory scales down to smaller technology nodes, new failure mechanisms emerge that threaten its correct operation. If such failure mechanisms are not anticipated and corrected, they can not only degrade system reliability and availability but also, perhaps even more importantly, open up security vulnerabilities: a malicious attacker can exploit the exposed failure mechanism to take over the entire system. As such, new failure mechanisms in memory can become practical and significant threats to system security.

We first discuss the RowHammer problem in DRAM, as a prime example of such a failure mechanism. We believe RowHammer is the first demonstration of how a circuit-level failure mechanism in DRAM can cause a practical and widespread system security vulnerability (Section II). After analyzing RowHammer in detail, we describe solutions to it (Section II-C). We then turn our attention to other vulnerabilities that may be present or become present in DRAM and other types of memories (Section III), e.g., NAND flash memory or Phase Change Memory, that can potentially threaten the foundations of secure systems, as the memory technologies scale to higher densities. We conclude by describing and advocating a principled approach to memory reliability and security research that can enable us to better anticipate and prevent such vulnerabilities (Section IV).

II. The RowHammer Problem

Memory isolation is a key property of a reliable and secure computing system. An access to one memory address should not have unintended side effects on data stored in other addresses. However, as process technology scales down to smaller dimensions, memory chips become more vulnerable to *disturbance*, a phenomenon in which different memory cells interfere with each others' operation. We have shown, in our ISCA 2014 paper [53], the existence of *disturbance errors* in commodity DRAM chips that are sold and used in the field today. Repeatedly reading from the same address in DRAM could corrupt data in nearby addresses. Specifically, when a DRAM row is opened (i.e., activated) and closed (i.e., precharged) repeatedly (i.e., *hammered*), enough times within a DRAM refresh interval, one or more bits in physically-adjacent DRAM rows can be flipped to the wrong value. This DRAM failure mode is now popularly called *RowHammer* [55, 99, 1, 2, 57, 10, 33, 89, 90, 13, 86, 98]. Using an FPGA-based experimental DRAM testing infrastructure, which we originally developed for testing retention time issues in DRAM [69],[1] we tested 129 DRAM modules manufactured by three major manufacturers (A, B, C) in seven recent years (2008–2014) and found that 110 of them exhibited RowHammer errors, the earliest of which dates back to 2010. This is illustrated in Figure 1, which shows the error rates we found in all 129 modules we tested where modules are categorized based on manufacturing date.[2] In particular, *all* DRAM modules from 2012–2013 were vulnerable to RowHammer, indicating that RowHammer is a recent phenomenon affecting more advanced process technology generations.

[1] This infrastructure is currently released to the public, and is described in detail in our HPCA 2017 paper [39]. The infrastructure has enabled many studies [63, 69, 53, 39, 28, 47, 46, 84, 48] into the failure and performance characteristics of modern DRAM, which were previously not well understood.

[2] Test details and experimental setup, along with a listing of all modules and their characteristics, are reported in our original RowHammer paper [53].

978-3-9815370-8-6/17/$31.00 ©2017 IEEE

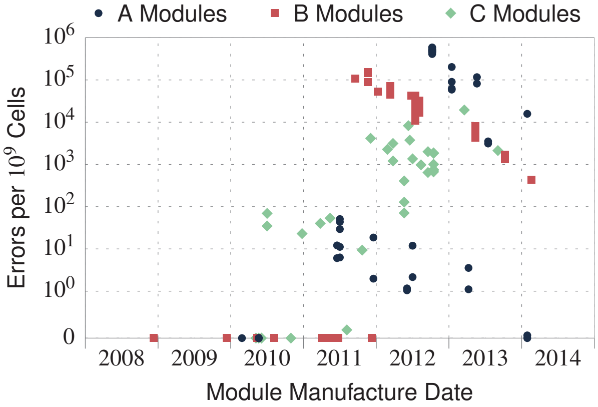

Fig. 1: RowHammer error rate vs. manufacturing dates of 129 DRAM modules we tested (reproduced from [53]).

A. User-Level RowHammer

We have also demonstrated that a very simple user-level program [53, 3] can reliably and consistently induce RowHammer errors in three commodity AMD and Intel systems using vulnerable DRAM modules. We released the source code of this program [3], which Google Project Zero later enhanced [4]. Using our user-level RowHammer test program, we showed that RowHammer errors violate two invariants that memory should provide: (i) a read access should not modify data at any address and (ii) a write access should modify data only at the address that it is supposed to write to. As long as a row is repeatedly opened, both read and write accesses can induce RowHammer errors, all of which occur in rows other than the one that is being accessed. Since different DRAM rows are mapped (by the memory controller) to different software pages, our user-level program could reliably corrupt specific bits in pages belonging to other programs. As a result, RowHammer errors can be exploited by a malicious program to breach memory protection and compromise the system. In fact, we hypothesized, in our ISCA 2014 paper, that our user-level program, with some engineering effort, could be developed into a *disturbance attack* that injects errors into other programs, crashes the system, or hijacks control of the system. We left such research for the future since our primary objective was to understand and prevent RowHammer errors [53].[3]

B. RowHammer as a Security Threat

RowHammer exposes a *security threat* since it leads to a breach of memory protection, where accesses to one row (e.g., an OS page) modifies the data stored in another memory row (e.g., another OS page). As indicated above, malicious software can be written to take advantage of these disturbance errors. We call these *disturbance attacks* [53], or *RowHammer attacks*. Such attacks can be used to corrupt system memory, crash a system, or take over the entire system. Confirming the predictions of our ISCA paper [53], researchers from Google Project Zero developed a user-level attack that exploits RowHammer to gain kernel privileges and thus take over an entire system [89, 90]. More recently, researchers showed that RowHammer can be exploited in various ways to take over various classes of systems. For example, RowHammer vulnerability in a remote server can be exploited to remotely take over the server via the use of JavaScript [33]. Or, a virtual machine can take over a victim virtual machine running on the same system by inducing RowHammer errors in the victim virtual machine's memory space [86]. Or, a malicious application that requires no permissions can take control of a mobile device by exploiting RowHammer, as demonstrated in real Android devices [98]. Or, an attacker can gain arbitrary read and write access in a web browser by exploiting RowHammer together with memory deduplication, as demonstrated on a real Microsoft Windows 10 system [13]. As such, the RowHammer problem has widespread and profound real implications on system security, threatening the foundations of memory isolation on top of which modern system security principles are built.

As described above, RowHammer has recently been the subject of many popular analyses and discussions on hardware-induced security problems [55, 99, 1, 2, 35, 57, 10, 33, 98, 86, 13, 90] as well as the prime vulnerability exploited by various software-level attacks that rely on no permissions or software vulnerabilities [33, 98, 86, 13]. As a result of the severity of the security problem, several major system manufacturers increased DRAM refresh rates to reduce the probability of occurrence of RowHammer [9, 40, 67, 32]. And, multiple memory test programs have been augmented to test for RowHammer errors [80, 8, 98]. Unfortunately, some recent reports suggest that even state-of-the-art DDR4 DRAM chips are vulnerable to RowHammer [57], which suggests that the security vulnerabilities might continue in the current generation of DRAM chips as well. As such, it is critical to investigate solutions to the RowHammer vulnerability.

C. Solutions to RowHammer

Given that it is such a critical vulnerability, it is important to find both *immediate* and *long-term* solutions to the RowHammer problem (as well as related problems that might cause similar vulnerabilities). The goal of the immediate solutions is to ensure that existing systems are patched such that the vulnerable DRAM devices that are already in the field cannot be exploited. The goal of the long-term solutions is to ensure that future DRAM devices do not suffer from the RowHammer problem when they are released into the field.

Given that immediate solutions require mechanisms that already exist in systems operating in the field, they are fundamentally more limited. A popular immediate solution, described and analyzed by our ISCA 2014 paper [53], is to increase the refresh rate of memory such that the probability of inducing a RowHammer error before DRAM cells get refreshed is reduced. Several major system manufacturers have adopted this solution and released security patches that increased DRAM refresh rates (e.g., [9, 40, 67, 32]) in the memory controllers. While this solution might be practical and effective in reducing the vulnerability, it has the significant drawbacks of increasing energy/power consumption, reducing system performance, and degrading quality of service experienced by user programs. Our paper shows that the refresh rate needs to be increased by 7X if we want to eliminate all RowHammer-induced errors we saw in our tests of 129 DRAM modules. Since DRAM refresh is already a significant burden [68, 26, 45, 46, 84] on energy consumption, performance, and quality of service, increasing it by any significant amount would only exacerbate the problem. Yet, increased refresh rate is likely the most practical *immediate* solution to RowHammer.

Other immediate solutions modify the software. For example, ANVIL proposes software-based detection of RowHammer attacks by monitoring via hardware performance counters and selective explicit refreshing of victim rows that are found to be vulnerable [10]. Unfortunately, such solutions require modifications to system software and might be intrusive to system operation (yet they are a promising area of research).

Our ISCA 2014 paper [53] discusses and analyzes seven long-term countermeasures to the RowHammer problem. The first six solutions are: 1) making better DRAM chips that are not vulnerable, 2) using error correcting codes (ECC), 3) increasing the refresh rate (as discussed above), 4) remapping RowHammer-prone cells after manufacturing, 5) remapping/retiring RowHammer-prone cells at the user level during operation, 6) accurately identifying hammered rows during runtime and refreshing their neighbors. None of these first six solutions are very desirable as they come at various significant power, performance or cost overheads. We already discussed the overheads of increasing the refresh rates across the board. Similarly,

[3] Our ISCA 2014 paper [53] provides a detailed analysis of various characteristics of RowHammer, including its data pattern dependence, relationship with leaky cells, repeatability of errors, and various circuit-level causes of the phenomenon. We omit these analyses here and focus on security vulnerabilities and prevention of RowHammer, and refer the reader to [53] for a rigorous treatment of the characteristics and causes of the RowHammer phenomenon.

simple SECDED ECC (an example of the second solution above), as employed in many systems, is not enough to prevent all RowHammer errors, as some cache blocks experience two or more bit flips, which are not correctable by SECDED ECC, as we have shown [53]. Thus, stronger ECC is likely required to correct RowHammer errors, which comes at the cost of additional energy, performance, cost, and DRAM capacity overheads. Alternatively, the sixth solution described above, i.e., accurately identifying a row as a hammered row requires keeping track of access counters for a large number of rows in the memory controller [50], leading to very large hardware area and power consumption, and potentially performance, overheads.

We believe the long-term solution to RowHammer can actually be very simple and low cost: when the memory controller closes a row (after it was activated), it, with a very low probability, refreshes the adjacent rows. The probability value is a parameter determined by the system designer or provided programmatically, if needed, to trade off between performance overhead and vulnerability protection guarantees. We show that this probabilistic solution, called *PARA (Probabilistic Adjacent Row Activation)*, is extremely effective: it eliminates the RowHammer vulnerability, providing much higher reliability guarantees than modern hard disks today, while requiring no storage cost and having negligible performance and energy overheads [53].

PARA is not immediately implementable because it requires changes to either the memory controllers or the DRAM chips, depending on where it is implemented. If PARA is implemented in the memory controller, the memory controller needs to obtain information on which rows are adjacent to each other in a DRAM bank. This information is currently *unknown* to the memory controller as DRAM manufacturers can internally remap rows to other locations [69, 53, 48, 47, 65] for various reasons including for tolerating various types of faults. However, this information can be simply provided by the DRAM chip to the memory controller using the serial presence detect (SPD) read-only memory present in modern DRAM modules, as described in our ISCA 2014 paper [53]. If PARA is implemented in the DRAM chip, then the hardware interface to the DRAM chip should be such that it allows DRAM-internal refresh operations that are not initiated by an external memory controller. This could be achieved with the addition of a new DRAM command, like the *targeted refresh* command proposed in a patent by Intel [11]. In 3D-stacked memory technologies [54, 66], e.g., HBM (High Bandwidth Memory) [43, 66] or HMC (Hybrid Memory Cube) [5], which combine logic and memory in a tightly integrated fashion, the logic layer can be easily modified to implement PARA.

All these implementations of the promising PARA solution are examples of much better cooperation between memory controller and the DRAM chips. Regardless of the exact implementation, we believe RowHammer, and other upcoming reliability vulnerabilities like RowHammer, can be much more easily found, mitigated, and prevented with better cooperation between and co-design of system and memory, i.e., *system-memory co-design* [77]. System-memory co-design is explored by recent works for mitigating various DRAM scaling issues, including retention failures and performance problems [68, 52, 77, 45, 79, 70, 46, 84, 48, 47, 63, 62, 91, 27, 28, 69, 26, 53, 65, 38, 93, 64, 92]. Taking the system-memory co-design approach further, providing more intelligence and configurability/programmability in the memory controller can greatly ease the tolerance to errors like RowHammer: when a new failure mechanism in memory is discovered, the memory controller can be configured/programmed to execute specialized functions to profile and correct for such mechanisms. We believe this direction is very promising, and several works have explored *online profiling* mechanisms for fixing retention errors [46, 84, 48, 47] and reducing latency [65]. These works provide examples of how an intelligent memory controller can alleviate the retention failures, and thus the DRAM refresh problem [68, 69], as well as the DRAM latency problem [62, 63].

D. Putting RowHammer into Context

Springing off from the stir created by RowHammer, we take a step back and argue that there is little that is surprising about the fact that we are seeing disturbance errors in the heavily-scaled DRAM chips of today. Disturbance errors are a general class of reliability problem that is present in not only DRAM, but also other memory and storage technologies. All scaled memory technologies, including SRAM [30, 34, 49], flash [16, 20, 19, 23, 31], and hard disk drives [44, 97, 102], exhibit such disturbance problems. In fact, our recent work at DSN 2015 [23] experimentally characterizes the read disturb errors in flash memory, shows that the problem is widespread in the flash devices, and develops mechanisms to correct such errors in the flash memory controller. Even though the mechanisms that cause the bit flips are different in different technologies, the high-level root cause of the problem, *cell-to-cell interference*, i.e., that the memory cells are too close to each other, is a fundamental issue that appears and will appear in any technology that scales down to small enough technology nodes. Thus, we should expect such problems to continue as we scale any memory technology, including emerging ones.

What sets DRAM disturbance errors apart from other technologies' disturbance errors is that in modern DRAM, as opposed to other technologies, error correction mechanisms are not commonly employed (either in the memory controller or the memory chip). The success of DRAM scaling until recently has *not* relied on a memory controller that corrects errors (other than performing periodic refresh). Instead, DRAM chips were implicitly assumed to be error-free and did *not* require the help of the controller to operate correctly. Thus, such errors were perhaps not as easily anticipated and corrected within the context of DRAM. In contrast, the success of other technologies, e.g., flash memory and hard disks, has heavily relied on the existence of an intelligent controller that plays a key role in correcting errors and making up for reliability problems of the memory chips themselves. This has not only enabled the correct operation of assumed-faulty memory chips but also enabled a mindset where the controllers are co-designed with the chips themselves, covering up the memory technology's deficiencies and hence perhaps enabling better anticipation of errors with technology scaling. This approach is very prominent in modern SSDs (solid state drives), for example, where the flash memory controller employs a wide variety of error mitigation and correction mechanisms [17, 16, 20, 19, 21, 23, 24, 22, 18, 72], including not only sophisticated ECC mechanisms but also targeted voltage optimization, retention mitigation and disturbance mitigation techniques. We believe changing the mindset in modern DRAM to a similar mindset of *assumed-faulty memory chip and an intelligent memory controller that makes it operate correctly* can not only enable better anticipation and correction of future issues like RowHammer but also better scaling of DRAM into future technology nodes [77].

III. Other Potential Vulnerabilities

We believe that, as memory technologies scale to higher densities, other problems may start appearing (or may already be going unnoticed) that can potentially threaten the foundations of secure systems. There have been recent large-scale field studies of memory errors showing that both DRAM and NAND flash memory technologies are becoming less reliable [76, 94, 95, 96, 75, 88]. As detailed experimental analyses of real DRAM and NAND flash chips show, both technologies are becoming much more vulnerable to cell-to-cell interference effects [53, 23, 21, 19, 16, 20, 78, 72, 24], data retention is becoming significantly more difficult in both technologies [68, 46, 69, 48, 84, 26, 45, 73, 22, 17, 71, 16, 20, 18, 78, 47], and error variation within and across chips is increasingly prominent [69, 63, 28, 25, 16, 20, 65]. Emerging memory technologies [77, 74], such as Phase-Change Memory [58, 106, 83, 82, 100, 85, 60, 59, 105, 104], STT-MRAM [29, 56], and RRAM/ReRAM/memristors [101] are likely to exhibit similar and perhaps even more exacerbated reliability issues. We believe, if not carefully accounted for and corrected, these reliability problems may surface as security problems as well, as in the case of RowHammer, especially if the technology is employed as part of the main memory system.

We briefly examine two example potential vulnerabilities. We believe future work examining these vulnerabilities, among others, are promising for both fixing the vulnerabilities and enabling the effective scaling of memory technology.

A. Data Retention Failures

Data retention is a fundamental reliability problem, and hence a potential vulnerability, in charge-based memories like DRAM and flash memory. This is because charge leaks out of the charge storage unit (e.g., the DRAM capacitor or the NAND flash floating gate) over time. As such memories become denser, three major trends make data retention more difficult [68, 69, 45, 22]. First, the number of memory cells increases, leading to the need for more refresh operations to maintain data correctly. Second, the charge storage unit (e.g., the DRAM capacitor) becomes smaller and/or morphs in structure, leading to potentially lower retention times. Third, the voltage margins that separate one data value from another become smaller (e.g., the same voltage window gets divided into more "states" in NAND flash memory, to store more bits per cell), and as a result the same amount of charge loss is more likely to cause a bit error in a smaller technology node than a larger one.

1) DRAM Data Retention Issues: Data retention issues in DRAM are a fundamental scaling limiter of the DRAM technology [69, 45]. We have shown, in recent works based on rigorous experimental analyses of modern DRAM chips [69, 46, 84, 48], that determining the minimum retention time of a DRAM cell is getting significantly more difficult. Thus, determining the correct rate at which to refresh DRAM cells has become more difficult, as also indicated by industry [45]. This is due to two major phenomena, both of which get worse (i.e., become more prominent) with technology scaling. First, Data Pattern Dependence (DPD): the retention time of a DRAM cell is heavily dependent on the data pattern stored in it and the neighboring cells [69]. Second, Variable Retention Time (VRT): the retention time of some DRAM cells can change drastically over time, due to a memoryless random process that results in very fast charge loss via a phenomenon called trap-assisted gate-induced drain leakage [103, 87, 69]. These phenomena greatly complicate the accurate determination of minimum data retention time of DRAM cells. In fact, VRT, as far as we know, is very difficult to test for because the there seems to be no way of determining that a cell exhibits VRT until that cell is observed to exhibit VRT and the time scale of a cell exhibiting VRT does not seem to be bounded given the current experimental data [69]. As a result, some retention errors can easily slip into the field because of the difficulty of the retention time testing. Therefore, data retention in DRAM is a vulnerability that can greatly affect both reliability and security of current and future DRAM generations. We encourage future work to investigate this area further, from reliability and security, *as well as* performance and energy efficiency perspectives. Various works in this area provide insights about the retention time properties of modern DRAM devices based on experimental data [69, 46, 84, 48, 39], develop infrastructures to obtain valuable experimental data [39], and provide potential solutions to the DRAM retention time problem [68, 69, 46, 84, 48, 47, 26], all of which the future works can build on.

Note that data retention failures in DRAM are likely to be investigated heavily to ensure good performance and energy efficiency. And, in fact they already are (see, for example, [68, 26, 47, 46, 48]). We believe it is important for such investigations to ensure no new vulnerabilities (e.g., side channels) open up due to the solutions developed.

2) NAND Flash Data Retention Issues: Experimental analysis of modern flash memory devices show that the dominant source of errors in flash memory are data retention errors [16]. As a flash cell wears out, its charge retention capability degrades [16, 22] and the cell becomes leakier. As a result, to maintain the original data stored in the cell, the cell needs to be refreshed [17, 18]. The frequency of refresh increases as wearout of the cell increases. We have shown that performing refresh in an adaptive manner greatly improves the lifetime of modern MLC (multi-level cell) NAND flash memory while causing little energy and performance overheads [17, 18]. Most high-end SSDs today employ refresh mechanisms.

As flash memory scales to smaller nodes and even more bits per cell, data retention becomes a bigger problem. As such, it is critical to understand the issues with data retention in flash memory. Our recent work provides detailed experimental analysis of data retention behavior of MLC NAND flash memory [22]. We show, among other things, that there is a wide variation in the leakiness of different flash cells: some cells leak very fast, some cells leak very slowly. This variation leads to new opportunities for correctly recovering flash data from a flash device that has experienced an uncorrectable error: by identifying which cells are fast-leaking and which cells are slow-leaking, one can probabilistically estimate the original values of the cells before the uncorrectable error occurred. This mechanism, called *Retention Failure Recovery*, leads to significant reductions in bit error rate in modern MLC NAND flash memory [23] and is thus very promising. Unfortunately, it also points out to a potential security and privacy vulnerability: by analyzing data and cell properties of a failed device, one can potentially recover the original data. We believe such vulnerabilities can become more common in the future and therefore they need to be anticipated, investigated, and understood.

B. Other Vulnerabilities in NAND Flash Memory

We believe other sources of error (e.g., cell-to-cell interference) and cell-to-cell variation in flash memory can also lead various vulnerabilities. For example, a similar type of variation (to the variation in cell leakiness that we described above) exists in the vulnerability of flash memory cells to read disturbance [23]: some cells are much more prone to read disturb effects than others. This wide variation among cells enables one to probabilistically estimate the original values of cells in flash memory after an uncorrectable error has occurred. Similarly, one can probabilistically correct the values of cells in a page by knowing the values of cells in the neighboring page [21]. These mechanisms [23, 21] are devised to improve flash memory reliability and lifetime, but the same phenomena that make them effective in doing so can also lead to potential vulnerabilities, which we believe are worthy of investigation to ensure security and privacy of data in flash memories.

As an example, we have recently shown [24] that it is theoretically possible to exploit vulnerabilities in flash memory programming operations on existing solid-state drives (SSDs) and to cause (malicious) data corruption. This particular vulnerability is caused by the *two-step programming* method employed in dense flash memory devices, e.g., MLC NAND flash memory. An MLC device partitions the threshold voltage range of a flash cell into four distributions. In order to reduce the number of errors introduced during programming of a cell, flash manufacturers adopt a two-step programming method, where the least significant bit of the cell is partially programmed first to some intermediate threshold voltage, and the most significant bit is programmed later to bring the cell up to its full threshold voltage. We find that two-step programming exposes new vulnerabilities, as both cell-to-cell program interference and read disturbance can disrupt the intermediate value stored within a multi-level cell before the second programming step completes. We show that it is possible to exploit these vulnerabilities on existing solid-state drives (SSDs) to alter the partially-programmed data, causing (malicious) data corruption. We experimentally characterize the extent of these vulnerabilities using contemporary 1X-nm (i.e., 15-19nm) flash chips [24]. Building on our experimental observations, we propose several new mechanisms for MLC NAND flash that eliminate or mitigate disruptions to intermediate values, removing or reducing the extent of the vulnerabilities, mitigating potential exploits and increasing flash lifetime by 16% [24]. We believe investigation of such vulnerabilities in flash memory will lead to more robust flash memory devices in terms of both reliability and security, as well as performance.

IV. Prevention

Various reliability problems experienced by scaled memory technologies, if not carefully anticipated, accounted for, and corrected, may surface as security problems as well, as in the case of RowHammer. We believe it is critical to develop principled methods to understand, anticipate, and prevent such vulnerabilities. In particular, principled methods are required for three major steps in the design process.

First, it is critical to understand the potential failure mechanisms and anticipate them beforehand. To this end, developing solid methodologies for failure modeling and prediction is critical. To develop such methodologies, it is essential to have real experimental data from past and present devices. Data available both at the small scale (i.e., controlled testing of individual devices, as in, e.g., [69, 63, 16, 20, 72]) as well as the large scale (i.e., data obtained during in-the-field operation of the device, under likely-uncontrolled conditions, as in, e.g., [76, 75]) can enable accurate models for failures, which could aid many purposes, including the development of better reliability mechanisms and prediction of problems before they occur.

Second, it is critical to develop principled architectural methods that can avoid, tolerate, or prevent such failure mechanisms that can lead to vulnerabilities. For this, we advocate co-architecting of the system and the memory together, as we described earlier. Designing intelligent, flexible, and configurable memory controllers that can understand and correct existing and potential failure mechanisms can greatly alleviate the impact of failure mechanisms on reliability, security, performance, and energy efficiency. Described in Section II-C, this *system-memory co-design* approach can also enable new opportunities, like performing effective processing near or in the memory device [91, 92, 6, 7, 93, 42, 41, 12, 27, 81, 36, 37]. In addition to designing the memory device together with the controller, we believe it is important to investigate mechanisms for good partitioning of duties across the various levels of transformation in computing, including system software, compilers, and application software.

Third, it is critical to develop principled methods for electronic design, automation and testing, which are in harmony with the failure modeling/prediction and system reliability methods, which we mentioned in the two above paragraphs. Design, automation and testing methods need to provide high and predictable coverage of failures and work in conjunction with architectural and across-stack mechanisms. For example, enabling effective and low-cost *online profiling of DRAM* [69, 46, 84, 48, 47] in a principled manner requires cooperation of failure modeling mechanisms, architectural methods, and design, automation and testing methods.

V. Conclusion

It is clear that reliability of memory technologies we greatly depend on reducing, as these technologies continue to scale to ever smaller technology nodes in pursuit of higher densities. These reliability problems, if not anticipated and corrected, can also open up serious security vulnerabilities, which can be very difficult to defend against if they are discovered in the field. RowHammer is an example, likely the first one, of a hardware failure mechanism that causes a practical and widespread system security vulnerability. As such, its implications on system security research are tremendous and exciting. Prevention of such vulnerabilities opens up new avenues for principled approaches to understanding of failures and failure modeling, architectural methods for ensuring reliable operation, and design, automation, and testing. We believe the future is very bright for research in reliable and secure memory systems, and many discoveries abound in the exciting yet complex intersection of reliability and security issues in such systems.

Acknowledgments

This paper, and the associated talk, are a result of the research done together with many students and collaborators over the course of the past 4-5 years. We acknowledge their contributions. In particular, three PhD theses have shaped the understanding that led to this work. These are Yoongu Kim's thesis entitled "Architectural Techniques to Enhance DRAM Scaling" [51], Yu Cai's thesis entitled "NAND flash memory: Characterization, Analysis, Modeling and Mechanisms" [15] and his continued follow-on work after his thesis, and Donghyuk Lee's thesis entitled "Reducing DRAM Latency at Low Cost by Exploiting Heterogeneity" [61]. We also acknowledge various funding agencies (NSF, SRC, ISTC, CYLab) and industrial partners (AMD, Google, Facebook, HP Labs, Huawei, IBM, Intel, Microsoft, Nvidia, Oracle, Qualcomm, Rambus, Samsung, Seagate, VMware) whom have supported the presented and other related work generously over the years. The first version of this talk was delivered at a CyLab Partners Conference in September 2015. A prior version of the talk was delivered as part of an Invited Session at DAC 2016, with a collaborative accompanying paper entitled "Who Is the Major Threat to Tomorrows Security? You, the Hardware Designer" [14].

References

[1] RowHammer Discussion Group. https://groups.google.com/forum/#!forum/rowhammer-discuss.

[2] RowHammer on Twitter. https://twitter.com/search?q=rowhammer&src=typd.

[3] rowhammer: Source code for testing the row hammer error mechanism in dram devices. https://github.com/CMU-SAFARI/rowhammer.

[4] Test DRAM for bit flips caused by the rowhammer problem. https://github.com/google/rowhammer-test.

[5] *Hybrid Memory Consortium*, 2012. http://www.hybridmemorycube.org.

[6] J. Ahn et al. A Scalable Processing-in-Memory Accelerator for Parallel Graph Processing. In *ISCA*, 2015.

[7] J. Ahn et al. PIM-Enabled Instructions: A Low-Overhead, Locality-Aware Processing-in-Memory Architecture. In *ISCA*, 2015.

[8] B. Aichinger. The Known Failure Mechanism in DDR3 Memory referred to as Row Hammer. http://ddrdetective.com/files/6414/1036/5710/The_Known_Failure_Mechanism_in_DDR3_memory_referred_to_as_Row_Hammer.pdf, September 2014.

[9] Apple Inc. About the security content of Mac EFI Security Update 2015-001. https://support.apple.com/en-us/HT204934, June 2015.

[10] Z. B. Aweke et al. Anvil: Software-based protection against next-generation rowhammer attacks. In *ASPLOS*, 2016.

[11] K. Bains et al. Row hammer refresh command. U.S. Patent Number 9117544 B2, 2015.

[12] A. Boroumand et al. LazyPIM: An Efficient Cache Coherence Mechanism for Processing-in-Memory. *IEEE CAL*, 2016.

[13] E. Bosman et al. Dedup Est Machina: Memory Deduplication as an Advanced Exploitation Vector. *S&P*, 2016.

[14] W. Burleson et al. Who Is the Major Threat to Tomorrow's Security? You, the Hardware Designer. *DAC*, 2016.

[15] Y. Cai. *NAND flash memory: Characterization, Analysis, Modeling and Mechanisms*. PhD thesis, Carnegie Mellon University, 2012.

[16] Y. Cai et al. Error patterns in MLC NAND flash memory: Measurement, characterization, and analysis. In *DATE*, 2012.

[17] Y. Cai et al. Flash Correct-and-Refresh: Retention-aware error management for increased flash memory lifetime. In *ICCD*, 2012.

[18] Y. Cai et al. Error Analysis and Retention-Aware Error Management for NAND Flash Memory. *ITJ*, 2013.

[19] Y. Cai et al. Program interference in MLC NAND flash memory: Characterization, modeling, and mitigation. In *ICCD*, 2013.

[20] Y. Cai et al. Threshold voltage distribution in MLC NAND flash memory: Characterization, analysis and modeling. In *DATE*, 2013.

[21] Y. Cai et al. Neighbor-cell assisted error correction for MLC NAND flash memories. In *SIGMETRICS*, 2014.

[22] Y. Cai et al. Data retention in MLC NAND flash memory: Characterization, optimization and recovery. In *HPCA*, 2015.

[23] Y. Cai et al. Read Disturb Errors in MLC NAND Flash Memory: Characterization, Mitigation, and Recovery. In *DSN*, 2015.

[24] Y. Cai et al. Vulnerabilities in MLC NAND Flash Memory Programming: Experimental Analysis, Exploits, and Mitigation Techniques. In *HPCA*, 2017.

[25] K. Chandrasekar et al. Exploiting Expendable Process-margins in DRAMs for Run-time Performance Optimization. In *DATE*, 2014.

[26] K. Chang et al. Improving DRAM performance by parallelizing refreshes with accesses. In *HPCA*, 2014.

[27] K. Chang et al. Low-Cost Inter-Linked Subarrays (LISA): Enabling Fast Inter-Subarray Data Movement in DRAM. In *HPCA*, 2016.

[28] K. Chang et al. Understanding Latency Variation in Modern DRAM Chips: Experimental Characterization, Analysis, and Optimization. *SIGMETRICS*, 2016.

[29] E. Chen et al. Advances and future prospects of spin-transfer torque random access memory. *IEEE Transactions on Magnetics*, 46(6), 2010.

[30] Q. Chen et al. Modeling and Testing of SRAM for New Failure Mechanisms Due to Process Variations in Nanoscale CMOS. In *VTS*, 2005.

[31] J. Cooke. The Inconvenient Truths of NAND Flash Memory. In *Flash Memory Summit*, 2007.

[32] T. Fridley and O. Santos. Mitigations Available for the DRAM Row Hammer Vulnerability. http://blogs.cisco.com/security/mitigations-available-for-the-dram-row-hammer-vulnerability, March 2015.

[33] D. Gruss et al. Rowhammer.js: A remote software-induced fault attack in javascript. *CoRR*, abs/1507.06955, 2015.

[34] Z. Guo et al. Large-Scale SRAM Variability Characterization in 45 nm CMOS. *JSSC*, 44(11), 2009.

[35] R. Harris. Flipping DRAM bits - maliciously. http://www.zdnet.com/article/flipping-dram-bits-maliciously/, December 2014.

[36] M. Hashemi et al. Accelerating Dependent Cache Misses with an Enhanced Memory Controller. In *ISCA*, 2016.

[37] M. Hashemi et al. Continuous Runahead: Transparent Hardware Acceleration for Memory Intensive Workloads. In *MICRO*, 2016.

[38] H. Hassan et al. ChargeCache: Reducing DRAM Latency by Exploiting Row Access Locality. In *HPCA*, 2016.

[39] H. Hassan et al. SoftMC: A Flexible and Practical Open-Source Infrastructure for Enabling Experimental DRAM Studies. In *HPCA*, 2017.

[40] Hewlett-Packard Enterprise. HP Moonshot Component Pack Version 2015.05.0. http://h17007.www1.hp.com/us/en/enterprise/servers/products/moonshot/component-pack/index.aspx, 2015.

[41] K. Hsieh et al. Accelerating Pointer Chasing in 3D-Stacked Memory: Challenges, Mechanisms, Evaluation. *ICCD*, 2016.

[42] K. Hsieh et al. Transparent Offloading and Mapping (TOM): Enabling Programmer-Transparent Near-Data Processing in GPU Systems. *ISCA*, 2016.

[43] JEDEC. JESD235 High Bandwidth Memory (HBM) DRAM, 2013.

[44] W. Jiang et al. Cross-Track Noise Profile Measurement for Adjacent-Track Interference Study and Write-Current Optimization in Perpendicular Recording. *Journal of Applied Physics*, 93(10), 2003.

[45] U. Kang et al. Co-architecting controllers and DRAM to enhance DRAM process scaling. In *The Memory Forum*, 2014.

[46] S. Khan et al. The efficacy of error mitigation techniques for DRAM retention failures: A comparative experimental study. *SIGMETRICS*, 2014.

[47] S. Khan et al. A Case for Memory Content-Based Detection and Mitigation of Data-Dependent Failures in DRAM. *CAL*, 2016.

[48] S. Khan et al. PARBOR: An Efficient System-Level Technique to Detect Data-Dependent Failures in DRAM. In *DSN*, 2016.

[49] D. Kim et al. Variation-Aware Static and Dynamic Writability Analysis for Voltage-Scaled Bit-Interleaved 8-T SRAMs. In *ISLPED*, 2011.

[50] D.-H. Kim et al. Architectural Support for Mitigating Row Hammering in DRAM Memories. *IEEE CAL*, 2015.

[51] Y. Kim. *Architectural Techniques to Enhance DRAM Scaling*. PhD thesis, Carnegie Mellon University, 2015.

[52] Y. Kim et al. A case for subarray-level parallelism (SALP) in DRAM. In *ISCA*, 2012.

[53] Y. Kim et al. Flipping bits in memory without accessing them: An experimental study of DRAM disturbance errors. In *ISCA*, 2014.

[54] Y. Kim et al. Ramulator: A Fast and Extensible DRAM Simulator. *IEEE CAL*, 2015.

[55] Y. Kim et al. RowHammer: Reliability Analysis and Security Implications. *ArXiV*, 2016.

[56] E. Kultursay et al. Evaluating STT-RAM as an energy-efficient main memory alternative. In *ISPASS*, 2013.

[57] M. Lanteigne. How Rowhammer Could Be Used to Exploit Weaknesses in Computer Hardware. http://www.thirdio.com/rowhammer.pdf, March 2016.

[58] B. C. Lee et al. Architecting phase change memory as a scalable DRAM alternative. In *ISCA*, 2009.

[59] B. C. Lee et al. Phase change memory architecture and the quest for scalability. *CACM*, 2010.

[60] B. C. Lee et al. Phase change technology and the future of main memory. *IEEE Micro*, 2010.

[61] D. Lee. Reducing DRAM Latency by Exploiting Heterogeneity. *ArXiV*, 2016.

[62] D. Lee et al. Tiered-latency DRAM: A low latency and low cost DRAM architecture. In *HPCA*, 2013.

[63] D. Lee et al. Adaptive-latency DRAM: Optimizing DRAM timing for the common-case. In *HPCA*, 2015.

[64] D. Lee et al. Decoupled Direct Memory Access: Isolating CPU and IO Traffic by Leveraging a Dual-Data-Port DRAM. In *PACT*, 2015.

[65] D. Lee et al. Reducing DRAM Latency by Exploiting Design-Induced Latency Variation in Modern DRAM Chips. *ArXiV*, 2016.

[66] D. Lee et al. Simultaneous Multi-Layer Access: Improving 3D-Stacked Memory Bandwidth at Low Cost. *TACO*, 2016.

[67] Lenovo. Row Hammer Privilege Escalation. https://support.lenovo.com/us/en/product_security/row_hammer, March 2015.

[68] J. Liu et al. RAIDR: Retention-aware intelligent DRAM refresh. *ISCA*, 2012.

[69] J. Liu et al. An experimental study of data retention behavior in modern DRAM devices: Implications for retention time profiling mechanisms. *ISCA*, 2013.

[70] Y. Luo et al. Characterizing application memory error vulnerability to optimize data center cost via heterogeneous-reliability memory. *DSN*, 2014.

[71] Y. Luo et al. WARM: Improving NAND Flash Memory Lifetime with Write-hotness Aware Retention Management. *MSST*, 2015.

[72] Y. Luo et al. Enabling Accurate and Practical Online Flash Channel Modeling for Modern MLC NAND Flash Memory. *JSAC*, 2016.

[73] J. Mandelman et al. Challenges and future directions for the scaling of dynamic random-access memory (DRAM). *IBM Journal of Research and Development*, 46, 2002.

[74] J. Meza et al. A case for efficient hardware-software cooperative management of storage and memory. In *WEED*, 2013.

[75] J. Meza et al. A Large-Scale Study of Flash Memory Errors in the Field. In *SIGMETRICS*, 2015.

[76] J. Meza et al. Revisiting Memory Errors in Large-Scale Production Data Centers: Analysis and Modeling of New Trends from the Field. *DSN*, 2015.

[77] O. Mutlu. Memory scaling: A systems architecture perspective. *IMW*, 2013.

[78] O. Mutlu. Error Analysis and Management for MLC NAND Flash Memory. In *Flash Memory Summit*, 2014.

[79] O. Mutlu and L. Subramanian. Research problems and opportunities in memory systems. *SUPERFRI*, 2014.

[80] PassMark Software. MemTest86: The original industry standard memory diagnostic utility. http://www.memtest86.com/troubleshooting.htm, 2015.

[81] A. Pattnaik et al. Scheduling Techniques for GPU Architectures with Processing-In-Memory Capabilities. *PACT*, 2016.

[82] M. K. Qureshi et al. Enhancing lifetime and security of phase change memories via start-gap wear leveling. In *MICRO*, 2009.

[83] M. K. Qureshi et al. Scalable high performance main memory system using phase-change memory technology. In *ISCA*, 2009.

[84] M. K. Qureshi et al. AVATAR: A Variable-Retention-Time (VRT) Aware Refresh for DRAM Systems. In *DSN*, 2015.

[85] S. Raoux et al. Phase-change random access memory: A scalable technology. *IBM Journal of Research and Development*, 2008.

[86] K. Razavi et al. Flip Feng Shui: Hammering a Needle in the Software Stack. *USENIX Security*, 2016.

[87] P. J. Restle et al. DRAM variable retention time. IEDM, 1992.

[88] B. Schroeder et al. Flash Reliability in Production: The Expected and the Unexpected. In *USENIX FAST*, 2016.

[89] M. Seaborn and T. Dullien. Exploiting the DRAM rowhammer bug to gain kernel privileges. http://googleprojectzero.blogspot.com.tr/2015/03/exploiting-dram-rowhammer-bug-to-gain.html.

[90] M. Seaborn and T. Dullien. Exploiting the DRAM rowhammer bug to gain kernel privileges. *BlackHat*, 2016.

[91] V. Seshadri et al. RowClone: Fast and Efficient In-DRAM Copy and Initialization of Bulk Data. In *MICRO*, 2013.

[92] V. Seshadri et al. Fast Bulk Bitwise AND and OR in DRAM. *CAL*, 2015.

[93] V. Seshadri et al. Gather-Scatter DRAM: In-DRAM Address Translation to Improve the Spatial Locality of Non-unit Strided Accesses. In *MICRO*, 2015.

[94] V. Sridharan, N. DeBardeleben, S. Blanchard, K. B. Ferreira, J. Stearley, J. Shalf, and S. Gurumurthi. Memory errors in modern systems: The good, the bad, and the ugly. In *ASPLOS*, 2015.

[95] V. Sridharan and D. Liberty. A study of DRAM failures in the field. In *SC*, 2012.

[96] V. Sridharan, J. Stearley, N. DeBardeleben, S. Blanchard, and S. Gurumurthi. Feng shui of supercomputer memory: positional effects in DRAM and SRAM faults. In *SC*, 2013.

[97] Y. Tang et al. Understanding Adjacent Track Erasure in Discrete Track Media. *Transactions on Magnetics*, 44(12), 2008.

[98] V. van der Veen et al. Drammer: Deterministic Rowhammer Attacks on Mobile Platforms. *CCS*, 2016.

[99] Wikipedia. Row hammer. https://en.wikipedia.org/wiki/Row_hammer.

[100] H.-S. P. Wong et al. Phase Change Memory. *Proceedings of the IEEE*, 2010.

[101] H.-S. P. Wong et al. Metal-oxide RRAM. In *Proceedings of the IEEE*, 2012.

[102] R. Wood et al. The Feasibility of Magnetic Recording at 10 Terabits Per Square Inch on Conventional Media. *Transactions on Magnetics*, 45(2), 2009.

[103] D. Yaney et al. A meta-stable leakage phenomenon in DRAM charge storage - Variable hold time. IEDM, 1987.

[104] H. Yoon et al. Row buffer locality aware caching policies for hybrid memories. In *ICCD*, 2012.

[105] H. Yoon et al. Efficient data mapping and buffering techniques for multi-level cell phase-change memories. *TACO*, 2014.

[106] P. Zhou et al. A durable and energy efficient main memory using phase change memory technology. In *ISCA*, 2009.