



Nethammer: Inducing Rowhammer Faults through Network Requests

Moritz Lipp, Misiker Tadesse Aga, Michael Schwarz, Daniel Gruss,
Clémentine Maurice, Lukas Raab, Lukas Lamster

► To cite this version:

Moritz Lipp, Misiker Tadesse Aga, Michael Schwarz, Daniel Gruss, Clémentine Maurice, et al..
Nethammer: Inducing Rowhammer Faults through Network Requests. 2018. hal-01872588v1

HAL Id: hal-01872588

<https://inria.hal.science/hal-01872588v1>

Preprint submitted on 12 Sep 2018 (v1), last revised 10 Nov 2020 (v2)

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Nethammer: Inducing Rowhammer Faults through Network Requests

Moritz Lipp
Graz University of Technology

Misiker Tadesse Aga
University of Michigan

Michael Schwarz
Graz University of Technology

Daniel Gruss
Graz University of Technology

Clémentine Maurice
Univ Rennes, CNRS, IRISA

Lukas Raab
Graz University of Technology

Lukas Lamster
Graz University of Technology

ABSTRACT

A fundamental assumption in software security is that memory contents do not change unless there is a legitimate deliberate modification. Classical fault attacks show that this assumption does not hold if the attacker has physical access. Rowhammer attacks showed that local code execution is already sufficient to break this assumption. Rowhammer exploits parasitic effects in DRAM to modify the content of a memory cell without accessing it. Instead, other memory locations are accessed at a high frequency. All Rowhammer attacks so far were local attacks, running either in a scripted language or native code.

In this paper, we present Nethammer. Nethammer is the first truly remote Rowhammer attack, without a single attacker-controlled line of code on the targeted system. Systems that use uncached memory or flush instructions while handling network requests, e.g., for interaction with the network device, can be attacked using Nethammer. Other systems can still be attacked if they are protected with quality-of-service techniques like Intel CAT. We demonstrate that the frequency of the cache misses is in all three cases high enough to induce bit flips. We evaluated different bit flip scenarios. Depending on the location, the bit flip compromises either the security and integrity of the system and the data of its users, or it can leave persistent damage on the system, *i.e.*, persistent denial of service.

We investigated Nethammer on personal computers, servers, and mobile phones. Nethammer is a security landslide, making the formerly local attack a remote attack. With this work we invalidate all defenses and mitigation strategies against Rowhammer build upon the assumption of a local attacker. Consequently, this paradigm shift impacts the security of millions of devices where the attacker is not able to execute attacker-controlled code. Nethammer requires threat models to be re-evaluated for most network-connected systems. We discuss state-of-the-art countermeasures and show that most of them have no effect on our attack, including the target-row-refresh (TRR) countermeasure of modern hardware.

Disclaimer: This work on Rowhammer attacks over the network was conducted independently and unaware of other research groups working on truly remote Rowhammer attacks. Experiments and observations presented in this paper, predate the publication of the Throwhammer attack by Tatar et al. [81]. We will thoroughly study the differences between both papers and compare the advantages and disadvantages in a future version of this paper.

1 INTRODUCTION

Hardware-fault attacks have been considered a security threat since at least 1997 [12, 13]. In such attacks, the attacker intentionally brings devices into physical conditions which are outside their specification for a short time. For instance, this can be achieved by temporarily using incorrect supply voltages, exposing them to high or low temperature, exposing them to radiation, or by dismantling the chip and shooting at it with lasers. Fault attacks typically require physical access to the device. However, if software can bring the device to the border or outside of the specified operational conditions, software-induced hardware faults are possible [50, 80].

The most prominent hardware fault which can be induced by software is the Rowhammer bug, caused by a hardware reliability issue of DRAM. An attacker can exploit this bug by repeatedly accessing (*hammering*) DRAM cells at a high frequency, causing unauthorized changes in physically adjacent memory locations. Since its initial discovery as a security issue [50], Rowhammer’s ability to defy abstraction barriers between different security domains has been improved gradually to develop more powerful attacks on various systems. Examples of previous attacks include privilege escalation, from native environments [27, 78], from within a browser’s sandbox [14, 25, 28], and from within virtual machines running on third-party compute clouds [86], mounting fault attacks on cryptographic primitives [11, 73], and obtaining root privileges on mobile phones [84].

Most Rowhammer attacks assume that two DRAM rows must be hammered to induce bit flips. The reason is that they assume that an “open-page” memory controller policy is used, *i.e.*, a DRAM row is kept open until a different row is accessed. However, modern CPUs employ more sophisticated memory controller policies that preemptively close rows [27]. Based on this observation, Gruss et al. [27] described a technique called *one-location* hammering.

In 2016, Intel introduced Cache Allocation Technology (CAT) to address quality of service in multi-core server platforms [32]. Intel CAT allows restricting cache allocation of cores to a subset of cache ways of the last-level cache, with the aim of optimizing workloads in shared environments, e.g., protecting virtual machines against performance degradation due to cache thrashing of a co-located virtual machine. However, with a lower number of cache ways available to the process, the probability to evict an address by accessing other addresses increases significantly. Aga et al. [4] showed that this facilitates eviction-based Rowhammer attacks.

All previously known Rowhammer attacks required some form of local code execution, e.g., JavaScript [14, 25, 28] or native code [4, 9, 11, 27, 50, 62, 69, 73, 78, 84, 86]. Moreover, all works on Rowhammer defenses assume that some form of local code execution is required [9, 14, 15, 17, 18, 26, 29, 31, 43, 49, 50, 59, 67, 74, 91]. In particular, we found that none of these works even mentions the theoretical possibility of truly non-local Rowhammer attacks. Consequently, it was a widely accepted assumption that remote Rowhammer attacks are not possible. More specifically, devices where an attacker could not obtain local code execution were so far considered to be safe. Yet, the following questions arise:

Are remote Rowhammer attacks possible? More specifically, is it possible for an attacker to induce bit flips and exploit them, without any local code execution on the system?

In this paper, we answer these questions and confirm that truly remote Rowhammer attacks are possible. We present Nethammer, the first Rowhammer attack that does not require local code execution. Nethammer requires only a fast network connection between the attacker and victim. It sends a crafted stream of size-optimized packets to the victim which causes a high number of memory accesses to the same set of memory locations. If the network driver or other parts of the network stack use uncached memory or flush instructions, e.g., for interaction with the network device, an attacker can induce bit flips. Furthermore, if Intel CAT is activated, e.g., as an anti-DoS mechanism, memory accesses lead to fast cache eviction and thus frequent DRAM accesses. This enables attacks even if there are no accesses to uncached memory or flush instructions while handling the network packet. Thus, the attacker implicitly hammers the DRAM through the code executed for processing the network packets. While an attacker cannot control the addresses of the bit flips, we demonstrate how an attacker can still exploit them.

Nethammer has several building blocks that we systematically developed. First, we measure whether handling network packets could at least, in theory, induce bit flips, and the influence of real-world memory-controller page policies. For this purpose, we present a new algorithm to observe and classify the memory-controller page policy. Second, based on these insights, we demonstrate that one-location hammering [27] does not require a closed-page policy, but instead, adaptive policies may also allow one-location hammering. Third, we investigate memory operations that occur while handling network requests. Fourth, we show that the time windows we observe between memory accesses from subsequent network requests enable Rowhammer attacks.

As previous work on Rowhammer showed, once a bit flips in a system, its security can be subverted. We present different attacks exploiting bit flips on victim machines to compromise various services, in particular, version-control systems, DNS servers and OCSP servers. In all cases, the triggered bit flips may induce persistent denial-of-service attacks by corrupting the persistent state, e.g., the file system on the remote machine. In our experiments, we observed bit flips using Nethammer already after 300 ms of running the attack and up to 10 000 bit flips per hour. Nethammer represents a significant paradigm shift, from local to remote attacks. Previous fault attacks required physical access or local code execution in the case of Rowhammer. Making Rowhammer possible over the network requires re-evaluating the threat model of virtually every

network-connected system. We discuss state-of-the-art countermeasures and show that most of them do not affect our attack, including the target-row-refresh (TRR) countermeasure in hardware. Furthermore, we evaluate the performance of different other proposed Rowhammer countermeasures against Nethammer. Nethammer is difficult to detect on systems where high network traffic is commonplace. Finally, we discuss how attacks like Nethammer can be mitigated.

Contributions. The contributions of this work are:

- We present Nethammer, the first truly remote Rowhammer attack, with not even a single line of attacker-controlled code running on the target device.
- We demonstrate Nethammer on devices that either use uncached memory or `clflush` while handling network packets.
- We demonstrate that even without uncached memory and `clflush`, attacks on cloud systems can still be practical.
- We illustrate how our attack invalidates assumptions from previous works, marking a paradigm shift, and requiring re-evaluation of the threat models of most network-connected systems.
- We show that many previously proposed defenses, e.g., TRR, do not work against our new attack.

Outline. The remainder of the paper is structured as follows. In Section 2, we provide background information. In Section 3, we overview the Nethammer attack. In Section 4, we describe the building blocks and obtain insights we need for Nethammer. In Section 5, we demonstrate how bit flips induced over the network can be exploited. In Section 6, we evaluate the performance of Nethammer in different scenarios on several different systems. In Section 7, we discuss and propose countermeasures. In Section 8, we discuss limitations of Nethammer. We conclude our work in Section 9.

2 BACKGROUND

In this section, we provide the necessary background information on DRAM, memory controller policies, and the Rowhammer attack. Furthermore, we discuss caches and cache eviction as well as the Intel CAT technology.

2.1 DRAM and Memory Controller Policies

DRAM Organization. Modern computers use DRAM as the main memory. To maximize data transfer rates, DRAM is organized for a high degree of parallelism, in a hierarchy of channels, DIMMs, ranks, bank groups, and banks. Most processors today support dual-channel or quad-channel configurations. The DIMMs are assigned to one of the channels. Each DIMM has one or more ranks, e.g., the two sides of the DIMM may form two ranks. Every rank is further subdivided into so-called *banks*, with each bank spanning over multiple chips. The number of banks in a rank is standardized [45], e.g., 8 banks on DDR3 and 16 banks on DDR4. Each bank is an array of *cells*, organized in *rows* and *columns*, storing the actual memory content. The row size, i.e., the amount of data that can be stored in all cells of one row, is defined to be 8 kB [45]. Each cell is made out of a capacitor and an access transistor. The charge of the capacitor represents the binary data value of the cell. Each cell in the grid is connected to the neighboring cells with a wire forming horizontal and vertical bit lines.

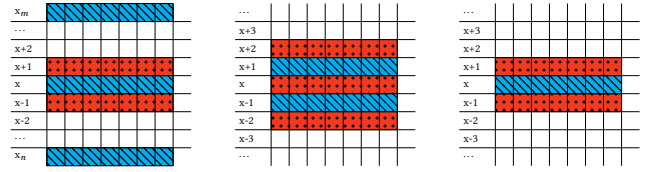
When accessing a physical address, the memory controller translates the physical address to channel, DIMM, rank, bank group, bank, row, and column addresses. While AMD publicly documents these addressing functions [3], Intel and ARM do not. Pessl et al. [68] reverse-engineered these addressing functions using an automated technique for several Intel and ARM processors.

As DRAM cells lose their charge over time, they must be re-freshed periodically. The maximum time interval between refreshes is defined through the *row refresh rate*, standardized by the JEDEC group for the different DRAM technologies [45]. Typically, the refresh interval is 64 ms but can vary depending on the device, on-the-fly adjustments due to the current temperature, or other external influences. With a 64 ms refresh interval, the memory controller issues the refresh command every 7.8 μ s for each bank.

Memory Controller Policies. Each bank has a *row buffer*, acting as a directly-mapped cache for the rows. To read data, the data is moved from the cells of a row to the row buffer before it is sent to the processor. Similarly, write accesses go to the row buffer instead of directly to the row. By raising the word line of a row, all access transistors in that row are activated to connect all capacitors to their respective bit line. This transfers the charge representing the data from the row to the row buffer. If the requested data from this bank is already stored in the row buffer, the data can be transmitted to the processor immediately, resulting in a fast access time (a *row hit*). However, if the requested data is not in the row buffer, a so-called *row conflict* occurs, and the bit lines must be *pre-charged* before the data can be read from the new target row (row-activate).

Consequently, there are three different cases leading to distinct access times: row hits are the fastest, an access to a row in a pre-charged bank is a few nanoseconds slower, row conflicts are significantly slower (*i.e.*, several nanoseconds). Hence, the memory controller can optimize the memory performance by deciding when to close a row preemptively and pre-charge the bank. Typically, memory controllers employ one of the three following page policies:

- (1) **Closed-page policy:** the page is immediately closed after every read or write request, and the bank is pre-charged and, thus, ready to open a new row (page-empty). If subsequent accesses are likely to be from other rows, a closed-page policy can achieve a better average system performance.
- (2) **Fixed open-page policy:** the page is left open for a fixed amount of time after a read or write request. If temporal locality is given, subsequent accesses are served with a low latency. This policy is also beneficial for power consumption and bank utilization [48].
- (3) **Adaptive open-page policy:** the adaptive open-page policy by Intel [21] is similar to the fixed open-page policy but dynamically adjusts the page timeout interval. Each row buffer has a timeout counter and a timeout register. A row remains open until the timeout counter reaches the value of the timeout register. As the initial timeout register value might not be the most efficient, an additional mistake counter is introduced to update the timeout register dynamically [26]. If a row conflict occurs, the memory controller kept the row open for too long and hence, the mistake counter is decremented. Whenever a page-empty access could have been a hit as the requested row is the same as the last accessed one, the mistake counter is incremented. Periodically, the value of the



(a) Single-sided (b) Double-sided (c) One-location

Figure 1: Different hammering strategies: blue rectangles (■) represent the hammered location, while red rectangles (■) represent the most likely location for bit flips to occur.

mistake counter is checked to decide if a less or more aggressive close-page policy should be used. If the mistake counter is higher than a certain threshold, the timeout register is incremented to keep the row open for a longer period of time, and conversely, if the mistake counter is lower than a certain threshold, the timeout register is decremented to close the row earlier.

As modern processors have many cores running independently as well as deploy large caches and complex algorithms for spatial and temporal prefetching, the probability that subsequent memory accesses go to the same row decreases. Awasthi et al. [8] proposed an access-based page policy that assumes a row receives the same number of accesses as the last time it was activated. Shen et al. [79] proposed a policy taking past memory accesses into account to decide whether to close a row preemptively. Intel suggested predicting how long a row should be kept open [47, 82]. Consequently, more complex memory controller policies have been proposed and are implemented in modern processors [26, 48]. Besides these memory controller policies, the memory controller can also reorder and combine memory accesses [76].

2.2 Rowhammer

With increasing DRAM cell density, the physical size of DRAM cells and their capacitance decreases. While this has the advantage of higher storage capacity and lower power consumption, cells may be more susceptible to disturbance errors. Disturbance errors are interferences between cells that cause memory corruption by unintentionally flipping the bit-value of a DRAM cell [62].

In 2014, Kim et al. [50] demonstrated that such bit flips could be reliably triggered in a DRAM row by accessing memory locations in adjacent DRAM rows in a high frequency, a technique known as *row hammering* [35]. Typically, subsequent memory accesses would be served from the CPU cache. However, in a Rowhammer attack, the cache is bypassed by either using specific instructions [50], cache eviction [4, 9, 25, 28] or uncached memory [69, 84].

To reliably induce bit flips, different techniques have been proposed using different memory access patterns as illustrated in Figure 1. While the name *single-sided hammering* suggests that only one memory location is accessed, Seaborn and Dullien [78] accessed 8 randomly chosen memory locations simultaneously. Seaborn and Dullien [78] focused on a typical DDR3 setup with 32 DRAM banks. Following the birthday paradox, the probability is quite high that at least 2 out of 8 random memory locations map into the same DRAM bank. By repeatedly accessing these 8 memory locations,

the attacker induces row conflicts at a high frequency. With single-sided hammering, bit flips most likely occur in some proximity to one of the 8 hammered rows.

With *double-sided hammering*, the attacker chooses three rows, where the two outer rows are hammered. Bit flips most likely occur in the row between the two rows. Double-sided hammering requires at least partial knowledge of virtual-to-physical mappings.

Finally, Gruss et al. [27] proposed *one-location hammering*, in which the attacker only accesses one single location at a high frequency. The attacker does not directly induce row conflicts but instead keeps re-opening one row permanently. As modern processors do not use strict open-page policies anymore, the memory controller preemptively closes rows earlier than necessary, causing row conflicts on the subsequent accesses of the attacker. Bit flips most likely occur in proximity to the hammered row.

Using these techniques, the Rowhammer bug has been exploited in different scenarios. Bhattacharya and Mukhopadhyay [11] exploited untargeted bit flips at random locations to produce faulty RSA signatures, allowing the recovery of the secret keys. However, as bit flips can be reproduced quite reliably, more deterministic attacks have been mounted. These attacks include privilege-escalation attacks, sandbox escapes and the compromise of cryptographic algorithms. They have been mounted from sandboxed environments [78], from native environments [27, 78], from virtual machines in the cloud [73, 86], as well as from within a web browser running JavaScript [14, 28]. Furthermore, attacks from native code [84] and JavaScript within the browser sandbox [25] have been demonstrated on mobile devices. To reliably induce a bit flip on a specific page, memory spraying [28, 78, 86], grooming [84], and page deduplication [14, 73] have been used.

To develop countermeasures, a large body of research focused on detecting [17, 18, 29, 31, 43, 67, 91], neutralizing [14, 15, 28, 73, 84], or eliminating [9, 15, 18, 26, 49, 50] Rowhammer attacks in software or hardware. Furthermore, the LPDDR4 standard [46] specifies two features to mitigate Rowhammer attacks: with Target Row Refresh (TRR) the memory controller refreshes adjacent rows of a certain row and with Maximum Activation Count (MAC) the number of times a row can be activated before adjacent rows have to be refreshed is specified. However, in 2018, Gruss et al. [27] showed that an attacker can bypass all software-based countermeasures and gain root privileges by mounting a one-location hammering Rowhammer attack from inside an Intel SGX enclave.

2.3 Caches and Cache Eviction

Caching is a fundamental concept that is used to reduce the latency of various operations, in particular computations and accesses to slower storage. Hardware caches keep frequently used data from main memory in smaller but faster memories.

Cache Organization. Modern CPUs have multiple levels of caches, varying in size and latency, where the level-1 (L1) cache is the smallest and fastest, and the L3 or last-level cache is the biggest but slowest cache. Modern caches are organized in cache sets consisting of a fixed number of cache ways. The cache set is determined by either the virtual or physical address. Addresses are called *congruent* if they map to the same cache set. The cache replacement policy

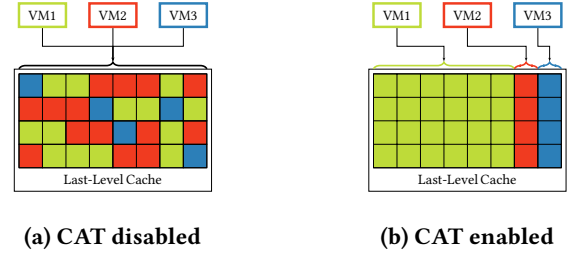


Figure 2: When Intel CAT is disabled in (a), the cache is shared among the virtual machines. In (b), Intel CAT is configured with 6 ways for VM1, and 1 way for VM2 and VM3.

decides which of the cache ways is replaced (evicted) when new data has to be loaded into the cache.

On most Intel CPUs, the last-level cache is inclusive, *i.e.*, data present in L1 or L2 cache must also be present in the last-level cache. Furthermore, the last-level cache is shared among all cores and divided into so-called cache slices. The hash function that maps physical addresses to slices is not publicly documented but has been reverse-engineered [39, 57, 88].

Cache Eviction. To mount a Rowhammer attack, memory accesses need to be directly served by the main memory. Thus, an attacker needs to make sure that the data is not stored in the cache. An attacker can use the unprivileged `clflush` instruction [87] to invalidate the cache line or use uncached memory if available [84]. On devices where no uncached memory and no unprivileged cache flush instruction is available, an attacker can instead evict a cache line by accessing congruent memory addresses [25, 28, 52], *i.e.*, addresses that map to the same cache set and same cache slice. Merely accessing a large number of different but congruent addresses in an arbitrary order typically does not lead to a high eviction rate. Gruss et al. [28] observed that to evict the victim address, a so-called eviction set of attacker-chosen congruent addresses has to be accessed in a specific pattern. The eviction set does not contain the victim address, which is consequently evicted from the cache.

Intel CAT. In 2016, Intel introduced Cache Allocation Technology (CAT) [41] to address quality of service in multi-core server platforms [32, 40]. Intel CAT allows system software to partition the last-level cache to optimize workloads in shared environments as well as to isolate applications or virtual machines in the cloud. When a virtual machine in the cloud thrashes the cache and therefore decreases the performance of other co-located machines, the hypervisor can restrict this virtual machine to a subset of the cache to retain the performance of other tenants. More specifically, Intel CAT allows restricting the number of cache ways available to processes, virtual machines, and containers, as illustrated in Figure 2. However, Aga et al. [4] showed that Intel CAT allows improving eviction-based Rowhammer attacks as it reduces the number of accesses required for cache eviction and consequently reduces the time required to evict an address from the cache.

3 NETHAMMER ATTACK

All previously published Rowhammer attacks rely on some form of code execution on the targeted device, be it the execution of a native binary [4, 27, 73, 78], an application [84] or using a scripted language in the web browser, like JavaScript [14, 25, 28]. In this section, we present Nethammer, the first Rowhammer attack that does not rely on any attacker-controlled code on the victim machine.

3.1 Attack Overview

Nethammer sends a crafted stream of network packets to the target device to mount a one-location or single-sided Rowhammer attack by exploiting quality-of-service technologies deployed on the device. For each packet received on the target device, a set of addresses is accessed, either in the kernel driver or a user-space application processing the contents. By repeatedly sending packets, this set of addresses is hammered and, thus, bit flips may be induced. As frequently-used addresses are served from the cache for performance, the cache must be bypassed such that the access goes directly into the DRAM to cause the row conflicts required for hammering. This can be achieved in different ways if the code that is executed (in kernel space or user space) when receiving a packet, (1) *flushes* (and later on reloads) an address; (2) *uses uncached* memory; (3) *evicts* (and later on reloads) an address.

All three scenarios are plausible. *Uncached* memory is often used on ARM-based devices for interaction with the hardware, e.g., access buffers used by the network controller. Intel x86 processors have the *clflush* instruction for the same purpose. We verified that an attack is practical in both scenarios, as we describe in Section 6.2.

As caches are large, and cache replacement policies try to keep frequently-used data in the cache, it is not trivial to mount an eviction-based attack without executing attacker-controlled code on the device. However, to address quality of service in multi-core server platforms, Intel introduced CAT (cf. Section 2.3), allowing to control the amount of cache available to applications or virtual machines dynamically as illustrated in Figure 2. If a virtual machine is thrashing the cache, the hypervisor limits the number of cache ways available to this virtual machine to meet performance guarantees given to other tenants on the same physical machine. Thus, if an attacker excessively uses the cache, its virtual machine is restricted to a low number of ways, possibly only one, leading to a fast self-eviction of addresses.

3.2 Attack Setup

In our attack setup, the attacker has a fast network connection to the victim machine, e.g., a gigabit connection. We assume that the victim machine has DDR2, DDR3, or DDR4 memory that is susceptible to one-location (or single-sided) hammering.

Personal Computers. For our attack on personal computers, tablets, smartphones, or devices with similar hardware configuration, we make no further assumptions.

Cloud Systems. For our attack on cloud systems, we assume that the victim is running a virtual machine on a cloud server providing an interface or API accessible over the network. Furthermore, to prevent denial-of-service situations due to cache thrashing, we

assume that the hypervisor on the cloud server uses Intel CAT to constrain the virtual machine of the victim to a subset of the cache.

Note that there are overlaps between the two attack setups. A personal computer can be susceptible to the attack we describe for the cloud scenario. Even more likely a cloud system can be susceptible to the attack we describe for personal computers.

3.3 Inducing Bit Flips over Network

To induce bit flips remotely, one requirement is to send as many packets as possible over the network in a short time frame. As defined in Section 3.2, we assume that either uncached memory or *clflush* is used when receiving a network packet or alternatively, that Intel CAT is active on the victim machine. Thus, every single packet processed by the network stack actively evicts and reloads data from the cache. By sending many packets, the corresponding addresses are hammered efficiently.

As an example, UDP packets without content can be used, allowing an overall packet size of 64 B, which is the minimum packet size for an Ethernet packet. This allows to send up to 1 024 000 packets per second over a 500 Mbit/s connection.

4 FROM REGULAR MEMORY ACCESSES TO ROWHAMMER

Naturally, several challenges need to be solved to induce Rowhammer bit flips over the network. Fundamentally, we need to investigate memory-controller page policies to determine whether regular memory accesses that occur while handling network packets could at least, in theory, induce bit flips. Note that these investigations are oblivious to the specific technique to access the DRAM row (i.e., eviction, flushing, uncached memory). Hence, in this section, we do not discuss *clflush*, uncached memory, or eviction strategies with [4] or without Intel CAT [28, 52]. We defer comparisons of Nethammer with these techniques to Section 6. In this section, we focus on the underlying behavior of the memory controller and what this means for possible attacks.

Gruss et al. [27] found that the memory-controller page policy has a significant influence on the way the Rowhammer bug can be triggered. In particular, they found that one-location hammering works and deduced from this that the memory-controller page policy must be similar to a closed-page policy. Most previous work on Rowhammer assumed an open-row policy [4, 9, 11, 50, 62, 69, 73, 78, 84, 86]. In Section 4.1, we propose a method to determine the memory-controller page policy on real-world systems automatically. We show that one-location hammering does not necessarily need a closed-page policy, but instead, adaptive policies may allow one-location hammering.

Based on these insights, we demonstrate the first one-location Rowhammer attack on an ARM device in Section 4.2, and draw the connection to the attack presented by Aga et al. [4]. Finally, we investigate whether Rowhammer via network packets is theoretically possible. Network packets do not arrive with the same speed as the memory accesses in an optimized tight loop.

4.1 Automated Classification of Memory-Controller Page Policies

Gruss et al. [27] stated that a requirement for one-location hammering is a policy similar to a closed-page policy. To get a more in-depth understanding of the memory-controller page policy used on a specific system, we present an automated method to detect the used policy. This is a significant step forward for Rowhammer attacks, as it allows to deduce whether specific attack variants may or may not work without an empiric evaluation. Pessl et al. [68] reverse-engineered the undocumented mapping functions of physical memory addresses to DRAM channels, ranks and banks. These mapping functions allow selecting addresses located in the same bank but in a different row. If we access these addresses consecutively, we will cause a row conflict in the corresponding bank. This row conflict induces latency for the second access because the currently active row must be closed (written back), the bank must be pre-charged, and only then the new row can be fetched with an activate command. This side-channel information can not only be used to build a covert communication channel [68], but as we show, it can also be used to detect the page policy used by the memory controller.

Automated Classification of the Page-policy. We assume knowledge of processor and DRAM timings. For the DRAM this means in particular, the tRCD latency (the time to select a column address), and the tRP latency (the time between pre-charge and row activation). These three timings influence the observed latency as follows:

- (1) we consider the case **page open / row hit** as the base line;
- (2) in the case **page empty / bank pre-charged**, we observe an additional latency of tRP over a row hit;
- (3) in the case **page miss / row conflict**, we observe an additional latency of (tRP + tRCD) over a row hit.

To compute the actual number of cycles we can expect, we have to divide the DRAM latency value by the DRAM clock rate. In case of DDR4, we have to additionally divide the latency value by factor two, as DDR4 is double-clocked. This yields the latency in nanoseconds. By dividing the nanoseconds by the processor clock speed, we obtain the latency in CPU cycles. Still, as we cannot obtain absolutely clean measurements due to out-of-order execution, prefetching, and other mechanisms that aim to hide the DRAM latency, the actually observed latency will deviate slightly.

As in our test we cannot simply measure the three different cases, we define an experiment that allows to distinguish the different policies. In the experiment we use for our automated classification, we select two addresses A and B that map to the same bank but different rows. Using the `clflush` instruction, we make sure that A and B are not cached, in order to load those addresses directly from main memory. We base our method on two observations for open-page policies:

Single By loading address A an increasing number of times ($n = 1..10\,000$) before measuring the time it takes to load the same address on a subsequent access, we can measure the access time of an address in DRAM if the corresponding row is already active. For an open-page policy the access time should be the same for any n . **Conflict** By accessing address A and subsequently measuring the access time to address B , we can measure the access time of an address in DRAM in the occurrence of a row conflict.

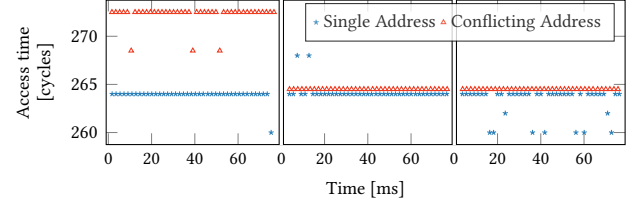


Figure 3: Measured access times over a period of time for a single address (blue) and an address causing a row conflict (red) for different page policies on the Intel Xeon D-1541: open policy (left), closed policy (middle), adaptive policy (right).

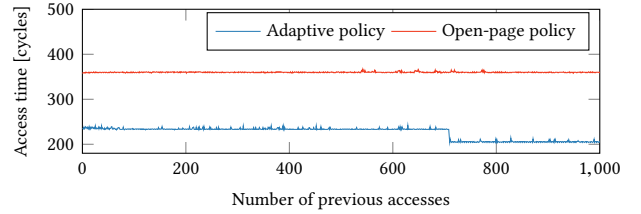


Figure 4: Open-page policy and adaptive page policy can be distinguished by testing increasing numbers of accesses to the same row. The open-page policy (Intel Core i7-4790) always has the same timing for subsequent accesses, since the row always remains open. The adaptive page policy (Intel Xeon E5-1630v4) only leaves the row open for a longer time after a larger number of accesses.

Our classification now works by running the following checks:

- (1) If there is no timing difference between the two cases described above (**Single** with a large n and **Conflict**), the system uses a closed-page policy. The closed-page policy immediately closes the row after every read or write request. Thus, there is no timing difference between these two cases. The timing observed corresponds to the row-pre-charged state.
- (2) Otherwise, if the timing for the **Single** case is the same, regardless of the value of n , but differs from the timing for **Conflict**, the system uses an open-page policy. The timing difference corresponds to the row hits and row conflicts. Following the definition of the open-page policy, the timing for row hits is always the same.
- (3) Otherwise, the timing for the **Single** case will have a jump at some n after which the page policy is adapted to cope better with our workload. Consequently, the timing differences we observe correspond to row hit and row-pre-charged states.

Figure 3 shows the memory access time measured on an Intel Xeon D-1541 with different page policies. The plot shows that closed-page policy can be distinguished from the other two using our method. We also verified our results by reading out the `CLOSE_PG` bit in the `mcmtr` configuration register of the integrated memory controller [42].

We validated that we can distinguish open-page policy and adaptive page policy by running our experiments on two systems with the corresponding page policies. Figure 4 shows the results of these

experiments. The difference between open-page policy and adaptive policy is clearly visible.

Our experiments show that adaptive page policies often behave like closed-page policies. This indicates the possibility of one-locating hammering on systems using an adaptive page policy.

4.2 One-location Hammering on ARM

To make Nethammer a more generic attack, it is essential to demonstrate it not only on Intel CPUs but also on ARM CPUs. This is particularly interesting as ARM CPUs dominate the mobile market, and ARM-based devices are predominant also in IoT applications. Gruss et al. [27] only demonstrated one-location hammering on Intel CPUs. However, as one-location hammering is the most plausible hammering variant for Nethammer, we need to investigate whether it is possible to trigger one-location hammering bit flips on ARM.

In our experiments, we used a LG Nexus 4 E960 mobile phone equipped with a Qualcomm Snapdragon 600 (APQ8064) [71] SoC and 2GB of LPDDR2 RAM, susceptible to bit flips using double-sided hammering. The page policy used by the memory controller is selected via the `DDR_CMD_EXEC_OPT_0` register: if the bit is set to 1, which is the recommended value [72], a closed-page policy is used. If the bit is set to 0, an open-page policy is used. Hence, we can expect the memory controller to preemptively close rows, enabling one-location hammering.

So far, bit flips on ARM-based devices have only been demonstrated in the combination of double-sided hammering, and uncached memory [84] or access via the GPU [25]. Even in the presence of a flush instruction [7] or optimal cache eviction strategies [52], the access frequency to the two neighboring rows is too low to induce bit flips. Furthermore, devices with the ARMv8 instruction set that allows exposing a flush instruction to unprivileged programs are usually equipped with LPDDR4 memory.

In our experiment, we allocated uncached memory using the Android ION memory allocator [90]. We hammered a single random address within the uncached memory region at a high frequency and then checked the memory for occurred bit flips. We were able to observe 4 bit flips while hammering for 10 hours. Thus, we can conclude that there are ARM-based devices that are vulnerable to one-location hammering.

4.3 Minimal Access Frequency for Rowhammer Attacks

A show stopper for Nethammer is if the frequency of memory accesses caused by processing network packets is not high enough to induce bit flips on one of our test systems successfully. As the system performs many memory accesses when handling a network packet, the attacker, in fact, cannot tell whether only one location in a bank is hammered (*i.e.*, one-location hammering) or multiple locations (*i.e.*, single-sided hammering). In particular, following the pigeon-hole principle, in our test setups with 32 bank (single DIMM) or 64 bank (dual DIMM) setups, we know that, if we access at least $n + 1$ different addresses, *i.e.*, 33 or 65 respectively, at least two addresses must be served from the same bank. Hence, we can assume that there is a good probability that the attacker

actually does single-sided hammering. Moreover, some addresses are accessed multiple times.

Previous work has investigated the minimal number of accesses that are necessary within a 64 ms refresh interval to still obtain bit flips. Kim et al. [50] reported bit flips starting at 139 000 row activations per refresh interval, which can be, depending on the page policy, identical to the number of memory accesses. Gruss et al. [28] reported bit flips starting at 43 000 and Aweke et al. [9] at 110 000 memory accesses per refresh interval.

In our experiments, we send 500 Mbit/s (and more) over the network interface. With a minimum size of 64 B for Ethernet packets, we can send 1 024 000 packets per second over a 500 Mbit/s connection. As described in Section 6.2, we found functions which are called multiple times, *e.g.*, 6 times in the case of once function. Hence, on a 500 Mbit/s connection, the attack can induce 6 144 000 accesses per second. Divided by the default refresh interval of 64 ms, we are at 393 216 accesses per refresh interval. This is clearly above the previously reported required number of memory accesses [9, 28, 50]. Hence, we conclude that in theory, if the system is susceptible to Rowhammer attacks, network packets can induce bit flips. In the following section, we will describe how an attacker can exploit such bit flips.

5 EXPLOITING BIT FLIPS OVER A NETWORK

In this section, we discuss Nethammer attack scenarios to exploit bit flips over the network in detail. We discuss the possible locations of bit flips in Section 5.1. We describe different Nethammer attacks in Section 5.2.

5.1 Bit Flip Location and Effect

As the Nethammer attack does not control where in physical memory a bit flip is induced and, thus, what is stored at that location, the bit flip can lead to different consequences. On a high level, we can divide bit flips into two groups, based on the location of the flip. We distinguish between bit flips in user memory, *i.e.*, memory pages that are mapped as `user_accessible` in at least one process, and bit flips in kernel memory, *i.e.*, memory pages that are never mapped as `user_accessible`. We can also distinguish the bit flips based on their high-level effect, again forming two groups. The first group consists of bit flips that lead to a denial-of-service situation. The second group consists of bit flips that do not lead to a denial-of-service situation. If a denial-of-service situation is temporary, a system reboot may be necessary. A denial-of-service situation can be persistent if the bit flip is written back to a permanent storage location. Then it may be necessary to reinstall the system software or parts of it from scratch, clearly taking more time than just a reboot. Denial-of-service attacks have a direct financial impact on companies due to unplanned downtimes and maintenance times. Moreover, studies show that their announcement can also have a negative impact on the stock prices [1]. Consequently, Nethammer poses a severe threat to servers vulnerable to the attack.

5.2 Bit Flip Targets

Nethammer may induce a bit flip in *kernel memory*. Depending on the modified location, parts of the operating system can behave

unexpectedly, or the entire system may even halt. Bit flips in *user memory* may have similar consequences.

5.2.1 File System Data Structures. File system data structures, e.g., inodes, are not directly part of the kernel code or data but are also in the kernel memory. An inode is a data structure defining a file or a directory of a file system. Each inode contains metadata such as the size of the file, owner and permission data as well as the disk block location of its data. If a bit flips in the inode structure, it corrupts the file system and, thus, causes persistent loss of data. This may again crash the entire system.

5.2.2 SGX Enclave Page Cache. If the victim machine supports Intel SGX [19], an x86 instruction-set extension that allows the execution of programs in so-called *secure enclaves* to run with integrity and confidentiality guarantees in untrusted environments, a bit flip easily causes a denial of service. Enclave memory is stored in a physically contiguous block of memory that is encrypted using a Memory Encryption Engine [30]. Jang et al. [44] and Gruss et al. [27] demonstrated that if a bit flip in enclave memory is induced, the Memory Encryption Engine locks the memory controller, preventing any future memory operations and thus, halting the entire system. While such a bit flip is not persistent itself, the unsafe halting of the entire system can leave permanent damage leading to a persistent denial-of-service.

5.2.3 Application Memory in General. If a bit flip occurs in memory of a user-space application, e.g., code or data, a possible outcome is the crash of the program. Such a flip may render the affected service unavailable.

Another outcome of a bit flip in the data of a user-space application, e.g., in the database of a service, is that the service delivers modified, possibly invalid, content. Depending on the service, its users cannot distinguish if the data is correct or has been altered.

Altering DNS Entries to redirect to Malicious Services. To resolve domain names to the corresponding IP address, a DNS request [60] is sent to a DNS server. DNS servers are organized in a tree-like structure, building a distributed system to store DNS records. A record consists of a type, a name, a class code, a time-to-live for caching, and the value. For instance, the A record holds a 32-bit IPv4 address for a specific domain. However, DNS allows defining aliases to map one domain name to another. This is used to define message transfer agents for a domain or to redirect domains.

In this attack, the attacker leverages Nethammer to induce a bit flip in a character of a DNS entry to make it point to a different domain. For instance, `domain.com` changes to `dnmain.com` if the least-significant bit of the “o” character is flipped from ‘1’ to ‘0’. Such an attack is also referred to as bitsquatting [20]. Such bit flips in domains have been successfully exploited before using Rowhammer attacks [73]. DNS zone transfers (AXFR queries) allow replicating DNS databases across different servers. Using zone transfers, an attacker can retrieve entries of an entire zone at once. The attacker queries the DNS server for its entries, mounts the attack and then verifies whether a bit flip at an exploitable position has occurred by monitoring changes in the queried entries. If so, the attacker can register the changed domain and host a malicious service on the domain, e.g., a fake website to steal login credentials or a mail server intercepting email traffic. Users querying the DNS server

for said entry connect to the server controlled by the attacker and are thus exposed to data theft. A flip might also change an MX entry, pointing it to a different domain. The attacker can then again register the domain and intercept connections that were intended to go to the original mail server.

Rebuilding Trust in Revoked Certificates. An attacker can also target OCSP servers. The Online Certificate Status Protocol (OCSP) is a protocol to retrieve the revocation status of a certificate [77]. In contrast to a certificate revocation list, where all revoked certificates are enumerated, the OCSP protocol enables to query the status of a single certificate. This protocol shifts the workload from the user to the OCSP server, so that users, or more specifically browsers, do not have to store huge revocation lists. Instead, the OCSP server manages a list of revoked certificate fingerprints.

Digital certificates are used to generate digital signatures that present the authenticity of digital documents or messages. They are typically obtained from a trusted party, e.g., a certificate authority. The certificate allows verifying that a specific signature was indeed issued by the signer. However, if the corresponding private key of a certificate is exposed to the public, everyone can sign data in the name of the signer. Hence, a user can revoke a certificate to avoid any abuse. Liu et al. [53] evaluated 74 full IPv4 HTTPS scans and found that 8% of 38 514 130 unique SSL certificates served have been revoked.

To process a certificate validity request, the server queries its database for the requested certificate identifier. The result can either be that the certificate is revoked, not revoked (*i.e.*, valid), or that the state is unknown (*i.e.*, it is not in the database). If a client tries to establish a secure connection to a server or check the validity of a signed document, it queries the OCSP server provided by the certificate. If the certificate has been revoked, the client aborts the connection or marks the signature as invalid.

In this attack, the attacker flips a bit in the memory of an OCSP server of a certificate authority where private keys of certificates have become public, and the certificates have thus been revoked. The attacker can either flip the status or the identifier of the certificate. As the status of the certificate is stored as an ASCII character in the OpenSSL OCSP responder [65], one bit flip is sufficient to flip the “R” (revoked) to “V” (valid). Assuming the memory is filled with revocation list entries, which are on average 100 B for this specific responder, an attacker has a chance of 0.125 % *per bit flip* to make a random certificate valid again. Thus, an attacker can again reuse that certificate (with the known private key) to sign documents or data and, thus, impersonate the original signer.

A weaker, but more likely attack scenario, is to flip a bit in the certificate identifier. Such a bit flip leads to the OCSP server not finding the certificate in its database anymore, thus, returning “unknown” as the state. Most browsers fall back to their own certificate revocation list in such a case [2, 56, 66]. However, only high-value revocations are kept in the browser’s list, making it very unlikely that the certificate is in the certificate revocation list of the browser [2]. Hence, an attacker can again reuse that certificate.

Other attacks. The attack scenarios described above are by far not exhaustive. With bit flips in applications, attackers have numerous possibilities to modify random data, yielding different, disastrous

consequences. However, the outlined attacks highlight the severity of remotely induced bit flips by Nethammer.

5.2.4 Cryptographic Material. Cryptographic material as part of the application memory is particularly interesting for attacks. In the past, it has been demonstrated that fault attacks on RSA public keys result in broken keys which are susceptible to key factorization [10, 16]. Therefore, also public key material has to be protected against faults. Muir [61] remarked that a bit flip in an RSA public key allows an attacker with a non-negligible probability to compute a private key corresponding to the modified key in a reasonable amount of time. Thus, an attacker can flip a bit of a public RSA key in memory using Nethammer, giving the attacker the same privileges and permissions as the owner of the original key. These permissions are only temporary, e.g., until the key is reloaded from the hard drive.

Distribution of Malicious Software on Version-Control Hosting Services. An attacker can compromise a hosting service to distribute malicious software. The number of organizations using hosting services for revision control to manage changes to their source code, documents or other information is increasing steadily. These services can either be subscription based, e.g., GitHub [36], or self-hosted, e.g., GitLab [37], and, thus, are deployed on many web servers to distribute their software.

To commit changes to a version-controlled repository, users authenticate with the service using public-key cryptography. Typically, users generate an SSH key pair [89], e.g., using RSA [75], upload the public key to the service, and store the private key securely on their local system. As the position of the bit flip cannot be controlled using Nethammer, an attacker can improve the probability to induce a bit flip in the modulus of a public key by loading as many keys as possible into the main memory of the server. Some APIs, e.g., the GitLab API [38], allow enumerating the users registered for the service as well as their public keys. By enumerating and, therefore, accessing all public keys of the service, the attacker loads the public keys into the DRAM.

In the first step of the attack, the attacker enumerates all keys of all users and stores them locally. In the second step, the attacker mounts Nethammer to induce bit flips on the targeted system. The more keys the attacker loaded into memory, the more likely it is that the bit flip corrupts the modulus of a public key of a user. For instance, with 80 % of the memory filled with 4096-bit keys, the chance to hit a bit of a modulus is 79.7 %. As the attacker does not know which key was affected by the bit flip, the attacker enumerates all keys again and compares them with the locally stored keys. If a modified key has been found, the attacker computes a new corresponding private key [61, 73]. The attacker uses this new key to authenticate with the service, impersonating the user. Consequently, the attacker can make changes to the software repository as that user and, thus, introduce bugs that can later be exploited if the software is distributed. The original public key will be restored after a while when the key is evicted from the page cache and has to be reloaded from the hard drive. As the correct key is restored, the attack leaves no traces. Furthermore, it also breaks the non-repudiation guarantee provided by the public-key authentication, making the victim whose public key was attacked the prime suspect in possible investigations.

6 EVALUATION

In this section, we evaluate Nethammer and its performance. We show that the number of bit flips induced by Nethammer depends on how the cache is bypassed and the memory-controller’s page policy. We evaluate which kernel functions are executed when handling a UDP network packet. We describe the bit flips we obtained when running Nethammer in different attack scenarios. Finally, we show that TRR does not protect against Nethammer or Rowhammer in general.

6.1 Environment

In our evaluation, we used the test systems listed in Table 1. We used the first system for our experiments with a non-default network driver implementation that uses `clflush` in the process of handling a network packet, and the second and third system for our experiments with Intel CAT. To mount Nethammer, we used a Gigabit switch to connect two other machines with the victim machine. The two other machines were used to flood the victim machine with network packets triggering the Rowhammer bug. We used the fourth system for our experiments on an ARM-based device that uses uncached memory in the process of handling a network packet.

6.2 Evaluation of the Different Cache Bypasses for Nethammer

In Section 4, we investigated the requirements to trigger the Rowhammer bug over the network. In this section, we evaluate Nethammer for the three cache-bypass techniques (see Section 3.1): a kernel driver that flushes (and reloads) an address whenever a packet is received, Intel Xeon CPUs with Intel CAT for fast cache eviction, and uncached memory on an ARM-based mobile device.

Driver with `clflush`. To verify that Nethammer can induce bit flips, we used a non-default network driver implementation that uses `clflush` in the process of handling a network packet on an Intel i7-6700K CPU. We sent UDP packets with up to 500 Mbit/s and scanned memory regions where we expected bit flips. We observed a bit flip every 350 ms showing that hammering over the network is feasible if at least two memory accesses are served from main memory, due to flushing an address while handling a network packet. Thus, in this scenario, up to 10 000 bit flips per hour can be induced.

Eviction with Intel CAT. The operating system will handle every network packet received by the network card. The operating system parses the packets depending on their type, validates their checksum and copies and delivers every packet to each registered socket queue. Thus, for each received packet quite some code is executed before the packet finally arrives at the application destined to handle its content.

We tested Nethammer on Intel Xeon CPUs with Intel CAT. The number of cache ways has been limited to a single one for code handling the processing of UDP packets, resulting in fast cache eviction. If a function is called multiple times for one packet, the same addresses are accessed and loaded from DRAM with a high probability, thus, hammering this location. To estimate how many different functions are called and how often they are called, we

Table 1: List of test systems that were used for the experiments.

Device	CPU	DRAM	Network card	Operating system
Desktop	Intel i7-6700K @ 4 GHz	8 GB DDR4 @ 2133 MHz	Intel 10G X550T	Ubuntu 16.04
Server	Intel Xeon E5-1630v4 @ 3.7 GHz	8 GB DDR4 @ 2133 MHz	Intel i210/i218-LM Gigabit	Xubuntu 17.10
Server	Intel Xeon D-1541 @ 2.1 GHz	8 GB DDR4 @ 2133 MHz	Intel i350-AM2 Gigabit	Ubuntu 16.04
LG Nexus 4	Qualcomm APQ8064 @ 1.5 GHz	2 GB LPDDR2 @ 533 MHz	USB Adapter	Android 5.1.1

use the *perf* framework [22] to count the number of function calls related to UDP packet handling. Appendix A shows the results of a system handling UDP packets. Out of 27 different functions we identified, most were called only once for each received packet. The function `__udp4_lib_lookup` is called twice. In a more extensive profiling scan, we found that `nf_hook_slow` is called 6 times while handling UDP packets on some kernels.

With this knowledge, we analyzed how many bit flips can be induced from this code execution. We observed 45 bit flips per hour on the Intel Xeon E5-1630v4. As TRR is active on this system (see Section 6.5), fewer bit flips occur in comparison to systems without TRR. In Section 6.3, we evaluate the number of bit flips on the Intel Xeon D-1541 depending on the configured page policy.

Uncached Memory. In Section 4.2, we demonstrated that ARM-based devices are vulnerable to one-location hammering in general. To investigate whether bit flips can also be induced over the network, we connect the LG Nexus 4 using an OTG USB ethernet adapter to a local network. Using a different machine, we send as many network packets as possible to the mobile phone. An application on the phone allocates memory and repeatedly checks the allocated memory for occurred bit flips. However, we were not able to observe any bit flips on the device within 12 hours of hammering. As the device does not deploy a technology like Intel CAT (Section 2.3), the cache is not limited for certain applications and, thus, the eviction of code or data used by handling memory packets has a low probability. As network drivers often use DMA memory and, thus, uncached memory, bit flips induced by the network are more likely if the network driver itself uses uncached memory. While we identified a remarkable number of around 5500 uncacheable pages used by the system, we were not able to induce any bit flips over the network. However, we found that the USB ethernet adapter only allowed for a network capacity of less than 16 Mbit/s, which is clearly too slow for a Nethammer attack. It is very likely that with a faster network connection, e.g., more than 200 Mbit/s, it is possible to induce bit flips. Nevertheless, we were successfully able to induce bit flips using Nethammer on the Intel Xeon E5-1630v4 where one uncached address is accessed for every received UDP packet.

6.3 Influence of Memory-Controller Page Policies on Rowhammer

In order to evaluate the actual influence of the used memory-controller page policy on Nethammer, *i.e.*, how many bit flips can be induced depending on the policy used, we mounted the Nethammer in different settings. The experiment was conducted on our Intel Xeon D-1541 test system, as the BIOS of its motherboard allowed to

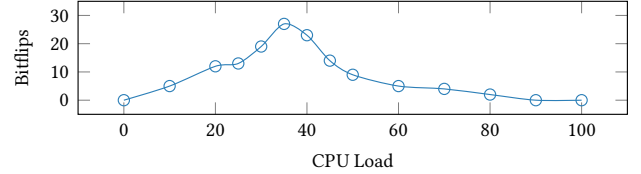


Figure 5: Number of bit flips depending on the CPU load with a closed-page policy after 15 minutes (Xeon D-1541).

chose between different page policies: *Auto*, *Closed*, *Open*, *Adaptive*. For each run, we configured the victim machine with one of the policies and Intel CAT, and, mounted a Nethammer attack for at least 4 hours. To detect bit flips, we ran a program on the victim machine that mapped a file into memory. The program then repeatedly scans the content of all allocated pages and reports bit flips if the content has changed.

We detected 11 bit flips in 4 hours with the *Closed* policy, with the first one after 90 minutes. We did not observe any bit flips with the *Open* policy within the first 4 hours. However, when running the experiment longer, we observed 46 bit flips within 10 hours. With the *Adaptive* policy, we observed 10 bit flips in 4 hours, with the first one within the second hour of the experiment. While this experiment was conducted without any additional load on the system, we see in Figure 5 that additional CPU utilization increases the number of bitflips drastically. Using the *Closed* policy, we observed 27 bitflips with a load of 35% within 15 minutes.

These results do not immediately align with the assumption that a policy that preemptively closes rows is required to induce bit flips using one-location hammering. However, depending on the addresses that are accessed and the constant eviction through Intel CAT, it is possible that two addresses map to the same bank but different rows and, thus, bit flips can be induced through single-sided hammering. In fact, the attacker cannot know whether the hammering was actually one-location hammering or single-sided hammering. However, as long as a bit flip occurs, the attacker does not care how many addresses mapped to the same bank. Finally, depending on the actual parameters used by a fixed-open-page policy, a row can still be closed early enough to induce bit flips.

6.4 Bit Flips induced by Nethammer

As described in Section 5.1, a bit flip can occur in user space or kernel space leading to different effects depending on the memory it corrupts. In this section, we present bit flips that we have observed in our experiments and the effects they have caused.

Kernel image corruption and kernel crashes. We observed Nethammer bit flips that caused the system not to boot anymore. It stopped responding after the bootloader stage. We inspected the kernel image and compared it to the original kernel image distributed by the operating system. As the kernel image differed blockwise at many locations, we assume that the Nethammer caused a bit flip in an inode of the file system. The inode of a program that wanted to write data did not point to the correct file any longer but to the kernel image and, thus, corrupted the kernel image.

Furthermore, we observed several bit flips immediately halting the entire system such that interaction with it was not possible any longer. By debugging the operating system over a serial connection, we detected bit flips in certain modules such as the keyboard or network driver. In these cases, the system was still running but did not respond to any user input or network packets anymore. We also observed bit flips that were likely in the SGX EPC region, causing an immediate permanent locking of the memory controller.

Bit flips in user space libraries and executables. We observed that bit flips crashed running processes and services or prevented the execution of others as the bit flip triggered a segmentation fault when functions of a library were executed. On one occasion, a bit flip occurred either in the SSH daemon or the stored passwords of the machine, preventing any user to login on the system. The system was restored to a stable state only by rebooting the machine and thus reloading the entire code from disk.

We also validated that an attacker can increase chances to flip a bit in a target page by increasing the memory usage of a user program. In fact, this was the most common scenario, overlapping with our general test setup to detect bit flips for our evaluation. Unsurprisingly, these bit flips equally occur when filling the memory with actual contents that the attacker targets.

6.5 Target Row Refresh (TRR)

Previous assumptions on the Rowhammer bug lead to the conclusion that only bit flips in the victim row adjacent to the hammering rows would occur. While the probability for bit flips to occur in directly adjacent rows is much higher, Kim et al. [50] already showed rows further away (even a distance of 8 rows and more) are affected as well. Still, the hardware vendors opted for implementing countermeasures focusing on the directly adjacent rows.

With the Low Power Double Data Rate 4 (LPDDR4) standard, the JEDEC Solid State Technology Association [46] defines a reliability feature called Target Row Refresh (TRR). The idea of TRR is to refresh adjacent rows if the targeted row is accessed at a high frequency. More specifically, TRR works with a maximum number of activations allowed during one refresh cycle, the maximum active count. Thus, if a double-sided Rowhammer attack (Section 2.2) is mounted, and two hammered rows are accessed more than the defined maximum active count, the adjacent rows (in particular the victim row of the attack) will be refreshed. As the potential victim rows are refreshed, in theory, no bit flip will occur, and the attack is mitigated. However, in practice, bit flips can be further away from the hammered rows and thus TRR may be ineffective.

With the Ivy Bridge processor family, Intel introduced Pseudo Target Row Refresh (pTRR) for Intel Xeon CPUs to mitigate the Rowhammer bug [55]. On these systems pTRR-compliant DIMMs

must be used; otherwise, the system will default into double refresh mode, where the time interval in which a row is refreshed is halved [55]. However, Kim et al. [50] showed that a reduced refresh period of 32 ms is not sufficient enough to impede bit flips in all cases. While pTRR is implemented in the memory controller [54], DRAM module specifications theoretically allow automatically running TRR in the background [58].

In our experiments, we were able to induce bit flips on a pTRR-supporting DDR4 module using double-sided hammering on an Intel i7-6700K. The bit flips occurred in directly adjacent rows and rows further away. We observed that when using the same DDR4 DRAM on the Intel Xeon E5-1630 v4 CPU, no bit flips occurred in the directly adjacent rows, but we observed no statistically significant difference in the number of bit flips for the rows further away. This indicates that TRR is active on the second machine but also that TRR does not prevent the occurrence of exploitable bit flips in practice. Thus, we conclude that the TRR hardware countermeasure is insufficient in mitigating Rowhammer attacks.

7 COUNTERMEASURES

Since Nethammer does not require any attack code in contrast to a regular Rowhammer attack, e.g., no attacker-controlled code on the system, most countermeasures do not prevent our attack.

Countermeasures based on static code analysis aim to detect attack code in binaries [43]. However, as our attack does not use any suspicious code and does not execute a program, these countermeasures do not detect the ongoing attack. Other countermeasures detect on-going attacks using hardware- and software-based performance counters [17, 18, 29, 31, 67, 91] and subsequently stop the corresponding programs. However, when hammering over the network, the large amount of memory accesses are executed by the kernel itself, and the kernel cannot just be terminated or stopped like a regular program. Hence, these countermeasures cannot cope with our attack. Modifying the system memory allocator to hinder the exploitability of bit flips [15, 28, 84] may generally work against Nethammer. However, the hammering is in practice done by the kernel, so the proposed isolation schemes are ineffective, and new schemes have to be proposed.

ANVIL [9] uses performance counters to detect and subsequently mitigate Rowhammer attacks. Since ANVIL, in its current form, does not detect one-location hammering [27], it also does not detect our attack. While we believe an adapted version of ANVIL could detect our attack, it would require evaluating whether the false positive and false negative rates allow for an application in practice. B-CATT [15] blacklists vulnerable locations, thus, effectively reducing the amount of usable memory, but fully eliminating the Rowhammer bug. B-CATT would work against Nethammer, but previous work has found that it is not practical as it would block too much memory [27, 50].

In general, we recommend reviewing any network stack and network services code. Uncached memory and `clflush` instructions should only be used with extreme care, and it may even be necessary to add artificial slow downs such that they cannot be exploited for Nethammer attacks anymore. If this is not possible for technical reasons, the threat model of the device should be revisited and reevaluated. Mitigating our eviction-based Nethammer attack

might be more straight-forward, as it requires a specific configuration for Intel CAT. Either avoiding the restriction to a low number of cache ways via Intel CAT on network-connected systems or installing ECC memory would likely be sufficient to make our attack very improbable to succeed. Hence, we also recommend using Intel CAT very carefully in network-connected systems.

8 DISCUSSION

Hardware requirements. To induce the Rowhammer bug, one needs to access memory in the main memory repeatedly and, thus, needs to circumvent the cache. Therefore, either native flush instructions [87], eviction [4, 28] or uncached memory [84] can be used to remove data from the cache. In particular, for eviction-based Nethammer, the system must use Intel CAT as described in Section 2.3 in a configuration that restricts the number of ways available to a virtual machine in a cloud scenario to guarantee performance to other co-located machines [40]. If none of these capabilities are available over the network, an attacker could not mount Nethammer in practice.

One limitation of our attack is that only DRAM susceptible to bit flips can be exploited using a Rowhammer attack and, thus, Nethammer. To reduce the risk of bit flips on servers, one would assume that cloud providers tend to deploy ECC RAM usually. However, many cloud providers offer to rent hardware without ECC RAM [23, 33, 34, 63, 64, 85], potentially allowing Nethammer attacks. DRAM with ECC can only be used in combination with Intel Xeon CPUs and can detect and correct 1-bit errors. Therefore it can deal with single bit flips. While non-correctable multi-bit flips can be exploitable [5, 6, 51], they often end up in a denial-of-service attack depending on the operating system’s response to the error.

Network traffic. Nethammer sends as many network packets to the victim machine as possible, aiming to induce bit flips. Depending on the actual attack scenario (see Section 5), additional traffic, e.g., by enumerating the public keys of the service, is generated. If the victim uses network monitoring software, the attack might be detected and stopped, due to the highly increased amount of traffic. In our experiments, we sent a stream of UDP packets with up to 500 Mbit/s to the target system. We were able to induce a bit flip every 350 ms and, thus, if the first random bit flip already hits the target or causes a denial-of-service, the attack could already be successful. However, as the rows are periodically refreshed, an attacker only needs an extraordinary high burst of memory accesses to a row between two refreshes, *i.e.*, within a period of 64 ms. Hence, an attacker could mount Nethammer for a few hundred milliseconds and then pause the attack for a longer time. These short network spikes may circumvent network monitoring software that might otherwise detect and prevent the on-going attack, e.g., by null routing the victim server.

Gigabit LTE on Mobile Devices.

While ethernet adapters in mobile phones are uncommon, many ARM-based embedded devices in IoT setups are equipped and connected with gigabit ethernet. However, we expect the maximum throughput of these network cards to be too low on many of these devices, e.g., the Raspberry Pi 3 Model B+ [24], and also WiFi chips

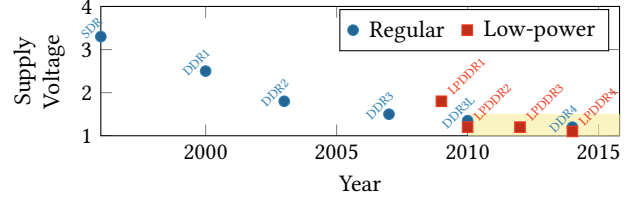


Figure 6: Minimum DRAM supply voltages for different DDR standards. The highlighted area marks the voltage and manufacturing years of DRAM modules where Rowhammer bit flips have been reported.

typically offer too little capacity. However, on more recent processors, e.g., the Qualcomm Snapdragon 845 chipset [70], and modems like the Qualcomm X20 Gigabit LTE modem, throughputs up to 1.2 Gbit/s are possible in practice. This would enable to send enough packets to hammer specific addresses to induce bit flips on the device and, thus, to successfully mount Nethammer.

Influence of DRAM Supply Voltage on Rowhammer Effect. Kim et al. [50] identified voltage fluctuations as the root cause of DRAM disturbance errors, e.g., the Rowhammer bug. However, no study so far has investigated the direct effect of the DRAM supply voltage on Rowhammer bit flips. In fact, we can already observe a direct correlation between a low DRAM supply voltage by reviewing related work. Figure 6 shows how the DRAM voltage has been reduced over the past years. Previous work observed that the vulnerability of DRAM modules is related to the manufacturing date, *i.e.*, no bitflips before 2010 [50, 78]. However, as shown in Figure 6 there are at least two possible correlations with the Rowhammer bug, the manufacturing date, and the supply voltage.

Indeed, Rowhammer has only been reported on DRAM modules with a voltage below 1.5 volts [50, 78], *i.e.*, DDR3 [50, 78], DDR4 [68], LPDDR2 and LPDDR3 [84], and LPDDR4 [83].

We investigated the influence of the DRAM voltage on the occurrence of bit flips on two systems. We tested voltage increases in 0.01 V steps. On three tested systems (2× DDR4, 1× DDR3), we observed no significant change in the number of bit flips, *i.e.*, the number of bit flips stayed in the same order of magnitude, even when increasing the voltage by 0.2 V. Future work should investigate whether other voltage-related parameters could lead to a straightforward elimination of the Rowhammer bug.

9 CONCLUSION

In this paper, we presented Nethammer, the first truly remote Rowhammer attack, without a single attacker-controlled line of code on the targeted system. We demonstrate attacks on systems that use uncached memory or flush instructions while handling network requests, and systems that don’t use either but are protected by Intel CAT. In all cases, we were able to induce multiple bit flips per hour on real-world systems, leading to temporary or persistent damage on the system. We showed that depending on the location, the bit flip compromises either the security and integrity of the system and the data of its users. In some cases, the system was rendered unbootable after the attack.

We presented a method to automatically identify the page policy used by the memory controller. Consequently, we found that adaptive page policies are also vulnerable to one-location hammering. While we were able to mount the first one-location hammering attack on an ARM device, the network capacity on this device was too low for Nethammer.

Transforming formerly local attacks into remote attacks is always a landslide in security. Assumptions that were true for the local scenario are largely invalid in a remote scenario. In particular, all defenses and mitigation strategies were designed against local Rowhammer attacks, *i.e.*, remote Rowhammer attacks were out of scope. Hence, Nethammer requires the re-evaluation of the security of millions of devices where the attacker is not able to execute attacker-controlled code. Finally, our work demonstrates that we need to develop countermeasures with the root cause of both local and remote Rowhammer attacks in mind.

ACKNOWLEDGEMENT

We thank Mattis Turin-Zelenko and Paul Höfler for help with some experiments. We thank Stefan Mangard, Anders Fogh, Thomas Dullien, and Jann Horn for fruitful discussions.

This work has been supported by the Austrian Research Promotion Agency (FFG) via the K-project DeSSnet, which is funded in the context of COMET – Competence Centers for Excellent Technologies by BMVIT, BMWFW, Styria and Carinthia. This project has received funding from the European Research Council (ERC) under the European Union’s Horizon 2020 research and innovation programme (grant agreement No 681402).

REFERENCES

- [1] Abhishta, Reinoud Joosten, and Lambert J.M. Nieuwenhuis. 2017. Comparing Alternatives to Measure the Impact of DDoS Attack Announcements on Target Stock Prices. (2017).
- [2] Adam Langley. 2014. Revocation still doesn’t work. (2014). <https://www.imperialviolet.org/2014/04/29/revocationagain.html>
- [3] Advanced Micro Devices. 2013. BIOS and Kernel Developer’s Guide (BKDG) for AMD Family 15h Models 00h-0Fh Processors. (2013). http://support.amd.com/TechDocs/42301_15h_Mod_00h-0Fh_BKDG.pdf
- [4] Misiker Tadesse Aga, Zelalem Birhanu Aweke, and Todd Austin. 2017. When good protections go bad: Exploiting anti-DoS measures to accelerate Rowhammer attacks. In *International Symposium on Hardware Oriented Security and Trust*.
- [5] Barbara Aichinger. 2015. DDR memory errors caused by Row Hammer. In *HPEC*.
- [6] Barbara Aichinger. 2015. Row Hammer Failures in DDR Memory. In *memcon*.
- [7] ARM Limited. 2013. *ARM Architecture Reference Manual ARMv8*. ARM Limited.
- [8] Manu Awasthi, David W. Nellans, Rajeev Balasubramanian, and Al Davis. 2011. Prediction Based DRAM Row-Buffer Management in the Many-Core Era. In *PACT’11*.
- [9] Zelalem Birhanu Aweke, Salessawi Ferede Yitbarek, Rui Qiao, Reetuparna Das, Matthew Hicks, Yossi Oren, and Todd Austin. 2016. ANVIL: Software-based protection against next-generation Rowhammer attacks. *ACM SIGPLAN Notices* 51, 4 (2016), 743–755.
- [10] Alexandre Berzati, Cécile Canovas, and Louis Goubin. 2008. Perturbating RSA Public Keys: An Improved Attack. In *Cryptographic Hardware and Embedded Systems - CHES 2008*.
- [11] Sarani Bhattacharya and Debdeep Mukhopadhyay. 2016. Curious Case of Rowhammer: Flipping Secret Exponent Bits Using Timing Analysis. In *Conference on Cryptographic Hardware and Embedded Systems (CHES)*.
- [12] Eli Biham. 1997. A fast new DES implementation in software. In *International Workshop on Fast Software Encryption*. 260–272.
- [13] Dan Boneh, Richard A. DeMillo, and Richard J. Lipton. 1997. On the Importance of Checking Cryptographic Protocols for Faults. In *Advances in Cryptology - EUROCRYPT ’97*.
- [14] Erik Bosman, Kaveh Razavi, Herbert Bos, and Cristiano Giuffrida. 2016. Dedup Est Machina: Memory Deduplication as an Advanced Exploitation Vector. In *S&P*.
- [15] Ferdinand Brasser, Lucas Davi, David Gens, Christopher Liebchen, and Ahmad-Reza Sadeghi. 2017. CAN’t Touch This: Software-only Mitigation against Rowhammer Attacks targeting Kernel Memory. In *USENIX Security Symposium*.
- [16] Eric Brier, Benoît Chevallier-Mames, Mathieu Ciet, and Christophe Clavier. 2006. Why One Should Also Secure RSA Public Key Elements. In *Cryptographic Hardware and Embedded Systems - CHES 2006*.
- [17] Marco Chiappetta, Erkay Savas, and Cemal Yilmaz. 2015. Real time detection of cache-based side-channel attacks using Hardware Performance Counters. Cryptology ePrint Archive, Report 2015/1034. (2015).
- [18] Jonathan Corbet. 2016. Defending against Rowhammer in the kernel. (Oct. 2016). <https://lwn.net/Articles/704920/>
- [19] Victor Costan and Srinivas Devadas. 2016. Intel SGX explained. (2016).
- [20] Artem Dinaburg. 2011. Bitsquatting: DNS Hijacking without Exploitation. (2011). http://media.blackhat.com/bh-us-11/Dinaburg/BH_US_11_Dinaburg_Bitsquatting_WP.pdf
- [21] James M. Dodd. 2003. Adaptive page management. (2003). <https://encrypted.google.com/patents/US7076617B2> US Patent Grant 2006-07-11.
- [22] Jake Edge. 2009. Perfcounters added to the mainline. (Jul 2009). <http://lwn.net/Articles/339361/>
- [23] fasthosts. 2018. High performance dedicated servers. (May 2018). <https://www.fasthosts.co.uk/dedicated-servers>
- [24] Raspberry Pi Foundation. 2018. Raspberry Pi 3 Model B+. (March 2018). <https://www.raspberrypi.org/products/raspberry-pi-3-model-b-plus>
- [25] Pietro Frigo, Cristiano Giuffrida, Herbert Bos, and Kaveh Razavi. 2018. Grand Pwning Unit: Accelerating Microarchitectural Attacks with the GPU. In *IEEE S&P*.
- [26] Mohsen Ghasempour, Mikel Lujan, and Jim Garside. 2015. ARMOR: A Run-time Memory Hot-Row Detector. (2015). <http://apt.cs.manchester.ac.uk/projects/ARMOR/RowHammer>
- [27] Daniel Gruss, Moritz Lipp, Michael Schwarz, Daniel Genkin, Jonas Juffinger, Sioli O’Connell, Wolfgang Schoecl, and Yuval Yarom. 2018. Another Flip in the Wall of Rowhammer Defenses. In *S&P’18*.
- [28] Daniel Gruss, Clémentine Maurice, and Stefan Mangard. 2016. Rowhammer.js: A Remote Software-Induced Fault Attack in JavaScript. In *DIMVA’16*.
- [29] Daniel Gruss, Clémentine Maurice, Klaus Wagner, and Stefan Mangard. 2016. Flush+Flush: A Fast and Stealthy Cache Attack. In *DIMVA*.
- [30] Shay Gueron. 2016. A Memory Encryption Engine Suitable for General Purpose Processors. Cryptology ePrint Archive, Report 2016/204. (2016).
- [31] Nishad Herath and Anders Fogh. 2015. These are Not Your Grand Daddys CPU Performance Counters – CPU Hardware Performance Counters for Security. In *Black Hat Briefings*. <https://www.blackhat.com/docs/us-15/materials/us-15-Herath-These-Are-Not-Your-Grand-Daddys-CPU-Performance-Counters-CPU-Hardware-Performance-Counters-For-Security.pdf>
- [32] Andrew Herdrich, Edwin Verplanke, Priya Autee, Ramesh Illikkal, Chris Gianos, Ronak Singhal, and Ravi Iyer. 2016. Cache QoS: From concept to reality in the Intel Xeon processor E5-2600 v3 product family. In *IEEE HPCA’16*.
- [33] Hetzner. 2018. Dedicated Root Server Hosting. (May 2018). <https://www.hetzner.com/dedicated-rootserver/>
- [34] DefineQuality Hosting. 2018. Highend Dedicated Rootserver. (May 2018). <https://definequality.net/dedicated.php>
- [35] Rei-Fu Huang, Hao-Yu Yang, Mango C.-T. Chao, and Shih-Chin Lin. 2012. Alternate hammering test for application-specific DRAMs and an industrial case study. In *Annual Design Automation Conference (DAC)*.
- [36] GitHub Inc. 2018. GitHub. (2018). <https://github.com>
- [37] GitLab Inc. 2018. GitLab. (2018). <https://gitlab.com>
- [38] GitLab Inc. 2018. GitLab Documentation: List SSH keys. (2018). <https://docs.gitlab.com/ee/api/users.html#list-ssh-keys>
- [39] Mehmet S Inci, Berk Gulmezoglu, Gorka Irazoqui, Thomas Eisenbarth, and Berk Sunar. 2016. Cache Attacks Enable Bulk Key Recovery on the Cloud. In *Cryptographic Hardware and Embedded Systems - CHES (LNCS)*, Vol. 9813. Springer, 368–388.
- [40] Intel. 2015. Improving Real-Time Performance by Utilizing Cache Allocation Technology: Enhancing Performance via Allocation of the Processor’s Cache. (April 2015). <https://www.intel.com/content/www/us/en/communications/cache-allocation-technology-white-paper.html>
- [41] Intel. 2016. Intel 64 and IA-32 Architectures Software Developer’s Manual, Volume 3 (3A, 3B & 3C): System Programming Guide. 253665 (2016).
- [42] Intel. 2016. Intel Xeon Processor E5 v4 Product Family: Datasheet Volume 2: Registers. 2 (2016).
- [43] Gorka Irazoqui, Thomas Eisenbarth, and Berk Sunar. 2017. MASCAT: Stopping Microarchitectural Attacks Before Execution. Cryptology ePrint Archive, Report 2016/1196. (2017).
- [44] Yeongjin Jang, Jaehyuk Lee, Sangho Lee, and Taesoo Kim. 2017. SGX-Bomb: Locking Down the Processor via Rowhammer Attack. In *SysTEX*.
- [45] Jecdec Solid State Technology Association. 2013. Low Power Double Data Rate 3. (2013). <http://www.jedec.org/standards-documents/docs/jesd209-4a>

- [46] JEDEC Solid State Technology Association. 2017. Low Power Double Data Rate 4. (2017). <http://www.jedec.org/standards-documents/docs/jesd209-4b>
- [47] Suryaprasad Kareenahalli, Zohar B. Bogin, and Mihir D. Shah. 2003. Adaptive idle timer for a memory device. (2003). <https://encrypted.google.com/patents/US7076617B2> US Patent Grant 2005-06-21.
- [48] Dimitris Kaseridis, Jeffrey Stuecheli, and Lizy Kurian John. 2011. Minimalist open-page: A DRAM page-mode scheduling policy for the many-core era. In *International Symposium on Microarchitecture (MICRO)*.
- [49] Dae-Hyun Kim, Prashant J Nair, and Moinuddin K Qureshi. 2015. Architectural support for mitigating row hammering in DRAM memories. *IEEE Computer Architecture Letters* 14, 1 (2015), 9–12.
- [50] Yoongu Kim, Ross Daly, Jeremie Kim, Chris Fallin, Ji Hye Lee, Donghyuk Lee, Chris Wilkerson, Konrad Lai, and Onur Mutlu. 2014. Flipping bits in memory without accessing them: An experimental study of DRAM disturbance errors. In *ISCA'14*.
- [51] Mark Lanteigne. 2016. How Rowhammer Could Be Used to Exploit Weaknesses in Computer Hardware. (March 2016). <http://www.thirdio.com/rowhammer.pdf>
- [52] Moritz Lipp, Daniel Gruss, Raphael Spreitzer, Clémentine Maurice, and Stefan Mangard. 2016. ARMageddon: Cache Attacks on Mobile Devices. In *USENIX Security Symposium*.
- [53] Yabing Liu, Will Tome, Liang Zhang, David Choffnes, Dave Levin, Bruce Maggs, Alan Mislove, Aaron Schulman, and Christo Wilson. 2015. An End-to-End Measurement of Certificate Revocation in the Web's PKI. In *IMC '15*.
- [54] Sreenivas Mandava, Brian S. Morris, Suneeta Sah, Roy M. Stevens, Ted Rossin, Mathew W. Stefaniw, and John H. Crawford. 2017. Techniques for determining victim row addresses in a volatile memory. (2017). <https://encrypted.google.com/patents/US9824754B2> US Patent Grant 2017-11-21.
- [55] Marcin Kaczmarek. 2014. Thoughts on Intel Xeon E5-2600 v2 Product Family Performance Optimisation – component selection guidelines. (August 2014). <http://infobazy.gda.pl/2014/pliki/prezentacje/d2s2e4-Kaczmarek-Optymalna.pdf> Infobazy 2014.
- [56] Mark Goodwin. 2015. Improving Revocation: OCSP Must-Staple and Short-lived Certificates. (2015). <https://blog.mozilla.org/security/2015/11/23/improving-revocation-ocsp-must-staple-and-short-lived-certificates/>
- [57] Clémentine Maurice, Nicolas Le Scouarnec, Christoph Neumann, Olivier Heen, and Aurélien Francillon. 2015. Reverse Engineering Intel Complex Addressing Using Performance Counters. In *RAID*.
- [58] Micron. 2014. DDR4 SDRAM. (2014). https://www.micron.com/~media/documents/products/data-sheet/dram/ddr4/4gb_ddr4_sdram.pdf Retrieved on February 17, 2016.
- [59] Microsoft. 2017. Cache and Memory Manager Improvements. (April 2017). <https://docs.microsoft.com/en-us/windows-server/administration/performance-tuning/subsystem/cache-memory-management/improvements-in-windows-server>
- [60] Paul V Mockapetris. 1987. Domain names-concepts and facilities. (1987).
- [61] James A Muir. 2006. Seifert's RSA fault attack: Simplified analysis and generalizations. In *International Conference on Information and Communications Security*.
- [62] Onur Mutlu. 2017. The RowHammer problem and other issues we may face as memory becomes denser. In *Design, Automation & Test in Europe Conference & Exhibition (DATE)*.
- [63] myLoc managed IT. 2018. The dedicated server in comparison. (May 2018). <https://www.mylloc.de/en/server-hosting/dedicated-server/dedicated-server-comparison.html>
- [64] netcup. 2018. Dedicated servers for professional applications. (May 2018). <https://www.netcup.eu/professional/dedizierte-server/>
- [65] OpenSSL. 2015. Online Certificate Status Protocol utility. (Jan. 2015). <https://www.openssl.org/docs/man1.0.2/apps/ocsp.html>
- [66] Paul Mutton. 2014. Certificate revocation: Why browsers remain affected by Heartbleed. (2014). <https://news.netcraft.com/archives/2014/04/24/certificate-revocation-why-browsers-remain-affected-by-heartbleed.html>
- [67] Matthias Payer. 2016. HexPADS: a platform to detect "stealth" attacks. In *ES-SoS'16*.
- [68] Peter Pessl, Daniel Gruss, Clémentine Maurice, Michael Schwarz, and Stefan Mangard. 2016. DRAMA: Exploiting DRAM Addressing for Cross-CPU Attacks. In *USENIX Security Symposium*.
- [69] Rui Qiao and Mark Seaborn. 2016. A New Approach for Rowhammer Attacks. In *International Symposium on Hardware Oriented Security and Trust*.
- [70] Qualcomm. 2017. Snapdragon 845 Mobile Platform Product Brief. (Dec. 2017). <https://www.qualcomm.com/documents/snapdragon-845-mobile-platform-product-brief>
- [71] Inc. Qualcomm Technologies. 2016. Qualcomm Snapdragon 600 APQ8064: Data Sheet. (2016).
- [72] Inc. Qualcomm Technologies. 2016. Qualcomm Snapdragon 600E Processor APQ8064E: Recommended Memory Controller and Device Settings. (2016).
- [73] Kaveh Razavi, Ben Gras, Erik Bosman, Bart Preneel, Cristiano Giuffrida, and Herbert Bos. 2016. Flip Feng Shui: Hammering a Needle in the Software Stack. In *USENIX Security Symposium*.
- [74] Red Hat. 2017. *Red Hat Enterprise Linux 7 - Virtualization Tuning and Optimization Guide*.
- [75] Ronald L. Rivest, Adi Shamir, and Leonard M. Adleman. 1977. Cryptographic communications system and method. (1977). <https://patents.google.com/patent/US4405829> US Patent Grant 1983-09-20.
- [76] Hemant G Rotithor, Randy B Osborne, and Nagi Aboulenein. 2006. Method and apparatus for out of order memory scheduling. (Oct. 2006). US Patent 7,127,574.
- [77] Stefan Santesson, Michael Myers, Rich Ankney, Ambarish Malpani, Slava Galperin, and Dr. Carlisle Adams. 2013. X.509 Internet Public Key Infrastructure Online Certificate Status Protocol - OCSP. RFC 6960. (2013). <https://doi.org/10.17487/RFC6960>
- [78] Mark Seaborn and Thomas Dullien. 2015. Exploiting the DRAM rowhammer bug to gain kernel privileges. In *Black Hat Briefings*.
- [79] X. Shen, F. Song, H. Meng, S. An, and Z. Zhang. 2014. RBPP: A row based DRAM page policy for the many-core era. In *IEEE ICPADS*.
- [80] Adrian Tang, Simha Sethumadhavan, and Salvatore Stolfo. 2017. CLKSCREW: Exposing the Perils of Security-Oblivious Energy Management. In *USENIX Security Symposium*.
- [81] Andrei Tatar, Radhesh Krishnan, Elias Athanasopoulos, Cristiano Giuffrida, Herbert Bos, and Kaveh Razavi. 2018. Throwhammer: Rowhammer Attacks over the Network and Defenses. In *USENIX ATC*.
- [82] Chee Hak Teh, Suryaprasad Kareenahalli, and Zohar Bogin. 2006. Dynamic update adaptive idle timer. (2006). <https://encrypted.google.com/patents/US7076617B2> US Patent Grant 2009-09-08.
- [83] Victor van der Veen. 2016. Drammer: Deterministic Rowhammer Attacks on Mobile Platforms. (2016). <http://vvdveen.com/publications/drammer.slides.pdf>
- [84] Victor van der Veen, Yanick Fratantonio, Martina Lindorfer, Daniel Gruss, Clémentine Maurice, Giovanni Vigna, Herbert Bos, Kaveh Razavi, and Cristiano Giuffrida. 2016. Drammer: Deterministic Rowhammer Attacks on Mobile Platforms. In *CCS'16*.
- [85] webtopia. 2018. Dedicated Server. (May 2018). <https://www.webtopia.com/en/dedicated-server/root-server-vergleich.html>
- [86] Yuan Xiao, Xiaokuan Zhang, Yinqian Zhang, and Radu Teodorescu. 2016. One Bit Flips, One Cloud Flops: Cross-VM Row Hammer Attacks and Privilege Escalation. In *USENIX Security Symposium*.
- [87] Yuval Yarom and Katrina Falkner. 2014. Flush+Reload: a High Resolution, Low Noise, L3 Cache Side-Channel Attack. In *USENIX Security Symposium*.
- [88] Yuval Yarom, Qian Ge, Fangfei Liu, Ruby B. Lee, and Gernot Heiser. 2015. Mapping the Intel Last-Level Cache. *Cryptology ePrint Archive, Report 2015/905* (2015), 1–12.
- [89] Tatu Ylonen and Chris Lonvick. 2006. The secure shell (SSH) protocol architecture. (2006).
- [90] Thomas M. Zeng. 2012. The Android ION memory allocator. (Feb. 2012). <https://lwn.net/Articles/480055/>
- [91] Tianwei Zhang, Yinqian Zhang, and Ruby B. Lee. 2016. CloudRadar: A Real-Time Side-Channel Attack Detection System in Clouds. In *RAID*.

A KERNEL ACCESSES FOR NETWORK PACKETS

Table 2 shows the results of the *funccount* script of the *perf* framework [22] for functions with *udp* in their name while the targeted system is flooded with UDP packets.

Table 2: Results of *funccount* on the victim machine for functions with udp in their name while the system is flooded with UDP packets.

Function	Number of calls
__udp4_lib_lookup	2 000 024
__udp4_lib_rcv	1 000 012
udp4_gro_receive	1 000 012
udp4_lib_lookup_skb	1 000 012
udp_error	1 000 012
udp_get_timeouts	1 000 013
udp_gro_receive	1 000 013
udp_packet	1 000 012
udp_pkt_to_tuple	1 000 012
udp_rcv	1 000 012
udp_v4_early_demux	1 000 012