Nethammer: Inducing Rowhammer Faults through Network Requests

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

Nethammer: Inducing Rowhammer Faults through Network Requests

Moritz Lipp
Graz University of Technology

Michael Schwarz
Graz University of Technology

Lukas Raab
Graz University of Technology

Lukas Lamster
Graz University of Technology

Misiker Tadesse Aga
University of Michigan

Clémentine Maurice
Univ Rennes, CNRS, IRISA

Daniel Gruss
Graz University of Technology

***Abstract*—In this paper, we present Nethammer, a remote Rowhammer attack without a single attacker-controlled line of code on the targeted system, *i.e.*, not even JavaScript. Nethammer works on commodity consumer-grade systems that either are protected with quality-of-service techniques like Intel CAT or that use uncached memory, flush instructions, or non-temporal instructions while handling network requests (e.g., for interaction with the network device). We demonstrate that the frequency of the cache misses is in all three cases high enough to induce bit flips. Our evaluation showed that 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 invalidate threat models of Rowhammer defenses building upon the assumption of a local attacker. Consequently, we show that most state-of-the-art defenses do not affect our attack. In particular, we demonstrate that target-row-refresh (TRR) implemented in DDR4 has no aggravating effect on local or remote Rowhammer attacks.**

1. Introduction

Hardware-fault attacks have been considered a security threat since at least 1997 [1], [2]. In such attacks, the attacker intentionally brings devices into physical conditions that 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 temperatures, 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 [3], [4].

The Rowhammer bug is a hardware reliability issue of DRAM [4]. An attacker can exploit this bug by repeatedly accessing (*hammering*) DRAM cells at a high frequency, causing unauthorized changes in physically adjacent memory locations. Examples of Rowhammer attacks include privilege escalation from native environments [5], [6], from within a browser's sandbox [7], and from within virtual machines running on third-party compute clouds [8], mounting fault attacks on cryptographic primitives [9], [10], and obtaining root privileges on mobile phones [11].

Intel CAT is a quality-of-service feature [12], allowing to restrict cache allocation of cores to a subset of cache ways of the last-level cache, removing interference of workloads in shared environments, e.g., protecting virtual machines against performance degradation due to cache thrashing of co-located virtual machines. However, Aga et al. [13] showed that Intel CAT facilitates eviction-based Rowhammer attacks.

The large majority of previous Rowhammer attacks required some form of local code execution, e.g., JavaScript [7] or native code [4]–[6], [8]–[11], [13]–[16]. Consequently, all works on Rowhammer defenses assume that some form of local code execution is required [4], [14], [17]–[24]. In contrast, Tatar et al. [25] utilized RDMA-enabled network cards to perform targeted memory accesses to specific physical addresses over a remote interface to induce bit flips.

In this paper, we challenge the requirements of remote Rowhammer attacks. We present Nethammer, a Rowhammer attack that does not require local code execution, nor RDMA-enabled network cards. Nethammer only requires a fast network connection between the attacker and the victim. It sends a crafted stream of size-optimized packets to the victim, causing a high number of memory accesses to the same set of memory locations. If any software processing the network request (e.g., user application, shared libraries, network stack, network driver) use uncached memory, non-temporal instructions 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, *i.e.*, Rowhammer. While, as in the first practical Rowhammer attacks [5], an attacker cannot control the addresses of the bit flips, we demonstrate how an attacker can still exploit them and reduce the probability of flips in non-attacker controlled regions by spraying.

To build Nethammer, we systematically analyzed the requirements to induce bit flips and, in particular, real-world memory-controller page policies. In most Rowhammer attacks, two DRAM rows are 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 [6]. We demonstrate one-location hammering [6] with adaptive page policies for the first time.

We also analyzed memory operations during network requests and analyzed the Nethammer bit flips we empirically obtained on our target systems and different potential target applications. 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. We empirically observed bit flips using Nethammer already after 300 ms runtime and up to 10 000 per hour.

Finally, we evaluate state-of-the-art defenses and show that most of them do not affect our attack. In particular, we show that TRR does not mitigate Rowhammer.

Contributions. The contributions of this work are:

- We present Nethammer, a remote Rowhammer attack that does not require attacker-controlled code on the target device, nor RDMA-enabled network cards.
- We demonstrate Nethammer on systems using uncached memory (or `clflush`) while handling network packets.
- We show how memory controller policies can automatically be identified.
- We show that the TRR countermeasure in DDR4 has no significant effect on Rowhammer attacks.

Outline. Section 2 provides background. Section 3 gives an attack overview. Section 4 describes the building blocks. Section 5 describes specific exploit strategies. Section 6 evaluates our empiric results. In Section 7, we discuss limitations and specific defenses. Section 8 concludes.

Responsible Disclosure. We responsibly informed Intel about Nethammer on March 20, 2018. We disclosed a full report of Nethammer, including the ineffectiveness of TRR on DDR4 to Intel, ARM, Qualcomm, on May 11, 2018.

2. Background and Related Work

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

DRAM and Memory Controller Policies. DRAM in modern computers is organized for a high degree of parallelism, in a hierarchy of 1–4 channels, one or more DIMMs, 1–4 ranks, 1–4 bank groups, and 8 or 16 banks. Each bank is an array of *cells*, organized in *rows* and *columns*, storing the actual memory content. The memory controller translates physical addresses to channel, DIMM, rank, bank group, bank, row, and column addresses. Pessl et al. [26] reverse-engineered these addressing functions using an automated technique for several processors.

As DRAM cells lose their charge over time, they must be refreshed periodically. The refresh interval is defined as 64 ms but can be adjusted to compensate, e.g., for temperature.

Each bank has a *row buffer*, buffering any read and write accesses to rows in this bank. Hence, depending on the state of the row buffer three different cases can occur: Row hits are the fastest, an access to a row in a pre-charged bank (*i.e.*, no row in the row buffer) is a few nanoseconds slower, row conflicts (*i.e.*, other row in row buffer) are measurably slower. 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, and the bank is pre-charged.
2) *Fixed open-page policy*: the page is left open for a fixed amount of time. This policy is beneficial for high-locality workloads, for power consumption and bank utilization [27].
3) *Adaptive open-page policy*: the adaptive open-page policy by Intel [28] is similar to the fixed open-page policy but dynamically adjusts the page timeout interval per bank.

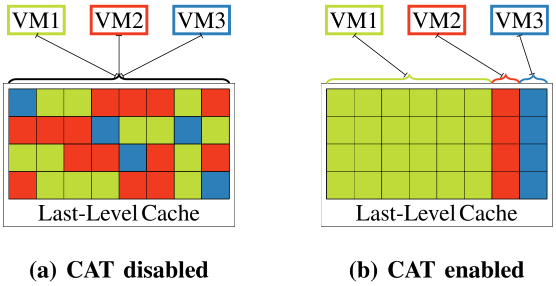
As modern processors have many cores running independently as well as deploy large caches and complex algorithms for spatial and temporal prefetching, the probability that subsequent memory accesses go to the same row decreases. Awashti et al. [29] proposed an access-based page policy that assumes a row receives the same number of accesses as the last time it was activated. Shen et al. [30] 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 [31], [32]. Consequently, more complex memory controller policies have been proposed and are implemented in modern processors [24], [27].

Rowhammer. Increasing DRAM cell density achieves higher storage capacity and lower power consumption, but cells may be more susceptible to disturbance errors [16], *i.e.*, bit flips. Such bit flips can be induced from software by bypassing the cache using specific instructions [4], cache eviction [7], [13], [14], [33], or uncached memory [11], [15]. Different access patterns have been developed to induce Rowhammer bit flips:

1) *Single-sided hammering* [5] accesses 8 randomly chosen memory locations simultaneously. The probability is high that at least 2 out of 8 random memory locations map into the same out of 32 DRAM banks on DDR3.
2) *Double-sided hammering* hammers two rows sandwiching a third. This requires at least partial knowledge of virtual-to-physical and physical-to-DRAM mappings.
3) *One-location hammering* [6] only accesses one single location at a high frequency. The attacker does not directly induce row conflicts but instead keeps reopening 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.

Using these techniques, the Rowhammer bug has been exploited in different scenarios and environments, e.g., attacking [10], sandboxes [5], [7], [33], native environments [5], [6], virtual machines [8], [9], mobile devices [11].

To develop defenses, a large body of research focused on detecting [17]–[21], [34], neutralizing [7], [9], [11], [22], or eliminating [4], [14], [22]–[24] Rowhammer attacks in software or hardware. The LPDDR4 standard [35] specifies two features to mitigate Rowhammer attacks:



(a) CAT disabled **(b) CAT enabled**

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

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. One-location hammering however bypasses all software-based defenses [6].

Tatar et al. [25] utilized RDMA-enabled network cards to induce bit flips remotely. RDMA enables remote access to specific physical addresses in a controlled way and, hence, can be used to implement Rowhammer memory access patterns. RDMA-enabled network cards are expensive and are only used by a few cloud providers [36]. In 2019, Cojocar et al. [37] demonstrated Rowhammer attacks bypassing ECC protection. In March 2020, Frigo et al. [38] analyzed TRR in more depth, confirming our findings of Section 6.

Caches and Cache Eviction. Hardware caches keep frequently used data from main memory in smaller but faster memories. Modern CPUs have multiple cache levels, with the L3 cache usually being the largest but slowest cache, shared across cores and inclusive to lower-level caches. The L3 cache on such CPUs has sets consisting of a fixed number of cache ways, where the set is determined by the physical address, and a replacement policy decides which way to replace (evict).

To mount a Rowhammer attack, an attacker needs to bypass the cache, e.g., via the unprivileged `clflush` instruction [39], or uncached memory [11]. An attacker can also resort to cache eviction by accessing congruent memory addresses [7], [33], [40], *i.e.*, addresses that map to the same cache set. Gruss et al. [7] observed that it is important to trick the replacement policy into keeping memory locations of the attacker cached, rather than the victim address that the attacker wants to evict.

In 2016, Intel introduced Cache Allocation Technology (CAT) [41] to address quality of service in multi-core server platforms [12], [42]. 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 on servers. When a virtual machine on a server 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 1. However, Aga et al. [13] showed that Intel CAT allows improving eviction-based Rowhammer attacks as it reduces the number of accesses, and thus the time, required for cache eviction.

3. Nethammer Attack

In this section, we present Nethammer, a Rowhammer attack not relying on any attacker-controlled code on the victim machine, nor RDMA-enabled network cards.

Attack Overview. Nethammer sends a crafted stream of network packets to the target device to mount a one-location or single-sided Rowhammer attack. For each packet received on the target device, a set of addresses is accessed, e.g., in the kernel driver, in a user-space application processing the contents, somewhere in between (e.g., network stack, shared libraries), or a combination of all. 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) *evicts* (and later on reloads) an address;
2) uses *uncached* memory;
3) uses *non-temporal* instructions;
4) *flushes* (and later on reloads) an address.

Non-temporal instructions perform their operations directly to the memory bypassing the cache [15]. *Uncached* memory is used on virtually all 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, and we found several open-source repositories where the `clflush` instruction was used for interaction with the hardware, but only one of them was an (outdated) network driver. We still describe this attack for completeness' sake, as it also applies to closed source drivers or user-space applications that handle the received packets and possibly use the `clflush` instruction. We verified that an attack is practical in both scenarios, as we describe in Section 6.

However, the main focus and contribution of this paper is an eviction-based remote Rowhammer attack. 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), allowing to control the amount of cache available to applications or virtual machines dynamically, as illustrated in Figure 1. 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 host. 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.

To induce bit flips remotely, one requirement is to send as many packets as possible over the network in a short time frame. As an example, UDP packets without content can be used, allowing an overall packet size of $64\,\mathrm{B}$, which is the minimum packet size for an Ethernet packet. This allows to send up to $1\,024\,000\,\mathrm{packets}$ per second over a $500\,\mathrm{Mbit/s}$ connection.

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. As DRAM with ECC can detect and correct single-bit errors and, thus, complicates Rowhammer attacks, we assume non-ECC memory on the victim machine. We did find server systems that have no ECC memory in the wild [43]–[46]. Note that this is not a real limitation, as Cojocar et al. [37] demonstrated Rowhammer attacks bypassing ECC protection. Furthermore, we assume that the victim machine uses either Intel CAT, available in Xeon CPUs, or uncached memory while handling network packets. We found $12.7\,\%$ of the dedicated hosts for sale on Hetzner [43] to have a Xeon CPU but non-ECC memory. Finally, we assume that the attacker has a sufficiently fast network connection to the victim, see Section 4.3. For our attack on personal computers, tablets, smartphones, or devices with similar hardware configuration, we make no further assumptions.

4. From Regular Memory Accesses to Rowhammer

We 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).

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. Finally, we investigate whether Rowhammer via network packets is theoretically possible. This is not trivial, as network packets do not arrive at the same speed as the memory accesses in an optimized tight loop.

4.1. Automated Classification of Memory-Controller Page Policies

Gruss et al. [6] 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. 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.

The undocumented mapping functions [26] allow to select addresses to access specific DRAM channels, ranks, banks, but also rows. Accessing same-bank different-row addresses consecutively causes a row conflict in the corresponding bank, incurring higher latency for the second access as 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.

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 baseline;
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 $(\texttt{tRP} + \texttt{tRCD})$ 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 the 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.

Our classification runs 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 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 2 shows the memory access time measured on an Intel Xeon D-1541 with different page policies, *i.e.*, the closed-page policy can be distinguished 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 [47].

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 3 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. [6] 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) 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 (the recommended value [48]), 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 [11], or access via the GPU [33]. Even in the presence of a flush instruction [49] or optimal cache eviction strategies [40], 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 [50]. 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

Nethammer requires a high frequency of memory accesses caused by processing network packets. Previous work indicated that at least 43 000 to 139 000 row activations [4], [7], [14] are required within one refresh interval to induce a bit flip.

In our experiments, we send 500 Mbit/s (and more) over the network interface. With a minimum size of 64 B for ethernet packets, this corresponds to 1 024 000 packets per second. Several kernel functions are called multiple times, e.g., up to 6 times (cf. Section 6). 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 [4], [7], [14]. 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

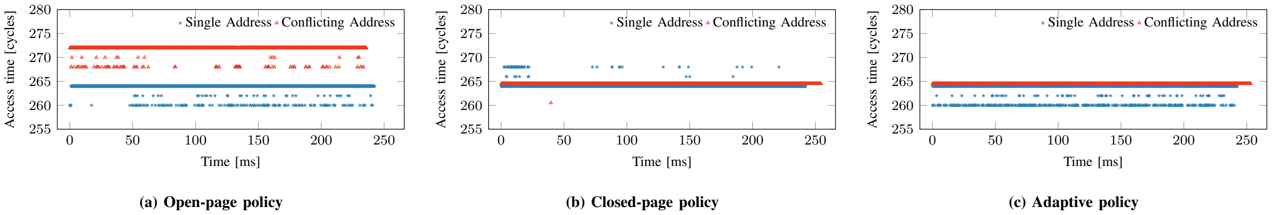
Nethammer does not control where in physical memory a bit flip is induced and, thus, what is stored at that location, the bit flip can have different consequences. We distinguish between bit flips in user memory, *i.e.*, memory pages that are mapped as `user_accessible` in at least one process and bit flips in kernel memory. We can also distinguish the bit flips based on their high-level effect, again forming two groups, depending on whether or not they lead to a denial-of-service situation. 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.

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. We empirically validated that this case occurs.[1]

SGX Enclave Page Cache. Bit flips in this region lock up the memory controller instantly (unsafely), halting the entire system [6], [51]. We empirically validated that this case occurs and found unsafe halting of the system to often leave permanent file system damage.

Application Memory. If a bit flip occurs in a user-space application, e.g., code or data, a possible outcome is the

[1] In fact, it was a problem when trying to trigger the other cases.



(a) Open-page policy

(b) Closed-page policy

(c) Adaptive policy

Figure 2: 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).

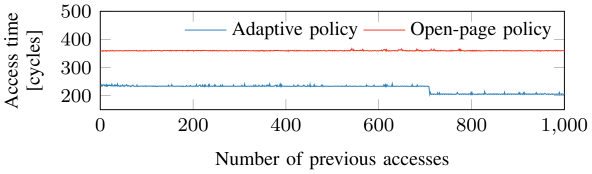


Figure 3: Open-page policy and adaptive page policy can be distinguished by testing increasing numbers of accesses to the same row.

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.

One example are DNS entries, which are altered such that a character of DNS entry points to a different domain, *i.e.*, bitsquatting [52]. Such bit flips in domains have been successfully exploited before using Rowhammer attacks [9]. Using zone transfers, an attacker can retrieve entries of an entire zone. 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 phishing website or a mail server intercepting email traffic. Users querying the DNS server for said entry connect to the attacker-controlled server and are thus exposed to data theft.

An attacker can also target Online Certificate Status Protocol (OCSP) servers that allow querying the status of a single certificate. The OCSP server manages a list of revoked certificate fingerprints. 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. 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 (a chance of 99.875 % *per bit flip* in the OCSP revocation list). A bit flip in the certificate identifier 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 [54]. 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. Hence, an attacker can again reuse that certificate.

We empirically observed bit flips in these applications, with a lower frequency than denial-of-service bit flips.

Cryptographic Material. Cryptographic material as part of the application memory is particularly interesting for attacks. To commit changes to a version-controlled repository, users authenticate with the service using public-key cryptography. 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, allow enumerating the registered users as well as their public keys. By enumerating and, thus, 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 [9], [55]. 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 is restored after a while when the key is evicted from the page cache and reloaded from the hard drive. As the correct key is restored, the attack leaves no traces. Furthermore, it 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, a countermeasure against

Rowhammer implemented in some DDR4 RAMs, does not protect against Nethammer or Rowhammer in general.

Environment. In our evaluation, we used the test systems listed in Table 1. We used the second and third system for our experiments with Intel CAT, which was configured exactly as recommended for quality-of-service purposes. For completeness' sake, on the first system, we ran an unprivileged server application which uses `clflush` while handling requests, and in another experiment, we installed a network driver which uses `clflush` while interacting with the network card. 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.

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 three cache-bypass techniques (cf. Section 3): Intel Xeon CPUs with Intel CAT for fast cache eviction, uncached memory on an ARM-based mobile device, and a single `clflush` instruction in the code running when receiving a packet.

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 use the *perf* framework 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 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 by 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), fewer bit flips occur in comparison to systems without TRR. In Section 6, we evaluate the number of bit flips depending on the configured page policy.

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 technology like Intel CAT (Section 2), the cache is not limited for certain applications and, thus, the eviction caused by handling memory packets has a low probability. As network drivers often use DMA memory and, thus, uncached memory, bit flips can be induced by network packets. While we identified a remarkable number of around 5500 uncacheable pages used by the system, we were not able to induce any bit flips remotely. However, the USB ethernet adapter allowed only a network capacity of less than $16\,\mathrm{Mbit\,s^{-1}}$, which is clearly too slow for a Nethammer attack. 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. Non-temporal instructions directly operate on the memory, thus, behaving similar to uncached memory.

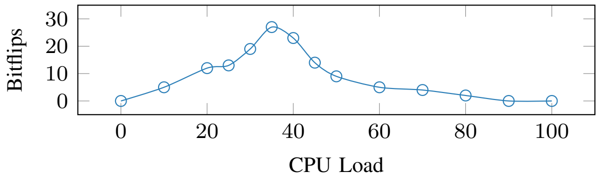


Figure 4: Number of bit flips depending on the CPU load with a closed-page policy after 15 minutes.

We implemented an unprivileged userspace server application which uses `clflush` while handling network requests. We then also tested a network driver implementation that uses `clflush` in the process of handling a network packet. Both tests were performed on an Intel i7-6700K CPU. While `clflush` is not very commonly used, our experiments provide valuable insights into the implications if it is used somewhere. We sent UDP packets with up to $500\,\mathrm{Mbit\,s^{-1}}$ and scanned memory regions where we expected bit flips. The results for both cases were very similar. For the driver variant, 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.

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 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.

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

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 4 that additional CPU utilization increases the number of bit flips drastically. Using the *Closed* policy, we observed 27 bit flips 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.

Bit Flips induced by Nethammer. As described in Section 5, 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 their effects.

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 Nethammer caused a bit flip in a file-system inode. The inode of a program that wanted to write data did not point to the correct file but to the kernel image and, thus, corrupted the kernel image.

Furthermore, we observed several bit flips immediately halting the entire system with no further interaction possible. 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.

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 to log in on the system. The system was restored to a stable state by rebooting the machine and thus reloading the entire code.

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

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. [4] already showed rows further away (even a distance of 8 rows and more) are affected as well. Still, the hardware vendors opted for implementing defenses focusing on the directly adjacent rows.

With the Low Power Double Data Rate 4 (LPDDR4) standard, the LPDDR4 standard 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) 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 [56]. 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 [56]. However, Kim et al. [4] 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 [57], DRAM module specifications allow automatically running TRR in the background.

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 defense is insufficient in mitigating Rowhammer attacks. In March 2020, Frigo et al. [38] analyzed TRR in more depth, confirming our findings that TRR does not prevent Rowhammer in practice.

7. Discussion

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 [39], eviction [7], [13] uncached memory [11] or non-temporal instructions [15] 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 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 [42]. Furthermore, the DRAM has to be susceptible to Rowhammer. We discovered in a brief market survey that many cloud providers offer hardware without ECC RAM [43]–[46], potentially allowing Nethammer attacks.

Nethammer sends as many network packets to the victim machine as possible, aiming to induce bit flips. Depending on the actual attack scenario (cf. 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 prevented due to the highly increased amount of traffic. In our experiments, we sent a stream of UDP packets with up to $500\,\mathrm{Mbit/s}$ to the target system. We could induce a bit flip every $350\,\mathrm{ms}$. Thus, if the first random bit flip already hits the target or causes a denial-of-service, the attack could already be successful. As the rows are periodically refreshed, an attacker only needs a burst of memory accesses to a row between two refreshes, *i.e.*, within a period of $64\,\mathrm{ms}$. Hence, an attacker could mount Nethammer for a few hundred milliseconds and then pause the attack for a longer time to prevent detection. While ethernet adapters in mobile phones are uncommon, many ARM-based embedded devices in IoT setups have gigabit ethernet.

The maximum throughput of these network cards we measured was too low on many of these devices, e.g., the Raspberry Pi 3 Model B+ [58], and WiFi chips typically offer too little capacity. However, on more recent modems, e.g., the Qualcomm X20 Gigabit LTE modem, throughputs up to $1.2\,\mathrm{Gbit/s}$ are possible in practice. This would enable sending enough packets to hammer specific addresses and potentially induce bit flips on the device.

8. Conclusion

In this paper, we presented Nethammer, a remote Rowhammer attack, with no attacker-controlled line of code on the target system. We demonstrate attacks on commodity consumer-grade systems, leading to temporary or persistent damage to the system. In some cases, the system was rendered unbootable after the attack. Our method to automatically identify the page policy used by the memory controller allowed us to show that adaptive page policies are also vulnerable to one-location hammering. We demonstrated the first one-location hammering attack on an ARM device, indicating their future exposure to Nethammer. Finally, we demonstrated that target-row-refresh (TRR) on DDR4 memory has no aggravating effect on local or remote Rowhammer attacks.

Acknowledgments

We thank our reviewers for their comments and suggestions that helped improving the paper. The project was 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. It was also supported by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No 681402). This work also benefited from the support of the project ANR-19-CE39-0007 MIAOUS of the French National Research Agency (ANR). Additional funding was provided by generous gifts from Intel and Red Hat. Any opinions, findings, and conclusions or recommendations expressed in this paper are those of the authors and do not necessarily reflect the views of the funding parties.

References

[1] D. Boneh, R. A. DeMillo, and R. J. Lipton, "On the Importance of Checking Cryptographic Protocols for Faults," in *EUROCRYPT*, 1997.

[2] E. Biham, "A fast new des implementation in software," in *International Workshop on Fast Software Encryption*, 1997.

[3] A. Tang, S. Sethumadhavan, and S. Stolfo, "CLKSCREW: Exposing the perils of security-oblivious energy management," in *USENIX Security Symposium*, 2017.

[4] Y. Kim, R. Daly, J. Kim, C. Fallin, J. H. Lee, D. Lee, C. Wilkerson, K. Lai, and O. Mutlu, "Flipping Bits in Memory Without Accessing Them: An Experimental Study of DRAM Disturbance Errors," in *ISCA*, 2014.

[5] M. Seaborn and T. Dullien, "Exploiting the DRAM rowhammer bug to gain kernel privileges," in *Black Hat Briefings*, 2015.

[6] D. Gruss, M. Lipp, M. Schwarz, D. Genkin, J. Juffinger, S. O'Connell, W. Schoechl, and Y. Yarom, "Another Flip in the Wall of Rowhammer Defenses," in *S&P*, 2018.

[7] D. Gruss, C. Maurice, and S. Mangard, "Rowhammer.js: A Remote Software-Induced Fault Attack in JavaScript," in *DIMVA*, 2016.

[8] Y. Xiao, X. Zhang, Y. Zhang, and R. Teodorescu, "One bit flips, one cloud flops: Cross-vm row hammer attacks and privilege escalation," in *USENIX Security Symposium*, 2016.

[9] K. Razavi, B. Gras, E. Bosman, B. Preneel, C. Giuffrida, and H. Bos, "Flip feng shui: Hammering a needle in the software stack," in *USENIX Security Symposium*, 2016.

[10] S. Bhattacharya and D. Mukhopadhyay, "Curious Case of Rowhammer: Flipping Secret Exponent Bits Using Timing Analysis," in *CHES*, 2016.

[11] V. van der Veen, Y. Fratantonio, M. Lindorfer, D. Gruss, C. Maurice, G. Vigna, H. Bos, K. Razavi, and C. Giuffrida, "Drammer: Deterministic Rowhammer Attacks on Mobile Platforms," in *CCS*, 2016.

[12] A. Herdrich, E. Verplanke, P. Autee, R. Illikkal, C. Gianos, R. Singhal, and R. Iyer, "Cache QoS: From concept to reality in the Intel Xeon processor E5-2600 v3 product family," in *IEEE HPCA*, 2016.

[13] M. T. Aga, Z. B. Aweke, and T. Austin, "When good protections go bad: Exploiting anti-DoS measures to accelerate Rowhammer attacks," in *HOST*, 2017.

[14] Z. B. Aweke, S. F. Yitbarek, R. Qiao, R. Das, M. Hicks, Y. Oren, and T. Austin, “ANVIL: Software-based protection against next-generation Rowhammer attacks,” *ACM SIGPLAN Notices*, 2016.

[15] R. Qiao and M. Seaborn, “A New Approach for Rowhammer Attacks,” in *International Symposium on Hardware Oriented Security and Trust*, 2016.

[16] O. Mutlu, “The RowHammer problem and other issues we may face as memory becomes denser,” in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2017.

[17] G. Irazoqui, T. Eisenbarth, and B. Sunar, “MASCAT: Stopping microarchitectural attacks before execution.” ePrint 2016/1196, 2017.

[18] N. Herath and A. Fogh, “These are Not Your Grand Daddys CPU Performance Counters – CPU Hardware Performance Counters for Security,” in *BlackHat US Briefings*, 2015.

[19] M. Payer, “HexPADS: a platform to detect “stealth” attacks,” in *ESSoS*, 2016.

[20] M. Chiappetta, E. Savas, and C. Yilmaz, “Real time detection of cache-based side-channel attacks using hardware performance counters.” ePrint 2015/1034, 2015.

[21] T. Zhang, Y. Zhang, and R. B. Lee, “Cloudradar: A real-time side-channel attack detection system in clouds,” in *RAID*, 2016.

[22] F. Brasser, L. Davi, D. Gens, C. Liebchen, and A.-R. Sadeghi, “CAn’t touch this: Software-only mitigation against Rowhammer attacks targeting kernel memory,” in *USENIX Security Symposium*, 2017.

[23] D.-H. Kim, P. J. Nair, and M. K. Qureshi, “Architectural support for mitigating row hammering in DRAM memories,” *IEEE Computer Architecture Letters*, vol. 14, 2015.

[24] M. Ghasempour, M. Lujan, and J. Garside, “ARMOR: A Run-time Memory Hot-Row Detector,” 2015.

[25] A. Tatar, R. Krishnan, E. Athanasopoulos, C. Giuffrida, H. Bos, and K. Razavi, “Throwhammer: Rowhammer Attacks over the Network and Defenses,” in *USENIX ATC*, 2018.

[26] P. Pessl, D. Gruss, C. Maurice, M. Schwarz, and S. Mangard, “DRAMA: Exploiting DRAM Addressing for Cross-CPU Attacks,” in *USENIX Security Symposium*, 2016.

[27] D. Kaseridis, J. Stuecheli, and L. K. John, “Minimalist open-page: A DRAM page-mode scheduling policy for the many-core era,” in *MICRO*, 2011.

[28] J. M. Dodd, “Adaptive page management,” 2003.

[29] M. Awasthi, D. W. Nellans, R. Balasubramonian, and A. Davis, “Prediction based dram row-buffer management in the many-core era,” in *PACT*, 2011.

[30] X. Shen, F. Song, H. Meng, S. An, and Z. Zhang, “RBPP: A row based DRAM page policy for the many-core era,” in *IEEE ICPADS*, 2014.

[31] S. Kareenahalli, Z. B. Bogin, and M. D. Shah, “Adaptive idle timer for a memory device,” 2003.

[32] C. H. Teh, S. Kareenahalli, and Z. Bogin, “Dynamic update adaptive idle timer,” 2006.

[33] P. Frigo, C. Giuffrida, H. Bos, and K. Razavi, “Grand Pwning Unit: Accelerating Microarchitectural Attacks with the GPU,” in *S&P*, 2018.

[34] D. Gruss, C. Maurice, K. Wagner, and S. Mangard, “Flush+Flush: A Fast and Stealthy Cache Attack,” in *DIMVA*, 2016.

[35] JEDEC Solid State Technology Association, “Low Power Double Data Rate 4,” 2017.

[36] Microsoft Azure, “High performance compute VM sizes,” 2018.

[37] L. Cojocar, K. Razavi, C. Giuffrida, and H. Bos, “Exploiting correcting codes: On the effectiveness of ecc memory against rowhammer attacks,” in *S&P*, 2019.

[38] P. Frigo, E. Vannacci, H. Hassan, V. van der Veen, O. Mutlu, C. Giuffrida, H. Bos, and K. Razavi, “TRRespass: Exploiting the Many Sides of Target Row Refresh,” in *S&P*, 2020.

[39] Y. Yarom and K. Falkner, “Flush+Reload: a High Resolution, Low Noise, L3 Cache Side-Channel Attack,” in *USENIX Security Symposium*, 2014.

[40] M. Lipp, D. Gruss, R. Spreitzer, C. Maurice, and S. Mangard, “ARMageddon: Cache Attacks on Mobile Devices,” in *USENIX Security Symposium*, 2016.

[41] Intel, “Intel 64 and IA-32 Architectures Software Developer’s Manual, Volume 3 (3A, 3B & 3C): System Programming Guide,” 2019.

[42] Intel, “Improving real-time performance by utilizing cache allocation technology: Enhancing performance via allocation of the processor’s cache,” 2015.

[43] Hetzner, “Dedicated root server hosting,” 2018.

[44] D. Hosting, “Highend dedicated rootserver,” 2018.

[45] myLoc managed IT, “The dedicated server in comparison,” 2018.

[46] netcup, “Dedicated servers for professional applications,” 2018.

[47] Intel, “Intel Xeon Processor E5 v4 Product Family: Datasheet Volume 2: Registers,” 2016.

[48] I. Qualcomm Technologies, “Qualcomm snapdragon 600e processor apq8064e: Recommended memory controller and device settings,” 2016.

[49] ARM, *ARM Architecture Reference Manual ARMv8*. ARM Limited, 2013.

[50] T. M. Zeng, “The android ion memory allocator,” 2012.

[51] Y. Jang, J. Lee, S. Lee, and T. Kim, “SGX-Bomb: Locking Down the Processor via Rowhammer Attack,” in *SysTEX*, 2017.

[52] A. Dinaburg, “Bitsquatting: DNS Hijacking without Exploitation,” in *BlackHat US Briefings*, 2011.

[53] Y. Liu, W. Tome, L. Zhang, D. Choffnes, D. Levin, B. Maggs, A. Mislove, A. Schulman, and C. Wilson, “An end-to-end measurement of certificate revocation in the web’s pki,” in *IMC*, 2015.

[54] Paul Mutton, “Certificate revocation: Why browsers remain affected by Heartbleed,” 2014.

[55] J. A. Muir, “Seifert’s RSA fault attack: Simplified analysis and generalizations,” in *International Conference on Information and Communications Security*, 2006.

[56] Marcin Kaczmarski, “Thoughts on Intel Xeon E5-2600 v2 Product Family Performance Optimisation – component selection guidelines,” August 2014. Infobazy 2014.

[57] S. Mandava, B. S. Morris, S. Sah, R. M. Stevens, T. Rossin, M. W. Stefaniw, and J. H. Crawford, “Techniques for determining victim row addresses in a volatile memory,” 2017.

[58] R. P. Foundation, “Raspberry pi 3 model b+,” 2018.

Appendix

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

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