Netherlands Organisation for Scientific Research

Vernieuwingsimpuls
Innovational Research Incentives Scheme
Grant application form 2018

Registration form (basic details)

1a. Details of applicant

1b. Title of research proposal

How I Learned to Stop Worrying and Love Unreliable Hardware

Research proposal

2a1 and 2a2. Description of the proposed research

2a1. Overall aim and key objectives

Software is becoming increasingly hard to compromise. Increasing maturity of formal verification is now allowing for practical development of secure software [1, 2] and deployment of sophisticated defences such as control-flow integrity protects existing software [3, 4]. Hardware attacks, however, are rapidly gaining ground as evident by variety of recently publicized attacks such as Rowhammer, Meltdown and Spectre [5, 6, 7, 8]. Hence, the number one challenge for system security is protecting against insecure hardware specially since they cannot easily be addressed using formal methods [9].

The most challenging component to protect is unreliable memory due to the way they are manufactured. The demand for higher performance and capacity is driving the industry to pack ever more transistors into memory chips, decreasing their reliability. A large proportion of currently deployed systems are already vulnerable to attacks [10] and the situation is projected to worsen as we build denser memory chips [11, 12]. For example, a recent study of Facebook datacentres found that the rate of server failures is directly proportionate to the density as shown in Figure 1. Server failures, however, are not the worst that can happen with unreliable memory. Attackers can exploit unreliable memory to compromise browsers, clouds and mobile devices [5, 14, 15, 16, 17, 18, 19]. These attacks are real: Bruce Schneier and Linus Torvalds have written about my findings while other researchers have also warned about them [20, 21, 22], and the hacker community awarded my work three Pwnie Awards[1]. We hence urgently need to address these reliability problems at either hardware or software.

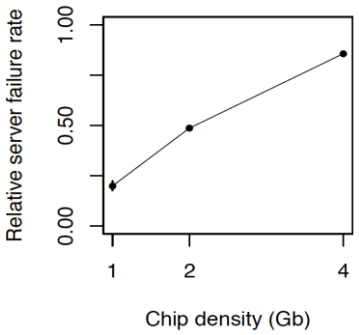

Figure 1. Server failures at Facebook data centers increase as memory becomes denser. Taken from [13].

Hardware protection: we should ideally address these issues directly in hardware by either building reliable hardware or implementing mitigations. Building fully reliable hardware as we scale transistors is becoming difficult and perhaps impossible [11, 12, 23]. Meanwhile, currently-deployed mitigations are ineffective because they make a trade-off between capacity, performance and power consumption. For example, error-correcting codes (ECC), while not always effective [10], require an additional chip, wasting capacity and power. They are hence only deployed in high-end servers. The mitigations that are being widely deployed, such as faster or targeted memory refreshing are not fully effective [24, 25], because they have to remain efficient in terms of power and performance. In fact, I recently discovered that recent Google Pixel devices with advanced mitigation in the form of targeted refresh are still vulnerable.

[1] Pwnie Awards are perhaps the most prestigious prizes in the hacking community. Only four have ever been awarded to researchers in the Netherlands.

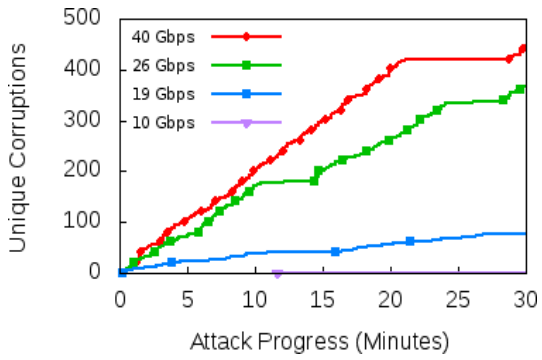

Figure 2. Attackers can corrupt memory over the network. Widely-deployed 10Gbps is already fast enough for this attack. (currently under submission)

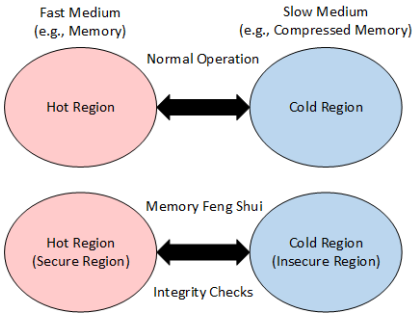

Figure 3. MFS incorporates existing memory management mechanisms to protect against unreliable memory.

Software protection: even if we could build cost-effective reliable hardware for the future, existing deployments remain exposed. Unlike software, upgrading hardware is costly, takes years and not always possible. The aim of this project is providing comprehensive software protection for arbitrary systems against unreliable memory. However, the problem with changing system software is sheer complexity that spans through many layers in the stack. This is the main reason why the few existing solutions cannot provide system-agnostic protection against all possible attacks [25, 26]. Figure 2 shows recent experiment corrupting memory used by a remote server application only by sending packets over the network. Existing solutions cannot protect against this new type of powerful attacks. Another example is recent work that uses integrated GPUs for compromising the browser on phones [16].

Memory Feng Shui: instead of building ad-hoc defences, I argue for a principled redesign of memory management which I term Memory Feng Shui (MFS). MFS is aimed at addressing wide-spread memory faults [10, 27] by partitioning the memory into secure and insecure regions. The invariant that I intend to enforce with MFS is that secure region can be accessed arbitrarily without any corruption, while the insecure region can contain corruptions and should be accessed with care to 1) avoid corrupting the secure region and 2) correct any potential corruption.

This partitioning for the entire software stack sounds like a major undertaking. Fortunately, a key insight makes this approach tractable: OSes have long implemented mechanisms to partition memory in areas that are frequently accessed (i.e., hot region) and areas that are not (i.e., cold region). They then store hot data on a fast medium and cold data on a slow medium to improve performance. Prime examples are swapping to disk and compressed in-memory caches in clouds [22]. As shown in Figure 3, I can repurpose these mechanisms to enforce MFS: the cold region can act as the insecure region to make a barrier that stops corruptions in the hot region. Hot region can act as the secure region and be accessed arbitrarily. I protect the cold region with software integrity checks without performance loss since it is infrequently accessed. Infrequent accesses to the cold region further helps protecting the integrity of the hot region.

By reusing existing memory management mechanisms, my solution does not require specific hardware features or substantial changes to the software stack, making it a compatible and practical solution. However, such a major redesign of memory

management requires a careful study of security, performance and compatibility which I will explore with instantiating MFS for memory modules vulnerable to Rowhammer.

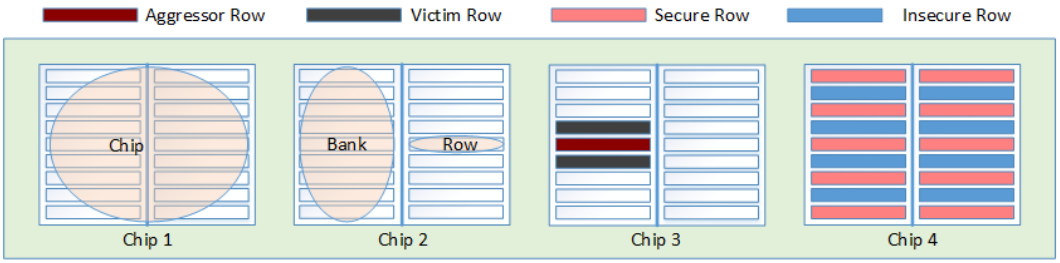

Figure 4. An example memory module consisting of four chips, each chip providing two banks. Data is stored on nine rows stacked together to form a bank. The Rowhammer causes corruption in adjacent rows (i.e., victims) of a frequently accessed row (i.e., aggressor). MFS assigns even/odd rows to secure/insecure regions.

Rowhammer: Figure 4 shows the organization of a sample memory module (Chip 1 and 2). As shown in Chip 3, attackers can place target data in a memory row (i.e., victim) and then later corrupt them by quickly accessing their adjacent rows (i.e., aggressors) to trigger Rowhammer. I now describe the challenges of applying MFS for protecting against Rowhammer.

Challenge 1: Secure Memory Partitioning

MFS can conceptually partition the memory into secure/insecure regions using even/odd rows (Chip 4). While it is possible to find out how memory addresses are mapped to different rows, banks and chips [29], MFS further needs to know whether they are mapped odd/even rows. Finding this information is challenging because it requires chip-level information which manufacturers do not provide. Solving this problem will be my first contribution. It will further enable research in Rowhammer attacks and defences beyond this project.

Approach: I propose using novel memory side channels for discovering this information. For example, it may be possible to use Rowhammer itself to find the ordering of rows similar to how data-dependent corruptions expose the proximity of memory bits [30]. Another possibility is a novel application of cold boot attacks [31, 32] by physically cooling down a small region of the memory chips and observing how bits get corrupted without power based on their local temperature.

Challenge 2: Operating System Support for Secure Memory

Once MFS can partition memory into secure and insecure regions, the OS needs to use this information for transparently exposing these regions to user applications. This sounds non-trivial in terms of engineering efforts. Fortunately, OSes already have mechanisms for moving data between different memory regions for improving performance. The main challenge is tapping into this capability for improving security without degrading performance.

Approach: I propose making OSes aware of how memory pages map to rows using the information I obtained earlier. This allows an MFS-enabled OS to partition the system into secure and insecure regions. Pages in the insecure region can be used for

bootstrapping an in-memory cache. It is possible to transparently use this cache for arbitrary applications in the system. In the background, the OS moves data between these two regions. Frequently accessed data will end up in the secure region and data that is rarely accessed will end up in the insecure region. While a basic page replacement policy might provide the desired performance for most workloads, latency-sensitive applications may require special treatment. I will explore the performance properties of this new design and investigate prefetching and caching optimizations when necessary [33].

Challenge 3: Mitigating Attacks

To mitigate attacks, MFS needs to ensure that accessing insecure region cannot corrupt memory in the secure region and that the integrity of data is preserved when it is moved from the insecure to the secure regions. Both are challenging with respect to performance.

Approach: I explore directions for limiting frequency of accesses and performing fast error correction in the insecure region. For example, I can use previous work in computer networking [35] to build efficient token-bucket rate limiters when accessing the insecure region. I can further interpose when pages move in and out the insecure region to ensure their integrity using software-based ECC. The ECC should be fast and space-efficient while providing strong guarantees against Rowhammer. Coming up with a good software-based ECC with these trade-offs is an interesting research direction which I will explore.

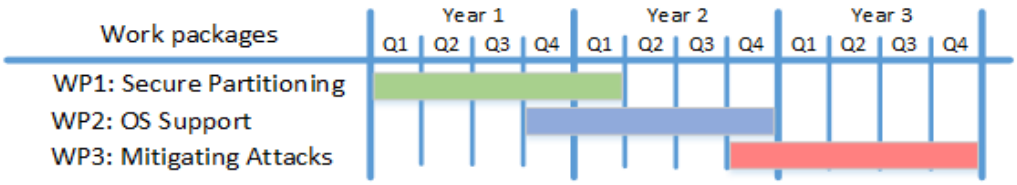

Figure 5. timeline of WPs in this project.

2a2. Research plan

The research plan is distributed over three WPs revolving around identified challenges. Figure 5 shows their timeline.

WP1: Secure Partitioning

I will use a representative set of memory modules to precompute the row ordering for the users. I will use the previously-developed Rowhammer testing tool [5] as a starting point. Further, I will prepare lab equipment for performing cold boot attacks and develop necessary analysis tools. I will use the results for the development of a Rowhammer emulator that will be of interest to other researchers in this area.

Risk assessment: if these approaches are not always successful, MFS can assume that the rows are mapped linearly to the banks based on their memory address. This assumption holds true for a large fraction of the deployed memory configurations which I will also quantify.

WP2: Operating System Support

Building on my previous experience, I will change Linux kernel's page allocator to make it aware of secure/insecure regions using the information derived from WP1. This WP can first assume a simplified linear row order and be later improved when the information provided by WP1 becomes more complete.

I will then look at different possibilities (e.g., ZSWAP [37]) for managing insecure memory in the Linux kernel. I intend to carry out evaluation on sensitive Rowhammer targets, including browsers, mobile devices and virtual machines.

Risk assessment: Some workloads may suffer from the latency of the cache which I can address using advanced caching and prefetching policies. If this is not enough, I will explore possibilities for directly exposing the insecure region to the applications.

WP3: Mitigating Attacks

I can use a conservative token bucket rate limiter for protecting the secure region against all possible Rowhammer corruptions. I will then look at fast, cryptographic hashes that can *detect* an ongoing attack in early stages. These steps provide an early prototype ready for testing.

I will then improve the prototype with ECC. I will explore the design space for software-ECC by looking at current hardware-based deployments [38] and current proposals [39].

Risk assessment: insecure region is accessed infrequently, but setting a conservative limit can still degrade the performance. If this is the case, I will look possibilities for using a secure cache for the insecure region. The downside will be added complexity which I will explore if necessary.

Beyond Protecting Memory

MFS will pave the way for a longer research agenda of dealing with future hardware unreliability issues beyond this project. For example, Onur Mutlu's group recently found out that it is possible to trigger hardware corruptions in Flash [27] and they can potentially be abused [40]. MFS can protect file systems or flash translation layers against these attacks. I intend to explore these directions towards the end of this project.

Collaborations

1) Onur Mutlu, ETH Zürich (WP1 and WP3)
2) Thorsten Holz, Ruhr-Universität Bochum (WP 2)
3) Paolo Costa, Microsoft Research Cambridge (WP3)

2b. Knowledge utilisation

Potential

This project provides an urgent software protection and its outcome will benefit end users, cloud providers, and other researchers.

1) **End users:** currently, the only way the end users can protect themselves is by buying new hardware without reliability issues. This is not cost-effective and sometimes not possible as hardware becomes more unreliable. For example, out of the two Pixel phones in my lab, both are vulnerable to Rowhammer. In the long term, my solution forms the basis for the users to immediately protect themselves without the need to upgrade or rely on their hardware to address the issues.

2) **Cloud providers:** cloud virtual machines rely on strong memory isolation in order to operate securely. Unfortunately, reliability issues compromise this isolation [5, 22]. My MFS prototype will address this problem by backing virtual machine memory from the secure region and exposing the insecure region as a memory-backed swap device.

3) **Researchers:** hardware researchers can draw principles from my research for better software support when dealing with unreliable hardware. For example, the memory subsystem can directly provide software with the abstraction of secure/insecure regions. In fact, I am already discussing these abstractions with Onur Mutlu during my current research visit. Security researchers who are working on the Rowhammer problem will benefit from the results obtained in WP1 and WP3 and OS researchers will benefit from studying the new memory management design point of WP2. Further, the principled design of MFS paves way for addressing emergent hardware issues [27, 40] and helps understanding system security in the broader context of computer science.

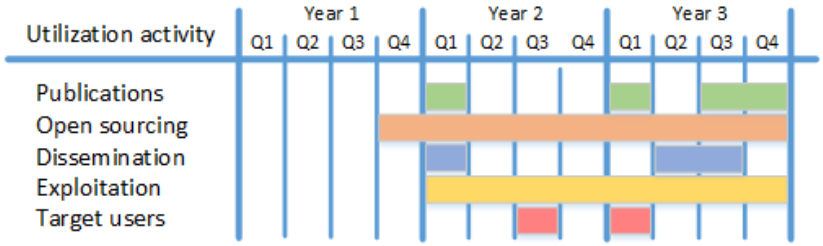

Figure 3. Distribution of utilization activities.

Implementation

I will make the results of my research available for experts through publications, for end-users through open sourcing and dissemination using my established news contacts and for specific target users through exploitation activities and collaboration. Figure 3 shows the timeline of these planned activities which I now discuss further.

1) **Conference publications:** I plan to publish four papers in top-tier security venues
    a. The mapping of memory addresses to rows and the novel techniques for obtaining them along with the Rowhammer emulator.

b. The architecture of an OS that supports secure/insecure regions and studying its security properties and performance.
c. The study of the design space considering different trade-offs with software-based ECC.
d. A systemization of knowledge paper about the outcome of this project for facilitating knowledge transfer between different researchers and starting new collaborations.

I further plan to give talks at the industry BlackHat conference similar to what I have done in the past to reach potential stakeholders.

2) **Open sourcing:** I will make the Rowhammer emulator and the MFS prototype available to the users as early as possible and will keep on updating them throughout the project.

3) **Dissemination:** my previous work in this area has generated more than 125 items in popular media: in newspapers (e.g., Volkskrant), TV (e.g., Universiteit van Nederlands and De Kennis van Nu), radio (e.g., BNR), popular online publications in Dutch (e.g., Tweakers) and international media (e.g., Ars Technica, WIRED, The Register). I intend these established contacts to reach the wider public about the outcome of this project.

4) **Exploitation:** I have already been contacted by multiple companies that expressed interest in my future mitigation efforts. Specifically, I am already working with Maarten Bodlaender on a framework with Philips for the exploitation of practical parts of my research.

5) **Specific target users:** My collaboration with NCSC-NL will allow me to reach out to affected companies. I routinely help companies such as Google, Microsoft, Mozilla, Apple and hardware manufacturers such as Intel and ARM to improve the security of their products. For example, my recent papers on memory deduplication prompted Microsoft to disable memory deduplication [14], GPG to increase the security of its key storage scheme, the Registrar of Last Resort to blacklist dangerous domains [5] and Google to secure its Android operating system against the Drammer attack [15]. I recently helped Apple to harden their Safari browser against my AnC attack. Here are some comments from influential individuals about my past research:

"Powerful Bit-Flipping Attack" Bruce Schneier, Resilient

"This is a significant finding. Thanks for Researching it." Mark Seaborn, Google

"This is an issue that will affect almost any modern computer system" Hugo Vincent, ARM Research

"We appreciate your assistance in helping to maintain and improve the security of our products." Cristina Formaini, Apple

2c. Number of words used

Section 2a: 2,000 words (max. 2,000 words)

Section 2b: 750 words (max. 750 words)

2d. Literature references

1. G. Klein, K. Elphinstone, G. Heiser, J. Andronick, D. Cock, P. Derrin, D. Elkaduwe, K. Engelhardt, R. Kolanski, M. Norrish, T. Sewell, H. Tuch and S. Windowwd, "*seL4: Formal Verification of an OS Kernel*", Symposium on Operating Systems Principles (SOSP), 2009.
2. C. Hawblitzel, J. Howell, J. R. Lorch, A. Narayan, B. Parno, D. Zhang and B. Zill, "*Ironclad Apps: End-to-end Security via Automated Full-system Verification*," USENIX Conference on Operating Systems Design and Implementation (OSDI), 2014.
3. M. Abadi, M. Budiu, Ú. Erlingsson, and J. Ligatti, "*Control-Flow Integrity*," The ACM Conference on Computer and Communication Security (CCS), 2005.
4. D. Wetson and M. Miller, "*Windows 10 Mitigation Improvements*," BlackHat Conference (BHUS), 2016.
5. K. Razavi, B. Gras, E. Bosman, B. Preneel, C. Giuffrida and H. Bos, "*Flip Feng Shui: Hammering a Needle in the Software Stack*," USENIX Security Symposium (SEC), 2016.
6. K. Razavi, B. Gras, E. Bosman, C. Giuffrida and H. Bos, "*ASLR on the Line: Practical Cache Attacks on the MMU from JavaScript*," The Network and Distributed System Security Symposium (NDSS), 2017.
7. A. Tang, S. Sethumadhavan and S. Stolfo, "*CLKSCREW: Exposing the Perils of Security-Oblivious Energy Management*," USENIX Security Symposium (SEC), 2017.
8. J. Horn, W. Haas, T. Prescher, D. Gruss, M. Lipp, S. Mangard, M. Schwarz, P. Kocher, D. Genkin, M. Hamburg and Y. Yarom, "*Meltdown and Spectre*," https://spectreattack.com, Accessed January 5, 2018.
9. D. Cock, Q. Ge, T. Murray and G. Heiser, "*The Last Mile: An Empirical Study of Timing Channels on seL4*," The ACM Conference on Computer and Communication Security (CCS), 2014.
10. 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*," International Symposium on Computer Architecture (ISCA), 2014.
11. O. Mutlu and L. Subramanian, "*Research Problems and Opportunities in Memory Systems*," Supercomputing Frontiers and Innovations (SUPERFRI), 2014.
12. V. Sridharan, N. DeBardeleben, S. Blanchard, K. B. Ferreira, J. Stearley, J. Shalf and S. Gurumurthi, "*Memory Errors in Modern Systems: The Good, The Bad, and The Ugly*," International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS), 2015.
13. J. Meza, Q. Wu, S. Kumar and Onur Mutlu, "*Revisiting Memory Errors in Large-Scale Production Data Centers: Analysis and Modeling of New Trends from the Field*", International Conference on Dependable Systems and Networks (DSN), 2016.
14. E. Bosman, K. Razavi, H. Bos and C. Giuffrida, "*Dedup Est Machina: Memory Deduplication as an Advanced Exploitation Vector*," IEEE Symposium on Security and Privacy (S&P), 2016.
15. 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*," The ACM Conference on Computer and Communication Security (CCS), 2016.
16. P. Frigo, C. Giuffrida, H. Bos and K. Razavi, "*Grand Pwning Unit: Accelerating Microarchitectural Attacks with the GPU*," accepted for publication in IEEE Symposium on Security and Privacy (S&P), 2018.
17. E. Bosman, K. Razavi, H. Bos and C. Giuffrida, "*Over the Edge: Silently Owning Windows 10's Secure Browser*," BlackHat Conference (BHUS), 2016.
18. K. Razavi, B. Gras, E. Bosman, B. Preneel, C. Giuffrida and H. Bos, "*Flip Feng Shui: Breaking the VM's Isolation*," in BlackHat Europe, 2016.
19. B. Bras, K. Razavi, E. Bosman, A. Barresi, "*Memory Deduplication: The Curse that Keeps on Giving*," 33rd Chaos Communication Congress (33C3), 2016.
20. M. Seaborn and T. Dullien, "*Exploiting the DRAM rowhammer bug to gain kernel privileges*," BlackHat Conference (BHUS), 2015.
21. D. Gruss, C. Maurice and S. Mangard, "*Rowhammer.js: A Remote Software-Induced Fault Attack in JavaScript*," Conference on Detection of Intrusions and Malware & Vulnerability Assessment (DIMVA), 2016.
22. Y. Xiao, X. Zhang, Y. Zhang and R. Teodorescu, "*One Bit Flips, One Cloud Flops: Cross-VM Row Hammer Attacks and Privilege Escalation*," USENIX Security Symposium (SEC), 2016.
23. O. Mutlu, "*The RowHammer Problem and Other Issues We May Face as Memory Becomes Denser*," Design, Automation, and Test in Europe Conference (DATE), 2017.
24. M. Lanteigne, "*How Rowhammer Could Be Used to Exploit Weaknesses in Computer Hardware,*" in SemiCON 2016.
25. Z. B. Aweke, S. F. Yitbarek, R. Qiao, R. Das, M. Hicks, Y. Orenm and T. Austin, "*ANVIL: Software-Based Protection Against Next-Generation Rowhammer Attacks,*" International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS), 2016.
26. 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,*" USENIX Security Symposium (SEC), 2017.
27. Y. Cai, S. Ghose, Y. Luo, K. Mai, O. Mutlu and E. F. Haratsch, "*Vulnerabilities in MLC NAND Flash Memory Programming: Experimental Analysis, Exploits, and Mitigation Techniques*," International Symposium on High-Performance Computer Architecture (HPCA), 2017.
28. S. Jennings, "*Transparent Memory Compression in Linux,*" In LinuxCon, 2013
29. P. Pessl, D. Gruss, C. Maurice, M. Schwarz and S. Mangard, "*DRAMA: Exploiting DRAM Addressing for Cross-CPU Attacks*," USENIX Security Symposium (SEC), 2016.
30. S. Khan, D. Lee, C. Wilkerson and O. Mutlu, "*PARBOR: An Efficient System-Level Technique to Detect Data Dependent Failures in DRAM*," in DSN, 2016.
31. J. A. Halderman, S. D. Schoen, N. Heninger, W. Clarkson, W. Paul, J. A. Calandrino, A. J. Feldman, J. Appelbaum and E. W. Felten, "*Lest We Remember: Cold Boot Attacks on Encryption Keys*," USENIX Security Symposium (SEC), 2008.
32. S. F. Yitbarek, M. T. Aga, R. Das and T. Austin, "*Cold Boot Attacks are Still Hot: Security Analysis of Memory Scramblers in Modern Processors*," International Symposium on High-Performance Computer Architecture (HPCA), 2017.
33. R. H. Patterson, G. A. Gibson, E. Ginting, D. Stodolsky and J. Zelenka, "*Informed Prefetching and Caching*," Symposium on Operating Systems Principles (SOSP), 1995.
34. K. Razavi and T. Kielmann, "*Scalable Virtual Machine Deployment Using VM Image Caches*", International Conference on High Performance Computing, Networking,

Storage and Analysis (SC), 2013.
35. P. Costa, H. Ballani, K. Razavi and I Kash, "*R2C2: A Network Stack for Rack-scale Computers*," ACM SIGCOMM Conference (SIGCOMM), 2015.
36. M. Oliverio, K. Razavi, H. Bos and C. Giuffrida, "*Secure Page Fusion with VUsion*," Symposium on Operating Systems Principles (SOSP), 2017.
37. Zswap in Linux kernel, https://www.kernel.org/doc/Documentation/vm/zswap.txt, Accessed 27.12.2017.
38. Advanced Micro Device, "*BIOS and Kernel Developer's Guide (BKDG) for AMD Family 15h Models 00h-0Fh Processors*," January 2013.
39. J. Kim and M. Sullivan and M. Erez, "*Bamboo ECC: Strong, safe, and flexible codes for reliable computer memory,*" International Symposium on High-Performance Computer Architecture (HPCA), 2015.
40. A. Kurmus, N. Ioannou, M. Neugschwandtner, N. Papandreou and T. Parnell, "*From random block corruption to privilege escalation: A filesystem attack vector for rowhammer-like attacks*," USENIX Workshop on Offensive Technologies (WOOT), 2017.