Hardware-Based Cyber Threats: Attack Vectors and Defence Techniques

Abstract: There are certain vulnerabilities associated with computing hardware that attackers can exploit to launch destructive attacks which often go undetected by the existing hardware and software countermeasures. Side Chanel Attacks (SCAs) and Rowhammer Attacks (RHAs), the consequences of hardware vulnerabilities, pose significant security and privacy threats to self-contained computing components and their end-users respectively. Such attacks compromise the security of computational environments, even those with advanced protection mechanisms such as virtualisation, sandboxes or robust encryptions. In light of these security threats against modern computing hardware, we perform an analysis overview of the modi operandi of SCAs and RHAs in hardware implementation and techniques that can be used to extract sensitive data such as secret keys. We then propose various countermeasures to safeguard against these attacks.

Keywords: side channels; microarchitectural attacks; cyber threats; hardware attacks; embedded systems; digital investigations; countermeasures;

1. Introduction

New technologies often claim to be secure by applying sandboxing techniques and performing processes in isolated software environments, i.e. Virtual Machines (VMs) (Wu et al., 2012; Aciimez, 2007). However, the implementation of isolated environments often does not adequately consider hardware emissions. The malicious use of hardware resources as side and covert channels can have a detrimental impact on the privacy of the end-users (victims), and modern processor architectures lack a thorough security analysis. Security vulnerabilities inherent in processor components are often due to their design and implementation (Aciimez, 2007; Aciimez and Koc, 2009; Bosman et al., 2016; Fournaris et al., 2017). For instance, although hardware manufacturers have been able to conceal the internal CPU architecture from programmers to a large extent, the CPU's timing behaviour is still highly visible (Ge et al., 2016; Bosman et al., 2016). All Microarchitectural attacks, irrespective of their type, can exploit security systems regardless of advanced partitioning methods (e.g. memory protection), sandboxing or even virtualisation. Hence, it is vital to identify every conceivable Microarchitectural susceptibility in order to comprehend the potential of Microarchitectural analysis and design to implement more secure systems. This can be achieved by adopting new approaches to develop appropriate software countermeasures and making specific hardware changes in future architectures.

Therefore, in light of the above discussion, our study will focus on three concepts in relation to Microarchitectural Attacks (MAs). To this end, we analyse (1) two MA variants, namely SCAs and the newly discovered Rowhammer Fault Injection attacks, (2) along with existing countermeasures, and (3) propose new mitigation strategies. From this analysis, we deduce insight in relation to the current state of knowledge observed in the literature, hypothesise means of attacks that have not been previously explored, predict possible future modus operandi of attacks, and propose effective future directions for defence mechanisms development.

The remainder of the paper is structured as follows: Section 2 provides a background for Microarchitectural analysis. Sections 3 and 4 analyse SCAs and RHAs respectively, while Sections 5 and 6 provide an analysis of the existing countermeasures to address the stated attacks. In Section 5, we provide our own recommendations on how to address such attacks. Finally, in Section 6, we conclude our study and discuss the future research direction in this field of study. Two main contributions of this paper are the scope of the discussion, since few works of similar scope currently exist, and the provision of an agenda for the direction of future research.

2. Attacks

2.1 Side-Channel Attacks

SCAs are almost undetectable hardware-targeted attacks that rely on the leakage produced by electronic circuits as by-products that render it possible for an adversary without access to circuit itself to find out how the circuit operates and what type of data it is processing. SCAs can be very detrimental if proper defence mechanisms have not been implemented on a targeted machine. They can be carried out to extract sensitive information from computing hardware by means of measurement and analysis of physical parameters such as execution time and electromagnetic emission. Heat and electromagnetic leakage are both feasible sources of information for an adversary. SCAs can also exploit

Microarchitectural performance of processor implementations, potentially exposing hidden hardware states. Through a SCA, the attacker takes advantage of the cache performance of a cryptosystem (Chiappetta et al., 2016; Brumley, 2015) by acquiring the execution time or power consumption variations created through cache hits and misses (Aciimez, 2007).

A SCA enables an unprivileged process to attack another process running side by side on the same processor (Aciimez and Koc, 2009; Neve and Seifert, 2006; Oren et al., 2015; Percival, 2005; Yarom and Falkner, 2014). This enables the adversary to extract sensitive data from the victim through shared CPU caches. This sensitive data is often related to both cryptographic processes such as signing or decryption and also other applications (Zhang et al., 2016, a). It is emitted via secret-dependent data flows that result in cache usage patterns that are visible to an adversary. With knowledge of this, an attacker can leverage various methods to manipulate data in the shared cache to infer the victim's cache usage patterns. As a result, the adversary will be able to deduce sensitive data that prescribes such patterns. Moreover, a SCA can also be carried out by monitoring operation such as AES T-table entry (Ashokkumar et al., 2016; Yarom and Falkner, 2014; Irazoqui et al., 2014) or modular exponentiation multiplicand accesses (Inci et al., 2016). The memory accesses of software cryptosystems use key-dependent table lookups. Exposing such memory access patterns through cache statistics and the knowledge of the processed message renders it possible to exploit these ciphers (Aciimez, 2007; Bluhm and Gueron, 2015; Mirvaziri et al., 2009).

In addition, there are certain cache attacks that can be conducted remotely over networks (Ma et al., 2014; Hashizume et al., 2013; Seibert et al., 2014). Within a Cloud Computing (CC) environment, SCAs can facilitate cryptographic/bulk key recovery (Genkin et al., 2017; Inci et al., 2016; Standaert et al., 2009). A Cloud's shared resources such as CPU and memory can provide the attackers with side-channels which leak sensitive data and facilitate key recovery attacks (Inci et al., 2016; Irazoqui et al., 2015, a; Sanfelix et al., 2015; Woodward, 2016). Those users that operate outdated libraries that are susceptible to leakage are more vulnerable to mass surveillance (Irazoqui et al., 2015, b). The most commonly exploited emission in the shared resource systems derive from the cache and the memory (Ge et al., 2016; Inci et al., 2016).

An RSA secret key from a co-located instance can be obtained by conducting a Prime+Probe Attack (Apecechea et al., 2014; Schwarz et al., 2017; Zhang et al., 2014). In this situation, to accelerate the attack, an adversary with advanced programming skills will be able to reverse engineer the cache slice selection algorithm, for instance for Xeon (a brand of x86 microprocessors targeted at the non-consumer workstation, server, and embedded systems) that is employed in distributed systems such as cloud computing. Furthermore, the attacker can also use noise reduction to infer the RSA private key from the monitored traces. By processing the noisy data, they will be able to retrieve the RSA key employed during decryption as demonstrated by (Agrawal et al., 2007; Apecechea et al., 2014; Genkin et al., 2014; Inci et al., 2016; Yarom and Falkner, 2014).

Based on the existing studies (Aciimez and Koc, 2009; Blomer and Krummel, 2007; Ge et al., 2016; Kong et al., 2013; Neve and Seifert, 2006; Zhang et al., 2016, b), we categorise cryptanalytic SCA into three classifications: trace-driven, time driven and a new category, access-driven, in accordance with the type of information that the adversary discovers about a victim's cipher. In trace-driven attacks, the attacker monitors the sequence of cache hits and misses during a cryptographic cipher execution. By observing

the execution details of the cipher, the adversary will be able to extract important information from specific key-dependent memory access results within a cache hit or a cache miss (Ashokkumar et al., 2016; Chen et al., 2013; Tiri et al., 2007). This denotes that trace-driven attacks enable the adversaries to discover the result of each of the victim's memory accesses in relation to cache hits and misses (Aciimez and Koc, 2009; Gallais et al., 2010; Page, 2005). They are often carried out in and against hardware because of the difficulty of obtaining the trace of cache hits and misses in software (Ge et al., 2016; Ashokkumar et al., 2016; Lauradoux, 2005).

In time-driven attacks, adversarial parties will monitor the entire runtime of a cryptographic cipher execution. They will then be able to extract important information since the runtime relies on the number of key-dependent memory accesses which lead to cache misses (Bernstein, 2005; Gullasch et al., 2011; Spreitzer and Plos, 2013, a; Spreitzer and Plos, 2013, b). Because the cache performance is only one part of the many aspects that impacts the total runtime of a cryptosystem, time-driven attacks necessitate statistical analysis employing a large number of samples to deduce important information (Bonneau, and Mironov, 2006; Spreitzer and Grard, 2014). Access-driven attacks monitor partial information of the addresses that the victim accesses. Similar to time-driven attacks, the timings of the cache are examined as a source of information emission. Access-driven attacks probe the cache behaviour with greater detail as opposed to assessing the overall runtime (Atici et al., 2013; Neve and Seifert, 2006; Zhang et al., 2016, a; Zhang et al., 2016, b). They are capable of identifying whether or not a cache line has been ejected as the main technique to launch an attack. Access-Driven Cache attacks themselves, can be classified into three subcategories, including: Evict+Time, Prime+Probe, Flush+Reload (Gruss et al., 2017; Osvik et al., 2006; Percival, 2005). Although many SCAs employ one of these three methods, there exist other variations to match the specific abilities of hardware and software environments (Gruss et al., 2017).

2.1 Rowhammer Attacks

There are certain vulnerabilities associated with computing hardware that attackers can exploit to launch destructive attacks which often go undetected by the existing software countermeasures against embedded device systems within cloud environments. One of the usual characteristics of a computer hardware component can be malfunction or some kinds of abnormal behaviour which can result in providing a backdoor access to a potential adversary (Fournaris et al., 2017). If a particular row of a Double Data Rate (DDR) memory bank is constantly activated (opened) and pre-charged (closed) within a DRAM refresh interval, one or more bit flips take place in physically-adjacent DRAM rows to an incorrect value. Such disturbance is known as Rowhammer (Kim et al., 2014). Google's Project Zero in discovered Rowhammer in 2015 establishing that that careful RAM bit-flipping in page table entries could allow an attacker to pwn Linux systems (Chirgwin, 2017; Seaborn and Dullien, 2015). To carry out a successful Rowhammer attack, the attacker will need to adopt four steps, including: (1) identifying the specific memory architecture characteristics of the targeted device (Fournaris et al., 2017), (2) activating rows in each bank in a swift manner to trigger the Rowhammer susceptibility, (3) accessing the aggressor physical address from userland (Seaborn and Dullien, 2015), and (4) taking advantage of Rowhammer vulnerability (Bit Flips) (Kim et al., 2014).

Advanced attackers (those with programming skills) can take advantage of Rowhammer to launch their attacks against the Dynamic Random-Access Memory

(DRAM) of a computing device. In this attack, through malicious codes, the adversaries can evade the defence mechanisms which are often deployed through traditional security software and features (such as memory isolation) to carry out the memory disturbance attack. Similarly, in relation to a CC environment, Rowhammer Fault Injection, which is a recently discovered real-time microarchitecture attack, can be launched remotely to obtain full access to the Dynamic Random-Access Memory (DRAM) of a CC device. Such an attack can pollute system memory, access and alter sensitive data and gain full control of the system.

Rowhammer is an issue with some new DRAM devices in which repeatedly accessing a row of memory can result in bit flips in adjoining rows. Through the Project Zero at Google, Seaborn and Dullien (2015) tested a collection of laptops and discovered the issue with some of those laptops. Their experiment was based on the implementation of two privileged escalation exploits that utilised such an impact. The first test involved employing Rowhammer-induced bit flips to gain kernel privileges on x86-64 Linux when executed as an unprivileged userland process. Seaborn and Dullien (2015) discovered that the process, when executed on a machine susceptible to the Rowhammer issue, was capable of triggering bit flips in page table entries (PTEs). It was capable of employing this to gain write access to its own page table, and as a result, gain read-write access to all of physical memory (Seaborn and Dullien, 2015). This test exploit on x86 systems employs CLFLUSH instruction to create many accesses to the underlying DRAM. Although creating bit flips in PTEs is just one vector of exploitation (as other methods for exploiting bit flips are possible too), nevertheless Seaborn and Dullien's experiment, which itself was based on a study by Kim et al. (2014), is a novel contribution to the field.

Repeatedly accessing two "aggressor" memory locations within the process's virtual address space can result in bit flips in a third, "victim" location, which likely to be outside the virtual address space of the process. The victim location is in a different DRAM row from the aggressor locations, and therefore in a different 4k page (Seaborn and Dullien, 2015; Kim et al., 2014). This works since DRAM cells have been growing smaller and closer together. Due to the fact that the DRAM production reduces features to smaller physical sizes to install more memory capacity onto a chip, it has become more problematic to stop DRAM cells interacting electrically with each other. Consequently, gaining access to one location in memory can disturb adjacent locations, resulting in charge leaking into or out of adjacent cells. With sufficient accesses, this can modify a cell's value from 1 to 0 or vice versa (Fournaris et al., 2017; Seaborn and Dullien, 2015; Kim et al., 2014).

3. Countermeasures

3.1 Cache-Based Side-Channel Attacks

The purpose of a SCA countermeasures must be to hide leakage or reduce it so that it holds minor or no valuable information for an adversary to launch an attack. Countermeasures against SCAs can be categorised into two groups of isolation measures and randomisation measures. Under the isolation measure, adversaries will no longer be able to gain access to the victim's machine's cache. Thus, carrying out an interference-related attack will become almost impossible. Isolation can be attained by dividing cache either statically or dynamically into zones associated with individual processes, for instance, by employing hardware virtualisation (Fournaris et al., 2017; Zhang and Lee,

2014). In randomisation approach, side channel information is randomised. Therefore, no accurate information is emitted from caches. This denotes that through the randomisation approach, sensitive data is disassociated from the emission trace and is concealed by hiding this data through a randomly produced number.

A variant of this technique is information hiding in which random noise is added to the leakage channel, thus rendering the associate to the leakage trace unusable (Fournaris, 2017; Fournaris et al., 2017; Zhang and Lee, 2014). There are two methods by which one can determine randomisation, including inserting random noise in the attacker's observations and randomising the mappings from memory addresses to cache sets. Often the proposed countermeasures included in the previously mentioned two categories focus on neutralising the cache features that a cache SCA exploits. Virtual time and black-box mitigation methods are also among the existing countermeasures against the SCAs (Yarom et al., 2016). Moreover, time partitioning can be employed through routine Cache Flushing, Lattice Scheduling, Memory Controller Partitioning, structuring Execution Leases and executing Kernel address space isolation (Fournaris et al., 2017).

Configurable cache architecture can be employed to provide hardware assisted defence against cache based channel attacks (Rani and Venkateswarlu, 2014; Wang and Lee, 2007; Page, 2005). The cache is dynamically divided into safeguarded regions and are capable of being configured for an application. In partitioned caches, there is a section of the cache that is assigned exclusively to the safeguarded process so as to avert information leakage. Therefore, partitioned cache mechanism can be deployed as a defence approach against cache based channel attacks. A partitioned cache will need to be included in devices that are susceptible to SCAs to separate the cache behaviour of one process to another. As a result, it will stop process interference through the provision of adequate space to store the entire S-box in cache, and it will be locked when it is pre-loaded. Segregation does not permit forcible flushing of the cache; furthermore, partitioned cache employs longer cache lines that render attacks more problematic. Moreover, a method called partition-locked cache (PLcache) (Wang and Lee, 2007) can be employed to deal with cache sharing issue. This method will employ a fine-grained locking control to isolate only the cache lines encompassing vital data. Through this method, only those cache lines that are of interest are locked by making private partitions.

The effectiveness of the defence mechanisms against SCAs proposed in the literature is open to question since many of them can be circumvented by employing the Rowhammer attacks through the techniques that were described in sub-section 2.2. Moreover, because SCAs leverage the physical elements of systems, many countermeasures take the approach of enhancing the security of the system design and development (Zhang and Lee, 2014). For example, to safeguard against cache SCAs different secure cache architectures have been presented in the literature (Domnitser et al., 2012; Wang and Ruby, 2008). These cache mechanisms are aimed at preventing specific kinds of SCAs without vast performance costs. Although the performance of such mechanisms can be assessed against criteria, nevertheless the efficiency of their security has only been examined qualitatively. However, it is necessary to carry out general quantitative methods of measuring the potential side-channel information leakage to be able to compare various cache architectures more accurately (Zhang and Lee, 2014). Such quantitative methods are necessary as they can expose which elements of the system are more vulnerable to emitting sensitive information, and facilitate the trade-off analysis among performance, power and security of various cache mechanisms.

3.1 Rowhammer Attacks

Configurable cache architecture can be employed to provide hardware assisted defence against cache based channel attacks (Wang and Lee, 2007; Page, 2005). The cache is dynamically divided into safeguarded regions and are capable of being configured for an application. To counteract a Rowhammer attack, Rowhammer-induced bit flips will need to be blocked by altering DRAM, memory controllers or the combination of both. A system must not also trigger any specific row repeatedly during a specific refresh point if the adjacent rows are not refreshed simultaneously. Moreover, Global Standards for the Microelectronics Industry (JEDEC, 2014) has recently published LPDDR4 standard for DRAM, which outlines two Rowhammer countermeasure methods that a memory controller must employ. These include "Targeted Row Refresh" (TRR) mode and "Maximum Activate Count" (MAC) metadata field. The TRR enables the memory controller to require the DRAM device to refresh a row's neighbours, whereas the MAC metadata field outlines the number of activations that a given row is safely capable of coping with before its neighbours require refreshing. Another countermeasure against Rowhammer arrack is to employ the physical probing the memory bus via a high-bandwidth oscilloscope by determining the voltage on the pins at the DIMM slots (Pessl et al., 2016). Another method of countering Rowhammer attack is to employ time analysis based on the Rowbuffer conflict to be able to determine address pair that is part of the same bank and then apply this address set to rebuild the precise map function automatically (Fournaris et al., 2017; Xiao et al., 2016; Pessl et al., 2016). Yet another simple solution requires that DRAM vendors build Rowhammer mitigations internally within a DRAM device, which does not need special memory controller support.

The CFLUSH command available on user space (userland) for x86 devices has also been employed (GitHub, 2017; Fournaris et al., 2017; Seaborn and Dullien, 2015; Kim et al., 2014) as a countermeasure to evict cache lines associated with the aggressor row addresses among its memory accesses (GitHub, 2017; Fournaris et al., 2017; Seaborn and Dullien, 2015; Kim et al., 2014).

The existing countermeasures against RHAs do not appear appropriate to address these attacks within CC environments. In the context of cloud, RHAs are executed on different attack interfaces (e.g. scripting language based attacks) by deploying web browsers that are activated remotely, a view supported also by Gruss et al. (2016) and Bhattacharya and Mukhopadhyay (2016). Therefore, new eviction methods will need to be developed to replace the existing flush instructions so that Rowhammer attacks within cloud environments can be addressed more effectively. The new methods must be able to facilitate the identification of an eviction set comprising of addresses that will be part of the same cache set of the aggressor rows. For instance, this can be achieved by employing a time attack to identify the eviction set. Another eviction method can be based on the reverse engineering analysis of the target system (Bosman et al., 2016). However, this can be a complex task considering the fact that modern Intel processors after Sandy Bridge presented an intricate hash function to partition the cache into slices even further (Maurice et al., 2015; Hund et al., 2013). Furthermore, in a recent paper (Van der Veen et al., 2016), employing the Direct Memory Access memory management technique has also been suggested as a method to bypass CPUs and their caches to address the Rowhammer attacks.

There are various other approaches that can be utilised as countermeasures to deal with Rowhammer attacks. For instance, by regularly refreshing the entire rows, one can expect the disturbance errors to be eradicated for adequately short refresh intervals (RI ≤ RIth) even though regular refreshing can reduce performance and energy-efficiency (Kim et al., 2014). Another method requires the manufacturers to retire DRAM cells that they determine as victim cells and remap them to spare cells (Kim et al., 2014). The end-users, themselves, could also retire DRAM sells by assessing and utilising system-level techniques for deactivating faulty addresses or remapping defective addresses to reserved addresses (Kim et al., 2014). Finding the hot row (i.e. specific row or aggressor row) and refreshing their adjacent rows are another way of dealing with a Rowhammer attack. A mechanism called Probabilistic Adjacent Row Activation (PARA), which is implemented in the memory Controller, is an approach proposed by Kim et al. (2014) to prevent DRAM disturbance errors. Through the PARA, when a row is constantly refreshed and closed, its neighbouring rows will also be refreshed with some low probability.

A Run-Time Memory Hot Detector (ARMOR), designed and developed by the researchers based in The University of Manchester's School of Computer Science (Ghasempour et al., 2015), is another approach to mitigate Rowhammer attacks. ARMOR is similar to that of DRAM in that it is implemented at the memory level. ARMOR is claimed to be able to detect all the possible Row Hammer errors, screen the activation flow at the memory level and also identify hot rows (specific rows) that might be hammered at run-time. Another solution to identify Rowhammer is to employ the last-level cache counter facility to produce an interrupt after N misses (Aweke et al., 2016). This method involves monitoring the last-level cache misses on a refresh interval and row access with high temporal locality on certain processors such as Intel/AMD (Fournaris et al., 2017). In cases where missing cache goes beyond a threshold, a selective refresh is carried out on the susceptible row.

Furthermore, software-based solutions have also been introduced to address Rowhammer attacks. These software-based mitigations containing ad-hoc defence techniques can be implemented on commodity systems. For instance, these include: rewriting the flush instructions that have been adopted in Google NaCl (Fournaris et al., 2017), doubling the RAM refresh rates (Kim et al., 2014), eliminating unprivileged access to the pagemap interface (Salyzyn, 2015; Shutemov, 2015; Seaborn and Dullien, 2015), and also eliminating the Clflush instruction (Seaborn and Dullien, 2015). Another example is that pagemap interface can be prevented from userland in new kernel Linux in order to stop Rowhammer attacks. However, there are emerging attacks that are capable of bypassing the above stated countermeasures by facilitating Rowhammer triggering through, for example, the use of JavaScript (Gruss et al., 2016) or native code (Qiao and Seaborn, 2016) without performing cache eviction. Likewise, the pagemap based countermeasure has been circumvented by employing various methods of establishing the machine address map to a physical address or the map to the virtual userland (Fournaris et al., 2017; Pessl et al., 2016; Xiao et al., 2015).

In a recent study, Gruss et al. (2017) proposed a novel Rowhammer attack and exploitation technique, Memory Waylaying, demonstrating that even the combination of all the existing countermeasures against Rowhammer attacks are ineffective. This new technique brings into question previous assumptions for activating the Rowhammer bug and ultimately the proposed countermeasures. As discussed previously, various DRAM rows will need to be hammered to trigger Rowhammer attack. However, based on Gruss et al.'s MW technique, only one DRAM row needs to be constantly open for such a

malicious purpose. Other ad-hoc approaches such as deactivating page duplication by default (Microsoft, 2017; Red Hat, 2017) as also cited by Gruss et al. (2017) can only stop certain Rowhammer attacks but not all Rowhammer attacks (Bosman et al., 2016). In addition, few hardware producers have taken steps to mitigate potential Rowhammer attacks against desktop and laptop machines by implementing ECC memories in these machines.

It has been argued (Fournaris et al., 2017; Lanteigne, 2016) that providing memories with ECC does not provide a better security as ECC cannot identify various bit flips on the same row so as to correct them. Likewise, it has also been contended (Kim et al., 2014) that the hardware-based solutions that take the approach of doubling the DRAM refresh rate cannot still prevent bit flips from being exploited, as this rate will require an upsurge of up to eight times to accomplish a capable mitigation. However, such arguments are in contradiction with the results of the tests that the researchers involved with the Google Project Zero (Seaborn and Dullien, 2015) performed on the desktop machines with ECC memory. According to the test results, no bit flips were detected on the desktop machines which had ECC memory implemented.

4. Recommendations

To counteract the discussed attacks, we suggest injecting dummy operations within AES algorithms, resulting in an upsurge in the runtime. With adequate dummy loads injected, it will be very difficult to determine whether a specific cache-hit or cache-miss is generated by genuine or false execution. Although injecting dummy operations will modify the runtime randomly, at the same time, it will be independent of cache activities. Note that this approach must be considered only in cases where attacks are carried out based on behaviour traces as opposed to the timing information. Reordering Memory Access can also be an effective countermeasure whereby the random reordering of memory access will decrease the connection among a captured behaviour trace (or runtime) and the input and algorithm. To achieve this, one would need to employ a non-deterministic processor architecture.

Efficient implementation of Advanced Encryption Standard (AES) algorithms in hardware can also be used as a countermeasure to safeguard modern hardware against CBSCAs. Few manufacturers have started implementing better hardware support in the design of their processor technologies to offer better constant-time cryptography operations. For instance, Intel has introduced AES New Instructions (AES NI), which is a new encryption instruction set that enhances the AES algorithm and speeds up the encryption of data in the two categories of Intel Xeon processor and the Intel Core processor. Thus, AES-NI provides an advantage in relation to speed over other implementations. Moreover, since AES-NI, which consists of seven new instructions, was specifically developed to be constant-time, it provides a better protection against SCAs over some other software implementations. The purpose of AES instructions is to alleviate all known timing and cache side channel emission of sensitive data from Ring 3 spy processes. Their latent period is data-independent, and because all the computations are carried out internally by the hardware, there will be no need for lookup tables. Thus, if AES instructions are utilised properly, the AES encryption/decryption and the Key Expansion will then have data-independent timing and involve only data-independent memory access.

As a result, the AES instructions will facilitate writing high performance AES software which is simultaneously safeguarded against the currently known software SCAs.

5. Discussion

In this study, we analysed Microarchitectural attack vectors, followed by a detailed examination of countermeasures that can be used to address these attacks. Through the findings of the study, we can deduce that despite the many advancements in computer security, SCAs continue to evolve and wreck a havoc on shared, modern computing hardware. Security researchers are simply playing a catch-up game with criminals as demonstrated by a wide range of sophisticated Microarchitectural attacks discussed in the literature. Everytime a new hardware vulnerability is discovered, a new mitigation research path also begins. As a result, security researchers have proposed several countermeasures and continue to suggest new ones. However, despite being useful, these suggestions are often restrictive or impractical. Furthermore, almost all of the existing solutions incur significant overheads or degradation in computing performance, which, in turn, weakens the optimisation of computing hardware such as processors. The level of such overheads and degradation in hardware performance, often caused by the methods, themselves, has not been adequately reported in the research papers. Thus, it is difficult to assess the effectiveness of such proposed countermeasures. It is also reasonable to state that an interaction between Microarchitectural side channels on one hand and inclination to optimise hardware performance via Microarchitectural improvements on the other hand have given rise to the absence of reliable defence mechanisms when producing hardware components. Adding to the stated issues, it is clear that the existing countermeasures for Microarchitectural attacks are not proactive but reactive in that they are there to mitigate single, specific attacks that are known to the research community. In addition, these countermeasures are not generic to be able to address future, unknown attacks that take advantage of different variations of Microarchitectural attacks.

6. Research Directions

From this work, it can be deduced that although there are a few studies that have proposed advanced countermeasures against the attack vectors discussed in this article, the effectiveness of these countermeasures are not fully known. Therefore, independent assessments of these countermeasures will need to be carried out to determine the efficacy of these defence mechanisms against a given attack vector, and to determine whether a mitigation mechanism can be bypassed or not. Hence, as a direction for future research, various exploitation techniques will need to be developed to establish the reliability of the proposed countermeasures present in the state-of-the-art, and then test them in a 'targeted' and 'predictable' manner within a dedicated laboratory. As future research direction, one can also conduct distinct studies with each focusing only one specific attack vector introduced in this paper. Each given study must carry out an independent assessment of one high-impact countermeasure (i.e. one that has been highly cited) previously proposed for each of the attack vectors. The purpose of each specific assessment will be to determine how effective the countermeasure against a given attack vector would be, and to determine whether or not that mitigation mechanism can be bypassed. To this end, exploitation techniques must be developed to establish whether the proposed countermeasures can be

undermined or not, and then tested in a ‘targeted’ and ‘predictable’ manner within a dedicated laboratory.

References

Aciimez, O. (2007). Yet Another Microarchitectural Attack: Exploiting I-Cache. Proceedings of the ACM Workshop on Computer Security Architecture, pp. 11-18.

Aciimez, O. and Koc, C. K. (2009). Microarchitectural Attacks and Countermeasures. In Cryptographic Engineering, pp. 475-504. Springer, Boston, US.

Agrawal, D., Baktir, S., Karakoyunlu, D., Rohatgi, P. and Sunar, B. (2007). Trojan Detection Using IC Fingerprinting. *IEEE Symposium on Security and Privacy (SP)*, pp. 296-310.

Apecechea, G.I., Inci, M.S., Eisenbarth, T. and Sunar, B. (2014). Fine Grain Cross-VM Attacks on Xen and VMware are possible!. IACR Cryptology ePrint Archive, p.248.

Ashokkumar, C., Giri, R.P. and Menezes, B. (2016). Highly Efficient Algorithms for AES Key Retrieval in Cache Access Attacks. *IEEE European Symposium on Security and Privacy (EuroS&P)*, pp. 261-275.

Atici, A.C., Yilmaz, C. and Savas, E. (2013). An Approach for Isolating the Sources of Information Leakage Exploited in Cache-Based Side-Channel Attacks. *7th IEEE International Conference on Software Security and Reliability-Companion (SERE-C)*, pp. 74-83.

Bernstein, D. (2005). Cache-timing attacks on AES. Available at: https://cr.yp.to/antiforgery/cachetiming-20050414.pdf (Accessed: 23rd May 2017).

Bhattacharya, S. and Mukhopadhyay. (2016). Curious Case of Rowhammer: Flipping Secret Exponent Bits Using Timing Analysis. In *International Conference on Cryptographic Hardware and Embedded Systems*, pp. 602-624. Springer, Berlin, Germany.

Blomer, J. and Krummel, V. (2007). Analysis of Countermeasures against Access Driven Cache Attacks on AES. *14th International Workshop on Selected Areas in Cryptography*, pp. 96-109.

Bluhm, M. and Gueron, S. (2015). Fast Software Implementation of Binary Elliptic Curve Cryptography. *Journal of Cryptographic Engineering*, 5(3), pp.215-226.

Bonneau, J. and Mironov, I. (2006). *Cache-Collision Timing Attacks against AES. International Workshop on Cryptographic Hardware and Embedded Systems*, pp. 201-215.

Bosman, E., Razavi, K., Bos, H. and Giurida, C. (2016). Dedup Est Machina: Memory Deduplication as an Advanced Exploitation Vector. *IEEE Symposium on Security and Privacy (SP)*, pp. 987-1004.

Brumley, B.B. (2015). Cache Storage Attacks. In Cryptographers' Track at the RSA Conference, pp. 22-34. Springer, Cham.

Chen, C., Wang, T., Kou, Y., Chen, X. and Li, X. (2013). Improvement of Trace-Driven I-Cache Timing Attack on the RSA Algorithm. *Journal of Systems and Software*, 86(1), pp. 100-107.

Chiappetta, M., Savas, E. and Yilmaz, C. (2016). Real Time Detection of Cache-Based Side-Channel Attacks Using Hardware Performance Counters. Applied Soft Computing, 49, pp.1162-1174.

Fournaris, A.P., Fraile, L.P. and Koufopavlou, O. (2017). Exploiting Hardware Vulnerabilities to Attack Embedded System Devices: A Survey of Potent Microarchitectural Attacks. Electronics, 6(3), p. 52.

Gallais, J.F., Kizhvatov, I. and Tunstall, M. (2010). Improved Trace-Driven Cache-Collision Attacks against Embedded AES Implementations. *WISA, 6513*, pp. 243-257.

Ge, Q., Yarom, Y., Cock, D. and Heiser, G. (2016). A Survey of Microarchitectural Timing Attacks and Countermeasures on Contemporary Hardware. Journal of Cryptographic Engineering, pp.1-27.

Genkin, D., Valenta, L. and Yarom, Y. (2017). May the Fourth Be with You: A Microarchitectural Side Channel Attack on Several Real-World Applications of Curve25519. ACM Conference on Computer and Communications Security (CCS), pp. 1-14.

Genkin, D., Shamir, A. and Tromer, E. (2014). RSA Key Extraction via Low-Bandwidth Acoustic Cryptanalysis. In *International Cryptology Conference*, pp. 444-461. Springer, Berlin, Germany.

GitHub. (2017). Test DRAM for Bit Flips Caused by the Rowhammer Problem. Available at: https://github.com/google/rowhammer-test (Accessed: 25th October 2017).

Gruss, D., Lipp, M., Schwarz, M., Genkin, D., Juffinger, J., O'Connell, S., Schoechl, W. and Yarom, Y. (2017). Another Flip in the Wall of Rowhammer Defenses. arXiv preprint arXiv:1710.00551, pp. 1-17.

Gruss, D., Maurice, C., Wagner, K. and Mangard, S. (2016). *Flush + Flush: A Fast and Stealthy Cache Attack. In Detection of Intrusions and Malware, and Vulnerability Assessment*, pp. 279-299. Springer International Publishing.

Gullasch, D., Bangerter, E. and Krenn, S. (2011). Cache Games-Bringing Access-Based Cache Attacks on AES to Practice. *IEEE Symposium on Security and Privacy (SP)*, pp. 490-505.

Hashizume, K., Rosado, D.G., Fernndez-Medina, E. and Fernandez, E.B. (2013). An Analysis of Security Issues for Cloud Computing. *Journal of Internet Services and Applications*, 4(1), p.5.

Hund, R., Willems, C. and Holz, T. (2013). Practical Timing Side Channel Attacks against Kernel Space ASLR. *IEEE Symposium on Security and Privacy (SP)*, pp. 191-205.

Inci, M.S., Gulmezoglu, B., Irazoqui, G., Eisenbarth, T. and Sunar, B. (2016). Cache Attacks Enable Bulk Key Recovery on the Cloud. In *International Conference on Cryptographic Hardware and Embedded Systems*, pp. 368-388. Springer, Berlin, Germany.

Irazoqui, G., Inci, M.S., Eisenbarth, T. and Sunar, B. (2014). Wait a Minute! A Fast, Cross-VM Attack on AES. *International Workshop on Recent Advances in Intrusion Detection*, pp. 299-319.

Irazoqui, G., Eisenbarth, T. and Sunar, B. (2015, a). S $ A: A Shared Cache Attack That Works Across Cores and Defies VM Sandboxing-and Its Application to AES. *IEEE Symposium on Security and Privacy (SP)*, pp.591-604.

Irazoqui, G., Inci, M.S., Eisenbarth, T. and Sunar, B. (2015, b). Know Thy Neighbor: Crypto Library Detection in Cloud. Proceedings on Privacy Enhancing Technologies, (1), pp.25-40.

Kim, Y., Daly, R., Kim, J., Fallin, C., Lee, J.H., Lee, D., Wilkerson, C., Lai, K. and Mutlu, O. (2014). Flipping Bits in Memory without Accessing Them: An Experimental Study of DRAM Disturbance Errors. *Proceedings of the 41st ACM/IEEE Annual International Symposium on Computer Architecture*, 42(3), pp. 361-372.

Kong, J., Aciicmez, O., Seifert, J.P. and Zhou, H. (2013). Architecting against Software Cache-Based Side-Channel Attacks. *IEEE Transactions on Computers*, 62(7), pp. 1276-1288.

Lauradoux, C. (2005). Collision Attacks on Processors with Cache and Countermeasures. *WEWoRC*, 5, pp.76-85.

Ma, C.G., Wang, D. and Zhao, S.D. (2014). Security Flaws in Two Improved Remote User Authentication Schemes Using Smart Cards. *International Journal of Communication Systems*, 27(10), pp. 2215-2227.

Maurice, C., Le Scouarnec, N., Neumann, C., Heen, O. and Francillon, A. (2015). Reverse Engineering Intel Last-Level Cache Complex Addressing Using Performance Counters. In *International Workshop on Recent Advances in Intrusion Detection*, pp. 48-65. Springer, Cham.

Microsoft Corporation. (2017). Data Deduplication Overview. Available at: https://technet.microsoft.com/en-us/library/hh831602(v=ws.11).aspx (Accessed: 25th November 2017).

Mirvaziri, H., Ismail, K.J. and Hanapi, Z.M. (2009). Message Based Random Variable Length Key Encryption Algorithm. *Journal of Computer Science*, 5(8), p.573.

Neve, M. and Seifert, J.P. (2006). Advances on Access-Driven Cache Attacks on AES. In Selected Areas in Cryptography, vol. 4356, pp. 147-162, Springer, Berlin, Germany.

Oren, Y., Kemerlis, V.P., Sethumadhavan, S. and Keromytis, A.D. (2015). The Spy in the Sandbox: Practical Cache Attacks in JavaScript and Their Implications. Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, pp. 1406-1418.

Osvik, D.A., Shamir, A. and Tromer, E. (2006). Cache Attacks and Countermeasures: The Case of AES. The RSA Conference Cryptographers' Track, pp. 1-20.

Page, D. (2005). Partitioned Cache Architecture as a Side-Channel Defence Mechanism. *IACR Cryptology ePrint Archive*, p. 280.

Percival, C. (2005) Cache Missing for Fun and Profit. Available at: http: //www.daemonology.net/papers/htt.pdf (Accessed: 19th June 2017).

Pessl, P., Gruss, D., Maurice, C., Schwarz, M. and Mangard, S. (2016). DRAMA: Exploiting DRAM Addressing for Cross-CPU Attacks. *Proceedings of the 25th USENIX Security Symposium*, pp. 565-581.

Schwarz, M., Weiser, S., Gruss, D., Maurice, C. and Mangard, S. (2017). Malware Guard Extension: Using SGX to Conceal Cache Attacks. arXiv preprint arXiv:1702.08719.

Seaborn, M. and Dullien, T. (2015). Exploiting the DRAM Rowhammer Bug to Gain Kernel Privileges. *Black Hat*, pp. 1-71.

Seibert, J., Okhravi, H. and Sderstrm, E. (2014). Information Leaks without Memory Disclosures: Remote Side Channel Attacks on Diversified Code. *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security*, pp. 54-65.

Spreitzer, R. and Plos, T. (2013, a). On the Applicability of Time-Driven Cache Attacks on Mobile Devices. In International Conference on Network and System Security, pp. 656-662. Springer Berlin Heidelberg.

Spreitzer, R. and Plos, T. (2013, b). Cache-Access Pattern Attack on Disaligned AES T-Tables. In *International Workshop on Constructive Side-Channel Analysis and Secure Design*, pp. 200-214. Springer, Berlin, Germany.

Spreitzer, R. and Grard, B. (2014). Towards More Practical Time-Driven Cache Attacks. In *IFIP International Workshop on Information Security Theory and Practice*, pp. 24-39. Springer, Berlin, Heidelberg.

Standaert, F.X., Malkin, T. and Yung, M. (2009). A Unified Framework for the Analysis of Side-Channel Key Recovery Attacks. Eurocrypt, 5479, pp. 443-461.

Tiri, K., Aciimez, O., Neve, M. and Andersen, F. (2007). An Analytical Model for Time-Driven Cache Attacks. In *International Workshop on Fast Software Encryption, pp. 399-413*. Springer, Berlin, Germany.

Van der Veen, V., Fratantonio, Y., Lindorfer, M., Gruss, D., Maurice, C., Vigna, G., Bos, H., Razavi, K. and Giuffrida, C. (2016). Drammer: Deterministic Rowhammer Attacks

on Mobile Platforms. Proceedings of the ACM SIGSAC Conference on Computer and Communications Security, pp. 1675-1689.

Wang, Z. and Lee, R.B. (2007). New Cache Designs for Thwarting Software Cache-Based Side Channel Attacks. *Proceedings of the 34th Annual ACM International Symposium on Computer Architecture*, 35(2), pp. 494-505.

Wu, Z., Xu, Z. and Wang, H. (2012). Whispers in the Hyper-space: High-speed Covert Channel Attacks in the Cloud. USENIX Security symposium, pp. 159-173.

Xia, W., Wen, Y., Foh, C. H., Niyato, D. and Xie, H. (2015). *A Survey on Software-Defined Networking*. IEEE Communications Surveys and Tutorials, 17(1), pp. 27-51.

Xiao, Y., Zhang, X., Zhang, Y. and Teodorescu, R. (2016). 'One Bit Flips, One Cloud Flops: Cross-VM Row Hammer Attacks and Privilege Escalation'. In *USENIX Security Symposium*, pp. 19-35.

Yarom, Y. and Falkner, K. (2014). FLUSH + RELOAD: A High Resolution, Low Noise, L3 Cache Side-Channel Attack. The Proceedings of the 23rd USENIX Security Symposium, pp. 719-732.

Yarom, Y. and Benger, N. (2014). Recovering OpenSSL ECDSA Nonces Using the FLUSH + RELOAD Cache Side-channel Attack. IACR Cryptology ePrint Archive, pp. 1-11.

Zhang, Y., Juels, A., Reiter, M. K. and Ristenpart, T. (2012, a). Cross-VM Side Channels and Their Use to Extract Private Keys. Proceedings of the ACM Conference on Computer and Communications Security, pp. 305-316.

Zhang, Y., Li, M., Bai, K., Yu, M. and Zang, W. (2012, b). Incentive Compatible Moving Target Defense against VM-Colocation Attacks in Clouds. *Information Security and Privacy Research*, pp. 388-399.

Zhang, Y., Juels, A., Reiter, M. K. and Ristenpart, T. (2014). Cross-Tenant Side-Channel Attacks in PaaS Clouds. *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security*, pp. 990-1003.

Zhang, T., Zhang, Y. and Lee, R.B. (2016, a). Cloudradar: A Real-Time Side-Channel Attack Detection System in Clouds. In I*nternational Symposium on Research in Attacks, Intrusions, and Defenses*, pp. 118-140. Springer International Publishing.

Zhang, L., Ding, A. A., Fei, Y. and Jiang, Z. H. (2016, b). Statistical Analysis for Access-Driven Cache Attacks Against AES. *IACR Cryptology ePrint Archive*, p. 970.

Zhang, T. and Lee, R.B. (2014). Secure Cache Modeling for Measuring Side-Channel Leakage. Technical Report, Princeton University, pp. 1-27.