Exploiting Hardware Performance Counters

Leif Uhsadel[†]
[†] K.U. Leuven,
Dept. ESAT/SCD-COSIC, IBBT
Kasteelpark Arenberg 10,
B-3001 Leuven-Heverlee, Belgium
firstname.lastname@esat.kuleuven.be

Andy Georges[‡]

Ingrid Verbauwhede[†]
[‡] Ghent University,
Dept. ELIS
St.-Pietersnieuwstraat 41
B-9000 Gent, Belgium
ageorges@elis.ugent.be

Abstract

We introduce the usage of hardware performance counters (HPCs) as a new method that allows very precise access to known side channels and also allows access to many new side channels. Many current architectures provide hardware performance counters, which allow the profiling of software during runtime. Though they allow detailed profiling they are noisy by their very nature; HPC hardware is not validated along with the rest of the microprocessor. They are meant to serve as a relative measure and are most commonly used for profiling software projects or operating systems. Furthermore they are only accessible in restricted mode and can only be accessed by the operating system. We discuss this security model and we show first implementation results, which confirm that HPCs can be used to profile relatively short sequences of instructions with high precision. We focus on cache profiling and confirm our results by rerunning a recently published time based cache attack in which we replaced the time profiling function by HPCs.

1 Introduction

Side-channel attacks are known since several years. Recently several side-channel attacks appeared, which exploit performance increasing features like caches of current desktop CPUs. In 1996 timing attacks were proposed by Kocher [19]. Many micro architectural side-channels have been accessed by measuring time since then. Typical examples are data caches [11, 12], instruction caches [8], and branch predictions [9, 10]. All of these side channels leak information because the according feature of the CPU is designed to improve performance and thus influences execution time. Not all attacks measure the execution time of the victim process directly. Some are remotely driven, some measure time of the adversary's process, that is running on the same machine, but they are all based on measuring elapsed time.

Besides time there are also approaches to access these side channels by measuring power consumption [20, 16]. Power consumption measurement has the advantage that it can display more leaked information, because it collects data during the entire execution of the cipher. A significant drawback is that the adversary needs physical access to the hardware.

In this paper we examine the use of micro architectural side-channels by using hardware performance counters. These are available on current CPUs, e.g., Intel Pentium and AMD Athlon chips, and they are intended to be used to profile software in order to allow for performance improvement. Events that can be monitored are for example branch prediction, data and instruction cache hits and misses (see Section 2 for more details). However, in terms of cryptographic security there are also threats linked with HPCs, because any event or combination of events that is supported by them can be used as a side channel.

All current x86 CPUs are equipped with HPCs and they can also be found in Cray, PowerPC, Ultrasparc and MIPS architectures. The events that can actually be measured heavily depend on the CPU. The fact that HPCs are used to log detailed information about how the microprocessor components features of the CPU are used at runtime makes them an extremely dangerous side-channel. Their usage can improve many existing side channel attacks or even enable new attacks on many platforms. Furthermore, they can also be used to profile a process continuously, which should result in attacks equally powerful as power analysis attacks. Potentially, HPCs can increase the measurement accuracy of a given information leak, because they can be used to focus on a specific CPU event, while not being polluted by other events. This has been mentioned in [21], but their work was restricted to using data cache-misses. This option seems even more powerful when taking into account that usually several HPCs are located in a single CPU allow-

ing to profile a process in multiple ways at the same time. To the best of our knowledge HPCs have not been used for cryptanalysis yet.

In this paper, we demonstrate that HPCs form a feasible side-channel and discuss their advantages, disadvantages and general properties. The remainder of this paper is organized as follows: In Section 2 we discuss HPCs in general. In Section 3 we describe the setup for our measurements in Section 4 and 5. Further on, in Section 4, we estimate the precision of cache-miss profiling with HPCs. Subsequently we will confirm these results by testing our setup with a known cache collision attack [1]. This attack is designed as a time based cache attack and will not use the full potential of HPCs, yet it shows that the HPC side-channel is a real threat and offers great potential for attacks.

2 Performance Counters

In this section we briefly discuss general properties of HPCs and illustrate one specific example using HPCs on the AMD Athlon. It is beyond the scope of this paper to provide details on HPCs for all platforms.

Essentially, hardware performance counters are a tool that is used by software engineers for measuring performance and for allowing software vendors to enhance their code such that performance improves. HPCs are implemented on the CPU die, and have both a control and a counting register. On the AMD Athlon, HPC counting registers are 40 bits wide. Through the control registers they can be used to count a large number of performance events, such as cache accesses, misses, TLB[2] misses, stalls in various components of the chip, etc. Each time the programmed event occurs, the count register is incremented. On most contemporary platforms, the control registers can be programmed to throw a non-maskable interrupt if the corresponding counting register overflows, i.e., when its value reaches 0. To allow this, the counting registers can be set up to start with any value, including negative ones (they count up). The interrupt can then be handled immediately in software, allowing for example to increase performance by steering compilation decisions in a virtual machine [13]. At run time, the counters can be accessed, read and changed if necessary. More information regarding the implementation of HPCs on the AMD Athlon can be found in [1].

As we mentioned earlier, the events that can be counted using HPCs depends on the hardware platform. For example, on the AMD Athlon, there are over 50 events which can be classified into several classes: events related to data cache usage, events related to branch prediction, events related to the processor front-end (e.g., instruction cache fetches, misses, instruction TLB hits and misses), the processor core (stalls), and more general events such as cycles burned, retired instructions, etc. Several of these events (data and instruction cache, branch prediction) are quite interesting to exploit with respect to known side-channel attacks. Recall that the number of events that leak information and thus can be exploited is quite high, leading us to believe that research evaluating possible attacks using the HPCs on various platforms is imperative to gain in-depth understanding of the danger that can be associated with them.

	kernel mode	user mode	both
global	√	√	√
virtual	√	√	√

Table 1: Different contexts in which the HPCs can count events.

Other than the system cycle counter or the time stamp counter[3], the HPCs belong to ring-0 in the ring security model [18], which means that they can only be programmed by the kernel. Depending on the instruction set architecture (ISA), it is possible that the HPC registers can be read from user-space. For example, the IA-32 ISA [2] offers the RDTSC and RDPMC instructions for reading the time stamp counter and the performance monitoring counters, respectively. It is however possible that such access is also restricted to the OS.

In many cases, to program the performance counting hardware, the kernel has to offer a driver that assumes control of the HPCs. Usually patching the kernel requires root privileges, yet a driver may be already present in the kernel. For example, Linux kernels from 2.6 on come standard with the Oprofile tool [5]. Additionally, depending on the privileges set by the OS, a regular user may or might not be able to access the driver. For example, on Linux, programming the HPCs requires write access to the device file corresponding to the driver or module that controls them. This means that an adversary who can gain sufficient privileges can set up the HPCs in a way that allows him to exploit the side channels presented in this paper.

Once activated, HPCs are incremented globally, irrespective of the process that has control of the CPU. It is possible to make a distinction between events that occur when the CPU operates at privilege level 0 (when the kernel is executing, e.g., on behalf of the process), or when it operates at a lesser privilege level. One can count either one or the other, or both at once. However, using the process context data structures in the kernel, it is possible to isolate

[1] We are using a cache collision attack based on time measurement present by Bonneau et al. [11]

[2] TLB stands for translation lookaside buffer; this is a structure in which virtual to physical address translation are cached.

[3] The time stamp counter is incremented each clock cycle occurring in the CPU, irrespective of any HPCs that are programmed.

the counts for one or more specific processes. To achieve this, the kernel must be patched, such that upon each context switch, the HPCs are read and their values are accumulated with the total events counts that are stored in the process context. Usually, a process will count events on behalf of itself, but a malicious attacker can set up the execution environment – for example, using the LD_PRELOAD variable in a Linux shell – such that a precompiled library is loaded prior to loading the application and event counts can be communicated back to the attacker. Of course, the reading of counter information at context switched perturbates the data somewhat, thus adding noise to the measurement. However, a virtual counter avoids the noise that is created by other processes running on the system, which potentially have a much larger impact on the event counts. Table 1 gives an overview of the different contexts in which an event can be counted.

By definition, HPCs are not exact [1]. Their hardware circuits are not validated like the rest of the CPU, and they are not part of the critical execution paths in the microprocessor. Consequently, sometimes an event does not get counted, although it should have been or an event is counted when it did not occur. Fortunately these cases are rare, and generally one can ignore them if the event counts are sufficiently large. However, for our purposes, we are dealing with small event counts, making this *natural* noise all the more bothersome. Additionally, accessing the software to read the counters may also cause an event to occur, further perturbing the results. The impact of these different sources of noise likely also depends on the event under observation, the architecture of the system, as well as the used driver and an optional used application programming interface (API). In the next section 4, we explore the perturbation for cache events on the AMD Athlon.

3 Experimental Setup

We have used two systems in our experiments, all belonging to the x86 family. Table 2 gives an overview of the machines we have used. To determine the relevant CPU information, such as cache line size, we use the x86info tool [7]. On each machine we use the Linux operating system, albeit different versions on different machines.

To access the HPC hardware, we have patched the kernel with Perfctr [3]. On top of that driver, PerfAPI [6] offers a portable user-level library that eases access to the counter infrastructure, but which also gives some extra overhead when reading the counter values.

The Perfctr patch allows the accumulation of counter events at each OS context switch, effectively counting the events that occur for a single process in the system. However, global counting remains possible.

Our main test system is the AMD Athlon Model 6 with a 64-Kb, 2-way associative L1 data cache and a 256-Kb, 8-way associative L2 data cache, see also Table 2. We verified our results on the second machine, an AMD Athlon XP model 10.

component	information
AMD Athlon XP	model 6
clock speed	1.6Ghz
L1 D-cache	64KB, 2-way set-associative, 64-byte line size
L2 cache	256KB, exclusive, 8-way set-associative, 64-byte line size, used for both instructions and data
operating system	Linux 2.6.22.9
HPC driver	perfctr 2.6.30
AMD Athlon XP	model 10
clock speed	2.1Ghz
L1 D-cache	64KB, 2-way set-associative, 64-byte line size
L2 cache	512K, exclusive, 8-way set-associative, 64-byte line size, used for both instructions and data
operating system	Linux 2.6.18
HPC driver	perfctr 2.6.25

Table 2: The platforms we used in our experiments.

4 Exploiting Cache Events of HPCs

As mentioned before, HPCs have two sources of noise by definition: a design dependent noise component and noise caused by accessing the HPCs. To determine the noise impact, we run several tests and compare the result with the conventionally used system cycle counter or time stamp counter (tsc). The HPC events used are L1 data cache-misses (l1dcm), L2 data cache-misses (l2dcm), and the HPC cycle counter (cycles).

To eliminate other sources of noise (other programs running, context switches etc.) as much as possible we use virtual counters. Recall that global usage of HPCs would require a context switch to read values from the HPC registers and would therefore add noise depending on the operating system.

To reduce noise of other instructions than those used in the lookup, we avoid loops, branches etc. within the measurement as much as possible. Furthermore we switch off compiler optimizations to make sure the table lookups really occur in the way we expect it to. Due to the fact that the

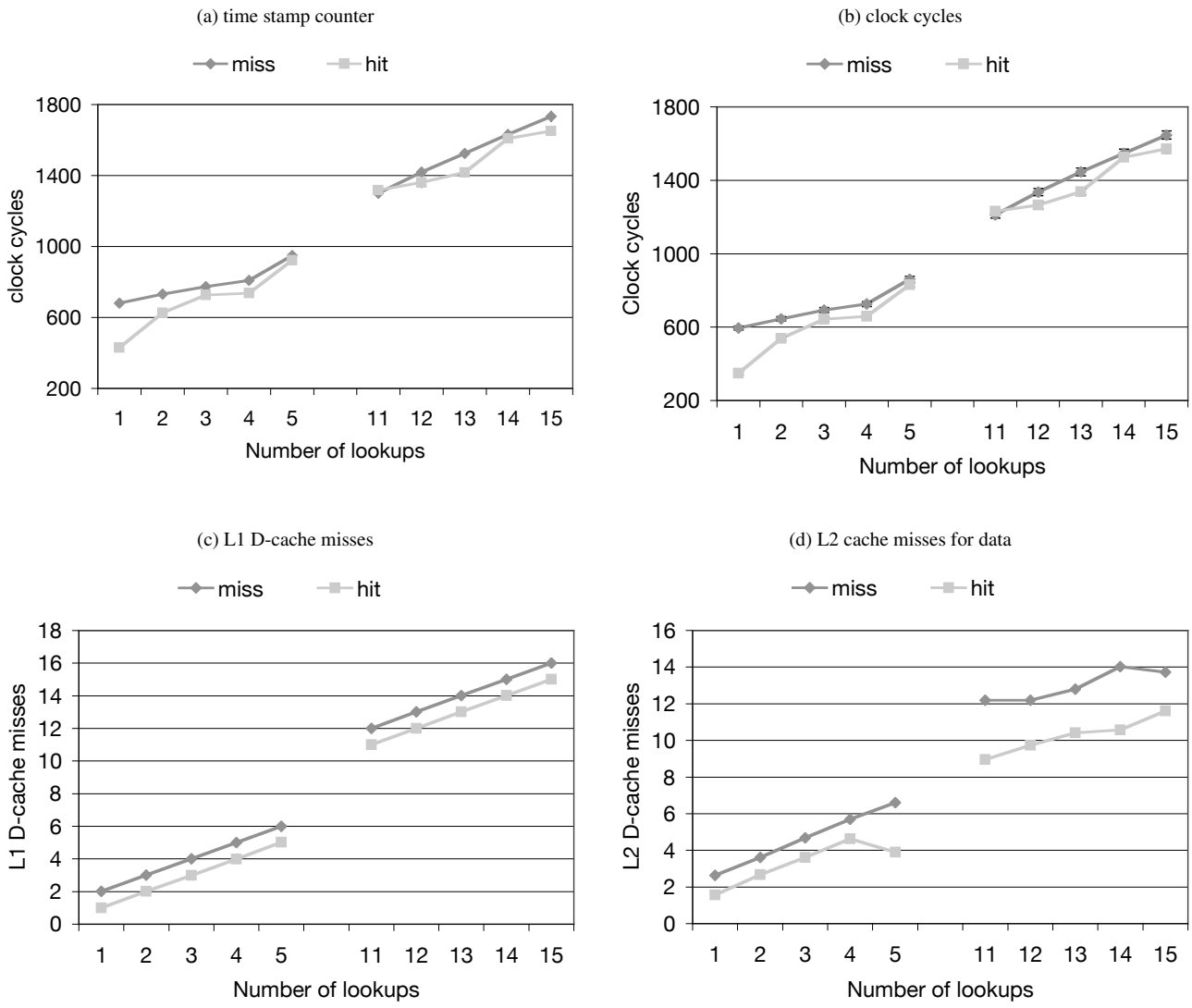


Figure 1: Noise for the various side channels used in our experiments as measured on the AMD Athlon model 6 using Perfctr: (a) TSC, (b) cycle count event, (c) L1 D-cache misses, (d) L2 D-cache misses. For each lookup number (on the X-axis), the graphs show 95% confidence intervals, but these are quite tight.

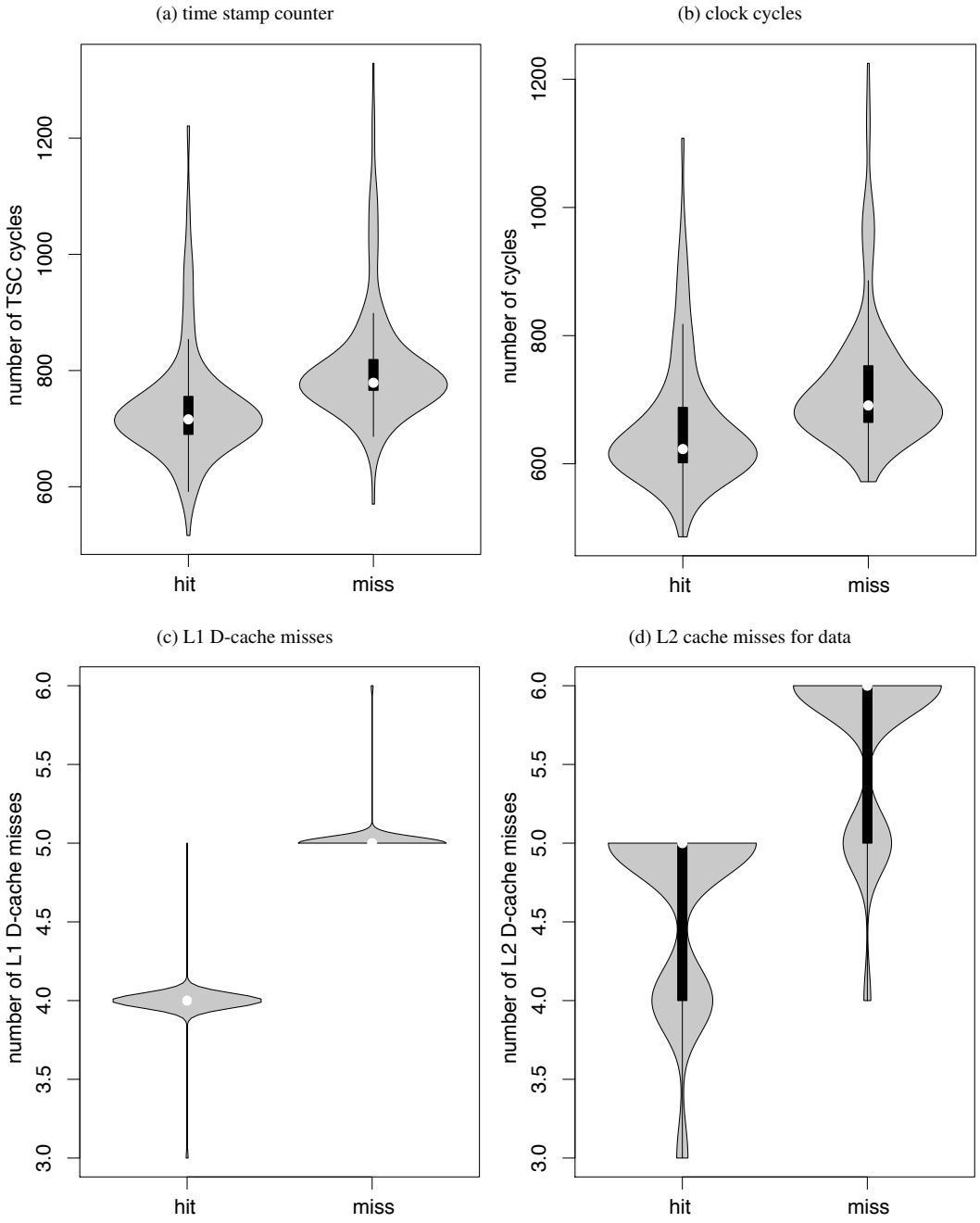


Figure 2: Violin plots of the event counts for the various side channels when doing 4 lookups as measured on the AMD Athlon model 6 using Perfctr: (a) TSC, (b) cycle count event, (c) L1 D-cache misses, (d) L2 D-cache misses. We show the results for the *hit* and *miss* experiments.

tsc also counts when the kernel is executing on behalf of the process, this metric incurs more noise than the other metrics that are programmed to only count in user mode. The HPCs in virtual mode have the advantage that they are independent of the general system noise. This is interesting for an adversary, because the noise caused by other programs running is hard to control in general.

We run several tests to clearly illustrate the quality of different profilings: First we execute a specific amount of table lookups and then log the estimated number of clock cycles, cache-misses etc. We test each side channel in a separate experiment. This means there is no information spill between runs, as the caches, TLBs, etc. are cleared when a new process gets the CPU. For each experiment we assemble a sufficient number of samples such that the power of the test [14] when comparing results in the presence of a hitting first lookup is 0.9 with an $\alpha = 0.05$ significance level in the presence of small effects. It turns out that taking 235 samples in this case suffices.

For this experiment, we use the AES [15] s-box tables used in the last AES round as implemented in the SSH library. Given that this table has a size of 1KB and consists of 256 entries of four-byte elements each single lookup fetches $f = 256/\text{linesize}$ consecutive elements at once. This means that after $256/f$ misses, the entire table is stored in the cache. During our experiments each lookup occurs at a distance of 16 to the prior lookup. This means that on the AMD model 6 for example, we only measure cache misses.

We are mainly interested in measuring cache events that allow for making a distinction between hits and misses. Therefore, we perform two analogous experiments. In the first experiment – the *miss* experiment – we simply perform all the lookups in the table and measure how many cache misses occur. In the second experiment – the *hit* experiment – we perform one extra lookup of the first element in the S-Box before the measurement starts. This means that the cache line containing that particular element should be present in the cache during the measurement, i.e., what would be a miss in the first experiment now becomes a hit. Figure 1 shows the resulting event count for both experiments for the time stamp counter, cycles, L1 and L2 data cache misses on the AMD model 6. In each graph, the X-axis shows the number of lookups performed, while the Y-axis gives the corresponding event count. For each lookup number, the experiments were repeated 235 times. The graphs show the mean value of these 235 samples, along with a 95% confidence interval. Note that these intervals are quite tight, indicating there is little variability across the measurements. Figure 2 shows violin plots [17] for the noise when 4 lookups are done for each of these side channels. In addition to the information conveyed in a regular box plot, the shape of a violin plot represents the density. The middle point shows the median; the thick vertical line represents the first and third quartiles (50% of all the data points are between the first and third quartile); the thin vertical line represents the upper and lower adjacent values (representing an extreme data point or 1.5 times the interquartile range from the median, whichever comes first); and the top and bottom values show the maximum and minimum values. There is some variability, in the TSC and cycles side channels, yet we see that both the time stamp counter and the cycle event counts are lower in the *hit* experiment, which is to be expected. We also expect there to be a single cache miss less in the *hit* experiment. To verify this, we conduct a one-sided Student t-test, with null hypothesis $H_0 \equiv \mu_{\text{miss}} - \mu_{\text{hit}} = 1$. Similarly, we expect the same to be true for the L2 cache misses.

Table 3 shows the resulting p-value, which indicates if the null hypothesis H_0 should be rejected – this is the case if $p < 0.05$ at a 0.05 significance.

The results indicate that both the TSC and the cycle count are significantly larger at the 95% confidence level for the *miss* experiment than for the *hit* experiment, except in the case of 11 lookups for both event types and in the case of 14 lookups for the cycle event. In all cases, except for 4 lookups, the difference in L1 data cache miss events seems to be a single event. Strangely, the same does not hold for L2 data cache misses. For 5 or more lookups, there is significant evidence to support the conclusion that there is more than a single L2 cache miss more in case of the *miss* experiment. Thus, for L2 misses, there seems to be additional noise perturbing the results. We can conclude that the L1 data cache miss event is the most accurate event we used in our experiments. In the next section, we will use all four event types in an actual attack. Based on the evidence presented above, we expect the L1 data cache misses to help us break the key fastest.

5 Evaluating Hardware Performance Counter in a Real Attack

The attack we use in this experiment is based on the Bonneau's attack [11], where we essentially replace the time measuring function. The changes made to the attack implementation do not affect the attack itself; they are simply needed for setting up the HPCs and reading their values at the appropriate moment.

In their paper, Bonneau et al. have introduced several attacks based on measuring the encryption time that varies due to cache misses and hits. We use the last round attack, as this seems to be the most reliable in our environment. Note that this attack only has been conducted on a Pentium 3 platform, and not on an AMD platform. The purpose of using this attack is to compare the effectiveness of using different side channels.

The attack is aimed at the AES implementation of Open

lookups	TSC	cycles	L1 D-cache misses	L2 D-cache misses
1	$1.01\ 10^{-119}$	$6.03\ 10^{-121}$	0.08	0.40
2	$4.49\ 10^{-36}$	$3.29\ 10^{-38}$	0.21	0.74
3	$1.49\ 10^{-06}$	$1.01\ 10^{-06}$	0.13	0.07
4	$2.31\ 10^{-13}$	$1.12\ 10^{-10}$	0.03	0.13
5	$1.34\ 10^{-02}$	$5.64\ 10^{-03}$	0.91	$3.07\ 10^{-102}$
11	0.875	0.896	0.84	$3.35\ 10^{-82}$
12	$5.74\ 10^{-03}$	$3.45\ 10^{-05}$	0.35	$1.23\ 10^{-54}$
13	$4.88\ 10^{-12}$	$1.59\ 10^{-11}$	0.84	$1.94\ 10^{-44}$
14	0.04	0.0586	0.16	$6.68\ 10^{-90}$
15	$2.48\ 10^{-05}$	$2.87\ 10^{-05}$	0.24	$2.16\ 10^{-21}$

Table 3: Results of a one-sided Student t-test with 95% confidence level for comparing the *hit* and *miss* experiments on the AMD model 6. p-values smaller than 0.05 indicate the null hypothesis should be rejected in favor of the alternative hypothesis that the *miss* event count is significantly larger than the *hit* event count. For the cache misses the expected difference is a single event. For the cycle event, we simply check the equality of the means.

SSL v. 0.9.8.(a) [4]. Like many fast AES implementations large tables are used to speed up the algorithm. For the last round a unique table is used, which is not used in the prior rounds and thus is not present in the cache at the beginning of the last round. A collision in this table reveals information about the used key. A cache hit results in a shorter execution time and indicates a collision.

In phase one the attack takes a large amount of samples that consist of the cipher text of a single AES encryption and the according execution time. Before each encryption the cache is evicted so the tables are not present in the cache.

In phase two of the attack a number of samples is used for a statistical analysis and a key guess is made. The number of used samples is constantly increased until the correct key is found. A more detailed description of the attack can be found in [11].

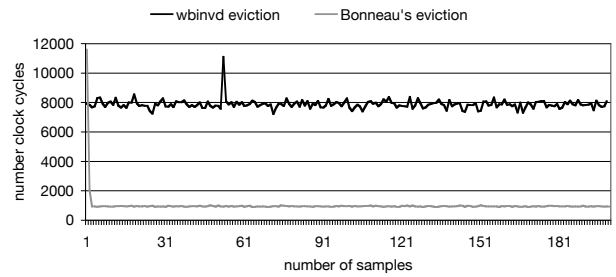
We expect that if the measurement technique can reveal a cache hit more precise, the amount of samples required to reveal the key is less.

In the original Bonneau implementation, the cached table entries were invalidated by reading data from a second (large) table, thus overwriting the data in the cache. Different routines for PIII, PIV, as well as L1 cache and L2 cache are provided. We carefully adapted this method to our platform by changing the according parameters like cache size, line size, or associativity to fit our CPU. However, we found that using this approach the number of L2 misses drops to 0, indicating that the cache is not properly evicted: we measure a lot of misses during the first encryption, but little or none after several encryptions. This is shown in Figure 3. The number of instructions executed for each test remains fairly constant at 1242.71 on average with a standard deviation of 0.15. However, the cycles, the number of L1 D-cache misses and the number of L2 D-cache misses drops down after the first few samples were gathered. The latter event drops down to 0, showing that the eviction strategy as implemented by Bonneau is not clearing our L2 cache at all, while the L1 cache is partially cleared.

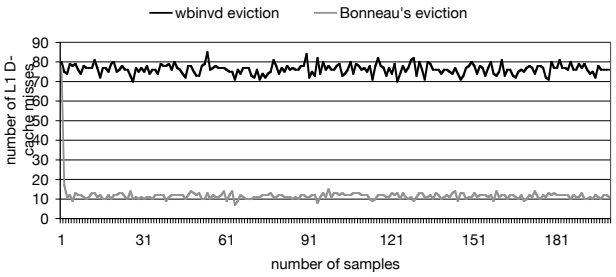
To address this issue, we use an alternative approach, that offers more certainty of actually invalidating both the L1 and L2 caches. In the IA-32 ISA (both for the Intel and AMD CPUs, the WBINVD instruction writes back the dirty cache lines to main memory before invalidating all cache lines of the internal caches. Following that, a bus cycle is started to signal the external cache to also write back their data and invalidate their lines. Because this instruction operates at privilege level 0, we wrote a kernel module that executes the WBINVD instruction when the corresponding device is written to.

We run the attack 50 times for each side channel. Table 4 lists the average amount of samples required to recover the key. In Bonneau's implementation, the RDTSC instruction is used for measuring the (global) time stamp counter values before and after the last encryption round. We provide additional results when measuring (virtual) TSC values as well as the (virtual) HPC cycle event. The former counts kernel cycles, the latter does not. Furthermore the table shows results for measurements with the (virtual) HPC L1 and L2 D-cache miss counter. The table shows that the L1 HPC performs best, as expected. The virtual cycle counters give very similar results and the they perform equal or worse than the global counter. It might be, due to the short execution time of AES, that the overhead virtualising the counter is bigger than than the gain in accuracy. Since the L1 and L2 D-cache miss counter ignore these extra burned cycles, they are not affected. In this case the virtual use of the cycle counters should show better performance when used to attack algorithms with sufficient long execution time. As

(a) cycles



(b) L1 D-cache misses



(c) L2 D-cache misses

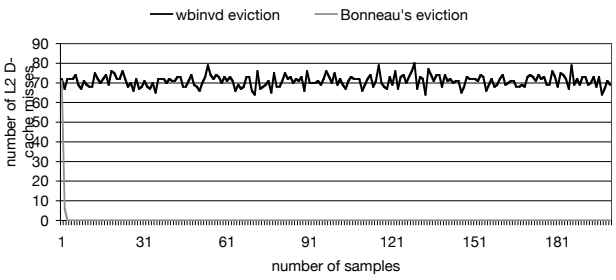


Figure 3: Comparing the number of events seen using Bonneau's eviction strategy and the WBINVD instruction strategy we implemented on the AMD Athlon model 6 for (a) cycles, (b) L1 D-cache misses and (c) L2 D-cache misses.

explained earlier in this section the L2 cache is only evicted by the WBINVD, therefore the attack only terminate for this clearing strategy. Surprisingly in this case the attack is not terminating with the RDTSC counter, yet we see that the L2 D-cache miss counter performs better than the two virtual cycle counters.

The table also shows a great impact of the different cache eviction strategies on the required amount of samples for the attack. The L2 cache eviction strategy for the PIII performs best on our AMD platform, while the WBINVD method performs worst, though it is the only one completely clearing the cache.

6 Conclusion and future work

We introduced HPCs as side-channel for multiple CPU depends information leaks. We furthermore explored its power for the case of cache-misses on AMD platforms and showed in several tests that HPCs are a side-channel of great potential. We furthermore convinced these results in a time based cache attack.

For future work these experiments should be widened to multiple platforms exploring multiple events. In doing so also events should be evaluated that access information that has not yet been used for side channel attacks.

7 Acknowledgement

We would like to thank the anonymous reviewers for their comments and suggestions. We are also grateful for support by Benedikt Gierlichs and Joris Plessers.

This work was supported in part by the IAP Programme P6/26 BCRYPT of the Belgian State (Belgian Science Policy), by the FWO project G.0300.07, and by the K.U. Leuven-BOF. Andy Georges is a post-doctoral researcher at Ghent University.

References

[1] AMD Athlon Processor Code Optimization Guide Appendix D Performance-Monitoring Counters. http://www.amd.com/us-en/assets/content_type/white-papers-and-tech-docs/22007.pdf.

[2] Intel 64 and IA-32 Architectures Software Developers Manual. download.intel.com/design/processor/manuals/253666.pdf.

[3] Linux Performance Counters Driver. http://sourceforge.net/projects/perfctr/.

[4] OpenSSL. http://www.openssl.org/.

[5] OProfile - A System Profiler for Linux. http://oprofile.sourceforge.net.

[6] Performance Application Programming Interface. http://icl.cs.utk.edu/papi/.

Cache eviction method	RDTSC	TSC	Cycles	L1	L2
L2 evict on PIII	14,8	13,6	13,2	13,6	n/a
L1 evict on PIII	45,7	58,7	57,3	33,7	n/a
L1 evict on PIV	45,3	60,0	60,4	34,1	n/a
WBINVD	n/a	156,4	244,2	35,6	76,1
L2 evict on PIV	not possible	not possible	not possible	not possible	not possible

Table 4: Required samples in thousand for Bonneau's attack using different cache evictions and side channels. The L2 cache evict for PIV uses the CFLUSH instruction from the instruction set extension of the PIV, which is not available on AMD. Those combinations of parameters, than did not terminate with 1M of samples are marked n/a.

[7] x86info, a CPU Identification Utility. `http://www.codemonkey.org.uk/projects/x86info/`.

[8] O. Aciiçmez. Yet another MicroArchitectural Attack:: exploiting I-Cache. In *CSAW '07: Proceedings of the 2007 ACM workshop on Computer security architecture*, pages 11–18, New York, NY, USA, 2007. ACM.

[9] O. Aciiçmez, Çetin Kaya Koç, and J.-P. Seifert. On the power of simple branch prediction analysis. In *ASIACCS '07: Proceedings of the 2nd ACM symposium on Information, computer and communications security*, pages 312–320, New York, NY, USA, 2007. ACM.

[10] O. Aciiçmez, Çetin Kaya Koç, and J.-P. Seifert. Predicting Secret Keys Via Branch Prediction. In M. Abe, editor, *CT-RSA*, volume 4377 of *Lecture Notes in Computer Science*, pages 225–242. Springer, 2007.

[11] J. Bonneau and I. Mironov. Cache-Collision Timing Attacks Against AES. In L. Goubin and M. Matsui, editors, *CHES*, volume 4249 of *Lecture Notes in Computer Science*, pages 201–215. Springer, 2006.

[12] D. Brumley and D. Boneh. Remote timing attacks are practical. *Comput. Netw.*, 48(5):701–716, 2005.

[13] D. Buytaert, A. Georges, M. Hind, M. Arnold, L. Eeckhout, and K. D. Bosschere. Using hpm-sampling to drive dynamic compilation. In *OOPSLA '07: Proceedings of the 22nd annual ACM SIGPLAN conference on Object oriented programming systems and applications*, pages 553–568, New York, NY, USA, 2007. ACM.

[14] J. Cohen. *Statistical power analysis for the behavioral sciences*. Psychology Press, 1988.

[15] J. Daemen and V. Rijmen. AES Proposal: Rijndael. NIST AES Homepage: `http://csrc.nist.gov/encryption/aes/round2/r2algs.htm`, 1999.

[16] P. Grabher, J. Groschdl, and D. Page. Cryptographic Side-Channels from Low-Power Cache Memory. In *Cryptography and Coding, volume 4887 of Lecture Notes in Computer Science*, pages 170–184. Springer Verlag, December 2007.

[17] J. Hintze and R. Nelson. Violin plots: A box plot-density trace synergism. *The American Statistician*, 52(2):181–184, 5 1998.

[18] P. A. Karger and A. J. Herbert. An Augmented Capability Architecture to Support Lattice Security and Traceability of Access. In *Proceedings of the 1984 IEEE Symposium on Security and Privacy*, April 1984.

[19] P. C. Kocher. Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems. In *CRYPTO '96: Proceedings of the 16th Annual International Cryptology Conference on Advances in Cryptology*, pages 104–113, London, UK, 1996. Springer-Verlag.

[20] Onur Aciiçmez and Çetin Kaya Koç. Trace-Driven Cache Attacks on AES. In P. Ning, S. Qing, and N. Li, editors, *ICICS*, volume 4307 of *Lecture Notes in Computer Science*, pages 112–121. Springer, 2006.

[21] K. Tiri. Side-Channel Attack Pitfalls. In *DAC*, pages 15–20. IEEE, 2007.