CACHEOMS: Practical Exploitation of **Cache** Side Channel **O**n **M**ultiprocessor **S**ystems

Aravind Machiry & Varun Kulkarni Somashekhar

Abstract—**Cache side channel is well known attack on cryptographic implementations[1][2][3]. This is primarily based on observation that infrequently used information incurs a large timing penalty, thus revealing some information about the frequency of use of the memory blocks. This attack is straight forward on a single processor system with single level of cache, but current systems (both desktop and mobile) are multiprocessor where each processor has multilevel caches also there are some cache levels which are common to a subset of processors. Moreover, latest ARM processors have TrustZone support[4] which ensures System-On-Chip (SOC) wide security by avoiding shared cache access. In addition to this, there has been lot of work done both on Hardware and Software side to mitigate cache side channel[5][6]. It is high time we revisit and evaluate cache side channel, its possibility on latest processors and implications[7].**

I. Introduction

Memory access time is the main bottleneck for achieving maximum processor performance. Memory hierarchies are used to bridge this gap. Memory cache is the main component of the hierarchy. Most of the Processors have memory caches whose access time is several times less than main memory. On latest Intel I7 processor, access to L1 cache takes 2 cycles, where as it takes 360-400 cycles to access a byte from main memory. Memory Caches are continuously evolving and lot of research have been done to make these as fast as the corresponding processor. On the other hand, main memory access time is not decreasing accordingly. This makes an access to main memory to be easily distinguishable by measuring access time. The difference in the access time is the main idea based on which most of the cache side channel attacks are discovered. Recent improvements in cache technologies seems to decrease cache access time thus making main memory access extremely easy to detect.

All known attacks are implemented on previous generation processors like Pentium III (cite from challenges), Pentium 4E (cite from challenges) and Athlon 64 (cite from challenges). Although, recent improvements in caches make cache side attacks more reliable, multi-core processors with multiple levels of cache makes exploiting cache side-channels non trivial. Moreover, recent processors have pre-fetching[8] which thwarts most of the known attacks. In this project, we explore the feasibility cache timing attacks on latest multiprocessor based systems, discuss our findings and results. Specifically, following are our contributions:

Authors are with the Department of Computer Science, University of California, Santa Barbara, CA 93106. E-mail: {machiry,varun_kulkarni}@cs.ucsb.edu

- Investigate, understand and report cache side channel with examples. We aim to make the documentation simple enough to be understood by a computer science graduate without much knowledge of cryptography or computer architecture.
- Implement cache timing attack on Latest Intel i7-930 processor and report our findings.
- Research on the recent improvements and state of art on cache side channel and clearly report the findings.
- Evaluate the applicability of these attacks on well known protocols on current multiprocessor systems, both on x86 and ARM.
- Investigate and report existing defense strategies to prevent cache side channel.

II. Background

In this section we give a brief overview of the background needed to understand the attack presented in this work. After summarizing the design of cache architecture, a short explanation about different types of cache attacks are provided.

A. Cache Design and Operation

A cache is a small memory placed between the CPU and RAM to reduce the big latency added by retrieval of data. Modern processors usually have more than one level of cache to improve the efficiency of memory accesss.Most modern processes offer 3 levels of cache.with the first level (L1) being the closest to the CPU registers. Generally, L1 and L2 caches are each split into instruction caches and separate data caches. L3 usually stores both instructions and data.The cache size is much smaller than the number of directly addressable bytes in main memory. Hence a mapping strategy needs to be adopted.The cache associativity determines how the main memory blocks map into blocks of the cache.

Varun: Put some picture of cache operation, explain associativity, cache replacement policy(invalidate, dirty bit)

When a memory reference is made by the CPU, the tag address of the main memory block is compared with all the tags in the corresponding set. If the tag is found then this reference qualifies as a cache hit. Meaning that there is no need to retrieve the data from main memory since it is already located in the cache, and the data can be immediately provided to the CPU. Conversely, if the tag is not found in the corresponding set then the memory reference qualifies as a cache miss.The caching mechanism operation is based on the principals of spatial and temporal locality, which help to minimize the number of cache misses. Temporal locality states that the same data blocks will likely be requested repeatedly during the execution of a process. Spacial locality states that data blocks from nearby addresses are likely to be subsequently accessed. Even though the number of cache misses is reduced by these principles, they are not eliminated.

Obviously, data in the cache can be accessed much faster than data present only in memory.This is also true for multi-level caches where data accessed from the L1 cache will experience lower latencies, than data accessed from subsequent cache levels. These time differences are used to decide whether a specific portion of memory resides in the cache - implying that the corresponding data has been accessed recently. This resulting information leakage stemming from microarchitectural time differences when data is retrieved from cache rather than memory forms the basis of cache side channel attacks.

B. Types of Cache Side Channel Attacks

Cache misses occur at every cache level and they are categorized as:

• **cold start misses**, which occur when data is first referenced;

• **capacity misses**, which occur if the size of the LUTs Look Up Table used by the cipher is larger than the size of the cache; this type of cache misses do not occur for most cryptographic cipher implementations.

• **conflict misses**, which occur when at least two elements of LUTs that map to the same cache block are used.

Based on this idea, cache attacks can be classified into three groups:

• **Reset attacks** require that all or most LUTs used by the cryptographic cipher are not to be loaded in the cache before the attack commences. Therefore this type of attacks is mainly based on cold start misses.

• **Initialization attacks** require the adversary to be able to set the cache into a known state before the attack commences. Therefore this type of attacks is based both on cold start misses and conflict misses.

• **Micro-architecture attacks** require the cache to hold all or most LUTs that will be used by the cipher,before the attack commences. Therefore this type of attacks is partially based only on conflict misses and partially on other timing penalties that strongly depend on the CPU microarchitecture.

These three types of attacks are not limited to timing channels. Measuring power dissipation levels during the encryption process also offers more information about data access. However, this work would focus only on timing attacks.

III. Attack Implementation

In this section, we explain our implementation of an attack based on cache timing on Intel i7-930 processor. First, we describe the cache organization on this processor. Second, we explain our attack goal and strategy. Finally, our implementation along with challenges faced and results.

A. Cache Organization

Processor Intel i7-930 has 4 physical cores and has Hyper-threading support[9], which exposes 8 (4 * 2) virtual cores. Cache on the processor has 3 levels, which is shared among these 8 virtual cores. Organization of various cache levels and corresponding properties is as shown in the below Table:

Cache Level	Type	Shared Virtual Cores	Size(Bytes)	Sets	Associativity	Line Size(Bytes)
L1	Data	0,4	32K	64	8	64
L1	Instruction	0,4	32K	64	8	64
L1	Data	1,5	32K	64	8	64
L1	Instruction	1,5	32K	64	8	64
L1	Data	2,6	32K	64	8	64
L1	Instruction	2,6	32K	64	8	64
L1	Data	3,7	32K	64	8	64
L1	Instruction	3,7	32K	64	8	64
L2	Unified	0,4	256K	512	8	64
L2	Unified	1,5	256K	512	8	64
L2	Unified	2,6	256K	512	8	64
L2	Unified	3,7	256K	512	8	64
L2	Unified	All	8192K	8192	16	64

Most of the known methods to find cache information of a system requires additional (usually `root`) privileges. However we wrote a python script(`get_cpu_cache_info.py`) which get details of entire cache organization of any system running *nix operating system, moreover *this script doesn't require `root` privileges.* As you can see cache organization is non trivial, you may or may not be able to exploit cache timing side channel depending on virtual cores on which attack and victim process gets executed. Also note that cache sizes are considerably larger then key size of most of the cryptographic algorithms, which makes attacks based on capacity misses practically impossible.

B. Attack Goal and Strategy

In this subsection, we explain goal of the attack and strategy we used to perform the attack.

B.1 Attack Goal

The goal of the attack is to predict the cache set of a data item which the victim process is repeatedly accessing. To emulate the possibility of attacker ability to measure memory access time, the victim process outputs access time of the data item.

C. Strategy

Our strategy is to exploit delay due to conflict misses. Specifically, we launch an attack process which tries to fill up one cache set at a time. If the data item of victim process maps to the same cache set then we should see considerable increase in access time, otherwise access time should not be affected. We repeat this for all the sets in the cache and observe the access time of victim process, an increase in access time implies the cache set filled by attack process is same as the cache set of the data item which the victim process is trying to access.

D. Implementation

We measured the access time of data item as seen by a process(p1) when another process(p2) is trying to access data that belongs to both same and different cache set. We see following differences:

Description	P1 Data item set	P2 Data item set	P1 Access Time
P2 Doesn't access any data	x	N/A	**6 ns**
P2 Access data belongs to different cache set as P1 data	x	y (y≠x)	**8-9ns**
P2 Access data that belongs to same cache set as P1 data	x	x	**10-11 ns**

We expected values of last column to be same for Row 1 and Row 2, as they should not affect access time for P1, but the reason they are different is because

D.1 Challenges

D.2 Results

IV. Attacks

V. Defence

Unlike physical side channel attacks, software cache-based side channel attacks can impact a much wider spectrum of systems and users as these software attacks are very easy to perform and are effective on various platforms. This makes cache-based side channel attacks extremely attractive as a new weapon in the attackers arsenal. A thourough analysis of cache side channel attacks have revelaed cache interfearance as the main root cause. Several mitigation methods mostly in hardware have been devised to prevent leakage of side channel information. Some of them are

- **Partition Locked Cache** essentially achieves the effect of cache partitioning, but much more flexibly with less performance degradation. In partition locked cache, the cache lines of interest are locked in cache, creating a flexible private partition; these cache lines can not be evicted by other cache accesses not belonging to this private partition. This prevents both internal and external cache inerferences.
- **Random Permutation Cache** allows cache sharing by randomizing the resulting interference, so that no useful information about which cache line was evicted can be inferred. An attacker can observe another processs cache access only if that process changes the attackers cache usage, i.e., evicts the attackers cache lines. If the process evicts its own cache lines, the attacker has no way to know that. In random permutation cache, each time cache interference occurs, the interference is randomized in a way that carries no useful information.

The above methods have been described in detail in [6]. Another mitigation strategy employed in ARM achitectures is the presence of distinguished trustzone. In .... to be completed by Arvind.

References

[1] Daniel J Bernstein, "Cache-timing attacks on aes," 2005.

[2] Onur Aciçmez and Çetin Kaya Koç, "Trace-driven cache attacks on aes," 2006.

[3] Sebastian Banescu, "Cache timing attacks," 2011.

[4] Torsten Frenzel, Adam Lackorzynski, Alexander Warg, and Hermann Härtig, "Arm trustzone as a virtualization technique in embedded systems," in *Proceedings of Twelfth Real-Time Linux Workshop, Nairobi, Kenya*, 2010.

[5] Taesoo Kim, Marcus Peinado, and Gloria Mainar-Ruiz, "Stealthmem: System-level protection against cache-based side channel attacks in the cloud," in *Presented as part of the 21st USENIX Security Symposium (USENIX Security 12)*, Bellevue, WA, 2012, pp. 189–204, USENIX.

[6] Zhenghong Wang and Ruby B Lee, "New cache designs for thwarting software cache-based side channel attacks," in *ACM SIGARCH Computer Architecture News*. ACM, 2007, vol. 35, pp. 494–505.

[7] Yuval Yarom and Katrina Falkner, "Flush+reload: A high resolution, low noise, l3 cache side-channel attack," in *23rd USENIX Security Symposium (USENIX Security 14)*, San Diego, CA, Aug. 2014, pp. 719–732, USENIX Association.

[8] John Tse and Alan Jay Smith, "Cpu cache prefetching: Timing evaluation of hardware implementations," *Computers, IEEE Transactions on*, vol. 47, no. 5, pp. 509–526, 1998.

[9] Intel, "Intel Hyperthreading," `http://www.intel.com/content/www/us/en/architecture-and-technology/hyper-threading/hyper-threading-technology.html`.