Covert Timing Channels Exploiting Non-Uniform Memory Access based Architectures

Fan Yao, Guru Venkataramani and Miloš Doroslovački
Department of Electrical and Computer Engineering
The George Washington University, Washington, DC, USA
{albertyao, guruv, doroslov}@gwu.edu

ABSTRACT

Covert timing channels are a class of information leakage attacks where two processes, namely the *trojan* and *spy*, collude with intent to stealthily exfiltrate privileged information even when the underlying system security policy prohibits any direct communication between the two processes. In this paper, we present a new type of covert timing channel that exploits the access timing difference between various caches in Non-Uniform Memory Access (NUMA)-based architectures, especially multi-socket CPUs. We demonstrate a realistic covert timing channel implemented on a dual-socket Intel Xeon server. We then explore use of statistical analysis techniques to characterize and quantify the presence of covert timing channel activity. Our experimental results show that such quantification techniques could be a useful first step in formulating an effective defense against NUMA-based covert timing channels.

1. INTRODUCTION

Computer security attacks, especially those that leak sensitive data, are a fast growing concern in our everyday lives. Covert channels are a class of information leakage attacks where two colluding processes, namely the trojan and the spy, intentionally subvert the underlying system security policy to exfiltrate privileged information. Compared to side channels in which the activities of an unsuspecting victim process are monitored stealthily by a spy, covert channels operate through an *insider* trojan process that communicates the secrets to the spy even when the underlying system policy prohibits such interaction [5]. Covert channels are usually classified into two broad categories: storage-based (that hide secrets within legitimately communicated data) and timing-based (that simply manipulate the timing of access to certain hardware or software resources). Among these covert channel implementations, timing-based attacks are notoriously hard to detect since they do not leave any physical trace of an attack for inspection and forensic analysis [18].

GLSVLSI '17, May 10-12, 2017, Banff, AB, Canada
© 2017 ACM. ISBN 978-1-4503-4972-7/17/05...$15.00
DOI: http://dx.doi.org/10.1145/3060403.3060417

With rapid improvements in software confinement and isolation mechanisms, it is becoming increasingly difficult for covert timing channels to exploit software resources. Naturally, malicious attackers are turning to exploit hardware. The multi-core processors, that have multitude of shared hardware resources (e.g., caches, interconnect and functional units), offer rich space for such exploits. Note that these multi-core processors dominate the entire computing landscape today ranging from mobile phones to cloud computing. Therefore, it becomes critical to understand how malicious attackers exploit such shared resources, and devise strategies to neutralize such efforts by the adversary.

Recently, there have been a growing number of studies on hardware covert timing channel attacks that manipulate the access timing on caches [15, 23, 14], functional units such as integer multipliers [19], processor-memory bus [22] and branch predictors [7]. In all of these attacks, the trojan creates contention on a particular shared resource by creating a certain number of events, and lets the spy observe the altered access timing on that resource, and infer the transmitted bit.

In this paper, we investigate and present a new type of hardware vulnerability to covert timing channels exposed by the difference in access timing across multiple levels of the cache hierarchy in Non-Uniform Memory Access (NUMA)-based architectures. We implement a realistic covert timing channel based on this vulnerability, and demonstrate the attack on a dual-socket Intel Xeon server where the trojan modulates the cache access timing by locating itself on a socket different from the spy. We then explore statistical techniques to characterize and quantify the possible presence of covert timing channel activity. We utilize Degree of Sparseness [13] to quantify and analyze the pattern of time intervals for inter-socket cache data transfers when observed during covert channel activity and regular application execution (with no known covert information leakage channels). We note that developing such quantification techniques will be a useful first step in mounting successful defense against such timing channels.

The contributions of our paper are:

1. We present a new hardware-based covert timing channel implementation where the trojan, in order to communicate the secrets to the spy, exploits the timing differences in cache read accesses exposed by NUMA-based architectures.
2. We demonstrate the viability of our new covert timing channel on a real system testbed using Intel Xeon X5650 dual-socket processors.
3. We explore statistical methods to quantify NUMA-based

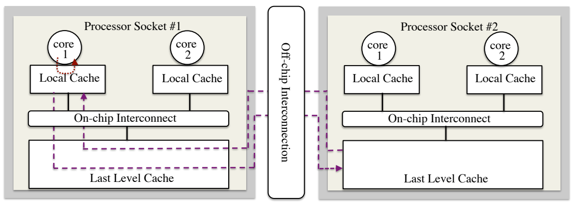


Figure 1: Local and remote cache accesses in NUMA system.

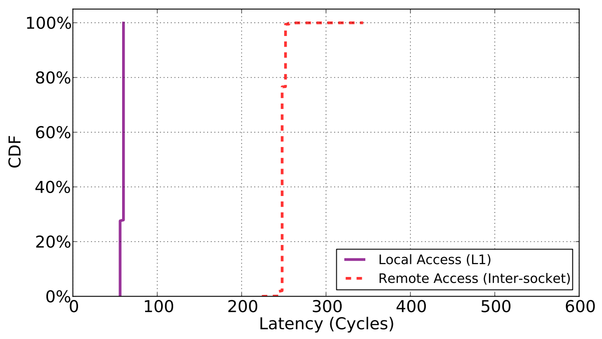


Figure 2: Cumulative Distribution Function for local (Level 1) and remote/cross-socket (Last Level) cache access latency

covert timing channel activity, that can eventually help guide defense strategies against such channels.

2. BACKGROUND

The memory hierarchy in recent processors includes several levels of caches and DRAM, some that are used privately by the individual cores and some that are shared between multiple cores. Also, with the use of multi-socket CPUs communicating via high speed interconnect (e.g., Quick Path Interconnect (QPI) in Intel architectures [2] and HyperTransport links in AMD architectures [6]), processor cores can now share cache contents across processors with coherence protocols running between them. Due to these hierarchical memory levels, there is usually a shorter latency period for read/write requests satisfied by a cache situated locally within the processor (local socket) when compared to memory requests that are satisfied by a cache belonging to a different processor (cross-socket or inter-socket). An unintended side-effect of these timing differences in NUMA-based machines is that, a malicious hacker can exploit such timing variations to force data to be placed in different caches, and ultimately implement covert timing channels. Figure 1 illustrates the difference between cache accesses that are satisfied locally within the same socket (local cache hit– shown using a dotted line) versus those that are satisfied by the remote cache (cache hit in another socket– shown using a dashed line).

2.1 NUMA Latency

To understand the access latency variations in NUMA, we perform experiments that read data from local and remote processor caches in a multi-socket processor. For this study, we use a dual-socket Intel Xeon X5650 server with 6 cores in each socket running at 2.67 GHz frequency. Each processor has a 32 KB private L1, 256 KB private L2 caches, 12 MB shared L3 cache within each socket. All of the caches are kept coherent in hardware. To model real system settings, applications such as browser, dropbox, code editors are run alongside our experiments.

We generate 1,000 memory read operations that target caches in the local socket (specifically, L1 cache) and remote socket (specifically, Last Level Cache or LLC in another socket). To target local caches for reads, our test profiler runs a tight loop with repeated reads that are satisfied by the local L1 cache. To target remote LLC for reads, our profiler issues periodic read requests. Meanwhile, a control program, that is pinned to a socket different from where the test profiler resides, explicitly issues a flush operation (using x86 ISA-supported instructions such as *clflush* that clears the block from all caches) and then loads the block into corresponding socket's cache hierarchy (that includes its LLC). When the test profiler issues its read access to the cache block, this block is guaranteed to be routed to the remote socket where the control program is located. All of the cache accesses and the associated instructions are timed using *rdtsc* instruction. Figure 2 shows the Cumulative Distribution Function (CDF) for cache access latencies from local and remote sockets. As we can see, the cache accesses to local and remote caches show distinct bands of latency distributions. This shows the viability for exploiting the latency difference between caches in different sockets.

3. THREAT MODEL

In our covert timing channel, we assume that the trojan and spy are running on the same machine that features two or more multi-core processors. The trojan intentionally modulates the cache access timing through issuing a flush operation, and later places a data block in its local cache so that the spy can infer the covertly transmitted information. We assume that a compromised trojan, that has sufficient privileges to access sensitive data, is able to run inside the target multi-socket CPU. Our attack model fits into *flush+reload* category of attacks [26], where the trojan clears its memory blocks and reloads them to alter the access times.

As confinement mechanisms continue to offer stronger isolation between software applications, hardware structures will become natural targets for covert timing channels. In this vein, we illustrate the vulnerabilities exposed by NUMA-based architectures (specifically, multi-socket CPU systems) that provide for timing difference in cache accesses depending on the caches that satisfy the memory requests.

4. NUMA-BASED TIMING CHANNELS

In this section, we demonstrate a covert timing channel implementation that exploits the cache access timing differences in NUMA. The trojan and the spy are two separate physical processes with the trojan having higher privileges than the spy in terms of access to sensitive secrets such as Operating System- or user-related data. In order to sufficiently modulate the timing of cache accesses, the trojan process is run in a socket different from the spy.

Figure 3 illustrates the communication protocol between the trojan and the spy for bit transmission. The spy issues the *load* or read instructions periodically to a memory address that is known (and accessible) to both the trojan and spy. Typically, this memory address points to a shared code region such as a library function shared between

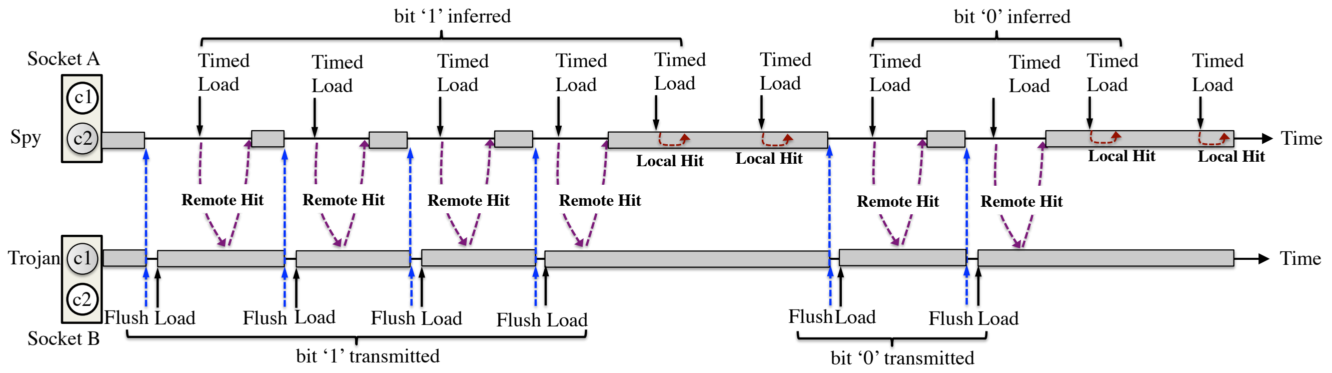


Figure 3: Illustration of the communication protocol between the spy and trojan showing a transmission of bit sequence '10'.

processes [26]. There are two possibilities in NUMA-based CPUs when load operations are issued by the spy: 1. the load hits in the local L1 cache or LLC of the spy's socket, 2. the load misses in the cache hierarchy of the spy's socket, and the requested memory address is resident in the remote socket or DRAM. Our experiments show two distinct bands of latencies for the above two possibilities (Figure 2). The spy *times* these loads to infer whether the loads resulted in a *local cache hit* or a *remote cache hit*. If the loads are satisfied by the DRAM (i.e., neither socket's cache can satisfy), the spy observes a longer latency compared to remote cache hit and ignores them.

The trojan manipulates the access timing to the shared memory address to communicate the bits covertly to the spy. Whenever the trojan wants to communicate a bit, it performs a *cache flush* operation using instructions such as *clflush*. This clears the target memory address from all of the caches that are kept coherent in the multi-socket CPU, including that of the spy located in another socket. The trojan, then immediately, performs a load from that memory location. This populates the shared memory address data in the cache hierarchy of the trojan's socket (i.e., the corresponding L1 cache and LLC). When the spy performs its periodic load from this memory address, the spy's cache cannot service this load since the corresponding contents have been previously flushed by the trojan. Effectively, this enables the spy to detect the timing difference in cache access compared to its local cache hit.

Now that, we have seen how the trojan manipulates the timing of cache access for the spy, it is important to understand how the transmission of '1' and '0' bits can be accomplished. In our current implementation, this is done through trojan performing a specific number of *consecutive* flush+reload operations to be observed by the spy. In our current design, for transmitting a '1', the trojan performs *four* consecutive flush+reload; and for transmitting a '0', the trojan performs *two* consecutive flush+reload operations. The spy infers the end of bit transmission when it observes its load operation result in a local cache hit (via timing the load), i.e., the trojan has not performed flush+reload on the memory block.

Finally, we note that our implementation does not have explicit synchronization between the trojan and the spy. Based on our experimental measurements, a flush+reload operation by the trojan took 350 cycles. To allow for sufficient time, the spy performs its periodic timed load accesses every 1,000 cycles. This minimizes the possibility of the spy missing the trojan's transmitted bit. We note that, in order to further improve the reliability of transmission while maximizing the effective bit rate of covert channel, the spy could reduce the time interval between its load requests, and the trojan could utilize one or more of the following features: 1. add parity bits, 2. packetize the transmission and include packet headers containing metadata, 3. synchronize using existing system features such as CPU clock.

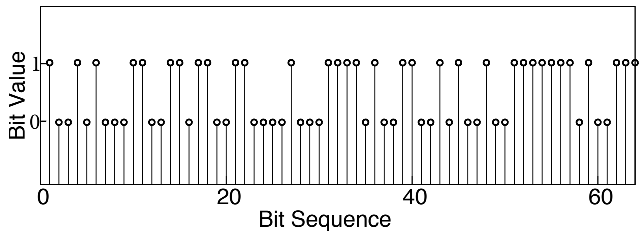


Figure 4: Bit pattern (64 bits) transmitted by the trojan.

5. TIMING CHANNEL DEMONSTRATION

Using the Linux system call *sched_setaffinity*, the spy and the trojan thread are pinned to two different sockets of Intel Xeon X5650 server. To create shared memory pages, we programmed the trojan and spy to load a shared page from the *libgcrypt library* [1], a widely used Linux cryptography library.

The spy runs a *while* loop with *timed* loads to a cache block in the shared libgcrypt library. Figure 4 shows a random 64 bits that are transmitted from the trojan. Correspondingly, Figure 5 shows the sequence of latencies measured using *timed load* by the spy. As we can see, there are several consecutive tall bars (corresponding to *remote cache hits*) fenced by consecutive short bars (corresponding to *local cache hits*). The spy deciphers the bit based on the number of consecutive tall bars (see Section 4). In our experiments, the spy observes 1-2 tall bars during '0' transmission, and 3-4 tall bars during '1' transmission. This small variance stems from tail latencies in remote cache latency distribution and the background noise from other processes. Despite this minor variation, the spy could still correctly distinguish between '1' and '0' with 100% accuracy. Overall, the transmission achieves an effective bit rate of 190 Kbit/sec. Note that it is possible to further improve the speed through ad-

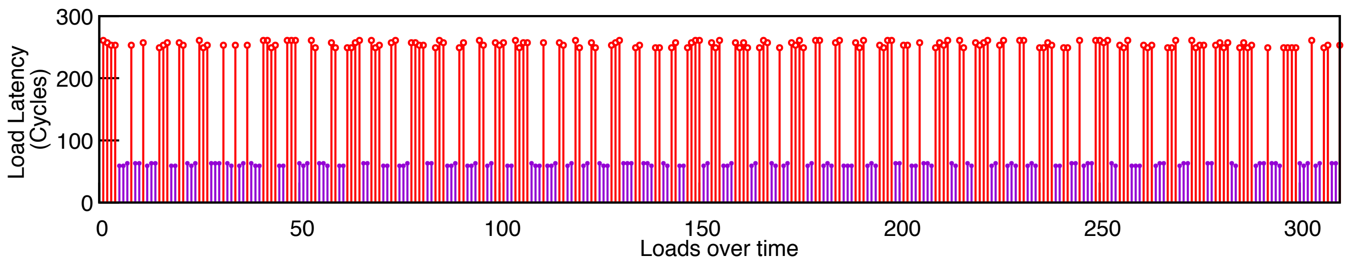


Figure 5: Latency sequence for load operations measured by the spy. The (taller) red bar and (shorter) purple bars correspond to remote cache and local cache hits respectively.

ditional optimizations to the trojan and spy described in Section 4.

6. TIMING CHANNEL ANALYSIS

To prepare for an effective defense, we first attempt to characterize the NUMA-based timing channels. We envision that any detection strategy would need to monitor the activities of multiple NUMA components, and capture the interactions between individual components (e.g., cache miss requests and data transfers). Through analyzing the activity of cache hierarchy during covert transmission, we made an important observation: *The time-interval between two consecutive remote cache accesses (i.e., time-interval between inter-socket cache data transfers) exhibits a concentrated distribution in the case of a covert channel attack compared to regular applications with legitimate communication.* The concentration is due to the fact that the spy relies on observing a consecutive number of remote load accesses in order to infer the transmitted bits. As such, the spy issues a number of load operations at a pre-determined sampling interval that can be observed over the interconnect. Our experiments show that covert channels have only a *few possible values for time-intervals between remote cache accesses* (see Section 6.1 for details). We note that it is conceivable for a spy to change the sampling interval at runtime to evade from being noticed. However, in asynchronous environments, the spy may begin to see a rapid rise in bit error rate due to lack of synchronization. On the contrary, for regular applications that do not intentionally manipulate the timing between inter-socket data transfers, a higher level of randomness is expected in time between inter-socket cache transfers, which is verified through our experiments.

6.1 Time-Intervals between Remote Accesses

Since real hardware does not support measuring the time-intervals between remote cache accesses, we setup *Gem5* [4], a cycle-accurate, full system simulator to perform our measurements. We configure Gem5 with eight x86 cores, and use a minimal Linux distribution with kernel version 2.6.32. We build Parsec-2.1 benchmarks [3] with 8 threads, where each thread is pinned to a separate core. We also configure the spy and trojan to run on Gem5, where we record the timestamps for inter-core cache data transfers. Since any core pair can be used in a covert channel attack, the monitor filters the timestamps associated with cache data transfers for each source-destination core pair. We generate histograms for time-intervals between inter-socket cache accesses.

As expected, our experiments in Figure 6 show that regular (Parsec-2.1) benchmarks exhibit higher randomness in time-intervals. We show results for a representative core in each benchmark. Also, we eliminate time-intervals greater than 5,000 cycles since they represent lengthy idle periods without inter-core cache transfers, and collectively constitute less than 0.5% of the population. We observe that, though some time interval bins are more favored than other bins (i.e., higher probability), the probability of any single bin does not exceed 25% and the expected probability in almost all of the bins are non-trivial. In contrast, for covert channel scenario (Figure 7), the time-interval distribution is highly concentrated and is non-zero only for a very few bins. Notably, the bins between 1,500 and 2,000 cycles correspond to *bit transmission phases* when more frequent remote cache accesses are observed, and the bins between 4,300 and 5,000 cycles correspond to *idle phases* when time interval between remote cache accesses are long. *We note that, though the absolute locations of the histogram bins corresponding to bit transmission and idle phases may change, the characteristic of concentration in certain bins over others would remain, as they are inherently needed to covertly communicate bits between the trojan and the spy.*

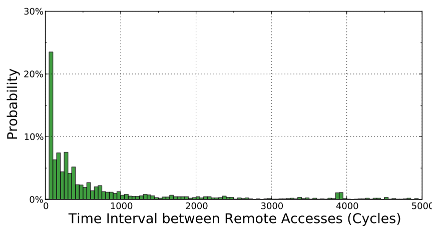
6.2 Quantifying NUMA Covert Channels

From Section 6.1, we see that the time-interval histograms for regular applications and covert channels are significantly different. Specifically, the time-interval distribution in covert channel is highly concentrated within a small number of bins to improve *bit inference accuracy* for the spy. In other words, the highly probable bins are *sparse* for covert channels compared to regular applications. Therefore, we use the metric *Degree of Sparseness* (S) [13] to capture this phenomenon. An S value of 1 denotes that the distribution is very sparse, and a value of 0 means that the distribution is not sparse (i.e., uniform). Given the probability vector for the histogram bins as P, Degree of Sparseness (S) is defined as:

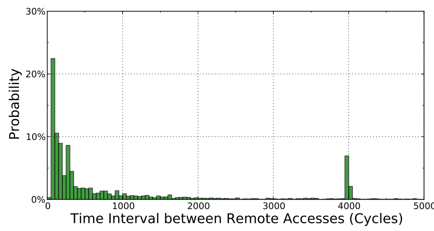
$$S = \frac{M}{M - \sqrt{M}}\left(1 - \frac{\|P\|_1}{\sqrt{M} \times \|P\|_2}\right) \tag{1}$$

where M equals to the number of histogram bins, and $\|P\|_1$, $\|P\|_2$ are norm-1 and norm-2 for vector P respectively. It is worth noting that there are a number of histogram bins in regular applications which have *near-zero* probabilities. Since S depends on the absolute probability values, the histogram bins with near-zero probabilities may be outweighed. To account for such bins and amplify their influence during computation of S, we pre-process the probability values with the μ-law compression function [16] to get a new vector Q:

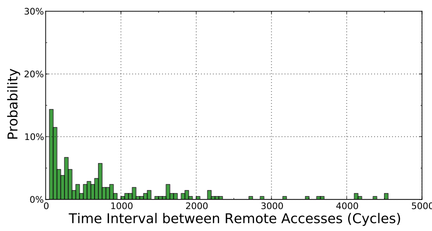
$$Q_i = \frac{\log(1 + \mu P_i)}{\log(1 + \mu)} \tag{2}$$



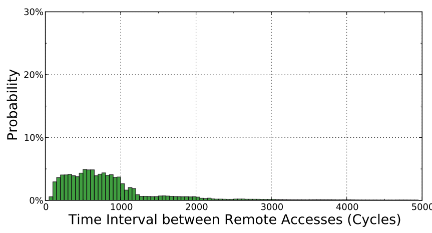
(a) Bodytrack



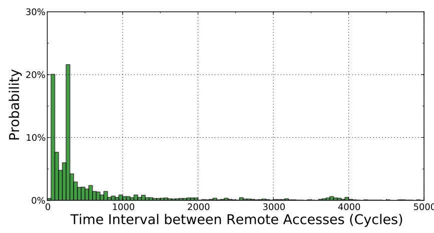
(b) Dedup



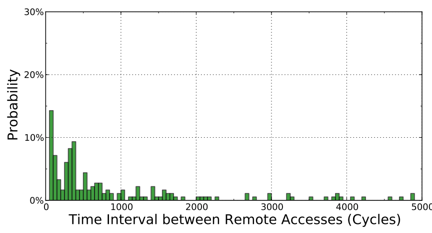
(c) Fluidanimate



(d) StreamCluster



(e) Swaptions



(f) x264

Figure 6: Histograms of time-intervals between remote cache accesses in Parsec-2.1 benchmarks for representative cores.

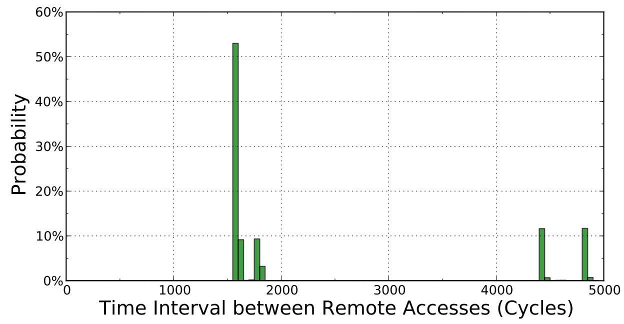


Figure 7: Histogram of time-intervals between remote cache accesses in the covert channel.

where μ is a tunable parameter that controls the degree of compression for large values while increasing the level of amplification for small values. We set $\mu = 500$ in our experiments.

We compute values of S on our histograms for time-intervals for remote cache accesses. Figure 8 shows the *Sparseness* measurement for each of the (source, destination) core pairs for six Parsec-2.1 benchmarks. We observe the S values to be less than 0.4 in all regular benchmarks. On the other hand, S value is very high, and around 0.8 for covert channels (which is at least 2× compared to regular applications). This proves that our quantification technique can indeed be applied as an effective indicator for the possible presence of NUMA-based covert timing channels.

7. RELATED WORK

Several prior works have studied hardware-based side- and covert channels [26, 23, 11, 24, 10]. Recently, Irazoqui et al. [9] demonstrate a side channel implementation that takes advantage of the cache access timing difference exposed by the high speed point-to-point interconnect between processors compared to DRAM accesses. This attack manipulates accesses to the remote cache and DRAM. Different from this attack, our paper demonstrates an attack that exploits local/remote cache access latency differences.

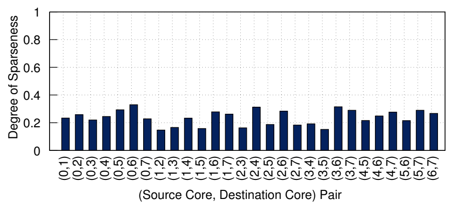
Architecture support for safeguarding computer systems from information leakage attacks have been widely studied [20, 25]. Venkataramani et al. [18] study how to detect contention-based covert timing channels through dynamically tracking conflict patterns on shared hardware resources. Liu et al. [12] propose a cache design that provides dynamic and randomized memory-to-cache mapping to mitigate contention based side channels. Gu et al. [8] have shown the potential of leveraging the merging 3D die-stacking techniques for secure system design. Hardware-based side channel resistant S-Box designs for the AES crypto-systems are studied in prior works [17, 21]. Such techniques can help detect and mitigate information leakage channels, and work synergistically with our detection mechanism to improve the overall system security.

8. CONCLUSIONS

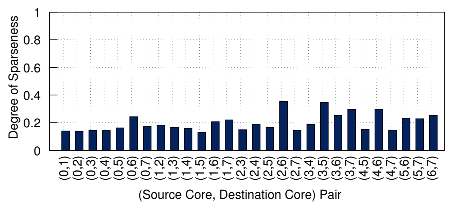
In this paper, we presented a new type of covert timing channel that exploits the cache access timing difference in NUMA-based architectures. Unlike most prior timing channel attacks that exploit a single cache, the proposed attack manipulates different levels of caches across multiple sockets. We implemented a realistic covert timing channel on a dual-socket Intel Xeon CPU. We explored statistical analysis techniques to characterize and quantify the presence of covert timing channel activity using *Degree of Sparseness* for inter-socket cache data transfers. Our experimental results showed that such quantification techniques can sufficiently distinguish covert timing channel activity from regular applications, and ultimately help design defense mechanisms.

9. ACKNOWLEDGMENTS

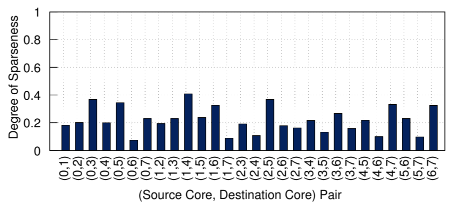
This material is based on work supported by the US National Science Foundation under CAREER Award CCF-1149557 and CNS-1618786, and Semiconductor Research Corp. (SRC) contract 2016-TS-2684. Any opinions, findings, con-



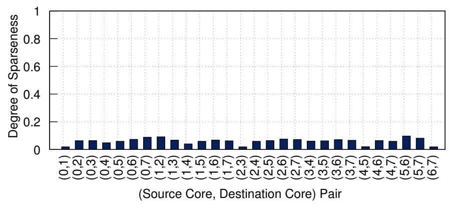
(a) Bodytrack



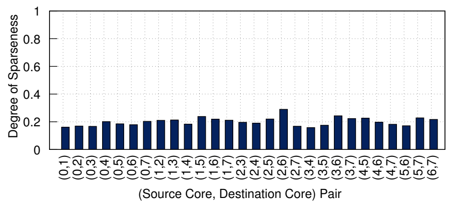
(b) Dedup



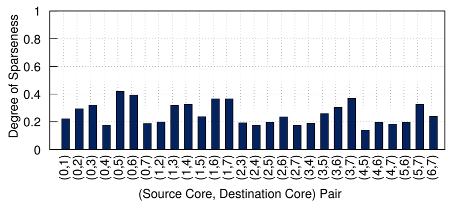
(c) Fluidanimate



(d) Streamcluster



(e) Swaptions



(f) x264

Figure 8: Degree of Sparseness for (Source, Destination) pairs for the Parsec-2.1 Benchmarks.

clusions, or recommendations expressed in this article are those of the authors, and do not necessarily reflect those of the NSF or SRC.

10. REFERENCES

[1] Libgcrypt project. https://www.gnu.org/software/libgcrypt/.

[2] Intel QuickPath Architecture, 2012. http://www.intel.com/pressroom/archive/reference/whitepaper_QuickPath.pdf.

[3] C. Bienia, S. Kumar, J. P. Singh, and K. Li. The PARSEC benchmark suite: Characterization and architectural implications. In *Proceedings of the 17th International Conference on Parallel Architectures and Compilation Techniques*, pages 72–81. ACM, 2008.

[4] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti, et al. The gem5 simulator. *ACM SIGARCH Computer Architecture News*, 39(2):1–7, 2011.

[5] J. Chen and G. Venkataramani. Cc-hunter: Uncovering covert timing channels on shared processor hardware. In *47th Annual International Symposium on Microarchitecture (MICRO)*. IEEE, 2014.

[6] P. Conway, N. Kalyanasundharam, G. Donley, K. Lepak, and B. Hughes. Cache hierarchy and memory subsystem of the AMD Opteron processor. *IEEE micro*, 30(2):16–29, 2010.

[7] D. Evtyushkin, D. Ponomarev, and N. Abu-Ghazaleh. Understanding and mitigating covert channels through branch predictors. *ACM Transactions on Architecture and Code Optimization*, 13(1):10, 2016.

[8] P. Gu, S. Li, D. Stow, R. Barnes, L. Liu, Y. Xie, and E. Kursun. Leveraging 3D technologies for hardware security: Opportunities and challenges. In *Proceedings of the 26th edition of the Great Lakes Symposium on VLSI*, pages 347–352. ACM, 2016.

[9] G. Irazoqui, T. Eisenbarth, and B. Sunar. Cross processor cache attacks. In *Proceedings of the 11th Asia Conference on Computer and Communications Security*, pages 353–364. ACM, 2016.

[10] Z. H. Jiang, Y. Fei, and D. Kaeli. A complete key recovery timing attack on a GPU. In *Proceedings of the 22nd International Symposium on High Performance Computer Architecture*, pages 394–405. IEEE, 2016.

[11] J. Lee, M. Tehranipoor, C. Patel, and J. Plusquellic. Securing designs against scan-based side-channel attacks. *IEEE Transactions on Dependable and Secure Computing*, 4(4):325–336, 2007.

[12] F. Liu, H. Wu, K. Mai, and R. B. Lee. Newcache: Secure cache architecture thwarting cache side-channel attacks. *IEEE Micro*, 36(5):8–16, 2016.

[13] P. Loganathan, A. W. Khong, and P. A. Naylor. A class of sparseness-controlled algorithms for echo cancellation. *IEEE Transactions on Audio, Speech, and Language Processing*, 17(8):1591–1601, 2009.

[14] D. A. Osvik, A. Shamir, and E. Tromer. Cache attacks and countermeasures: the case of AES. In *Proceedings of the Cryptographers' Track at the RSA Conference*, pages 1–20. Springer, 2006.

[15] T. Ristenpart, E. Tromer, H. Shacham, and S. Savage. Hey, you, get off of my cloud: exploring information leakage in third-party compute clouds. In *Proceedings of the 16th Conference on Computer and Communications Security*, pages 199–212. ACM, 2009.

[16] K. Sayood. *Introduction to data compression.* Newnes, 2012.

[17] C. Teegarden, M. Bhargava, and K. Mai. Side-channel attack resistant rom-based aes s-box. In *Proceedings of the International Symposium on Hardware-Oriented Security and Trust*, pages 124–129. IEEE, 2010.

[18] G. Venkataramani, J. Chen, and M. Doroslovacki. Detecting hardware covert timing channels. *IEEE Micro*, 36(5):17–27, 2016.

[19] Z. Wang and R. B. Lee. Covert and side channels due to processor architecture. In *Proceedings of the 22nd Annual Computer Security Applications Conference*, pages 473–482. IEEE, 2006.

[20] Z. Wang and R. B. Lee. New cache designs for thwarting software cache-based side channel attacks. In *ACM SIGARCH Computer Architecture News*, volume 35, pages 494–505. ACM, 2007.

[21] J. Wu, Y.-B. Kim, and M. Choi. Low-power side-channel attack-resistant asynchronous s-box design for aes cryptosystems. In *Proceedings of the 20th edition of the Great Lakes Symposium on VLSI*, pages 459–464. ACM, 2010.

[22] Z. Wu, Z. Xu, and H. Wang. Whispers in the hyper-space: high-speed covert channel attacks in the cloud. In *USENIX Security 12*, 2012.

[23] Y. Xu, M. Bailey, F. Jahanian, K. Joshi, M. Hiltunen, and R. Schlichting. An exploration of l2 cache covert channels in virtualized environments. In *Proceedings of the 3rd Workshop on Cloud Computing Security Workshop*, pages 29–40. ACM, 2011.

[24] B. Yang, K. Wu, and R. Karri. Scan based side channel attack on dedicated hardware implementations of data encryption standard. In *Proceedings of the International Test Conference*. IEEE, 2004.

[25] F. Yao, J. Chen, and G. Venkataramani. Jop-alarm: Detecting jump-oriented programming-based anomalies in applications. In *Proceedings of the 31st International Conference on Computer Design*, pages 467–470. IEEE, 2013.

[26] Y. Yarom and K. Falkner. Flush+ reload: a high resolution, low noise, L3 cache side-channel attack. In *USENIX Security*, 2014.