Leveraging Cache Management Hardware for Practical Defense against Cache Timing Channel Attacks

Fan Yao, *Member, IEEE,* Hongyu Fang, *Student Member, IEEE,* Miloš Doroslovački, *Member, IEEE,* Guru Venkataramani, *Senior Member, IEEE*

Abstract—Sensitive information leakage is becoming a growing security concern exacerbated by processor's shared hardware structures. Recent studies have shown how adversaries can exploit cache timing channel attacks to exfiltrate secret information. To effectively guard computing systems against such attacks, it is essential to build *practical defense techniques* that are readily deployable and introduce only minimal performance overhead. In this work, we propose a new protection framework against cache timing channel attacks by leveraging Commercial Off-The-Shelf (COTS) hardware support in Last Level Caches (LLC) for cache monitoring and partitioning. We apply signal processing techniques on per-domain LLC occupancy data (available through LLC monitor) to identify suspicious contexts. LLC's dynamic way partitioning is then used to disband domains that are involved in timing channels. We build a prototype of our proposed framework on a real system, and evaluate our design on a number of cache timing channel attacks and on virtualized environment. Results show that our framework can thwart several classes of cache timing channels with negligible performance overheads.

Index Terms—cache timing channels, COTS hardware, cache monitoring, cache allocation, signal processing, practical defense.

1 Introduction

Timing channels are a form of information leakage attacks where adversaries modulate and (or just) observe access timing to shared resources in order to exfiltrate secrets. Among various hardware-based information leakage attacks, cache timing channels have become notorious, since caches presenting the largest on-chip attack surface for adversaries to exploit combined with high bandwidth transfers [13]. Previously proposed detection and defense techniques against cache timing attacks either explore hardware modifications or incur non-trivial performance overheads [17], [12], [16], [6]. For more effective system protection and widescale deployment, it is critical to explore ready-to-use and performance-friendly practical protection against cache timing channel attacks.

In this article, we propose a new framework that makes novel use of COTS hardware to thwart cache timing channels. We observe that cache block replacements by adversaries in cache timing channels reveal a distinctive pattern in their cache occupancy profiles, which could be a strong indicator for the presence of cache timing channels. We leverage Intel's Cache Monitoring Technology (CMT [3]) available in recent server-class processors to perform fine-grained monitoring of LLC occupancy for individual application domains. We then apply signal processing techniques that characterize the communication strength with spy processes in cache timing channels. We further leverage LLC way allocation (i.e., CAT [3]) and re-purpose it as a secure cache manager to dynamically partition LLC for suspicious domains and disband their timing channel activity. Our mechanism *avoids pre-emptively separating domains* and consequently, does not result in high performance overheads to benign application domains.

In comparison to our recent work—COTSknight [19], the novel contributions of this article are as follows:

1) To defend against sophisticated adversaries that randomize interval times between transmissions, we augment our COTSKnight design to remove irrelevant occupancy trace segments using *time warping* (Section 6.3).

2) We perform new experimental studies on virtualized environments that are prone to cache timing channel attacks, and demonstrate the efficacy of our approach (Section 6.4).

3) We identify futuristic threats (like multiple spies and evidence tampering), and discuss potential defense mechanisms using our proposed defense framework (Section 7).

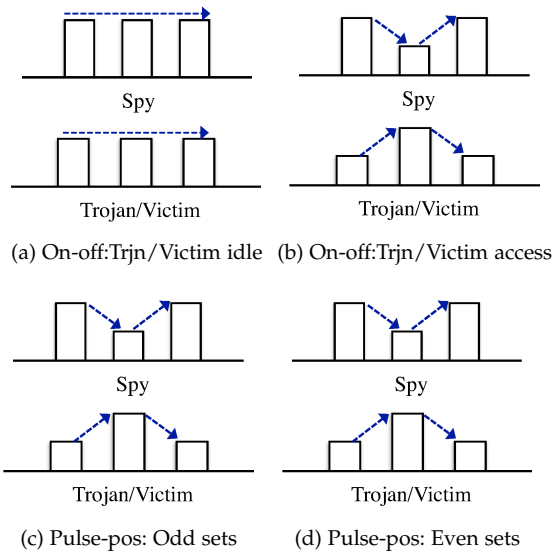
2 Background

There are typically two processes involved in cache timing channels, namely, the trojan and spy in covert channels, and victim and spy in side channels. The spy infers secrets from the trojan or the victim by observing the modulated latency of cache accesses [2]. To exfiltrate secrets, the spy needs to determine a communication channel. In case of covert channels, the trojan and spy may alternate their accesses to the cache temporally, while in side channels, the spy has to run in parallel to the victim process [17]. This may vary along space dimension as well (i.e., access single cache location or alternate among multiple cache locations).

Recently, Intel's CMT allows for uniquely identifying each logical core with a Resource Monitoring ID (RMID) [3], and track the LLC usage for the mapped domains. CMT enables flexible monitoring of LLC occupancy at user-desired *domain* granularity such as a core, a multi-threaded application or a virtual machine. With CAT, caches can be configured to have several different partitions on cache ways, called Classes of Service (CLOS), where evicting cache lines from other CLOS is restricted for a given domain.

3 Threat Model

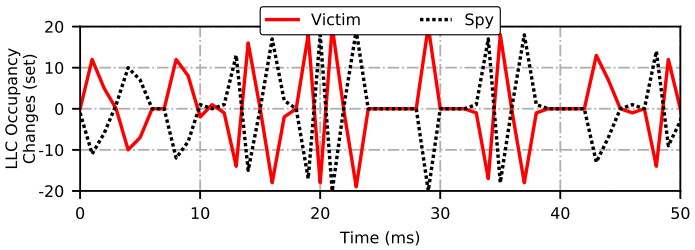
In this work, we focus on the sophisticated form of attacker that does not rely on any prior memory sharing, and utilizes Prime+Probe-based techniques to launch attacks on LLC simply by creating conflict misses (replacement) on cache



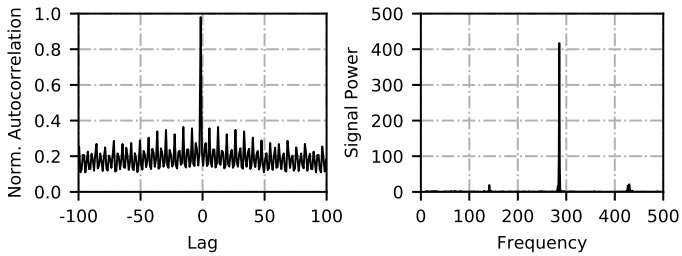
(a) On-off:Trjn/Victim idle (b) On-off:Trjn/Victim access

(c) Pulse-pos: Odd sets (d) Pulse-pos: Even sets

Fig. 1: LLC occupancy changes for trojan/victim and spy.



(a) Parallel Protocol with pulse-position encoding



(b) Autocorrelogram (left) and power spectrum (right)

Fig. 2: LLC occupancy traces, autocorrelogram and power spectrum for a cache timing channel channel (with parallel, pulse-position) [13].

sets. Attacks such as Flush+Reload require shared memory blocks either through shared libraries or data sharing that may be prohibited in practical settings. Therefore, we do not consider such forms of attacks. However, for evict+reload attacks, where cache replacements alter access latencies, our design would still be applicable (See Section 7 for details).

4 Why Cache Occupancy Patterns Matter?

Regardless of whether a trojan intentionally communicates or a victim unintentionally leaks secrets to a spy, cache timing channels use one of the following encoding schemes: 1) on-off encoding (where spy uses timing profile of a *single* cache set group to infer bits/symbols [13]), and 2) pulse-position encoding (where spy leverages access timing of *distinct* cache set groups for inferring each bit/symbol [14]).

Figure 1 illustrates the changes in cache occupancy under the two encoding methods. In on-off encoding, when trojan/victim accesses cache and fetches its blocks, the trojan's cache occupancy should first *increase* and then *decrease* during spy's probe when trojan/victim-owned blocks are replaced. Similarly, the spy's cache footprint would first *decrease* due to trojan/victim's filling in the cache blocks and then *increase* when spy probes and fills the cache with its own data. When trojan/victim doesn't access the cache, neither of the processes change their respective cache occupancies. Under pulse-position encoding with two distinct cache set groups used by trojan/victim (e.g., odd and even sets), we observe swing patterns in their cache occupancies.

We make the following key observation here: Cache timing channels fundamentally rely on cache block replacements, that create swing patterns in participating domain's cache occupancy regardless of the specific timing channel protocols. By analyzing these repetitive swing patterns, there is a potential to uncover the communication strength in such attacks. We note that merely tracking cache misses on an adversary will not be sufficient as an attacker may inflate cache misses (through issuing additional cache loads that create self-conflicts) on purpose to evade detection.

5 System Design

Here, we first discuss CMT-based cache occupancy monitoring and trace analysis for cache timing channel detection, and then outline cache partitioning strategy to prevent information leakage.

5.1 Cache Occupancy Monitor and Pattern Analyzer

We leverage Intel CMT [10] to obtain LLC occupancy data for each domain/context, that allows the system administrators to flexibly define monitoring granularity, e.g., hardware threads, applications or even VMs.

The occupancy pattern analyzer performs the following steps to determine whether there is a cache timing channel between two domains:

First, the analyzer generates the time-differentiated cache occupancy changes for each domain. Assume that x_i and y_i are the cache occupancy sample vectors obtained within the i^{th} window, we can then get the time-differentiated cache occupancy traces for each domain, denoted as $\Delta x_{i,j}$ and $\Delta y_{i,j}$ (i.e., the LLC occupancy difference between two consecutive samples). Figure 2a shows time-differentiated LLC occupancy traces for a timing channel that implements parallel protocol with pulse-position encoding.

In the second step, to capture the unique pair-wise cache occupancy swing pattern in timing channels, we compute the product of Δx_i and Δy_i as z_i. Based on the discussion in Section 4, negative values of z_i occur when the cache occupancy patterns of the two processes move in opposite directions due to mutual cache evictions.

In the third step, our analyzer checks if z series contains repeating negative pulses that may be caused by intentional eviction over a longer period of time (denoting illegal communication activity). To capture the repetitive swing patterns, we perform power spectrum analysis in frequency domain on r_i, which is the autocorrelogram of z.

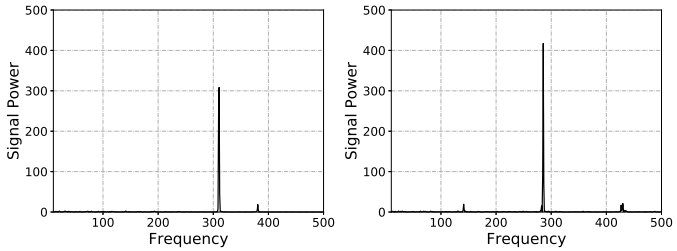
Figure 2b illustrates the autocorrelogram and power spectrum for a *(victim, spy)* pair in timing channels [13]. We can visually observe a sharp peak around frequency of 290 in the power spectrum, which represents a strong communication strength indicating timing channel activity (See COTSKnight [19] for further details on this algorithm).

5.2 Cache Way Allocation Manager

After the way allocation manager (*allocator*) is notified of identified suspicious domains from the analyzer, it will configure LLC using CAT to isolate the suspicious pairs by heuristically assigning non-overlapping cache ways to each domain based on their ratio of LLC occupancy sizes during the last observation period. Our allocator evaluates two candidate policies, namely, 1. Aggressive Policy, that keeps suspicious domains separated until one of them finishes execution and 2. Jail Policy, that partitions the two domains until a timeout period.

5.3 Implementation

We implement our framework prototype on a real system with Intel Xeon E5-2698 v4 processor. The processor comes with 16 CLOS and 20 LLC slices, and each LLC slice has 20×2048 64-byte blocks. The LLC occupancy MSR reading is sampled at 1,000 per second.



(a) serial-onoff (b) para-pp

Fig. 3: Power Spectrum in attack variants including trojan/victim-spy pairs.

6 Evaluation

6.1 Power Spectra for Cache Timing Channels

We setup attack variants of cache timing channels [14], [2], [15], [13] that utilize on-off and pulse-position encoding for spy reception and perform accesses to cache either serially (trojan and spy) or in parallel (victim and spy) as described in Section 4. In each case, we also ran along side with at least two SPEC2006 benchmarks with high LLC activity [8]. The analyzer performs power spectrum analysis based on time-differentiated LLC occupancy traces for 6 combination pairs of processes. In all cases, our framework correctly identified trojan/victim-spy processes since the pair consistently had the highest power in the frequency domain. In fact, our experiments show that the attacker pair's peak power spectrum values are at least an order of magnitude higher than benign application pairs.

Figure 3a and Figure 3b show the analyzer's results on representative windows for two trojan/victim-spy pairs. In the *serial-onoff* attack, we observe a single concentrated and sharp peak with the power value in the frequency domain, while the other data points are almost all zeros. This indicates the existence of a dominating signal in the time domain corresponding to the repetitive gain-loss occupancy pulses due to timing channel activity (Figure 3a). We also observe a similar isolated peak for the trojan/victim-spy pair in *para-pp*, as shown in Figure 3b where the signal power is even higher compared to serial-onoff case.

We repeated several experiments with 60 benign workload pairs with high LLC activity time-overlapping at various random phases, and observed the peak signal power to be less than 5 about 80% of the time, and around 50 for only about 2% of the time. This shows that a vast majority of benign workload samples do not exhibit isolated peaks in the frequency domain, and the maximum signal power is significantly less than any known timing channels (that have signal strengths at well over 100).

6.2 Effectiveness of Our Framework

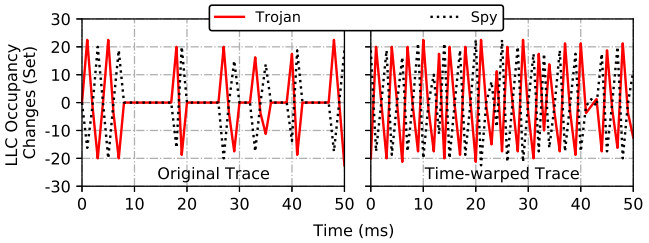
Defeating cache timing channels. We conservatively set signal power threshold at 50 to trigger cache partitioning. Note that we have analyzed the attack variants with different transmission bit rates (i.e., ranging from a few *bps* to several *kbps*), numbers of cache sets and probe intervals. Our results showed that our framework identifies all of the trojan-spy domain pairs within five consecutive analysis windows after they start execution.

Partition trigger rate for benign workloads and the corresponding performance impact. Among all benign workloads (each runs 4 SPEC2006 applications), only 6% of the domain pair population had LLC partitioning - these benchmarks covered 2% of the analysis window samples. Even when there are only two benign applications, it is worth noting that cache occupancy change patterns are typically random. Therefore, signal power (that captures the periodic gain-loss patterns) will not be any higher. Our experiment shows that the LLC partitioning only minimally impacts applications that *trigger partition* (with less than 5% slowdown), and interestingly, we observe performance boost for many of them (up to **9.2**%). The overall average impact on all the applications that ran with partitioned LLC was positive (about **1**%). This shows that our framework can even help benign workloads while safeguarding systems against cache timing channels.

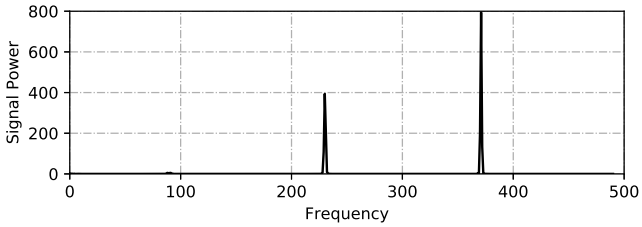
Runtime Overhead. Our framework implements the non-intrusive LLC occupancy monitoring for *only* mutually distrusting domains identified by the system administrator. Overall, the mechanism incurs less than 4% CPU utilization with 4 active mutually-distrusting domains.

6.3 Defense against Advanced Adversaries

In theory, advanced adversaries may use randomized interval times between bit transmissions. Let us imagine a trojan and spy that setup a pre-determined pseudo-random number generator to decide the next waiting period before bit transmission. Even in such cases, our framework can be adapted to recognize them through a signal pre-processing procedure called *time warping* [5], that removes irrelevant segments from the occupancy traces (for which $\Delta x, \Delta y$ are



(a) LLC occupancy changes for transmision with random intervals



(b) Power spectrum on time-warped LLC occupacy trace

Fig. 4: Analysis of bit transmission at random intervals. In (a), left half shows a snippet of original trace with random bit intervals and right half shows time-warped trace.

close to 0 and aligns the swing patterns. After this step, the periodic patterns are reconstructed, and the cadence of cache accesses from adversaries will be recovered. Figure 4 demonstrates the detection of this attack scenario. For illustration, we implement a prototype of this attack by setting up the trojan and spy as two threads within the same process, and configure the main thread to control the synchronization. In reality, two separate trojan/victim and spy need to be synchronized. Figure 4a shows the LLC occupancy trace for this attack with random distances between the swing pulses. We can see that, with time warping, high signal power peaks are observed (Figure 4b). Additionally, when this signal compression pre-processing step is applied on benign workloads, we do not observe any increase in partition trigger rate.

6.4 Case Study on Virtualized Environments

To evaluate the efficacy of our proposed framework, we perform a case study on virtualized environment. This study is motivated by growing trend in studying timing channel attacks in the cloud environment. We implement the *para-onoff* attack that works cross-VM (similar to Maurice et al. [14]).

We setup four KVM virtual machines where the trojan and spy run on two of the VMs, and simultaneously, two other VMs co-run representative cloud benchmarks, namely video streaming (*stream*) and memcached (*memcd*) from CloudSuite [7], both of which are highly cache-intensive. The trojan and spy is set to start the *para-onoff* attack at a random time between 0 to 300 seconds.

We configure the allocator to use the *Aggressive* policy to demonstrate the effectiveness of LLC partitioning. Figure 5 shows the peak signal power between the trojan and spy VM pair and the way allocation determination during the entire execution. We can see that the trojan and spy start to *initiate* communication at around 188 s (when we start to observe increasing signal power). The peak signal power between the trojan and spy domain pair quickly climbs up

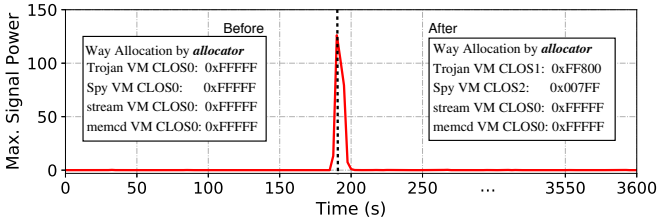


Fig. 5: Peak signal power values for the trojan/spy pair and the allocator's way allocation for one hour execution.

to 126 at time 192.5 s, which is when steady covert communication has began. This quickly triggers the allocator's action that splits the LLC ways between trojan and spy VMs. Consequently, the maximum signal power drops back to nearly zero for the rest of execution, effectively preventing any further timing channels. Note that during the one hour experiment, the peak signal power values for the other domain pairs (involving Cloudsuite applications) remained flat at values less than 3.

7 Discussion

We propose a new framework that builds on COTS hardware and can be augmented with a host of signal processing techniques to eliminate noise, randomness or distortion to unveil the timing channel activity. In this section, we discuss additional monitoring support and signal processing to detect futuristic attacks with sophisticated adversaries.

Using multiple spy processes. A spy may try to evade our defense through potentially involving multiple processes that perform either time-multiplexing (each process is active for a short period of time iteratively) or space multiplexing (each process touches a sub-region of target sets simultaneously) for timing channels. Our proposed framework can still effectively identify such malicious activities as it essentially monitors swing patterns in cache occupancy usage that could purposefully change cache access latencies for domains as discussed in Section 5. Further, CMT+CAT allows for dynamically defining security domains that can best isolate the capability and access boundary for each party (e.g., threads and processes run by the same user belong to the same domain). The *cumulative* LLC occupancy pattern among all the spy's processes in the same domain would preserve the correlated swing pattern that can be recognized by the analyzer.

Using clflush to deflate LLC occupancy. An adversary may attempt to tamper evidence of its cache occupancy changes by compensating the increase in its own cache occupancy through issuing *clflush* instruction. To handle such scenarios, *clflush*'s usage by suspicious domains may be tracked and the associated memory sizes can be accounted back to the issuing core, thus restoring original occupancy data for analysis. Also, many system-level protections against clflush instruction have been proposed, including constraining clflush to only be used in kernel space or just disable it (e.g., Google NaCl). Therefore, clflush-based cache occupancy deflation can be handled easily.

Applicability of our technique to other cache attacks. While we mainly evaluated the proposed technique using

Prime+Probe-based attacks, our proposed framework can be applied to other cache attacks using evictions as well. For instance, in Evict+Reload attacks, the repetitive data loads by the victim and subsequent evictions by the spy will also introduce cache occupancy gain-loss patterns, which can be detected by our proposed framework.

Current Hardware limitations and opportunities. We observe that CMT currently supports a minimum precision of 20 cache sets. If attackers were to leverage less number of sets to carry out attack, they may potentially evade COTSknight's detection. While such attacks are possible, they are prone to high noise. As such, the limitations mentioned above are an artifact of the current CMT hardware, and not of our analysis approach per se. That said, we note that CMT was designed for improving performance bottlenecks, and not to detect cache timing channels. Our study highlights a novel use case for LLC monitoring, and we strongly believe that it would motivate processor vendors to support improved precision and bolster system security.

8 Related Work

Cache-based timing channels have been widely studied [13], [18], and hardware-based solutions have been proposed. CC-Hunter [2] detects covert timing channel in caches by capturing fined-grained cache conflict miss patterns in hardware. ReplayConfusion [17] records program's memory accesses and replays them on a different machine to uncover covert channels on caches. Hunger et al. [9] observe the destructive read property in contention-based covert channel and propose a solution based on anomaly detection. Demme et al. [4] apply machine learning techniques on architectural-level statistics to detect malware including side channels. Fang et al. [6] use hardware prefetchers to defend against cache timing channels.

CATalyst [12] utilizes the CAT technology to reserve static cache partitions where secure pages are pinned upon request from applications. Differently, our proposed mechanism successfully defeat cache timing channels without application/user-level inputs and partition reservation. Bazm el al. [1] leverage cache occupancy information to detect side channels behavior in conjunction with other performance counters such as cache misses. However, their proposed technique makes anomalous behavior determination based on cache footprint, which is be subject to high false positive alarms. In contrast, our framework analyzes cache occupancy gain-loss patterns that are shown to be the unique characteristic for parties involving timing channel activity, which is both effective and efficient. Recently, DAWG [11] has proposed secure cache partitioning by strictly isolating both cache hits and misses between application domains.

9 Conclusion

In this article, we proposed a novel framework to protect caches against timing channel attacks through smartly leveraging COTS support for cache monitoring and performance tuning. We implemented a prototype of our proposed technique on Intel Xeon v4 server and our experiments showed that our framework can successfully thwart several classes of cache timing channels in both native and virtualized environment with minimal performance overhead. We also discussed several futuristic threats and mechanisms to defeat such timing channels.

Acknowledgments

This material is based on work supported by the US National Science Foundation under grant# CNS-1618786, and Semiconductor Research Corporation contract 2016-TS-2684. Fan Yao performed this work as a graduate student at GWU.

References

[1] M. Bazm, T. Sautereau, M. Lacoste, M. Sudholt, and J. Menaud. Cache-based side-channel attacks detection through Intel Cache Monitoring Technology and Hardware Performance Counters. In *Proc. of FMEC*. IEEE, 2018.

[2] Jie Chen and Guru Venkataramani. CC-hunter: Uncovering covert timing channels on shared processor hardware. In *Proc. of MICRO*, pages 216–228. IEEE, 2014.

[3] Intel Corporation. *Intel 64 and IA-32 Architectures Software Developer's Manual, Vol.3B*, 2016.

[4] John Demme, Matthew Maycock, Jared Schmitz, Adrian Tang, Adam Waksman, Simha Sethumadhavan, and Salvatore Stolfo. On the feasibility of online malware detection with performance counters. In *Proc. of ISCA*. ACM, 2013.

[5] M. G. Elfeky, W. G. Aref, and A. K. Elmagarmid. Warp: Time warping for periodicity detection. In *Proc. of ICDM*. IEEE, 2005.

[6] Hongyu Fang, Sai Santosh Dayapule, Fan Yao, Miloš Doroslovački, and Guru Venkataramani. Prefetch-guard: Leveraging hardware prefetches to defend against cache timing channels. In *Proc. of HOST*. IEEE, 2018.

[7] Michael Ferdman, Almutaz Adileh, Onur Kocberber, Stavros Volos, Mohammad Alisafaee, Djordje Jevdjic, Cansu Kaynak, Adrian Daniel Popescu, Anastasia Ailamaki, and Babak Falsafi. Clearing the clouds: a study of emerging scale-out workloads on modern hardware. In *ACM SIGPLAN Notices*, pages 37–48. ACM, 2012.

[8] John L Henning. SPEC CPU2006 benchmark descriptions. *ACM SIGARCH Computer Architecture News*, 34(4):1–17, 2006.

[9] Casen Hunger, Mikhail Kazdagli, Ankit Rawat, Alex Dimakis, Sriram Vishwanath, and Mohit Tiwari. Understanding contention-based channels and using them for defense. In *Proc. of HPCA*. IEEE.

[10] Intel. Intel-CMT-CAT Pacakage, 2017. http://https://github.com/01org/intel-cmt-cat.

[11] Vladimir Kiriansky, Ilia Lebedev, Saman Amarasinghe, Srinivas Devadas, and Joel Emer. DAWG: A defense against cache timing attacks in speculative execution processors. In *Proc. of MICRO*. IEEE, 2018.

[12] Fangfei Liu, Qian Ge, Yuval Yarom, Frank Mckeen, Carlos Rozas, Gernot Heiser, and Ruby B Lee. CATalyst: Defeating last-level cache side channel attacks in cloud computing. In *Proc. of HPCA*. IEEE, 2016.

[13] Fangfei Liu, Yuval Yarom, Qian Ge, Gernot Heiser, and Ruby B Lee. Last-level cache side-channel attacks are practical. In *Proc. of SP*. IEEE, 2015.

[14] Clémentine Maurice, Manuel Weber, Michael Schwarz, Lukas Giner, Daniel Gruss, Carlo Alberto Boano, Stefan Mangard, and Kay Römer. Hello from the other side: Ssh over robust cache covert channels in the cloud. In *Proc. of NDSS*, 2017.

[15] Thomas Ristenpart, Eran Tromer, Hovav Shacham, and Stefan Savage. Hey, you, get off of my cloud: exploring information leakage in third-party compute clouds. In *Proc. of CCS*. ACM, 2009.

[16] Guru Venkataramani, Jie Chen, and Miloš Doroslovački. Detecting hardware covert timing channels. *IEEE Micro*, 36(5):17–27, 2016.

[17] Mengjia Yan, Yasser Shalabi, and Josep Torrellas. Replayconfusion: Detecting cache-based covert channel attacks using record and replay. In *Proc. of MICRO*. IEEE, 2016.

[18] Fan Yao, Miloš Doroslovački, and Guru Venkataramani. Are Coherence Protocol States Vulnerable to Information Leakage? In *Proc. of HPCA*. IEEE, 2018.

[19] Fan Yao, Hongyu Fang, Miloš Doroslovački, and Guru Venkataramani. COTSknight: Practical Defense against Cache Timing Channel Attacks using Cache Monitoring and Partitioning Technologies. In *Proc. of HOST*. IEEE, 2019.

Fan Yao is an assistant professor of Electrical and Computer Engineering at the University of Central Florida. His research interests are in the areas of computer architecture, hardware and system security and cloud computing. Contact him at fan.yao@ucf.edu.

Hongyu Fang is currently a graduate student at George Washington University. His research interests are signal processing, computer architecture and security. Contact him at hongyufang_ee@email.gwu.edu.

Miloš Doroslovački is an associate professor of Electrical and Computer Engineering at George Washington University. His research area is adaptive signal processing with focus on communications and distributed estimation. Contact him at doroslov@gwu.edu.

Guru Venkataramani is an associate professor of Electrical and Computer Engineering at George Washington University. His research area is computer architecture, security and energy optimization. Contact him at guruv@gwu.edu.