

# Leveraging Cache Management Hardware for Practical Defense Against Cache Timing Channel Attacks

**Fan Yao**

University of Central Florida

**Hongyu Fang**

George Washington University

**Miloš Doroslovački**

George Washington University

**Guru Venkataramani**

George Washington University

**Abstract**—Sensitive information leakage through shared hardware structures is becoming a growing security concern. In this article, we propose a practical protection framework against cache timing channel attacks by leveraging commercial off-the-shelf hardware support in last level caches for cache monitoring and partitioning.

■ **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.<sup>13</sup> Previously proposed detection and defense techniques against cache timing attacks either explore hardware modifications or incur nontrivial performance overheads.<sup>6,12,16,17</sup> 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.

*Digital Object Identifier 10.1109/MM.2019.2920814*

*Date of publication 4 June 2019; date of current version 23 July 2019.*

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<sup>3</sup>) 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<sup>3</sup>) and repurpose it as a secure cache manager to dynamically partition LLC for suspicious domains and disband their timing channel activity. Our mechanism avoids preemptively separating domains and consequently, does not result in high performance overheads to benign application domains.

In comparison to our recent work—COTSknight,<sup>19</sup> 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* (see the “Defense Against Advanced Adversaries” section).
- 2) We perform new experimental studies on virtualized environments that are prone to cache timing channel attacks, and demonstrate the efficacy of our approach (see the “Case Study on Virtualized Environments” section).
- 3) We identify futuristic threats (like multiple spies and evidence tampering), and discuss potential defense mechanisms using our proposed defense framework (see the “Discussion” section).

## BACKGROUND

There are typically two processes involved in cache timing channels, namely, the trojan and

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.

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.<sup>2</sup> 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.<sup>17</sup> This may vary along space dimension as well (i.e., access single cache location or alternate among multiple cache locations).

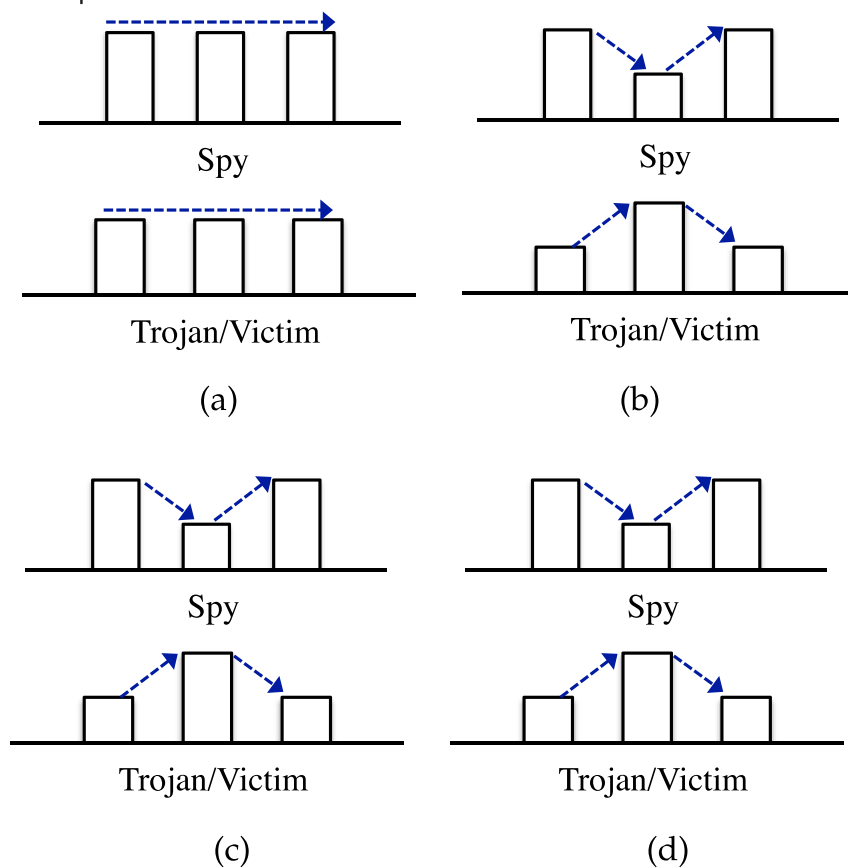
Recently, Intel's CMT allows for uniquely identifying each logical core with a resource monitoring ID,<sup>3</sup> and track the LLC usage for the mapped domains. CMT enables flexible monitoring of LLC occupancy at user-desired *domain* granularity such as a core, a multithreaded application, or a virtual 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.

## THREAT MODEL

In this article, 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 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 the “Discussion” section for details.)

## 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



**Figure 1.** LLC occupancy changes for trojan/victim and spy. (a) ON-OFF: Trjn/Victim idle. (b) ON-OFF: Trjn/Victim access. (c) Pulse-pos: Odd sets. (d) Pulse-pos: Even sets.

single cache set group to infer bits/symbols<sup>13</sup>), and 2) pulse-position encoding (where spy leverages access timing of *distinct* cache set groups for inferring each bit/symbol<sup>14</sup>).

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 does not 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.

## 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.

### Cache Occupancy Monitor and Pattern Analyzer

We leverage Intel CMT<sup>10</sup> 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 2(a) shows time-differentiated LLC occupancy traces for a timing channel

that implements parallel protocol with pulse-position encoding.

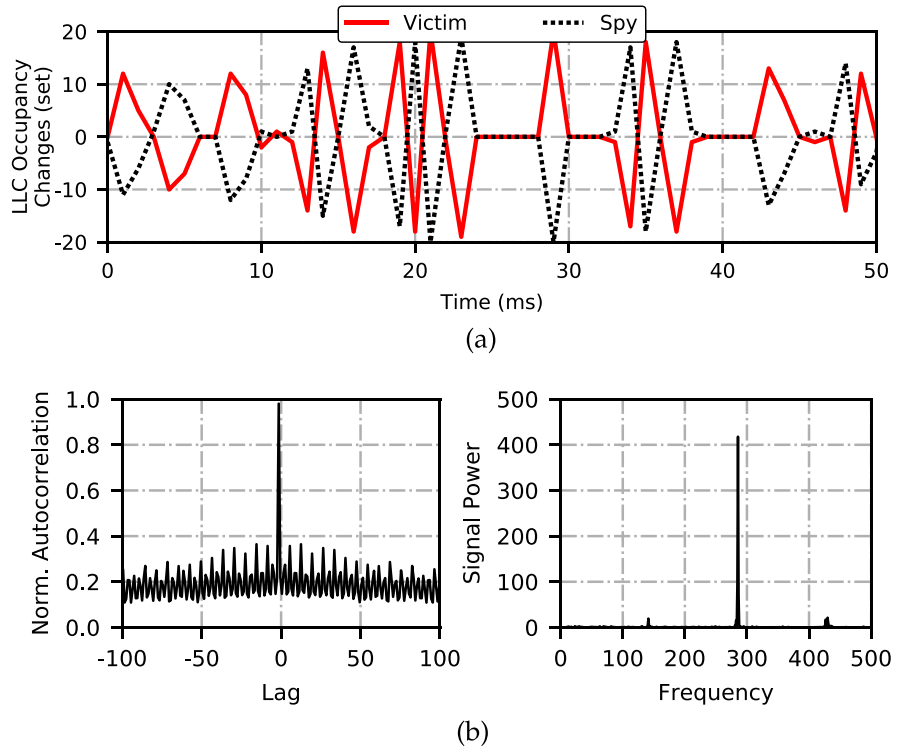
In the second step, to capture the unique pairwise cache occupancy swing pattern in timing channels, we compute the product of  $\Delta x_i$  and  $\Delta y_i$  as  $z_i$ . Based on the discussion in the “Why Cache Occupancy Patterns Matter?” section negative values of  $z_i$  occur when the cache occupancy 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 2(b) illustrates the autocorrelogram and power spectrum for a (*victim*, *spy*) pair in timing channels.<sup>13</sup> 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<sup>19</sup> for further details on this algorithm).

#### 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 nonoverlapping 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, Aggressive Policy, that keeps suspicious domains separated until one of them finishes execution and Jail Policy, that partitions the two domains until a timeout period.



**Figure 2.** LLC occupancy traces, autocorrelogram, and power spectrum for a cache timing channel (with parallel pulse-position).<sup>13</sup> (a) Parallel Protocol with pulse-position encoding. (b) Autocorrelogram (left) and power spectrum (right).

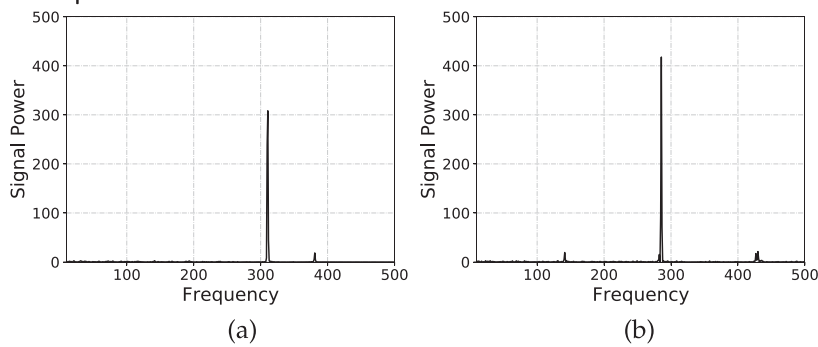
#### Implementation

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  $20 \times 2048$  64-byte blocks. The LLC occupancy MSR reading is sampled at 1000/s.

## EVALUATION

#### Power Spectra for Cache Timing Channels

We setup attack variants of cache timing channels<sup>2,13–15</sup> 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 the “Why Cache Occupancy PATTERNS MATTER?” section. In each case, we also ran along side with at least two SPEC2006 benchmarks with high LLC activity.<sup>8</sup> The analyzer performs power spectrum analysis based on time-differentiated LLC occupancy traces for six combination pairs of processes. In all



**Figure 3.** Power spectrum in attack variants including trojan/victim-spy pairs.  
(a) serial-on-off (b) para-pp

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 3(a) and (b) shows the analyzer's results on representative windows for two trojan/victim-spy pairs. In the *serial-on-off* 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 [see Figure 3(a)]. We also observe a similar isolated peak for the trojan/victim-spy pair in *para-pp*, as shown in Figure 3(b) where the signal power is even higher compared to *serial-on-off* 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).

#### 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 four 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% slow-down), 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 nonintrusive LLC occupancy monitoring for *only* mutually distrusting domains identified by the system administrator. Overall, the mechanism incurs less than 4% CPU utilization with four active mutually distrusting domains.

#### 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 predetermined pseudorandom 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 preprocessing procedure called *time warping*,<sup>5</sup> that removes irrelevant segments from the occupancy traces (for which  $\Delta x, \Delta y$  are close to 0 and aligns the swing patterns). After this step, the



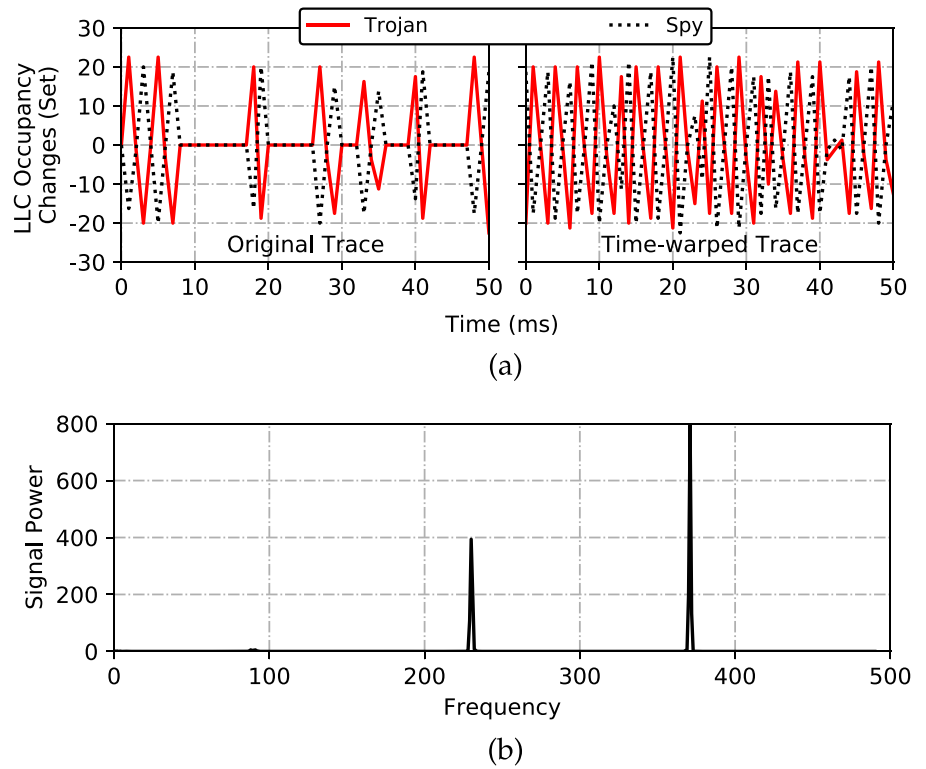
periodic patterns are reconstructed, and the cadence of cache accesses from adversaries will be recovered. Figure 4 demonstrates the detection of this attack scenario. For illustration, we implement a prototype of this attack by setting up the trojan and spy as two threads within the same process, and configure the main thread to control the synchronization. In reality, two separate trojan/victim and spy need to be synchronized. Figure 4(a) 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 [see Figure 4(b)]. Additionally, when this signal compression preprocessing step is applied on benign workloads, we do not observe any increase in partition trigger rate.

#### 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.*<sup>14</sup>)

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

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

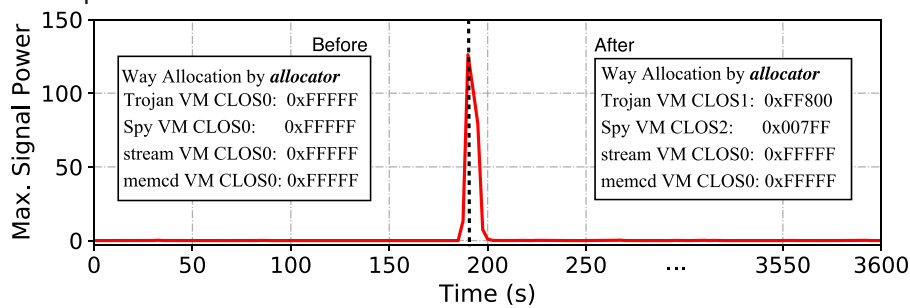


**Figure 4.** Analysis of bit transmission at random intervals. (a) Left half shows a snippet of original trace with random bit intervals and right half shows time-warped trace. (a) LLC occupancy changes for transmission with random intervals. (b) Power spectrum on time-warped LLC occupancy trace.

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 to 126 at time 192.5 s, which is when steady covert communication has begun. 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 1-h experiment, the peak signal power values for the other domain pairs (involving Cloudsuite applications) remained flat at values less than 3.

#### DISCUSSION

We propose a new framework that builds on COTS hardware and can be augmented with a host of signal processing techniques to eliminate



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

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 subregion 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 the “System Design” section. 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.

## RELATED WORK

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

channels. Fang *et al.*<sup>6</sup> use hardware prefetchers to defend against cache timing channels.

CATalyst<sup>12</sup> 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 *et al.*<sup>1</sup> 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 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<sup>11</sup> has proposed secure cache partitioning by strictly isolating both cache hits and misses between application domains.

## 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

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.

also discussed several futuristic threats and mechanisms to defeat such timing channels.

## ACKNOWLEDGMENT

This work was supported by the U.S. National Science Foundation under Grant CNS-1618786, and by the Semiconductor Research Corporation Contract 2016-TS-2684. F. 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. 3rd Int. Conf. Fog Mobile Edge Comput.*, 2018, pp. 7–12.
2. J. Chen and G. Venkataramani, "CC-hunter: Uncovering covert timing channels on shared processor hardware," in *Proc. 47th Annu. IEEE/ACM Int. Symp. Microarchit.*, 2014, pp. 216–228.
3. Intel Corporation, *Intel 64 and IA-32 Architectures Software Developer's Manual*, vol. 3B, 2016.
4. J. Demme *et al.*, "On the feasibility of online malware detection with performance counters," in *Proc. 40th Annu. Int. Symp. Comput. Archit.*, 2013, pp. 559–570.
5. M. G. Elfeky, W. G. Aref, and A. K. Elmagarmid, "Warp: Time warping for periodicity detection," in *Proc. 5th IEEE Int. Conf. Data Mining*, 2005, pp. 138–145.
6. H. Fang, S. S. Dayapule, F. Yao, M. Doroslovački, and G. Venkataramani, "Prefetch-guard: Leveraging hardware prefetches to defend against cache timing channels," in *Proc. IEEE Int. Symp. Hardware Oriented Secur. Trust*, 2018, pp. 187–190.
7. M. Ferdman *et al.*, "Clearing the clouds: A study of emerging scale-out workloads on modern hardware," *ACM SIGPLAN Notices*, vol. 47, pp. 37–48, 2012.
8. J. L. Henning, "SPEC CPU2006 benchmark descriptions," *ACM SIGARCH Comput. Archit. News*, vol. 34, no. 4, pp. 1–17, 2006.
9. C. Hunger, M. Kazdagli, A. Rawat, A. Dimakis, S. Vishwanath, and M. Tiwari, "Understanding contention-based channels and using them for defense," in *Proc. IEEE 21st Int. Symp. High Perform. Comput. Archit.*, 2015, pp. 639–650.
10. Intel, "Intel-CMT-CAT Package," 2017. [Online]. Available: <http://https://github.com/01org/intel-cmt-cat>



11. V. Kiriansky, I. Lebedev, S. Amarasinghe, S. Devadas, and J. Emer, "DAWG: A defense against cache timing attacks in speculative execution processors," in *Proc. 51st Annu. IEEE/ACM Int. Symp. Microarchit.*, 2018, pp. 974–987.
12. F. Liu *et al.*, "CATalyst: Defeating last-level cache side channel attacks in cloud computing," in *Proc. IEEE Int. Symp. High Perform. Comput. Archit.*, 2016, pp. 406–418.
13. F. Liu, Y. Yarom, Q. Ge, G. Heiser, and R. B. Lee, "Last-level cache side-channel attacks are practical," in *Proc. IEEE Symp. Secur. Privacy*, 2015, pp. 605–622.
14. C. Maurice *et al.*, "Hello from the other side: SSH over robust cache covert channels in the cloud," in *Proc. Netw. Distrib. Syst. Secur. Symp.*, 2017, pp. 8–11.
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 *Proc. 16th ACM Conf. Comput. Commun. Secur.*, 2009, pp. 199–212.
16. G. Venkataramani, J. Chen, and M. Doroslovački, "Detecting hardware covert timing channels," *IEEE Micro*, vol. 36, no. 5, pp. 17–27, Sep.–Oct. 2016.
17. M. Yan, Y. Shalabi, and J. Torrellas, "Replayconfusion: Detecting cache-based covert channel attacks using record and replay," in *Proc. 49th Annu. IEEE/ACM Int. Symp. Microarchit.*, 2016, pp. 1–14.
18. F. Yao, M. Doroslovački, and G. Venkataramani, "Are coherence protocol states vulnerable to information leakage?" in *Proc. IEEE Int. Symp. High Perform. Comput. Archit.*, 2018, pp. 168–179.
19. F. Yao, H. Fang, M. Doroslovački, and G. Venkataramani, "COTSknight: Practical defense against cache timing channel attacks using cache monitoring and partitioning technologies," in *Proc. HOST*, 2019.

**Fan Yao** is an assistant professor of electrical and computer engineering at the University of Central Florida. His research interests include 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 include 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.