NEGATIVE CORRELATION, NON-LINEAR FILTERING, AND DISCOVERING OF REPETITIVENESS FOR CACHE TIMING CHANNEL DETECTION

Hongyu Fang[1], *Fan Yao*[2], *Miloš Doroslovački*[1], *Guru Venkataramani*[1]

[1]The George Washington University, Washington, DC
[2] University of Central Florida, Orlando, FL

ABSTRACT

Physically shared micro-architecture can be exploited by adversaries to communicate covertly via timing modulation without leaving any physical traces. Among different micro-architecture units, caches provide one of the largest attack surfaces because it is frequently accessed by multiple processes and it cannot be disabled. In this work, we show that by collecting cache occupancy traces, we can distinguish adversary from benign workloads through multiple signal processing techniques. When two processes are communicating by creating conflict misses, they would take cache memory space from each other. Consequently. the cache occupancies of two involved processes would be negatively correlated. Besides, the activity of the adversary in occupying the victim's cache space would be repetitive as a result of long-term, continuous transmission of secret information in a covert manner. By filtering the non-negatively correlated part and analyzing the repetitiveness of cache occupancy trace, we can achieve zero false negative rate and 4% false positive rate in cache timing channel detection.

Index Terms— Cache Timing Channel, Non-linear Filtering, Correlation, Fourier Analysis

1. INTRODUCTION

With rapid growth in the use of computer systems for storing and accessing user data, protecting sensitive information and shielding them from malicious entities is an important task for computer designers. The recent attacks exploiting micro-architecture [1, 2, 3] have further stressed the need for hardware and information security to be considered as an important and critical hardware design requirement. Among the many forms of information leakage attacks, timing channels are particularly notorious for their stealthy exfiltration of sensitive information, leaving no physical evidence for forensic examination. These timing channels simply rely on the modulation of access times on shared hardware resources, that cannot be easily audited by software security monitors.

This material is based on work supported by the US National Science Foundation under CNS-1618786, and Semiconductor Research Corporation contract 2016-TS-2684.

Among various architectural units in a microprocessor, cache memory is a critical performance resource that is organized in an array of rows (blocks of 64 bytes) and ways (sets of blocks within each row; typically 4 or 8). Caches are one of the largest attack surfaces due to two major reasons: 1. CPU caches (especially, last level cache) are one of the most commonly shared hardware resources among processing cores (pipelines), and hence different application domains typically have access to it. 2. Cache hierarchy is tightly coupled with processor pipelines that cater to data and instruction accesses, and cannot be simply disabled for security reasons.

As shown in Figure 1, there are two application domains (processes) involved in a cache timing channel attacks: the sender and the receiver of secrets. The sender has access to sensitive data that is typically not allowed to be communicated to processes with lesser privileges or outside entities as per the system security policy. In other words, the sender is an insider process that is looking to leak secrets. The receiver is a process that has lesser privileges and acts as a spy to receive sensitive information stealthily. Any software-based communication between sender and receiver is prohibited by Operating Systems due to the sender having access to sensitive data, and hence, the sender and receiver exploit the shared cache through manipulating its access timing in an orchestrated manner in order to communicate indirectly.

As shown in Figure 2, at the beginning of transmission, the receiver occupies all of the cache ways with its own memory lines (prime phase). Then the sender either replaces receiver's memory lines with its own memory lines or stays idle. After sender's activity, the receiver accesses its memory lines again, and measures the access latency (probe phase). A high latency indicates that sender has replaced the memory lines, while a low latency means sender did not access the memory lines. The sender can encode bit by controlling receiver's access latency. For instance, the sender can encode bit '1' by

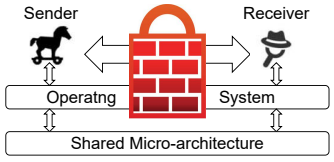


Fig. 1: The scenario of cache timing channel.

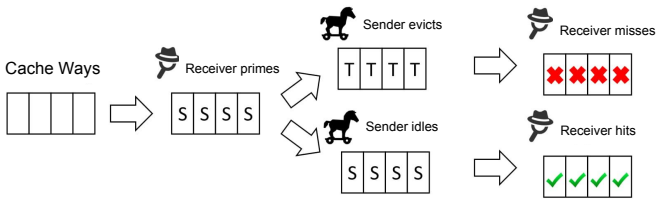
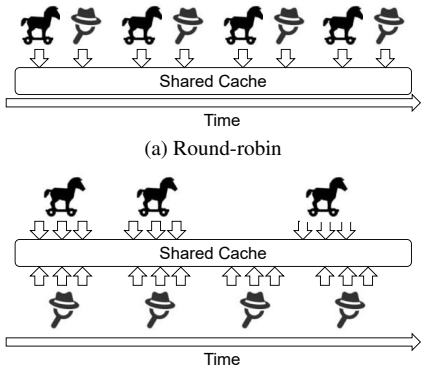


Fig. 2: Sender/Receiver activity in Cache Timing Channels

evicting receiver's memory lines and encode bit '0' by staying idle.

There are multiple protocols to implement cache timing channels. The operations shown in the Figure 2 is the basis of them. The adversary can exploit multiple cache sets to either transmit multiple bits in one prime+probe operation or enhance its robustness against background noise in cache. Besides, the sender and receiver may act in either round-robin fashion or in parallel as shown in Figure 3. To implement round-robin protocol, the sender and receiver synchronize before transmission. Then they act only after the other one finishes its activity. As for parallel protocol, the sender and receiver are not synchronized. The sender accesses the shared cache multiple times to guarantee the receiver can observe its activity. The receiver keeps accessing the shared cache in a constant frequency. In this paper, we will demonstrate the efficiency of our approach against both of these protocol classes.

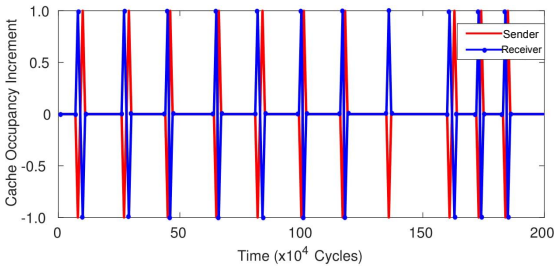
Prior studies have considered cache timing channel attacks that manipulate accesses on various cache levels [4, 5, 6, 7, 8, 9, 10]. CC-hunter [11] detects covert timing channel by capturing cache memory conflict misses between two processes, and needs modest hardware to provide fine-grained information about mutual cache evictions. Recent works identify cache timing channel through analyzing the cache miss patterns using existing performance counters [12, 13]. While these techniques may be more cost-effective without additional hardware modifications, such cache miss-based detection could be evaded by sophisticated adversaries which create self-eviction intentionally.



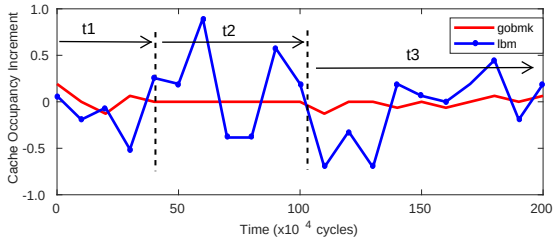
(a) Round-robin

(b) Parallel

Fig. 3: Protocols to implement Cache Timing Channels



(a) Cache Occupancy Changes for Sender and Receiver



(b) Cache Occupany Changes for Benign Workloads

Fig. 4: Cache occupancy changes of adversary and benign workloads show significantly different pattern.

In this paper, we propose the use of a new statistic–*cache occupancy*–that can be utilized for improved detection of cache timing channels without extra hardware modifications. Cache occupancy records the number of cache blocks owned by a specific process in a certain cache during the observation period. We find that, not only are the cache occupancy patterns between the covert communicators highly correlated, they also reveal repetitive and frequent pulses that can be recognized through signal processing techniques. To experimentally demonstrate our observation, we implement a cache timing channel attack and two cache-intensive benign workloads, and compare cache occupancy changes. Figure 4a shows a representative window capturing change in cache occupancy over time for cache timing channel attack. As we can see, the sender's cache occupancy gain is in proportion to receiver's loss and vice versa. In other words, each gain in cache occupancy by the sender is coupled by a corresponding loss of occupancy on the receiver's side, and vice versa. Moreover, we can also observe *a repetitive pattern of gain-loss events* in timing channels due to a series of consecutive transmission. Figure 4b shows that cache occupancy of benign workloads could be positively correlated or non-correlated overtime. The changes of cache occupancy of benign workloads do not have obvious repetitiveness. To detect potential cache timing channel attack between two processes, we focus on the negative correlation between two traces of change of cache occupancy while filtering the positive correlation segments in traces since they are not related to cache timing channel attack. Then we manage to reveal the repetitiveness between the negatively correlated traces. With this method, our detection mechanism achieve zero false negative rate and less than 4% false positive rate.

In summary, the major contributions of our article are:
1. We show that the adversaries' attempt to modulate cache access latencies using conflict misses generates distinct cache occupancy patterns during cache timing channels. By analyzing the cache occupancy profiles between two suspicious processes, cache timing channel may be inferred.
2. We design and demonstrate a detection mechanism of cache timing channel, and evaluate using real-world cache timing channel attacks. Our results show that our method can achieve zero false negative rate and less than 4% false positive rate.

2. DETECTION STEP

To detect cache timing channel, we collect the raw cache occupancy traces of all applications. All traces are separated into n windows and analyzed pairwise. x_i and y_i ($0 \leq i \leq n-1$) are the cache occupancy vectors obtained for applications X and Y, respectively (we assume that there are $p+1$ samples within each window). As discussed in Section 1, the covert communication would have suspicious pattern of cache occupancy changes. The cache occupancy changes of two applications are computed by:

$$\begin{aligned} \Delta x_{i,j} &= x_{i,j+1} - x_{i,j} \\ \Delta y_{i,j} &= y_{i,j+1} - y_{i,j} \end{aligned} \tag{1}$$

where $x_{i,j}$ and $y_{i,j}$ are the j^{th} samples ($0 \leq j \leq p-1$) in the i^{th} window for applications X and Y.

During communication, the sender and receiver would evict each other's memory lines from cache. One application would gain cache occupancy while the other one is losing the same amount of it. To extract this essential pattern, we analyze the negative correlation between cache occupancy traces of the applications. There would be multiple applications running together in a machine at the same time. To filter the noise effects from background applications, we take the product (z_i) of Δx_i and Δy_i and zero-out all non-negative values that do not correspond to gain-loss swing patterns in cache occupancy:

$$z_{i,j} = \begin{cases} \Delta x_{i,j}\Delta y_{i,j} & , \ \Delta x_{i,j}\Delta y_{i,j} < 0 \\ 0 & , \ \Delta x_{i,j}\Delta y_{i,j} \geq 0. \end{cases} \tag{2}$$

Equation (2) captures the interacting behavior between the two applications and filter the background noise that is not related to potential cache timing channels. If cache occupancy of one application changes while cache occupancy of the other one remains the same, they cannot communicate because the application is either occupying empty cache or being influenced by a third-party application. The product at the point would be zero. The cache occupancy change in a same direction indicates the influence of third-party application. In this case, the cache access latency of the pair of applications is influenced by the background instead of the other one in the pair, hence they cannot communicate. The product at the point would be positive and set to zero according to Equation (2).

The series z_i contains information about mutual eviction behavior between the two applications. In the most cases, the communication would contains multiple bits. The cache timing channel would involve multiple gain-loss pattern as shown in Figure 4a. The next step now is to check the repetitiveness of z series. The autocorrelation function r_i of z_i is computed by:

$$r_i(m) = \begin{cases} \sum_{j=0}^{p-m-1} z_{i,j} z_{i,j+m} & , \ m \geq 0 \\ r_i(-m) & , \ m < 0 \end{cases} \tag{3}$$

where m is the lag of series z_i and $m \in [-p+1, p-1]$. The autocorrelation function is normalized by:

$$r'_i(m) = \frac{r_i(m)}{\sqrt{(\sum_{j=0}^{p-1} \Delta x_{i,j}^4)(\sum_{j=0}^{p-1} \Delta y_{i,j}^4)}}. \tag{4}$$

$r'_i(0)$ would equal to 1 when the traces of cache occupancy changes are strictly linearly dependent according to the Cauchy-Schwarz Inequality [14]. $r'_i(0)$ would be close to zero if the traces of cache occupancy changes lacks linear dependency.

The benign applications may also have short-term influence on the cache occupancy. To cancel the influence of the background and benign applications, we take average of the autocorrelation computed in every time window. The mean autocorrelation function is defined as:

$$r'(m) = \frac{1}{n} \sum_{i=0}^{n-1} r'_i(m). \tag{5}$$

When the lag value (m) increase from 0 to half of the length of the complete pattern (wavelength, m_w), the gain-loss pattern would begin to mismatch. Consequently, $r'(m)$ would begin to decrease. When the lag m increase from $m_w/2$ to m_w, some of the patterns would rematch and $r'(m)$ would rise back. Note that there still might exist a small offset in the repetitive pattern, and this may cause $r'(m_w)$ to be not as high as $r'(0)$. However, $r'(m_w)$ is extremely likely to be a local maximum in the presence of timing channel activity. As m increases further, the cycle of pattern mismatch and rematch would begin to appear repeatedly.

Fourier transform is a powerful tool to extract the repetitive patterns in signals. We compute discrete Fourier transform of the autocorrelation function r':

$$R(k) = \sum_{m=-p+1}^{p-1} r'(m) W_{2p-1}^{m \cdot k} \tag{6}$$

where $W_{2p-1} = e^{-2\pi i/(2p-1)}$ and i is the imaginary constant ($i^2 = -1$). Here R is the *power spectrum* of z. The

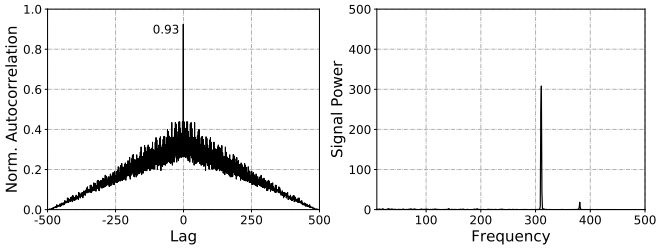


Fig. 5: Normalized autocorrelation and power spectral density (excluding zero-frequency) of cache timing channel

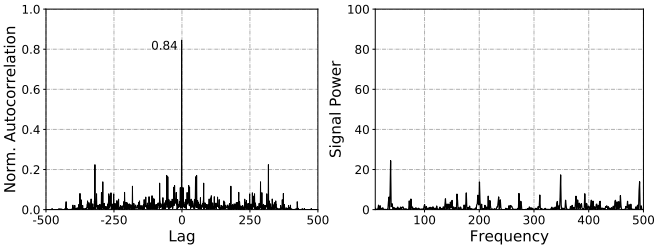


Fig. 6: Normalized autocorrelation and power spectral density (excluding zero-frequency) of Benign Workloads (soplex, omnetpp)

presence of a single or equally-spaced multiple spikes with concentrated (very high) signal power outside of frequency 0 in R indicates repetitive pattern in the underlying sequence. Note that this is a typical characteristic of timing channels.

In theory, one may think of a sender-receiver pair that pseudo-randomizes the intervals between two consecutive bits to obscure their communication pattern. However, cache timing channels with randomized bit intervals are very hard to synchronize at these random times in a real system environment amidst noise stemming from hardware, Operating System and external processes. As such, these attacks (if at all, possible) may be subject to high transmission errors. We are not aware of any such cache attacks with pseudo-random intervals in the literature. Even in such hypothetical cases, the repetitive swing pattern can be recovered with proper signal filtering (See Section 4.2).

3. EXPERIMENTAL SETUP

Our experimental testbed is an Intel Xeon V4 with 20 Last Level Cache (LLC) slices, and each LLC slice has 20×2048 64-byte blocks.

We run sender and receiver with two benign workloads from SPEC2006 [15] to demonstrate that our work can distinguish cache timing channel in a complicated environment. We run four variants of cache timing channel protocols as we introduced in Section 1: single-group round-robin, single-group parallel, multi-group round-robin and multi-group parallel. Each variant is configured to perform the prime+probe attack using 32 cache sets. We implement cache timing channel protocols as we discuss in Section 1.

To evaluate our proposal on benign workloads, we utilize benign workloads from SPEC2006. We run combinations of four workloads with different cache-intensity level. The four workloads are deployed on different cores of the processor.

4. EVALUATION

4.1. Efficiency of Proposed Method

As shown in Figure 5, the zero-lag normalized autocorrelation of cache timing channel is closed to 1 and the peak signal power density in the frequency domain is higher than 300. For 80% of benign workloads, the zero-lag normalized autocorrelation is lower than 0.5. Even some of the benign workloads show high zero-lag normalized autocorrelation as presented in Figure 6, the peak signal power density would be much lower than those in the case of cache timing channel because two benign benchmarks would not have repetitive mutual behavior. Our result shows that *all* cache timing channel variants can be detected by our method if we set the signal power density threshold as 50.

Figure 7 shows the ROC curve of the proposed method. The result shows that a vast majority of benign workloads do neither exhibit strong negative correlation in terms of cache occupancy change nor high repetitiveness of cache occupancy gain-loss pattern. All the attacks can be detected for the false positive rate as low as 4%.

4.2. Discussion: Transmission at Random Interval

In theory, sophisticated adversaries may use randomized interval times between bit transmissions. Let us imagine a sender and receiver that setup a pre-determined pseudo-random number generator to decide the next waiting period before bit transmission. Even if such attacks were feasible, our method can be adapted to recognize them through a signal pre-processing procedure called *time warping* [16, 17], that removes irrelevant segments from the occupancy traces (for which $\Delta x, \Delta y$ are 0 in Equation (1)) and aligns the swing patterns. After this step, the periodic patterns are reconstructed, and the cadence of cache accesses from adversaries will be recovered.

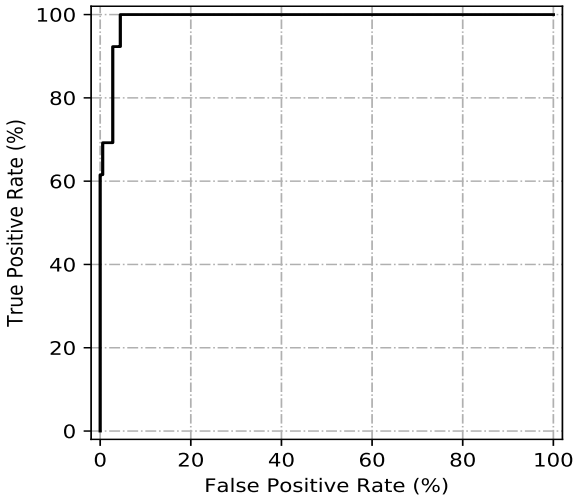


Fig. 7: ROC curve of the proposed detection method.

5. REFERENCES

[1] Google Project Zero, "Reading privileged memory with a side-channel," 2018, https://googleprojectzero.blogspot.com/2018/01/reading-privileged-memory-with-side.html.

[2] Fan Yao, Guru Venkataramani, and Miloš Doroslovački, "Covert timing channels exploiting non-uniform memory access based architectures," in *Proceedings of the on Great Lakes Symposium on VLSI 2017*. ACM, 2017, pp. 155–160.

[3] Murugappan Alagappan, Jeyavijayan Rajendran, Miloš Doroslovački, and Guru Venkataramani, "Dfs covert channels on multi-core platforms," in *Very Large Scale Integration (VLSI-SoC), 2017 IFIP/IEEE International Conference on*. IEEE, 2017, pp. 1–6.

[4] Fangfei Liu, Yuval Yarom, Qian Ge, Gernot Heiser, and Ruby B Lee, "Last-level cache side-channel attacks are practical," in *Proceedings of Symposium on Security and Privacy*. IEEE, 2015, pp. 605–622.

[5] Fan Yao, Miloš Doroslovački, and Guru Venkataramani, "Are Coherence Protocol States Vulnerable to Information Leakage?," in *Proceedings of the International Symposium on High Performance Computer Architecture*. IEEE, 2018.

[6] Hongyu Fang, Sai Santosh Dayapule, Fan Yao, Miloš Doroslovački, and Guru Venkataramani, "Prodact: Prefetch-obfuscator to defend against cache timing channels," *International Journal of Parallel Programming*, pp. 1–24, 2018.

[7] Fan Yao, Miloš Doroslovački, and Guru Venkataramani, "Covert timing channels exploiting cache coherence hardware: Characterization and defense," *International Journal of Parallel Programming*, pp. 1–26, 2018.

[8] Guru Venkataramani, Jie Chen, and Miloš Doroslovački, "Detecting hardware covert timing channels," *IEEE Micro*, vol. 36, no. 5, pp. 17–27, 2016.

[9] Hongyu Fang, Sai Santosh Dayapule, Fan Yao, Miloš Doroslovački, and Guru Venkataramani, "A noise-resilient detection method against advanced cache timing channel attack," in *Proceedings of Asilomar Conference on Signals, Systems, and Computers*, 2018.

[10] Hongyu Fang, Sai Santosh Dayapule, Fan Yao, Miloš Doroslovački, and Guru Venkataramani, "Prefetch-guard: Leveraging hardware prefetches to defend against cache timing channels," in *2018 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*. IEEE, 2018, pp. 187–190.

[11] Jie Chen and Guru Venkataramani, "CC-hunter: Uncovering covert timing channels on shared processor hardware," in *Proceedings of International Symposium on Microarchitecture*. IEEE, 2014, pp. 216–228.

[12] Marco Chiappetta, Erkay Savas, and Cemal Yilmaz, "Real time detection of cache-based side-channel attacks using hardware performance counters," *Applied Soft Computing*, vol. 49, pp. 1162–1174, 2016.

[13] Mathias Payer, "Hexpads: a platform to detect stealth attacks," in *Proceedings of International Symposium on Engineering Secure Software and Systems*. Springer, 2016, pp. 138–154.

[14] Alberto Leon-Garcia and Alberto Leon-Garcia, *Probability, statistics, and random processes for electrical engineering*, Pearson/Prentice Hall 3rd ed., 2008.

[15] John L Henning, "SPEC CPU2006 benchmark descriptions," *ACM SIGARCH Computer Architecture News*, vol. 34, no. 4, pp. 1–17, 2006.

[16] John R Deller Jr, John G Proakis, and John H Hansen, *Discrete time processing of speech signals*, Prentice Hall PTR, 2000.

[17] M. G. Elfeky, W. G. Aref, and A. K. Elmagarmid, "Warp: Time warping for periodicity detection," in *IEEE International Conference on Data Mining*, 2005, pp. 1–8.