A Noise-resilient Detection Method against Advanced Cache Timing Channel Attack

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

***Abstract*—Recent researches show that computers which are physically shared by multiple users are vulnerable to microarchitecture-based information leakage. Among all microarchitecture components, cache provides the largest attack surface. Cache timing channels manipulate the cache access latency to leak information leaving no physical trace. To mitigate cache timing channels, various detection methods are proposed. However, with the knowledge of existing detection methods, an advanced adversary can intentionally inject noise to evade detection. For example, the detection based on correlation method which extracts the repetitive behavior of cache timing channels can be evaded by randomizing the gap between information transmitting and receiving activity. The classification based detection would be obfuscated if adversary imitate the behavior of benign applications. We propose a novel noise-resilient detection method which focuses on the dependency between behavior of two processes. For each process, we define a group of events and track the conditional probability of every event given the appearance of the events from another process. With this method, we are able to detect the existence of cache timing channels. Our detection method is hard to evade because the dependency of cache behavior is necessary for any communication through cache timing channels.**

I. Cache Timing Channel

With the development of cloud computing, more and more developers deploy their processes and data on shared platforms to reduce infrastructure cost and boost performance. The scenario where multiple processes from different owners running on a same physical machine becomes ubiquitous. To prevent information leakage, platform operators usually prohibit direct communication of processes from different user domains using Operating Systems or Hypervisors. However, as long as the hardware is still shared, the microarchitecture covert and side channels are still potential security harzard for information leakage. Among all microarchitecture covert and side channels, cache timing channel [8], [11], [12], [13], [15], [16] is one of the most notorious because cache provides the largest attack surface and the channels cannot be mitigated by software-based approach. Cache timing channel usually involves two processes: trojan/victim and spy. For the side channel scenario, a victim unconciously leaks informaiton through its cache access pattern while a malicious spy tries to track victim's cache access pattern and finally reveals the victim's secret. In a covert channel scenario, a trojan which has access to secret tries to leak information intentionally by modulating cache access and the spy manages to receive trojan's information by observing its cache access pattern. We note that the only difference between trojan and victim is whether the process leaks information on purpose. The behavior of trojan and victim is simillar. In the following paper, we use the word *trojan* to represent both *trojan* in the covert channel scenario and *victim* in the side channel scenario.

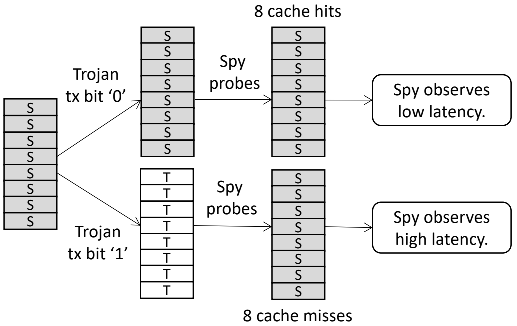


Fig. 1: Cache block replacement in Prime+Probe Attack.

Among various implementation of cache timing channels, the *prime+probe* is the most common protocol because it does not have prerequisites beyond physically shared cache. In prime+probe attack, the trojan manipulate the spy's cache access latency by creating conflict miss. Conflict miss is caused by cache block replacement. When one process accesses a memory line from DRAM which is mapped to a full cache set, the newly accessed memory line would be brought to cache and replace one cache block inside the cache set. If a process access the replaced cache block, it would suffer from cache conflict miss and observe higher latency.

As shown in Figure 1, to launch prime+probe attack, the spy firstly primes every cache block in cache sets with its own memory lines, then the trojan transmits bit '1' by evicting all cache blocks owned by spy and transmits bit '0' by staying idle. After trojan's activity, the spy probes the memory lines which were used to prime and measures the access latency. If the trojan transmits bit '1', the spy would observe high latency because of conflict misses. If the trojan transmits bit '0', the spy would observe low latency because all its memory lines remained in cache. From the observed cache access latency, the spy can infer the trojan's behavior and decipher the information from trojan. The measured cache access latencies of spy during transmission are shown in Figure 2, the latencies

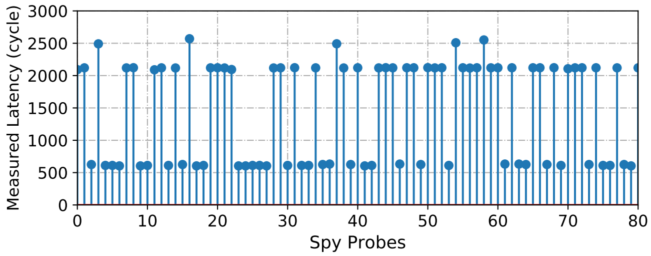
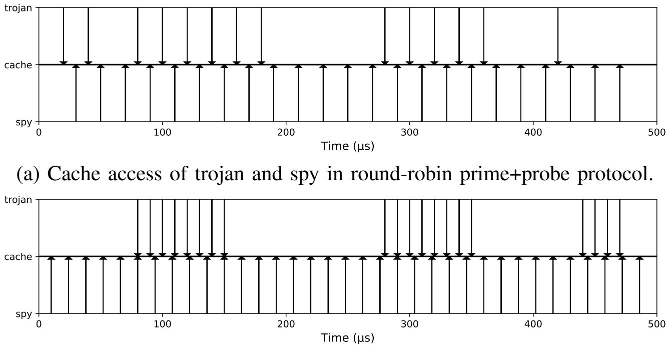


Fig. 2: Cache access latency observation of spy in Prime+Probe attack. The observed latencies which are higher than 2000 cycles indicate the bits '1' from trojan and the observed latencies which are lower than 1000 cycles indicate the bits '0' from trojan.



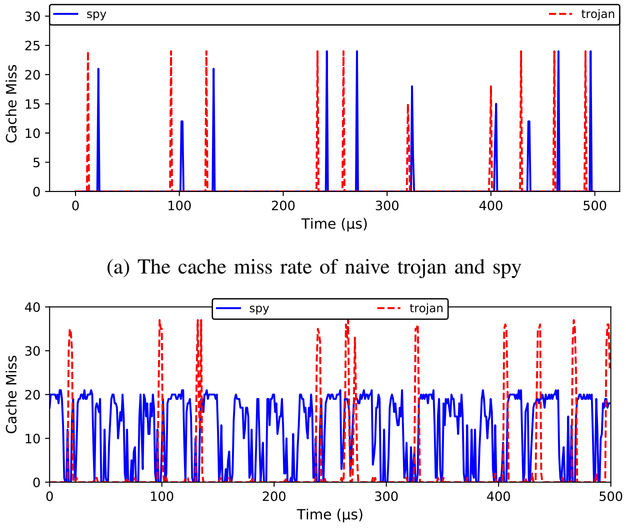
(a) Cache access of trojan and spy in round-robin prime+probe protocol.

(b) Cache access of trojan and spy in parallel prime+probe protocol.

Fig. 3: Cache access pattern of different cache timing channel protocols

of spy are either larger than 2000 cycles indicating bit '1' or lower than 1000 cycles indicating bit '0'.

The prime+probe based cache timing channel can be implemented using round-robin or parallel protocols. The cache access behavior of trojan and spy using round-robin cache timing channel is shown in Figure 3a. The spy synchronizes with trojan and only accesses cache after the trojan finishes encoding. The round-robin protocol guarantee high bit rates and low noise but it may take time for them to synchronize before communication. The cache access behavior of trojan and spy running parallel attack is shown in Figure 3b. While implementing parallel protocol, the trojan and spy act at the same time. The spy access cache more frequently than trojan to assure that it observes complete trojan's behavior. The parallel protocol does not need synchronization of trojan and spy and its transmission rate is lower than round-robin protocol.



(a) The cache miss rate of naive trojan and spy

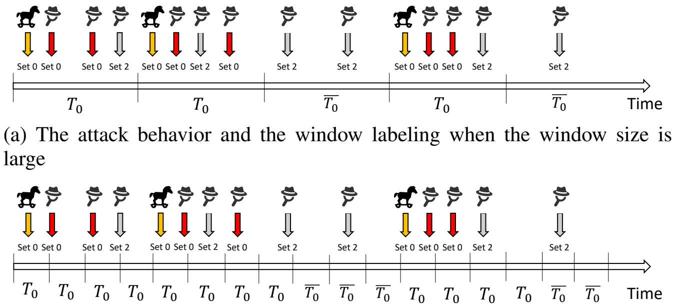
(b) The cache miss rate of advanced trojan and spy

Fig. 4: Advanced spy injects noise to make itself uncorrelated with trojan and hide its repetitive behavior.

II. Proposed Detection Approach

A. Existing Detection Methods and Limitation

To mitigate cache timing channels, various detection methods are proposed [14], [2], [7], [3], [4], [9], [10], [5], [6]. The existing detection methods mainly include two categories: 1. Correlation-based detection [14], [2] which aims at extracting the repetitive pattern of trojan and spy caused by information transmission, 2. Classification-based detection [7] which focuses on comparing characteristics of suspicious process and baseline benign processes. These detection methods can efficiently detect adversaries which are not aware of the existence of detection. However, with the knowledge of detection strategy, the smarter adversaries could manage to evade detection by noise injection and timing randomization. The repetitive pattern would be obfuscated by timing randomization so that correlation-based methods would fail. With noise injection, the adversary can pretend a benign cache-intensive process to evade classification-based detection. To illustrate how can an advanced adversary evade detection, we implemented an advanced cache timing channel on Gem5 and compared the observed cache miss of advanced cache timing channel with the naive cache timing channel.. The advanced spy and trojan exploit 16 cache sets to communicate using prime+probe protocol. To evade potential detection, the spy inflates its cache miss in another 16 cache sets. We note the cache sets exploited to communicate as *communication sets* and the cache sets for noise injection as *noise sets*. Besides noise injection, spy also randomize the timing of cache access. The number of misses of trojan and spy are shown in Figure 4. As shown in Figure 4a, the miss rates of naive trojan and spy are correlated. When trojan suffers from a number of cache misses, the spy also suffers from cache misses. When trojan doesn't have cache miss, the spy would not suffer from cache miss either. The misses of advanced trojan and spy are shown in Figure 4b. The spy's misses are not correlated with trojan's and its behavior is not repetitive compared to Figure 4a, hence the accuracy of correlation-based detection would degrade. Besides, the spy can create additional noise to pretend to be a cache-intensive benign application so that the classification-based detection would fail to detect it.


Set 0 Set 0 Set 0 Set 2 Set 0 Set 0 Set 2 Set 0 Set 2 Set 2 Set 0 Set 0 Set 0 Set 2 Set 2
T_0 T_0 $\overline{T_0}$ T_0 $\overline{T_0}$
Time
Set 0 Set 0 Set 0 Set 2 Set 0 Set 0 Set 2 Set 0 Set 2 Set 2 Set 0 Set 0 Set 0 Set 2 Set 2
T_0 T_0 T_0 T_0 T_0 T_0 T_0 T_0 $\overline{T_0}$ $\overline{T_0}$ $\overline{T_0}$ T_0 T_0 T_0 T_0 $\overline{T_0}$ $\overline{T_0}$
Time

(a) The attack behavior and the window labeling when the window size is large

(b) The attack behavior and the window labeling when the window size is small

Fig. 5: The labeling of time windows for cache timing channel detection

B. Conditional Probability in Cache Timing Channels

The communication of trojan and spy through cache timing channels is based on the fact that the trojan's behavior can indirectly influence the spy's observation. The trojan's access to cache would evict spy's memory lines from cache sets and result in the cache miss of spy. If trojan did not access cache, the spy would not suffer from cache miss. The spy's cache miss and hit in the communication sets have a significant dependency on trojan's cache miss and hit in the same sets. This property makes conditional probability a potential noise-resilient metric to detect advanced cache timing channel. During communication, the probability of spy's cache miss in communication sets given the appearance of trojan's cache miss in the same group of sets would be significantly higher than overall probability of spy's miss. On the other hand, when trojan did not access the communication sets, the probability of spy's misses in the sets would be lower. The conditional probability would not be influenced by the obfuscation from adversary or third-party application because the timing randomization or the noise does not change the dependency of spy's cache misses in communication set on trojan's cache misses in the same group of cache sets.

We define the event of spy suffering from cache miss in cache set i as s_i. For trojan, we define the condition where trojan suffer from cache miss in cache set j within a predetermined time window as t_j and the condition where trojan did not suffer from cache miss in the cache set j within a predetermined time window as $\overline{t_j}$. For i, j belonging to communication sets, we will observe:

$$|P(s_i|t_j) - P(s_i|\overline{t_j})| > \epsilon \tag{1}$$

where ϵ is a threshold of change of conditional probability. If cache set p belongs to noise set, the $P(s_p|t_j)$ would be approimately equal to $P(s_p|\overline{t_j})$ since the cache miss of spy in the noise sets are created by spy itself and independent to trojan's behavior.

C. Using Cache Miss Rate to Estimate Conditional Probability

To estimate the conditional probability we mention in the Section II-B by observing the cache miss pattern of trojan and spy, we divide the time axis into equal-size time windows. We add labels to each time window based on the following rule:

- *Trojan Label:* If the trojan suffered cache miss in the cache set j in the last M time window, add label T_j to current time window. Otherwise, add label $\overline{T_j}$ to the time window.
- *Spy Label:* If the spy suffered cache miss in the cache set j in the time window, add label S_j to the time window.

Figure 5a shows an example of time window sampling for cache timing channel. In this scenario, the time window is relatively large so that the number of window labeled as T_0 is set to one. The trojan and spy exploit cache set 0 and 1 to communicate while spy imitates benign application on the other cache set, for instance cache set 2. The trojan's cache miss is presented by the orange arrows. The spy's adversary behavior is represented by red arrows and its benign behavior is represented by gray arrows. We can observe that the spy has cache miss in the cache set 0 and 1 in every time window with label T_0 and has no cache miss in the same cache sets in every time window with label $\overline{T_0}$.

The conditional probabilities in Equation 1 can be estimated by the count of time windows:

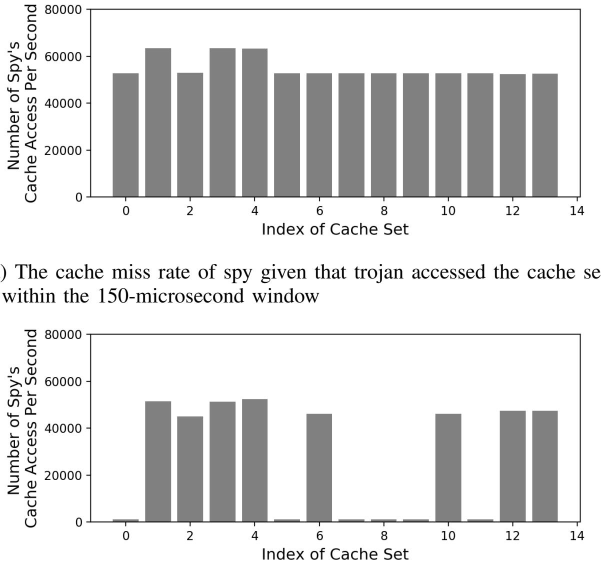
$$\begin{aligned} P(s_i|t_j) &= \frac{\#\text{time windows with both labels: } S_i, T_j}{\#\text{time windows with label } T_j} \\ P(s_i|\overline{t_j}) &= \frac{\#\text{time windows with both labels: } S_i, \overline{T_j}}{\#\text{time windows with label } \overline{T_j}}. \end{aligned} \tag{2}$$

As shown in Figure 5b, when the time interval is small enough such that at most one cache miss could happen within one time interval, the first conditional probability can be represented using cache miss rate:

$$P(s_i|t_j) = R_{ij} t_{window} \tag{3}$$

where R_{ij} is the number of spy's cache misses in cache set i in time windows with label T_j divided by the time elapse of all time windows with label T_j and t_{window} is the size of each time window. The second conditional probability in (2) can be expressed in the similar way where R_{ij} is replaced by $\overline{R_{ij}}$. We note that it is more practical to track cache miss rate in real machine. In the following paper, we will use cache miss rate to estimate conditional probabilities we mentioned in (1)

To illustrate the efficiency of detection, we do the cache miss rate analysis on the advanced cache timing channel introduced the Section II-A. Figure 6a shows the cache miss rate of spy given that trojan accessed the cache set 0 within 150-microsecond time interval. And the Figure 6b shows the cache miss rate of spy given that trojan did not access the cache set 0 within the 150-microsecond window. We can observe that the conditional probability on set 0, 5, 7, 8, 9, 10 and 11 change significantly when trojan has different behavior indicating they are communication sets. On the other hand, the conditional probabilities of the other sets do not change a lot when trojan access or not cache set 0, indicating that they are noise sets.



(a) The cache miss rate of spy given that trojan accessed the cache set 0 within the 150-microsecond window

(b) The cache miss rate of spy cache miss given that trojan did not access the cache set 0 within the 150-microsecond window

Fig. 6: Cache Miss Rate of spy in round-robin attack

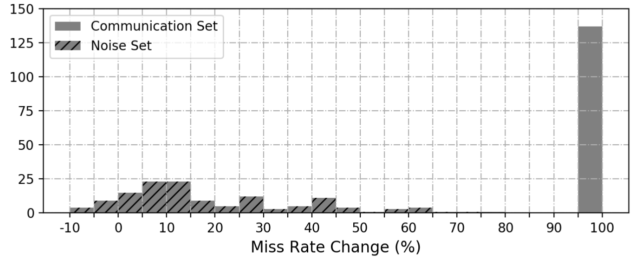


Fig. 7: Histogram of cache miss rate decrease of spy when trojan does not access the cache set 0 within given time window.

III. Experimental Setup

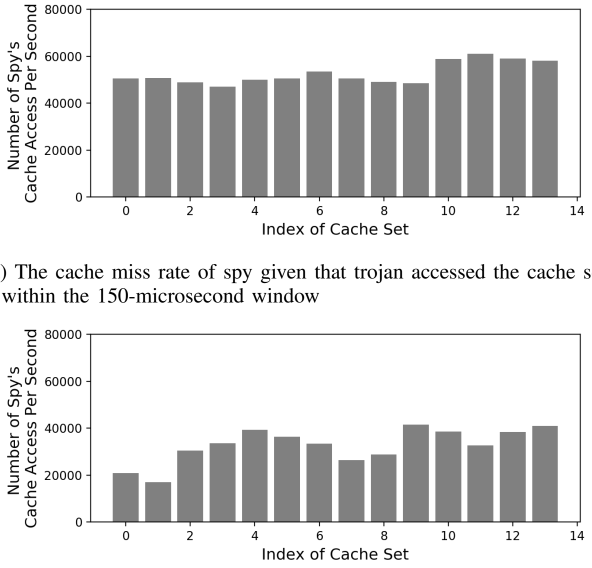
We implement the cache timing channel attack using *Gem5* [1], a cycle-accurate, full-system simulator. We configure Gem5 with four x86 cores, 32 KB private L1 and 512 KB, 8-way shared L2 caches. All the experiments are run on full system mode under Linux kernel version 2.6.32. We collect the cache access traces from simulator and implement the detection methods on the traces.

We deploy the trojan and spy processes on separated CPU cores. The trojan and spy communicate in either round-robin or parallel fashion as we discussed in Section I. For the round-robin attack, the trojan and spy use 7 communication sets. The spy injects noise in 7 noise sets by creating self evictions. For the parallel attack, the trojan and spy use 14 communication sets to transmit information.

IV. Evaluation

A. Round-robin Noisy Attack

As we discussed in Section II-B, the cache miss rates of spy in communication sets drop significantly when trojan does not access communication sets while the cache miss rates in noise sets drops much less than those in communication sets. Figure 7 shows the histogram of cache miss rate decrease of spy given different behaviors of trojan. The miss rate decrease is normalized with the cache miss rate when the trojan accesses the communication sets within a given time interval. Cache miss rate drops 100% in communication sets which indicates the spy's behavior is strongly dependent on trojan's activity. The cache miss decrease in noise set ranges from -10% to 50% which is much lower than the cache miss rate decrease in communication sets. The miss rate decrease can efficiently distinguish communication sets from noise sets and finally capture the covert communication in noisy background. We note that as demonstrated in previous work [2], the benign applications would not have this correlated behavior, so our design is also able to identify adversary process when it is running with benign workloads.



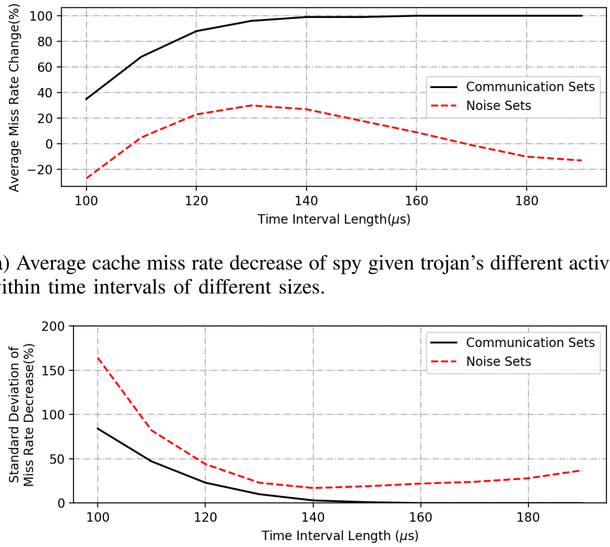
(a) The cache miss rate of spy given that trojan accessed the cache set 0 within the 150-microsecond window

(b) The cache miss rate of spy cache miss given that trojan did not access the cache set 0 within the 150-microsecond window

Fig. 8: Cache Miss Rate of spy in parallel attack

B. Parallel Attack

The cache miss rates of spy given different trojan activity implementing parallel protocol are shown in Figure 8. The cache miss rate of spy in cache set 0 drops 60% when trojan does not access the same cache set in the last 150 microsecond. The cache miss rate decrease in other cache sets may drop less than the cache miss rate in cache set 0 because the activities of trojan and spy are not synchronized. When spy is scanning the cache set 1 to 13, it is possible that the trojan has already start next transmission in cache set 0. For parallel attack, examining the spy's cache miss rate given different trojan's activity in the same cache sets can still distinguish the adversary behavior.



(a) Average cache miss rate decrease of spy given trojan's different activity within time intervals of different sizes.

(b) Standard deviation of cache miss rate decrease of spy given trojan's different activity within time intervals of different sizes.

Fig. 9: Cache miss rate decrease of spy in round-robin attack as function of the size of influence interval of trojan activity.

C. *Time Interval Analysis*

In the previous experiments, we have analyzed the cache miss rate decrease of spy given trojan's different activity within 150 microseconds. The choice of the time interval length would influence the detection accuracy. Figure 9 shows the mean and standard deviation of cache miss rate decrease of spy given the trojan's different activity within time intervals of different sizes. As shown in Figure 9a, the average miss rate decrease in communication sets is always larger than the average miss rate decrease in noise sets despite the time interval. The miss rate decrease in communication sets becomes larger as the time interval become larger. When the time interval is shorter than the gap between trojan and spy's activity, the communication-related cache miss of spy would not appear in the time interval so that the cache miss rate decrease could not capture the dependent behavior between trojan and spy. The miss rate decrease of communication set becomes larger than 90% when the time interval is larger than 125 microsecond which is sufficient for cache timing channel detection. As shown in Figure 9b, the standard deviation of miss rate decrease becomes lower when the time interval length increases which proves that the time interval length should be longer than a threshold to make noise sets and communication sets distinguishable.

V. CONCLUSION

In this paper, we propose a novel noise-resilient detection method based on conditional probability to prevent information leakage through cache timing channel. We demonstrated our work on both round-robin and parallel cache timing channel protocol. Our experiments shows that the proposed method is able to distinguish cache sets which are exploited by adversary in noisy background by calculating cache miss rate in different condition.

VI. ACKNOWLEDGMENT

This manuscript is based on work supported by the US National Science Foundation under CAREER Award CCF-1149557 and grant CNS-1618786, and Semiconductor Research Corp. (SRC) contract 2016-TS-2684. Any opinions, findings, conclusions, or recommendations expressed in this article are those of the authors, and do not necessarily reflect those of the NSF or SRC.

REFERENCES

[1] Nathan Binkert, Bradford Beckmann, Gabriel Black, Steven K Reinhardt, Ali Saidi, Arkaprava Basu, Joel Hestness, Derek R Hower, Tushar Krishna, Somayeh Sardashti, et al. The gem5 simulator. *ACM SIGARCH Computer Architecture News*, 2011.

[2] Jie Chen and Guru Venkataramani. Cc-hunter: Uncovering covert timing channels on shared processor hardware. In *IEEE/ACM International Symposium on Microarchitecture*, 2014.

[3] 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)*, pages 187–190. IEEE, 2018.

[4] 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*, pages 1–24, 2018.

[5] Vladimir Kiriansky, Ilia Lebedev, Saman Amarasinghe, Srinivas Devadas, and Joel Emer. Dawg: A defense against cache timing attacks in speculative execution processors.

[6] Fangfei Liu and Ruby B Lee. Random fill cache architecture. In *Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 203–215. IEEE Computer Society, 2014.

[7] Mathias Payer. Hexpads: a platform to detect stealth attacks. In *International Symposium on Engineering Secure Software and Systems*, pages 138–154. Springer, 2016.

[8] 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 *Conference on Computer and communications security*, 2009.

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

[10] Guru Venkataramani, Christopher J Hughes, Sanjeev Kumar, and Milos Prvulovic. Deft: Design space exploration for on-the-fly detection of coherence misses. *ACM Transactions on Architecture and Code Optimization (TACO)*, 8(2):8, 2011.

[11] Zhenghong Wang and Ruby B Lee. Covert and side channels due to processor architecture. In *Annual Computer Security Applications Conference*, 2006.

[12] Zhenyu Wu, Zhang Xu, and Haining Wang. Whispers in the hyper-space: high-speed covert channel attacks in the cloud. In *USENIX Security Symposium*, 2012.

[13] Yunjing Xu, Michael Bailey, Farnam Jahanian, Kaustubh Joshi, Matti Hiltunen, and Richard Schlichting. An exploration of l2 cache covert channels in virtualized environments. In *ACM workshop on Cloud computing security workshop*, 2011.

[14] Mengjia Yan, Yasser Shalabi, and Josep Torrellas. ReplayConfusion: Detecting cache-based covert channel attacks using record and replay. In *IEEE International Symposium on Microarchitecture*, 2016.

[15] Fan Yao, Milos Doroslovacki, and Guru Venkataramani. Are coherence protocol states vulnerable to information leakage? In *High Performance Computer Architecture (HPCA), 2018 IEEE International Symposium on*, pages 168–179. IEEE, 2018.

[16] 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*, pages 155–160. ACM, 2017.