# Prefetch-guard: Leveraging hardware prefetchers to defend against cache timing channels

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*—**Cache timing channels are a form of information leakage that operate through modulating cache access latencies and ultimately exfiltrate sensitive user information to adversaries. Among the many forms of timing channels, covert channels are particularly dangerous as they involve two insider processes (trojan and spy) colluding with each other to send out sensitive information, and are often difficult to detect or prevent. In this paper, we propose Prefetch-guard, an efficient and low-cost mitigation mechanism against cache-based timing channels. Prefetch-guard leverages hardware prefetchers to obfuscate the effect of timing modulation intentionally created by the trojan and spy. Our detection mechanism identifies the target cache sets that are being exploited for information leakage, and cache blocks are prefetched to fuzz the pattern of cache misses and hits created to construct timing channel between the trojan and the spy. With prefetch-guard, we observe that the cache timing channels suffer a 53% bit error rate which makes it very hard or impossible for the spy to decipher any useful information.**

## I. Introduction

With a vast majority of computer users relying on shared computing platforms for data storage and service needs, information security has become an important issue. To prevent information leakage, Operating Systems and Hypervisors prohibit any direct communication between untrusted users or processes in different security domains. As software confinement mechanisms continue to improve, adversaries are turning to hardware resources for exploits. Among the many forms of information leakage through shared hardware, timing channels are particularly notorious for their stealthy exfiltration of sensitive information leaving no physical evidence for future forensic examination [8]. These hardware-based timing channels rely on the modulation of resource access timing such that sensitive data can be secretly transmitted. Note that timing channels can manifest either as *side channels*, where a benign victim unknowingly leaks sensitive data to a malicious spy, or as *covert channels*, where a malicious insider trojan process intentionally colludes with a spy process to manipulate access timing of a shared resource to exfiltrate secrets. Note that the system security policy explicitly prohibits any form of direct communication between trojan-spy pairs [9].

Caches are among the most exploited hardware structures for timing channel attacks because of the following reasons: 1. Caches are usually shared between multiple CPU cores, and hence, processes from different security domains have access to the same caches. 2. Caches cannot be usually disabled or isolated to avoid impacting application performance. Therefore, caches form an ideal target for timing channel exploits.

In this work, we propose Prefetch-guard, an efficient, low-cost approach to defend against cache timing channels by leveraging hardware prefetchers. Our solution analyzes the cache for suspicious cache access activity by using a low-cost trigger pattern detector. Prefetch-guard targets the suspicious cache sets, and then obfuscates any trojan-initiated timing modulation (corresponding to covert bit transmission). This is achieved through prefetching the cache blocks that counter any cache replacement (and non-replacement/hits) as instrumented by the trojan on suspected cache sets. The use of a front-end detector ensures that the performance of benign processes remain unaffected and the cache sets belonging to them are not targeted by Prefetch-guard.

Prefetch-guard provides two key advantages over prior solutions: 1. *Scalability*: Prefetch-guard targets misbehaving trojan-spy processes and the cache sets belonging to them. Therefore, regardless of the total number of processes running in the system, we can effectively annul the timing channel activity between any number of pairs without adversely affecting benign applications that utilize the rest of the cache sets. We note that cache partitioning-based defenses [20] are hard to scale because the number of cache partitions are limited. If the cache is partitioned heavily, benign processes may slow down because of insufficient cache capacity. 2. *Lower cost*: Prefetch-guard leverages existing hardware prefetchers, and makes minimal hardware changes to track conflict misses.

In summary, the major contributions of our paper are:

1) We propose Prefetch-guard, an efficient, low-cost and scalable solution to counter cache-based timing channels. Our approach leverages hardware prefetcher module to obfuscate the trojan-spy communication in a targeted manner such that the spy will not be able to correctly decipher the bits transmitted by the trojan.

2) We show the design of our Prefetch-guard framework that profiles for suspicious target cache sets, and illustrate obfuscation methods that perturb the cache access timing modulation orchestrated by the trojan. This effectively increases the error rate when the transmitted bits are deciphered by the spy, and disrupts the timing channel activity.

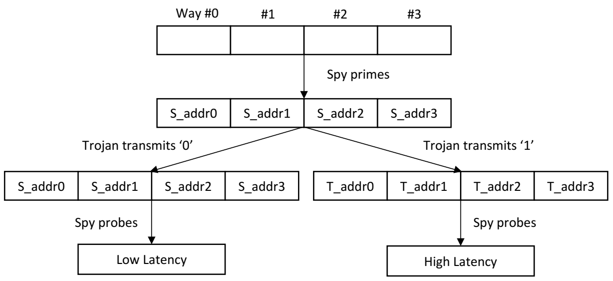


Fig. 1: Illustration of Prime+Probe cache attack. The spy first primes and fills the cache sets with its own data. Trojan may replace spy's contents depending on the transmitted bit. Spy probes infers the bit using cache access latency.

## II. Background & Related Work

### A. Cache Timing Channels

Timing channels usually involve two processes: trojan/victim and spy, where the spy learns of sensitive secrets from trojan/victim through timing modulation on caches. Access timing is altered through a pattern of cache hits and misses.

Among all implementations of cache timing channels, prime+probe techniques are among the most exploited class of attacks that do not have any need to have shared memory data (Figure 1). Therefore, we use these attacks in our study.

### B. Defenses Against Hardware Covert/Side Channels

Several side and covert channels have been studied by prior works. To name a few recent studies, power analysis [6], [16], program execution [13] or access latency [1], [7], [21], [22] are among a few examples. In many such channels, the adversary can reveal secrets about sensitive processes or endanger system security without leaving any trace. Prior works have proposed counter strategies for power and storage channels through memory safety and inspection [5], [15], [18]. For timing channels, mitigation techniques such as injecting noise may lead to severe performance degradation of all running processes.

Cache side- and covert timing channels have been demonstrated on real hardware in prior studies [2], [12]. Venkataramani et al. [17] propose a generic framework for cache covert timing channel detection using correlation between cache conflict misses. For L1 cache timing attack, Bao et al. [3] explore the implication of faster 3D integrated caches to perform low cost obfuscation. CATalyst [11] proposes a secure cache partition for applications where victim processes can voluntarily utilize such secure partitions.

Fuchs et al. [10] propose a disruptive prefetching scheme that utilizes existing prefetch policies to pollute caches and is aimed at L1 caches. In contrast, we perform targeted prefetching to counter cache blocks owned by the trojan and spy, and Prefetch-guard may be integrated with any cache.

### C. Prefetcher

Data prefetching has been utilized to bridge the performance gap created between processors and DRAM. The on-chip hardware prefetchers work by monitoring the cache misses, and predicting the memory addresses that satisfy CPU's data needs in the near future. The common types of hardware prefetchers based on spatial locality are *Stream*: which loads next sequential addresses in the page, and *Stride*: which brings the addresses at a fixed stride from the requested address. To satisfy temporal locality, prefetchers use global history buffer-based policy that predicts the next cache reference based on previous access pattern [14].

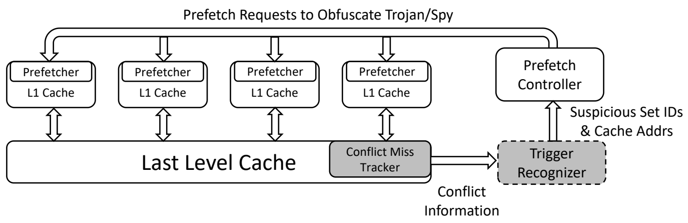


Fig. 2: Overview of Prefetch-guard design. Modifications needed for Prefetch-guard are shown using gray blocks, where solid frame denotes hardware add-ons and dotted frame denotes modules that can be implemented in software.

## III. Threat Model and Assumptions

Our attack model assumes that a trojan can access sensitive information that a spy is trying to steal. The spy and trojan run on different cores that share a cache (e.g., Last Level Cache or LLC) to implement their cache-based timing channels. Any form of inter-process communication between the trojan-spy is prohibited by the OS under the system security policy. Our threat model assumes the most elusive *Prime+Probe* technique (that does not require any shared memory between trojan and spy) to create conflict patterns for covert communication.

## IV. Prefetch-guard Design

The design of our Prefetch-guard framework involves three important modules: Conflict Miss Tracker, Trigger Pattern Recognizer, and Prefetch Controller as shown in Figure 2.

### A. Conflict Miss Tracker

Conflict misses occur exclusively in set-associative caches when blocks are pre-emptively replaced from the cache even before the full cache capacity is reached. That is, *if a core A's cache block is replaced by another core B prematurely, and when the core A accesses the same block again (which was recently replaced), a conflict miss occurs*. Conflict misses occur when many blocks that map to the same cache set are accessed successively, and the cache does not support enough associativity to accommodate all of the blocks. Note that such conflict misses would have never happened in a fully associative cache.

In order to track such conflict misses, a simple hardware buffer (Conflict Miss Tracker) maintains a list of addresses that are replaced during cache misses in LLC, along with the corresponding owner core ID for that block. If a currently replaced address in the cache is not in this hardware tracking buffer, it would be added. Upon every cache miss, the

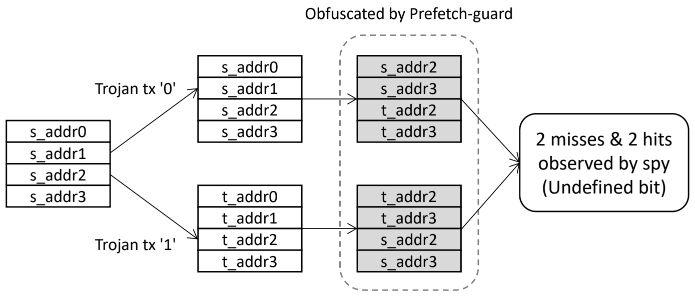


Fig. 3: Obfuscation of bits by Prefetch-guard

incoming address is checked against the buffer entries. If the incoming address to the cache is found in the buffer, we infer that this address was recently replaced by another cache block, and a conflict miss is recorded [19]. We note that the conflict miss tracker could achieve a high accuracy with less than 3% area overhead in L2 cache and 1.5% access time overhead.

Since the trojan and spy pairs will do the evictions repeatedly using multiple addresses to covertly communicate with each other, the identities of the trojan and victim can be easily tracked. The output of Conflict Miss Tracker is the list of evicted addresses, the owners of evicting and evicted memory addresses (likely trojan and spy processes).

### B. Trigger Pattern Recognizer

Prime+Probe typically involves three phases: 1) spy's prime, 2) trojan/victim's activity, and 3) spy's probe. Initially, the spy primes all of the cache sets by bringing its own blocks into those sets. Then the trojan transmits its encoded bits by evicting the spy's cache blocks or by staying idle. After trojan's activity, the spy probes the cache blocks it primed, and measures the access latency to infer the bits. We denote the trojan evicting spy's cache block as $trojan \rightarrow spy$. When the spy probes, it issues the same set of addresses used during prime phase, and measures cache access latency. Subsequently, the spy primes again to observe trojan's future transmissions, which we denote as $spy \rightarrow trojan$. In a successful Prime+Probe attack, we can observe repeated pattern of cache conflict misses in the form of $trojan \rightarrow spy$ and $spy \rightarrow trojan$. Once the rate of conflict misses exceeds a threshold, $T$ (determined empirically through observing the behavior of several benign workloads and cache timing channel exploits), the Trigger Pattern Recognizer module initiates the hardware prefetcher to launch counter-attack on the potential covert channel activity.

### C. Prefetch Controller

Based on the input from Trigger Pattern Recognizer module, the prefetch controller initiates prefetch requests to the L1 cache prefetcher hardware such that the underlying cache timing channel can be effectively annulled.

To thwart the trojan-spy communication, the prefetch controller needs information about the memory block addresses that the trojan and spy exploit to create conflict misses. During such covert communication, these memory addresses are frequently involved in several rounds of conflict misses, and therefore, would be recorded by Conflict Miss Tracker. After the suspicious cache sets are labeled by the Trigger Pattern Recognizer, these exploited addresses are sent to the prefetch controller. Then the prefetch controller issues requests to the L1 prefetcher to bring back memory addresses into the cache to disrupt spy's ability to correctly probe and decipher the transmitted bit.

Figure 3 shows our prefetch controller making the number of cache misses and hits observed by the spy to be the same, irrespective of trojan's activity. In our illustration, for a 4-way associative cache, the spy observes zero misses when trojan transmits bit '0' and four misses when trojan transmits bit '1'. In the absence of any defense, the spy could easily discern the bit because the latency difference between cache accesses will be easily distinguishable. To obfuscate the spy's probe phase, the prefetch controller makes spy suffer from two misses all the time. When trojan encodes bit '1', the prefetcher brings back half of the spy's memory lines after the trojan's activity (note that trojan-spy pair could be inferred by our Trigger Pattern Recognizer module IV-B). The spy would observe 2 conflict misses rather than 4 misses. And when trojan transmits bit '0', two of trojan-owned memory blocks are brought to the cache before spy's probe phase. With this strategy, the number of observed misses is made independent on trojan's activity, and thus, it becomes impossible for the spy to infer trojan's activity by measuring cache access latencies.

## V. Experiment Setup

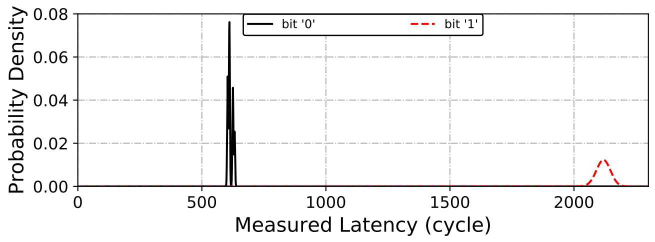
We evaluate Prefetch-guard using *Gem5* [4], a cycle-accurate, full-system simulator. We configure Gem5 with four x86 cores, 32 KB private L1 and 512 KB, 8-way shared L2 caches. All the experiments are run on full system mode under Linux kernel version 2.6.32.

We launch Prime+Probe attack on L2 cache shared by trojan and spy running on different cores. The trojan transmits '1' by replacing all of spy's cache blocks from the set with its own addresses, and transmits '0' by staying idle. The spy decodes bits by measuring access latency of cache blocks through re-issuing the same set of addresses used during its prime phase.

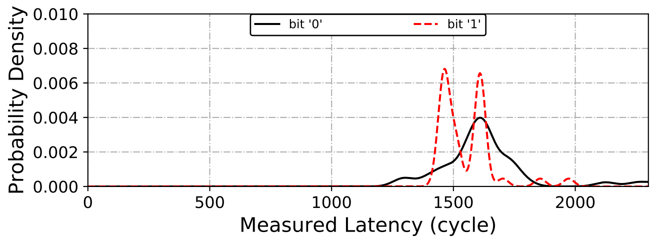
## VI. Evaluation

We record the spy's observed cache latencies with and without obfuscation strategy by Prefetch-guard (Section IV-C). The estimated conditional probability densities of spy's cache access latencies are shown in Figure 4. When there is no obfuscation (as shown in Figure 4a), the measured latencies for bit '0' and '1' transmissions are significantly distinguishable. Therefore, the spy can easily pick a threshold equal to the mean of all observations to decipher bits.

With Prefetch-guard enabled, Figure 4b shows that the estimated conditional probability densities change significantly. Specifically, the latency distributions of bit '0' and bit '1' overlap heavily, so that there is no clear boundary to separate them. With threshold-based detection mechanisms, the bit error rate for the spy is as high as 53%, which practically disables any communication. We note that spy and trojan can transmit predetermined symbol sequences to reveal these



(a) Attack



(b) With Prefetch-guard

Fig. 4: Conditional probability densities of spy's cache latency in single-group, round-robin attack

distributions, and find an optimum threshold. Even if this is the case, our experiments show that the lowest achievable error rate is 37%, still too high for any practical communication.

## VII. Conclusion

In this paper, we proposed Prefetch-guard that utilizes hardware prefetchers to prevent information leakage through cache timing channels. Prefetch-guard analyzes the conflict misses in shared caches, and targets the cache sets that are likely to be exploited by malicious processes. Hardware prefetchers are leveraged to obfuscate cache accesses on suspected cache sets used in timing channels. Prefetch-guard retrieves back the cache blocks owned by trojan and spy to disrupt the spy's probing of trojan's transmitted bits. With Prefetch-guard, we observe that the cache timing channels suffer a 53% bit error rate which makes it very hard or impossible for spy to decipher any useful information.

## VIII. Acknowledgement

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] Murugappan Alagappan, Jeyavijayan JV Rajendran, Miloš Doroslovacki, and Guru Venkataramani. Dfs covert channels on multi-core platforms. In *25th IFIP/IEEE International Conference on Very Large Scale Integration (VLSI-SoC)*, 2017.

[2] Alexandres Andreou, Andrey Bogdanov, and Elmar Tischhauser. Cache timing attacks on recent microarchitectures. In *IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, 2017.

[3] C. Bao and A. Srivastava. 3D integration: New opportunities in defense against cache-timing side-channel attacks. In *IEEE International Conference on Computer Design*, 2015.

[4] 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.

[5] Marco Bucci, Luca Giancane, Raimondo Luzzi, and Alessandro Trifiletti. Three-phase dual-rail pre-charge logic. In *International Workshop on Cryptographic Hardware and Embedded Systems*, pages 232–241. Springer, 2006.

[6] Abhishek Chakraborty, Ankit Mondal, and Ankur Srivastava. Correlation power analysis attack against stt-mram based cyptosystems. *IACR Cryptology ePrint Archive*, 2017:413, 2017.

[7] Jie Chen and Guru Venkataramani. An algorithm for detecting contention-based covert timing channels on shared hardware. In *ACM Proceedings of the Third Workshop on Hardware and Architectural Support for Security and Privacy*, 2014.

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

[9] Department of Defense Standard. *Trusted Computer System Evaluation Criteria*. US Department of Defense, 1983.

[10] Adi Fuchs and Ruby B. Lee. Disruptive prefetching: Impact on side-channel attacks and cache designs. In *ACM International Systems and Storage Conference*, 2015.

[11] 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 *IEEE International Symposium on High Performance Computer Architecture*, 2016.

[12] Fangfei Liu, Yuval Yarom, Qian Ge, Gernot Heiser, and Ruby B Lee. Last-level cache side-channel attacks are practical. In *Symposium on Security and Privacy*, 2015.

[13] Alireza Nazari, Nader Sehatbakhsh, Monjur Alam, Alenka Zajic, and Milos Prvulovic. Eddie: Em-based detection of deviations in program execution. In *ACM Proceedings of the 44th Annual International Symposium on Computer Architecture.*, pages 333–346, 2017.

[14] Kyle J Nesbit and James E Smith. Data cache prefetching using a global history buffer. In *IEEE MICRO*, 2004.

[15] Jianli Shen, Guru Venkataramani, and Milos Prvulovic. Tradeoffs in fine-grained heap memory protection. In *ACM Proceedings of the 1st workshop on Architectural and system support for improving software dependability*, 2006.

[16] Arvind Singh, Monodeep Kar, Anand Rajan, Vivek De, and Saibal Mukhopadhyay. Integrated all-digital low-dropout regulator as a countermeasure to power attack in encryption engines. In *IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, 2016.

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

[18] Guru Venkataramani, Ioannis Doudalis, Yan Solihin, and Milos Prvulovic. Memtracker: An accelerator for memory debugging and monitoring. *ACM Transactions on Architecture and Code Optimization (TACO)*, 6(2):5, 2009.

[19] Guru Prasadh V Venkataramani. *Low-cost and efficient architectural support for correctness and performance debugging*. Georgia Institute of Technology, 2009.

[20] Yao Wang, Andrew Ferraiuolo, Danfeng Zhang, Andrew C Myers, and G Edward Suh. Secdcp: secure dynamic cache partitioning for efficient timing channel protection. In *IEEE Design Automation Conference*, 2016.

[21] Fan Yao, Milos Doroslovacki, and Guru Venkataramani. Are coherence protocol states vulnerable to information leakage? In *24th IEEE International Symposium on High-Performance Computer Architecture*, 2018.

[22] Fan Yao, Guru Venkataramani, and Miloš Doroslovački. Covert timing channels exploiting non-uniform memory access based architectures. In *ACM Proceedings of the on Great Lakes Symposium on VLSI 2017*, 2017.