SPAM: Streamlined Prefetcher-Aware Multi-Threaded Cache Covert-Channel Attack

E. Kritheesh and Biswabandan Panda
Indian Institute of Technology Bombay

***Abstract*—Last-level cache (LLC) covert-channels exploit the cache timing differences to transmit information. In recent works, the attacks rely on a single sender and a single receiver. Streamline is the state-of-the-art cache covert channel attack that uses a shared array of addresses mapped to the payload bits, allowing parallelization of the encoding and decoding of bits. As multi-core systems are ubiquitous, multiple senders and receivers can be used to create a high bandwidth cache covert channel. However, this is not the case, and the bandwidth per thread is limited by various factors. We extend Streamline to a multi-threaded Streamline, where the senders buffer a few thousand bits at the LLC for the receivers to decode. We observe that these buffered bits are prone to eviction by the co-running processes before they are decoded. We propose SPAM, a multi-threaded covert-channel at the LLC. SPAM shows that fewer but faster senders must encode for more receivers to reduce this time frame. This ensures resilience to noise coming from cache activities of co-running applications. SPAM uses two different access patterns for the sender(s) and the receiver(s). The sender access pattern of the addresses is modified to leverage the hardware prefetchers to accelerate the loads while encoding. The receiver access pattern circumvents the hardware prefetchers for accurate load latency measurements. We demonstrate SPAM on a six-core (12-threaded) system, achieving a bit-rate of 12.21 MB/s at an error rate of 9.02% which is an improvement of over 70% over the state-of-the-art multi-threaded Streamline for comparable error rates when 50% of the co-running threads stress the cache system.**

***Index Terms*—Covert channels and noise resilience.**

I. Introduction

LAST-LEVEL cache can be exploited to create a covert channel transmitting data between two malicious processes. A covert channel by itself is significant for allowing an adversary to evade protections, such as sandboxing, and exfiltrate data from a device without network access [9]. It may also be used to exchange information in transient execution attacks or suggest the possibility of a side-channel attack. The covert-channel transmission rate establishes an upper bound on the information leakage rate of the side-channel [1]. Therefore, it is important to inspect high-bandwidth covert channels and investigate their bit rate limits.

Various countermeasures have been proposed to combat covert channels [10]. Bystander-based noise-injection strategies [13] provide a lightweight defense without radical hardware modifications or substantial performance degradation of non-malicious applications. They pollute the cache and cause eviction of the address(es) used by the covert channel. They are known to disrupt attacks that use a single shared address such as flush+reload [14]. In modern processors with many-cores, we observe that even a covert channel using a sequence of addresses such as Streamline [1] has a significant bandwidth drop if stressed with multiple cache-intensive co-running applications. Injecting noise succeeds if the address installed by the sender is evicted before the receiver accesses it. In order to weaken the impact of noise-injection, the time frame between the loading of an address and its subsequent decoding, has to be minimized. Faster decoding reduces this time frame, leading to stealthier channels.

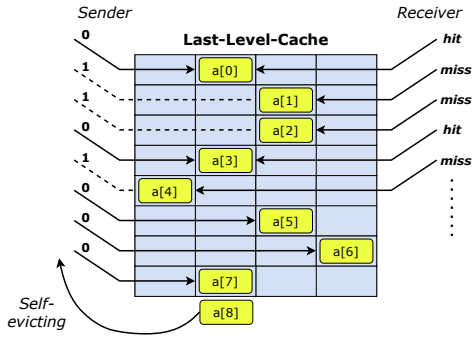

Fig. 1. State-of-the-art Streamline covert channel operates on a shared array for transmission of a stream of bits. The receiver follows the sender by maintaining a time gap, and cache thrashing is used to reset the channel.

Decoding bottleneck. Streamline employs a large shared array for transmission and attains a high bit rate by sequentially encoding each bit using a different address as shown in Figure 1. The receiver follows behind loading the addresses while timing them. It achieves a bit-rate of 2.11 MB/s and an error rate of 1.57% on an Intel® Core™ i7-10710U processor for a payload of 10^6 bits. The inability to measure the latency of simultaneously executing loads is emphasized by Streamline. This measurement bottleneck causes serialization of loads to measure time accurately while decoding and limits the bit period to be at least the LLC or DRAM latency.

A single thread's decoding is not fast enough to overcome the measurement bottleneck and we extend the Streamline channel with a multi-threaded sender process and a multi-threaded receiver process. We choose Streamline as its bit rate is about 3x-6x and 24x higher than the flush-based [2] and prime+probe-based [7] LLC covert channels, respectively. This shows that Streamline is an obvious choice for a multi-threaded LLC covert channel construction.

Our observations. We use multiple senders and receivers on a 12-threaded multi-core system (six cores with hyper-threading) with a 12MB LLC. Figure 2 shows the bit transmission rates of different combinations of sender and receiver

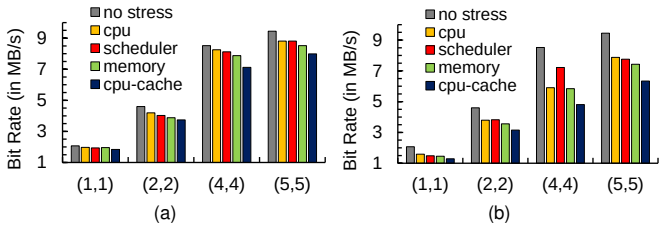

Fig. 2. Bit rates of different sender-receiver combinations of Streamline under four classes of *stress-ng* workloads running on (a) 25% and (b) 50% of the threads in a 12-threaded multi-core processor. The error rates of all these channels were restricted to 10%. (x, x) stands for x senders and x receivers.

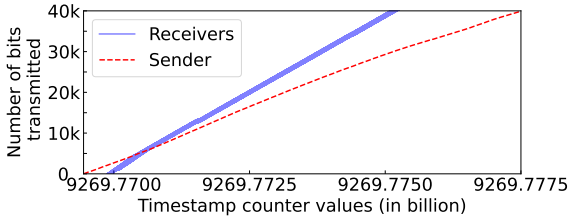

Fig. 3. Timestamp counter (TSC) values vs number of bits exchanged for one sender and two receiver threads following the access pattern of Streamline. The plot shows that around bit 5000, the receivers overtake the sender.

threads under four classes of co-running *stress-ng* workloads [8]. As the number of threads increases, Streamline's noise-resilience worsens, as shown by the larger drop in bit rates from no noise to different classes of noises. The addresses installed by the senders that are yet to be accessed by the receivers are prone to being evicted due to the interference caused by co-running processes in a many-core system. We argue that the time frame for eviction before decoding can be reduced by accelerating the decoding of buffered addresses using more receiver threads per sender thread.

Our challenges. We observe that if the senders are fewer than the receivers then senders tend to lag behind receivers and Streamline's mechanisms do not handle this case (refer Figure 3 for one sender and two receivers). In this case, two receivers independently decode the odd and even index payload bits. Note that for every transmitted bit, decoding should follow encoding. However, in Figure 3, this is true only for the first 5000 bits. The situation becomes worse if we further increase the number of receiver threads. Another challenge is that auditing or detection-based mitigations such as monitoring performance counters [15] pose a threat to multi-threaded covert channels. More cache misses will be observed in a short time window with a multi-threaded covert channel as compared to a single-threaded one.

Our approach. We propose a SPAM channel, where we make a case for faster but fewer senders and more receivers than senders. The major contribution through SPAM is to proactively exploit prefetchers to malicious parties' benefit. This is done by manipulating prefetchers to prefetch addresses necessary for covert communication. We accelerate sender threads with the usage of prefetcher-conscious access patterns for encoding. The access pattern plays a key role in circumventing the prefetchers on the receivers' side in order to get accurate load latencies, but the sender also employs the same pattern in Streamline and suffers from the inability to use prefetchers to its advantage. We propose to use a different sender access pattern that can exploit the prefetchers to its benefit. Prefetcher-friendly sender accesses mainly help in two ways: (i) fewer senders, that are faster, can sustain more receiver threads in order to accelerate decoding and make the channel stealthier against noise-injection based mitigation, and (ii) prefetched addresses reduce the number of cache misses on the sender's side and limit the exposure to detection-based defenses such as monitoring performance counters.

Prior works. Existing prefetcher-aware side channel attacks [11], [12] only aim to nullify/circumvent prefetchers whereas SPAM pursues the opposite direction and exploits prefetcher-triggering to benefit covert communication. Since SPAM is a covert channel, we can craft the access patterns on both sides of the channel, unlike the prior side channels. To the best of our knowledge, this is the first attempt to capitalize on hardware prefetchers for malicious parties' advantage. Another limitation of these past works is that their fooling techniques are limited to specific hardware. We demonstrated SPAM's effectiveness on Intel and AMD processors, verifying its universality. It should be noted that the prefetcher is not the medium for the covert channel as used in prior works [3]–[5].

Key results. On a six-core (12-threaded) Intel® i7-10710U processor (12 MB LLC), SPAM with four senders and eight receivers achieves a bit rate of 12.21 MB/s at an error rate of 9.02% as opposed to multi-threaded Streamline's 7.17 MB/s bit rate and an error rate of 12.53% for the *cpu-cache* stressors running on 50% of the threads. This is an improvement of over 70% in the bit rate as compared to multi-threaded Streamline.

II. SPAM

A. Encoding and decoding order

To achieve our goal of designing a noise-resilient high-bandwidth covert channel, we ask the following question: does the order in which the bits are encoded need to be the same as the order in which the bits are decoded? As long as bits i and $i+1$ are encoded before they are decoded, does it matter if the bit i was encoded before $i+1$? We argue that it does not matter if the subsequent address does not evict the preceding address from the LLC, which is the case in Streamline. What does it mean when we say the sender follows an encoding order different from the receiver? We argue that the sender differs from the receiver in the access pattern in which the sequence of addresses is accessed.

The heart of the SPAM covert channel is the usage of two different access patterns: one for sender(s) and one for receiver(s) in contrast to Streamline, which uses one access pattern for both sender(s) and receiver(s). We argue that there are two choices in terms of selecting the sequence of addresses of the shared array that will be used to transmit bits: (i) the exact mapping of payload bits to addresses (array indices) as shown in Figure 4a, and (ii) the access pattern through which the sender or the receiver accesses those addresses as shown in Figure 4b. Note that it is necessary for the sender and the receiver to agree upon the same mapping to ensure the correctness of the covert channel protocol. SPAM proposes that by using two different access patterns to load the addresses, the sender can be made faster. We achieve the same with the help of hardware prefetchers. The first access pattern is selected such that all the hardware prefetchers at the

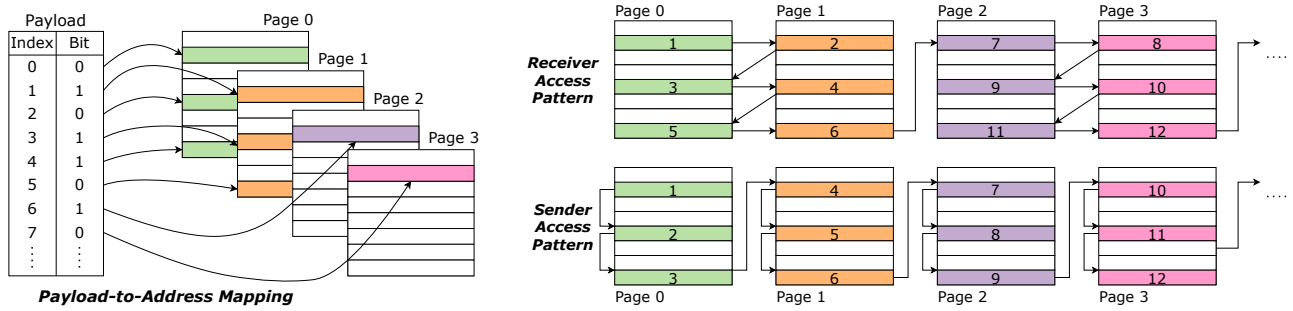

Fig. 4. (a) The mapping of payload bits to addresses (array indices), and (b) the pattern in which the sender or the receiver accesses those addresses. Note that consecutive accesses (connected by an arrow in (b)) of the same colour increase the confidence of the IP-stride prefetcher and trigger it. The subsequent identically coloured access is prefetched as a result. When the sender encodes the subsequent identically coloured address, the encoding is accelerated as the address was already installed by the prefetcher. However, consecutive accesses of different colours cross the page boundary and do not trigger prefetchers.

L1 and L2 are fooled. Fooling the prefetcher here refers to an access pattern preventing the prefetchers from being triggered. The receiver uses this access pattern. On the other hand, in the second access pattern, the sender triggers some of the L1 and L2 prefetchers to make the encoding faster.

We perform experiments with one sender following the access pattern of SPAM and k receivers where k = 2, 3, 4, and so on. We find that the accelerated sender is fast enough to encode for two receivers. The frequency of synchronization is one time for every 100k bits. However, three or more receivers decode faster than one sender. So we need to synchronize more often (once every 20k bits) to ensure that the sender always leads the receivers. Therefore, we argue that a sender-to-receiver ratio of 1:2 is the optimal ratio for SPAM to maximize the bit rate per thread.

B. Bit to address mapping

The objective behind the selection of the payload to address mapping is that there should exist an access pattern that can trigger the prefetchers and an access pattern that fools the prefetchers. For example, mapping the sequence of bits to addresses occupying consecutive cache lines can never be used because once cache line X is loaded, X+1 is also prefetched by the next-line prefetcher. So there is no way to find if X+1 is loaded by the sender or not. On the other hand, if a mapping is selected to access only one cache line per 4 KiB OS page, then an access pattern triggering prefetchers would not be possible as most of the prefetchers operate within the OS page boundaries. In general, a mix of accesses with variable strides that cross the page boundaries provides a way that can be used to trigger or not to trigger the prefetchers.

C. Prefetcher-conscious access patterns

Modern processors use multiple prefetchers to hide the costly DRAM latency. A few hardware prefetchers [6] used by Intel are documented, and some of them are reverse-engineered [3], [4]. The next-line [6] and data prefetch logic prefetchers [5] cannot be selectively triggered only for the prefetcher-friendly access pattern. These do not operate based on confidence. Therefore, these prefetchers are triggered for both the prefetcher-aware and prefetcher-evading patterns. Accesses need to trigger IP-stride or stream prefetcher to get prefetcher-conscious patterns. The streamer is more difficult to manipulate as it fetches multiple lines in the positive or

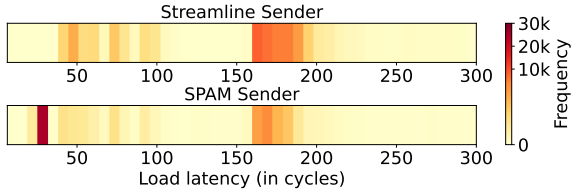

Fig. 5. Frequency of loads vs latency for the sender's access pattern of Streamline and SPAM on a 100k-bit payload.

negative direction in one go. While the pattern intended to fool prefetchers slides across pages, the prefetcher-aware accesses need to load all the addresses on a page before moving to the next page. The more the number of addresses mapped to one page, the more is the opportunity to trigger IP-stride prefetcher. So, an average stride as low as possible is needed to get an optimal access pattern. However, it was observed by Saileshwar et al. [1] that the prefetchers could not be fooled with stride two accesses.

D. Insights on the stride 3 across 2 pages access

Streamline accesses every third cache line spread across two pages. The receiver loads X, X+p, X+3, X+p+3, X+6, X+p+6, and so on where p is the page offset. To obtain a prefetcher-aware pattern from this mapping, we chose X, X+3, X+6, and so on till the end of the page followed by X+p, X+p+3, X+p+6, and so on. Dividing the intertwined cross-page accesses creates an opportunity to trigger prefetchers. Figure 5 shows the frequency of load latencies for the access patterns of Streamline and SPAM. It can be seen that the majority of the accesses by a Streamline sender take more than 150 cycles suggesting DRAM accesses. On the other hand, the dominant category of a SPAM sender is 20-30 cycles with a frequency of 26507 suggesting L1 or L2 hits. This validates our claim that SPAM reduces the number of cache misses on the sender's side by employing a prefetcher-conscious access pattern. Note that the access pattern depends on the payload and we used a uniformly distributed 100k-bit payload. For any payload, only zero bits involve loading the shared array. It was described by Saileshwar et al. [1] that any other payload distribution can be converted into a uniform distribution by modulating the payload bits with uniform random bits using the exclusive OR operation (XOR). The uniform random bits are generated through a pseudo-random generator whose seed is known to both the sender and the receiver.

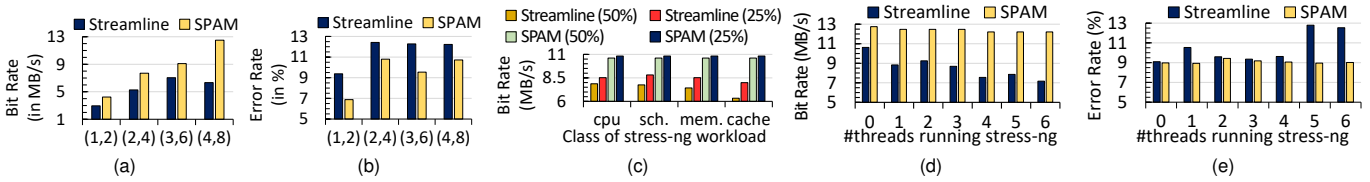

Fig. 6. Comparison of (a) the bit rates and (b) the error rates of Streamline vs SPAM for increasing difference between receivers and senders. (m, n) stands for m senders and n receivers. Comparison of the fastest combinations of Streamline and SPAM for (c) four classes (cpu, sch., mem., cache stand for cpu, scheduler, memory, and cpu-cache stress-ng classes respectively) of stressors on 25% and 50% of the threads. Comparison of (d) the bit rates and (e) the error rates of the fastest combinations of Streamline and SPAM for varying number of threads under stress-ng class cpu-cache workload.

III. Results

We compare SPAM with multi-threaded Streamline where we pin a thread to a core by setting affinity appropriately. Each thread on the sender process operates on a different physical core than every receiver thread. The channels were evaluated on the 12-threaded Intel® Core™ i7-10710U processor (12 MB LLC, 4 GHz frequency) on a payload of 10^6 bits. For the three-thread combination of one sender and two receivers, SPAM has a speedup of 2.006x over the single-threaded Streamline. The one-sender-and-two-receiver combination of Streamline attains a speedup of 1.189x over the single-threaded baseline. Therefore, the difference of 0.817x can be attributed to the prefetcher-friendly sender accesses. Additionally, the combination cannot be made faster further without uncovering a way to overcome the measurement bottleneck. Figures 6a and 6b show the performance of Streamline and SPAM for increasingly unequal sender and receiver combinations. We can see the poor scalability of Streamline for these combinations and SPAM's improvement of over 97% over Streamline for the four senders and eight receivers combination.

Figure 6c shows the transmission rates achieved by the fastest combinations of Streamline and SPAM on a 12-threaded multi-core system for two different extents (25% and 50%) of co-running stress. The quickest combination for SPAM is four senders and eight receivers and the quickest for Streamline is five senders and five receivers. We observed that the combination of six senders and six receivers could not meet the 10% error rate upper bound due to frequent synchronization misses. Additionally, the synchronization misses also slowed down the channel and the combination was slower than five senders and five receivers. The *cpu-cache* class stresses the cache and has the highest slowdown for Streamline, which is about 32% for 50% stress and over 15% for 25% stress. SPAM performed equally well across all the classes and has a dip of about 4% and 2% for 50% and 25% stress respectively.

Sources of errors. The error rates of SPAM are contributed by 0-to-1 error rate and 1-to-0 error rate. Multi-threaded SPAM has a higher 1-to-0 error rate (5.8%) than multi-threaded Streamline (3.59%) because the prefetchers hide the latency of the DRAM accesses (DRAM access is faster than our LLC-hit threshold), resulting in some of these accesses falling within the LLC threshold. On the other hand, the multithreaded SPAM has a lower 0-to-1 error rate (3.18%) than Streamline (5.51%) as the two extra receivers help in decoding the addresses present at the LLC faster.

Figures 6d and 6e show the bit rates and error rates, respectively, of the fastest combinations of SPAM and Streamline for no stress (zero threads stressed) to multiple threads co-running *cpu-cache* workload. SPAM improves the bandwidth of Streamline marginally by 15-20% under no-stress conditions. However, SPAM is resilient to noise and outperforms Streamline by over 43% for 25% of threads running stress workloads and over 70% for 50% of the threads stressed.

Universality of SPAM channel. So far, we have shown the effectiveness of a SPAM channel on Intel processors. We also performed the SPAM attack on AMD Ryzen 9 and Milan processors and observed similar bit and error rates. SPAM attack is universal as it exploits IP-stride prefetcher, commonly available in all commercial machines from Intel, AMD, and ARM. Also the effectiveness of SPAM over Streamline remains the same even for a large number of hardware threads (24 and 32) used for a covert channel.

IV. Conclusion

We proposed SPAM, a high bandwidth and noise-resilient LLC covert channel that uses prefetcher-friendly access patterns for the sender in multi-threaded covert channels. We showed that mounting a multi-threaded covert channel causes a dip in bit-rate because of noise coming from co-running applications. SPAM makes a case for faster but fewer senders and more receivers making it a competitive covert channel at the LLC even on a multi-threaded multi-core system.

References

[1] Saileshwar et al., Streamline: a fast, flushless cache covert-channel attack by enabling asynchronous collusion. In *26th ASPLOS*, pp. 1077-1090.

[2] Gruss et al., Flush+ Flush: a fast and stealthy cache attack. In *13th International Conference on DIMVA*, 2016, pp. 279-299.

[3] Chen et al., "AfterImage: Leaking control flow data and tracking load operations via the hardware prefetcher," in *28th ASPLOS*, pp. 16-32.

[4] Rohan et al., "Reverse engineering the stream prefetcher for profit," in *EuroS&PW 2020*, pp. 682–687.

[5] Chen et al., "PrefetchX: Cross-core cache-agnostic prefetcher-based side-channel attacks," in *HPCA 2024*, pp. 395-408

[6] Intel, "Disclosure of H/W prefetcher control on some Intel processors," https://radiable56.rssing.com/chan-25518398/article18.html, 2018.

[7] Liu et al., "Last-Level Cache Side-Channel Attacks are Practical," in *S&P*, pp. 605-622

[8] Colin King. (accessed April 17, 2024). Ubuntu Wiki: Stress-NG. https://wiki. ubuntu.com/Kernel/Reference/stress-ng.

[9] Naghibijouybari et al., "Constructing and Characterizing Covert Channels on GPGPUs," in 50th *MICRO*, 2017, pp. 354-366.

[10] Ge et al., "A survey of microarchitectural timing attacks and countermeasures on contemporary hardware," in *Journal of Cryptographic Engineering* 2018, pp. 1–27.

[11] Wang et al., "PAPP: Prefetcher-Aware Prime and Probe Side-channel Attack," in *56th ACM/IEEE DAC* 2019, pp. 1-6.

[12] Haas et al., "iTimed: Cache Attacks on the Apple A10 Fusion SoC," in *HOST* 2021, pp. 80-90.

[13] Zhang et al., "On Mitigating the Risk of Cross-VM Covert Channels in a Public Cloud," in *IEEE TPDS*, pp. 2327-2339, 2015.

[14] Liu et al. "Adaptive Noise Injection against Side-Channel Attacks on ARM Platform." EAI Endorsed Trans. Security Safety 2019.

[15] Payer, M., "HexPADS: a platform to detect "stealth" attacks." In *ESSoS*, 2016, pp 138–154.