Streamline: A Fast, Flushless Cache Covert-Channel Attack by Enabling Asynchronous Collusion

Gururaj Saileshwar
gururaj.s@gatech.edu
Georgia Tech

Christopher W. Fletcher
cwfletch@illinois.edu
University of Illinois, Urbana-Champaign

Moinuddin K. Qureshi
moin@gatech.edu
Georgia Tech

1. Motivation - Cache Covert Channels

Covert-channels allow malicious processes to collude and communicate with each other without detection. Among hardware covert-channels, cache covert-channels are one of the fastest and most robust channels. These channels emanate from the timing difference between accesses to processor caches (tens of ns) and DRAM (~100 ns). As caches are shared between multiple processes and cores, a sender process can covertly transmit information to a co-running receiver process, by influencing whether a (read-only) address shared with the receiver process is in the cache or not, and modulate the latency observed by the receiver for accessing the address. Such read-only sharing of addresses is possible via shared-libraries or via OS de-duplication (e.g. Linux KSM). Cache covert channel attacks have been heavily exploited in several recent attacks including Spectre [2], Meltdown [4], etc.

To understand the potential for information leakage via caches, it is important to bound the bit-rate for cache covert-channel attacks. Towards this goal, this paper focuses on understanding the limitations of existing attacks and designing new attacks that can achieve higher bit-rates than the state-of-the-art. Our default focus is on cross-core cache attacks, where a malicious sender and a receiver process execute on two different processor cores and attempt covert communication via accesses to the shared LLC (a setting typical for a virtualized environment with per-core resource allocation), although our attack is generally applicable even to a same-core setting (sender and receiver running on the same core).

2. Limitations of State-of-the-Art Attacks

The current fastest cross-core covert-channels are flush-based attacks, such as Flush+Reload [8] and Flush+Flush [1]. In these attacks (shown in Fig 1), the sender and receiver operate synchronously and transmit information in each bit-period epoch using the timing difference between cache hits and misses on a read-only shared address. The sender, in each bit-period epoch, executes a load to an address if the bit to be sent is 0, else it skips the load. The receiver in each bit-period, executes a load (or a `clflush`) to the address measuring its latency – and uses the difference in latency to decode the bit transmitted by the sender. The sender and receiver both wait till the end of a bit-period to ensure the other has finished its operations, typically synchronizing using `rdtscp` that provides both a shared notion of time, before communicating the next bit using the same address. Gruss et al. [1] showed these

Sender	Receiver-FR	Receiver-FF	Receiver-TW

foreach(bit)
{
if(bit==0)
load(x)
wait(end-epoch)
}
	foreach(bit)
	{
	t = rdtscp
	load(x)
	T = rdtscp-t
	bit=T<limit?0:1
	clflush(x)
	wait(end-epoch)
	}
		foreach(bit)
		{
		t = rdtscp
		clflush(x)
		T = rdtscp-t
		bit=T<limit?0:1
		wait(end-epoch)
		}
			foreach(bit)
			{
			t = rdtscp
			load(x_conflict)
			T = rdtscp-t
			bit=T<limit?1:0
			wait(end-epoch)
			}

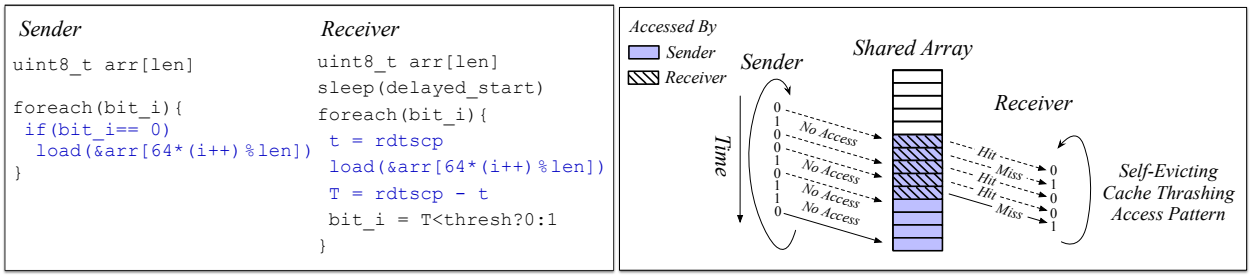
	(a) Flush+Reload	**(b) Flush+Flush**	**(c) Take-a-Way**
	Epoch-size = 3 Ops *(1 S-Load + 1 R-Load+1 R-Flush)*	***Epoch-size = 2 Ops*** *(1 S-Load + 1 R-Flush)*	***Epoch-size = 2 Ops*** *(1 S-Load + 1 R-Load)*

Figure 1: All existing covert-channel attacks operate the sender and receiver synchronously, where they transmit each bit in a synchronized epoch, and wait till the epoch ends before transmitting the next bit. The epoch-size also needs to be large enough to execute 2-3 operations (a load from a sender (S) and one or two operations from the receiver (R)) to avoid errors and subsequent channel breakdown.

attacks achieve bit-rates of 298 KB/s (Flush+Reload) and 496 KB/s (Flush+Flush) at <1% bit-error-rate. Take-a-way [3], an attack with a similar synchronous construction is the fastest same-core attack, with a bit-rate of 588 KB/s.

All of the existing attacks face two key limitations that prevent them from being faster and more universally applicable:

- **Bandwidth Limited by Synchronous Communication:** The bit-rate of all existing attacks is bottlenecked by the requirement of synchronous transfer of bits. The operations to encode and decode a bit and reset the channel for each bit, all need to be completed for a current bit by the sender and receiver, before the next bit can be communicated. Additionally, the synchronous window for each bit (epoch-size in Figure 1) has to be large enough to accommodate two or three operations. Any attempt to decrease the window results in loss of synchronization, and in channel breakdown.
- **Limited Applicability Based on ISA/Microarchitecture:** State-of-the-art attacks exploit features only present in a specific ISA or micro-architecture and have limited applicability. Flush+Reload and Flush+Flush require cacheline flush instruction whose unprivileged usage is permitted in x86 but not in ARM ISA by default (it is simply unavailable in ARMv7). Take-a-way exploits L1-cache way-prediction known to be only in AMD processors, making it infeasible on other CPUs. More generic attacks like Prime+Probe [5] are at least 7x slower than the fastest attacks.



(a) Streamline Attack Algorithm **(b) Access Pattern of Sender and Receiver to Shared Array**

Figure 2: Overview of the Streamline Attack. The sender and receiver communicate asynchronously via accesses to a shared array `arr` (larger than the LLC). The sender keeps transmitting on sequential entries of the array, without waiting for the receiver to decode. By the time the sequential access wraps around to the start of the array, the entries accessed in the previous iteration are evicted from the LLC due to the cache-thrashing access pattern.

Our goal is to design a fast attack that is also widely applicable to processors of all architectures and micro-architectures. Such an attack should not require cacheline flushes. At the same time, to be faster than current attacks, it must not require the sender and receiver to execute their operations synchronously. To that end, we design *Streamline*, our cache covert-channel attack, to be fast, flush-less, and asynchronous.

3. Key Insights of Streamline Attack

The key idea of our attack is to have the sender and receiver communicating asynchronously over a large number of lines, by using the cache to buffer data between the sender and the receiver, and rely on cache thrashing to naturally evict the resident lines previously used for communication (instead of expensive `clflush` operations). Our protocol is the following:

- The sender and receiver share a large array, few tens of MBs in size (larger than LLC size) rather than a single [1, 8], or a small number of addresses [3, 6] like prior attacks; successive bits are transmitted over a predetermined sequence.
- The sender transmits on successive addresses without waiting for the receiver; the receiver follows behind accessing the same addresses in a streamlined manner, as in Figure 2.
- The encoding is similar to prior works: the sender accesses an address to transmit Bit-0 and skips it to for Bit-1; the receiver infers Bit-0/Bit-1 based on if it gets an LLC-Hit/Miss.
- If the array is much larger than LLC, by the time the sender wraps around to the start of the array, addresses from the previous iteration are automatically evicted via cache-thrashing.

Streamline is faster than prior state-of-the-art as: (a) it is asynchronous (the sender does not wait for the receiver on every bit), and (b) it only requires a load per bit (no flushes). However, to orchestrate Streamline at low error-rates, we address two key challenges unique to asynchronous protocols:

Challenge-1. Ensuring the sequence of addresses occupies a significant fraction of the cache: Otherwise, successive addresses installed by the sender may evict previous addresses before the receiver accesses them, causing errors. We also work around hardware optimizations like LLC replacement policy (that can preemptively evict addresses) and the prefetcher (that can preemptively install lines) that can disrupt the channel.

Challenge-2. Tolerating rate-mismatch in sender and receiver: For an asynchronous protocol, any mismatch in the sender and receiver rates can breakdown the channel. We develop techniques to both reduce the mismatch, like pseudo-random channel encoding, and tolerate potential mismatch with coarse-grain synchronization (once every 200,000 bits), enabling a robust channel that can transmit billions of bits.

Table 1: Prior Cache Covert Channels (Bit-Rate>50 KB/s)

Attack	Attack Model	Bit-Rate	Bit Error Rate
Take-a-way [3]	Same-Core	588 KB/s	1–3%
Flush+Flush [1]	Cross-Core	496 KB/s	0.84%
Prime+Probe (L1) [6]	Same-Core	400 KB/s	–
Flush+Reload [1]	Cross-Core	298 KB/s	0%
Prime+Probe (LLC) [5]	Cross-Core	75 KB/s	1%
Xiaong and Szefer [7]	Same-Core	72 KB/s	<2%
Streamline (this work)	**Cross-Core**	**1788 KB/s**	**0.33%**

4. Key Results and Contributions

1. To our knowledge, we are the first to propose a high-bandwidth covert channel without relying on bit-level synchronization. Our *Streamline* attack, uses the entire cache to buffer data between sender and receiver, and uses thrashing to naturally evict data from the cache post transmission.
2. We overcome several obstacles for high-bandwidth attacks, circumventing LLC optimizations (prefetcher and replacement policy) and rate-mismatch in sender and receiver.
3. We demonstrate Streamline on an Intel Skylake CPU in a cross-core setting, with a bit-rate of 1788 KB/s and bit-error-rate of 0.3%. Our bit-rate is 3.6x that of Flush+Flush (496 KB/s), the prior-best cross-core attack, and 3x that of Take-a-Way (588 KB/s), the prior-best same-core attack.
4. We discover a fundamental limitation in existing load-latency measurement gadgets, that prevents latency measurement of multiple loads in parallel and limits bit-rate of Streamline (and any future attacks attempting to go faster).

References

[1] Daniel Gruss, Clémentine Maurice, Klaus Wagner, and Stefan Mangard. Flush+ Flush: a fast and stealthy cache attack. In *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, pages 279–299. Springer, 2016.

[2] Paul Kocher, Jann Horn, Anders Fogh, Daniel Genkin, Daniel Gruss, Werner Haas, Mike Hamburg, Moritz Lipp, Stefan Mangard, Thomas Prescher, et al. Spectre attacks: Exploiting speculative execution. In *2019 IEEE Symposium on Security and Privacy (SP)*, pages 1–19. IEEE, 2019.

[3] Moritz Lipp, Vedad Hadžić, Michael Schwarz, Arthur Perais, Clémentine Maurice, and Daniel Gruss. Take a way: Exploring the security implications of amd's cache way predictors. In *Proceedings of the 2020 ACM Asia Conference on Computer and Communications Security*, 2020.

[4] Moritz Lipp, Michael Schwarz, Daniel Gruss, Thomas Prescher, Werner Haas, Anders Fogh, Jann Horn, Stefan Mangard, Paul Kocher, Daniel Genkin, et al. Meltdown: Reading kernel memory from user space. In *27th USENIX Security Symposium (USENIX Security 18)*, pages 973–990, 2018.

[5] Fangfei Liu, Yuval Yarom, Qian Ge, Gernot Heiser, and Ruby B Lee. Last-level cache side-channel attacks are practical. In *2015 IEEE Symposium on Security and Privacy (SP)*, pages 605–622. IEEE, 2015.

[6] Colin Percival. Cache missing for fun and profit. In *Proceedings of BSDCan*, 2005.

[7] Wenjie Xiong and Jakub Szefer. Leaking information through cache lru states. In *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 139–152. IEEE, 2020.

[8] Yuval Yarom and Katrina Falkner. FLUSH+ RELOAD: a high resolution, low noise, L3 cache side-channel attack. In *23rd USENIX Security Symposium (USENIX Security 14)*, pages 719–732, 2014.