Opening Pandora's Box: A Systematic Study of New Ways Microarchitecture Can Leak Private Data

Jose Rodrigo Sanchez Vicarte, **Pradyumna Shome**, Nandeeka Nayak, Caroline Trippel*, Adam Morrison◦, David Kohlbrenner•, Christopher W. Fletcher

University of Illinois at Urbana-Champaign, *Stanford University, ◦Tel Aviv University, •University of Washington

Introduction

Microarchitectural attacks have plunged Computer Architecture into a security crisis. Yet, as the slowing of Moore's law justifies the use of ever more exotic microarchitecture, it is likely **we have only seen the tip of the iceberg**.

- Uncovers **seven classes of microarchitectural optimizations with novel security implications**
  1. Data Memory-Dependent Prefetchers
  2. Computation Simplification
  3. Pipeline Compression
  4. Computation Reuse
  5. Silent Stores
  6. Value Prediction
  7. Register-File Compression
- Proposes a **conceptual framework through which to study microarchitectural optimizations**
- Demonstrates **several Proofs-of-Concept** to show their efficacy

This paper's goal is to perform an early analysis to inform *secure and performant development* moving forward.

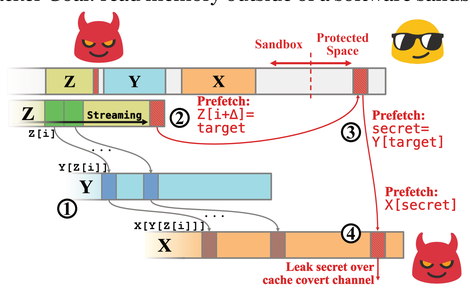
Motivating Example: Data Memory Dependent Prefetchers

Data Memory-Dependent Prefetchers (DMPs) are **effective in cases where stride prefetchers fail**, e.g., in applications dominated by indirections or "pointer chasing."

`for (i=0...N) X[Y[Z[i]]]` Multi-Level Indirection access pattern. The DMP prefetches entries in **Y** by using the *values* of **Z**, and entries in **X** by using the *values* of **Y**

DMPs in an Adversarial Setting

Attacker Goal: read memory outside of a software sandbox.



1) Attacker activates the prefetcher, with the now-malicious program `for(i=0…N): X[Y[Z[i]]]`
2) Attacker **tricks the prefetcher** into reading attacker-controlled data `z=Z[i+Δ]`, out of bounds of **Z**
3) The DMP reads any **attacker chosen** private value `y=Y[z]`, out of bounds of **Y**
4) Finally **leaks that value** over a traditional cache covert channel vis. the final prefetch for `&X[y]`

Augury: Follow up Work at Oakland'22

We demonstrate the existence of a pointer- chasing DMP on recent Apple processors, including the A14 and M1. We then *reverse engineer the details of this DMP* to *determine the opportunities for and restrictions it places on attackers using it*. Finally, **we demonstrate several basic attack primitives** capable of leaking pointer values using the DMP.

Read more about this follow up work at: **prefetchers.info**

Reasoning about Privacy Leakage

We present a framework for precisely characterizing privacy leakage. This framework:

- Highlights **necessary conditions** for data leakage
- **Hides low level details** needed for an attack

Why this type of model?

Consider Caches Attacks and the resources they modulate…

Cache Attacks	Modulated Resources
Prime+Probe, Flush+Reload, Flush+Flush, *and more!*	Cache Occupancy, Replacement Information, CPU Interconnect, *and more!*

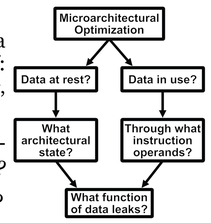
They *all* leak **the program memory access pattern**. *This is the relevant information for software defenses.*

Framework Flow

A. Does the optimization modulate a hardware channel as a function of: **data in use** (being operated on) or, **data at rest** (not being operated on)?

B. Refine the analysis to precisely indicate: *What is being leaked, and when?*

C. What *function of the operands* leak?



UArch with Novel Privacy Implications

We analyze each optimization relative to a "Baseline" architecture. The baseline represents a typical commercial server processor (out-of-order, speculative, multi- core) *based on known attacks today*. **Each column represents the same baseline plus the specified microarchitectural optimization.**

		Baseline	Silent Stores	Computation Simplification	Pipeline Compression	Computation Reuse	Value Prediction	Reg-File Compression	DMP
Data in use (operands/result of transmit instruction)									
Operands	Int Simple Ops	S	–	😈	😈	😈	–	–	–
	Int Multiplication	S	–	😈	😈	😈	–	–	–
	Int Division	😈	–	😈	😈	😈	–	–	–
	FP ops	😈	–	😈	–	😈	–	–	–
Result	Int Simple Ops	S	–	–	–	–	😈	😈	–
	Int Multiplication	S	–	–	–	–	😈	😈	–
	Int Division	S	–	–	–	–	😈	😈	–
	FP ops	S	–	–	–	–	😈	😈	–
Addr	Load	😈	–	–	–	–	–	–	–
	Store	😈	–	–	–	–	–	–	–
Data	Load	S	–	–	–	–	😈	–	–
	Store	S	😈	–	–	–	–	–	–
	Control Flow	😈	–	–	–	–	–	–	–
Data at rest									
	Register File	S	–	–	😈	–	–	😈	–
	Data Memory	S	😈	–	–	–	–	–	😈

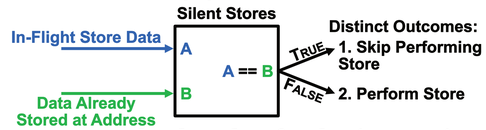
We found a range of microarchitectural optimizations with *novel security implications*

Microarchitectural Leakage Descriptors (MLDs)

The paper develops a novel conceptual framework and abstraction, called Microarchitectural Leakage Descriptors (MLDs), that **characterizes privacy leakage precisely**. An MLD is a map that **indicates when a microarchitectural optimization changes a program's observable execution** (e.g., based on timing or hardware resource usage) as a function of what changes to what program data.

Example MLD: Silent Stores

The silent stores optimization **skips stores which do not change the value already in memory**, reducing pressure on the memory system.

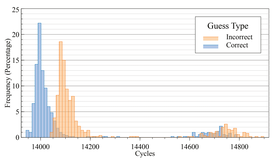


Silent stores **can be thought of as having two inputs**: The **in-flight store data** (in use) and the **data already in memory** (at rest). *Depending on these values*, **silent stores maps to two, distinct, observable outcomes**.

- **Active attack:** If the **attacker controls** either input, they can **perform a search over multiple replays** to leak the other operand.
- **Passive Attack:** The program can leak its own data. **All stores become potentially latent gadgets**.

Silent Stores PoC on Bitslice AES128 encryption

It's non-trivial to *convert a single silent store into a timing difference*. We develop an **amplification gadget**. Attacker **induces a silent store**, conditioned on the **data left behind in memory** by the victim's prior encryption operation.



Example MLD: Computation Reuse

Minor, performance-guided, decisions can result in significantly different security implications.

V1. One variant of *Computation Reuse* skips computations (forwards the result) by using operand ***values*** as memoization table keys. This ***leaks data values!***

V2. A second variant uses ***operand register IDs***. This ***only leaks control flow***.

Conclusion

Proactively understanding the security implications of 'exotic' microarchitecture is of fundamental importance for building holistic, long-term, defenses and anticipating future attacks.

This paper **performed a systematic study of the computer architecture literature through a security lens**. We found a range of microarchitectural optimizations with novel security implications—ranging from ones as devastating as Spectre/Meltdown (but without relying on speculative execution) to ones that render constant-time programming ineffective, or in need of overhaul.

Microarchitectural Leakage Descriptors (MLDs) **present a starting point towards communicating exactly the information essential for software** to implement *performant and secure* defenses, while hiding second-order details about the hardware.

Acknowledgements

This work was partially funded by an Intel ISRA grant, NSF grants #1816226 and #1942888, and ISF grant #2005/17.

The Grainger College of Engineering, University of Illinois Urbana-Champaign · Stanford University · University of Washington · Tel Aviv University