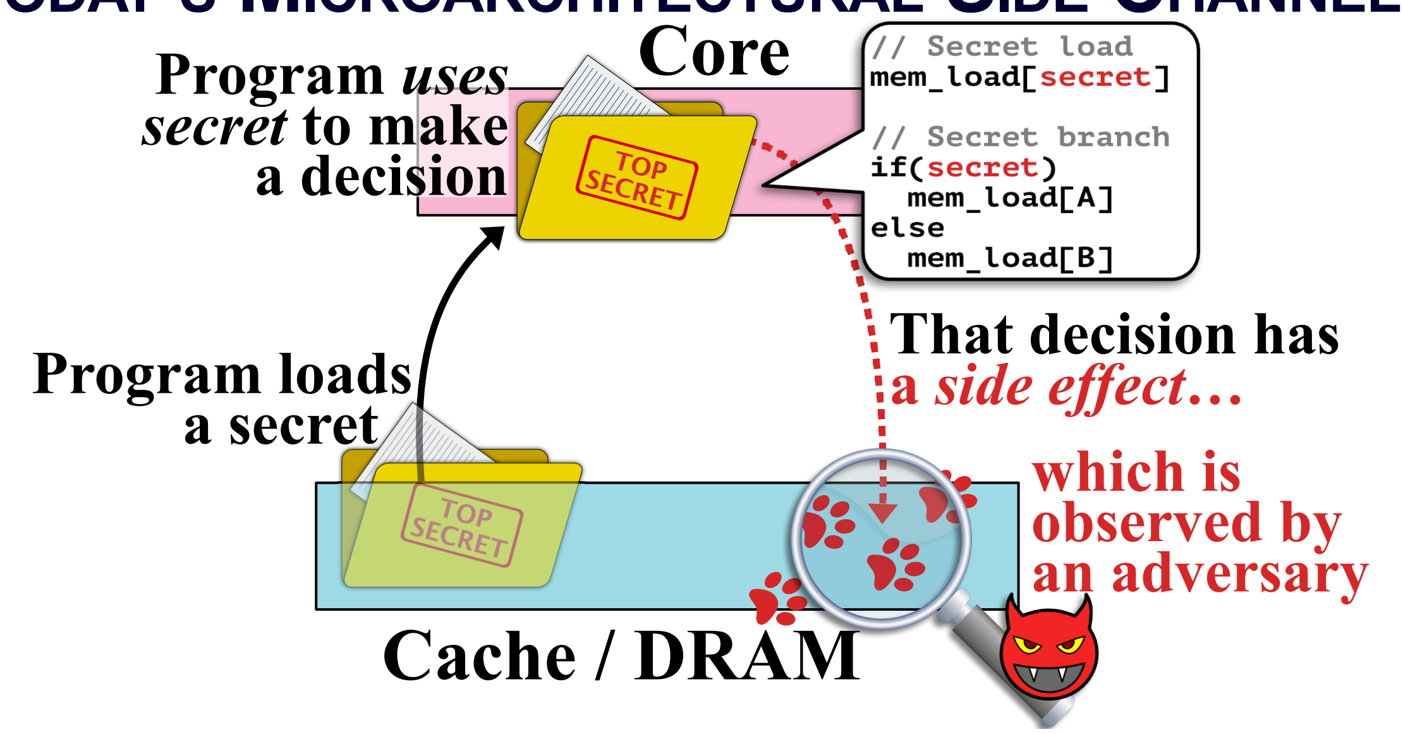
# Augury: Using Data Memory-Dependent Prefetchers to Leak Data at Rest

**Jose Rodrigo Sanchez Vicarte***, Michael Flanders*•, Riccardo Paccagnella, Grant Garrett-Grossman, Adam Morrison◦, Christopher W. Fletcher, David Kohlbrenner•

*The two first authors contributed equally to the paper

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

## Today's Microarchitectural Side Channels


Program *uses* *secret* to make a decision
Core
TOP SECRET
Program loads a secret
TOP SECRET
Cache / DRAM
That decision has a *side effect…*
which is observed by an adversary

```
// Secret load
mem_load[secret]

// Secret branch
if(secret)
  mem_load[A]
else
  mem_load[B]
```

**Data in Use Leakage:** **Leakage only occurs when data is *used* by *unsafe instructions*.**

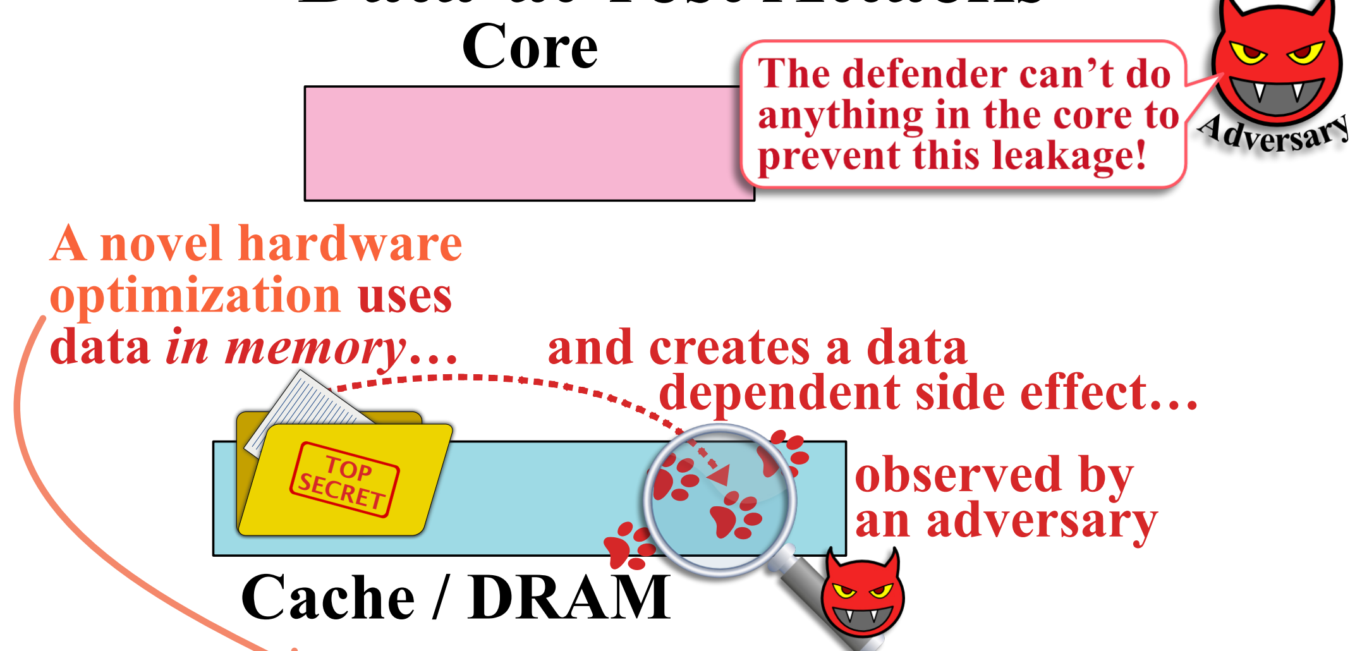
Adversary: I can *trick* instructions with side effects into using *secrets*!

Defender: I can *prevent* instructions with side effects from using *secrets*!

## A New Threat: Leaking Data at Rest

**This work presents the first microarchitectural attack which leaks *data at rest:*** data which was never directly read into the core.

### Data-at-rest Attacks

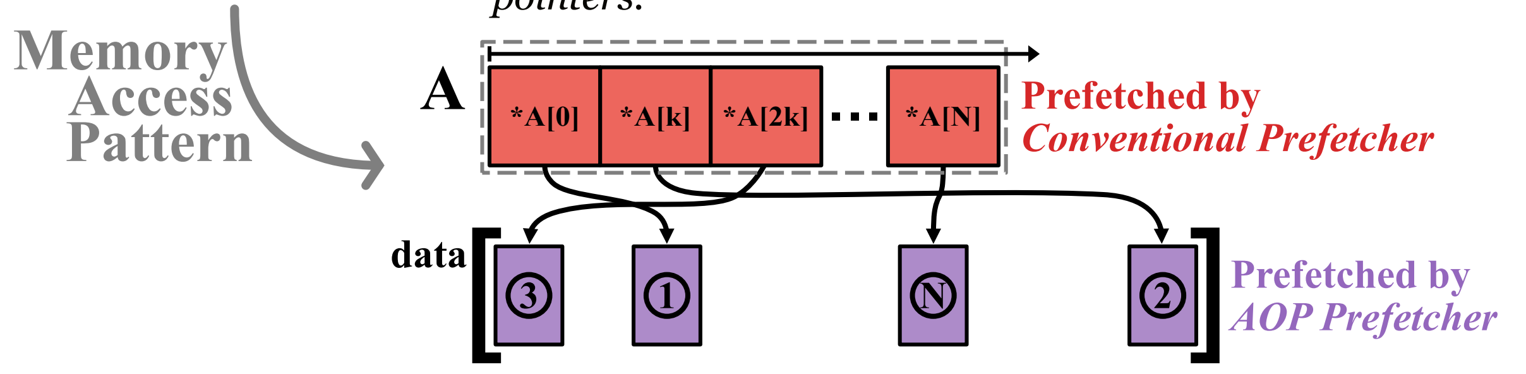

Core
A novel hardware optimization uses data *in memory…*
and creates a data dependent side effect…
TOP SECRET
observed by an adversary
Cache / DRAM

Adversary: The defender can't do anything in the core to prevent this leakage!

**We found an *Array-of-Pointers Prefetcher* in Apple's M1, M1 Max, M1 Pro, and A14 processors.** It's a previously unreported class of prefetcher for *irregular access patterns*: **Data Memory-Dependent Prefetcher (DMP).**

```
for (i=0...N)
    *A[k*i]
```

**Array of Pointers** access pattern. The DMP recognizes **streaming and striding reads** followed by **dereferences**, and then *prefetches the result of dereferencing future pointers.*

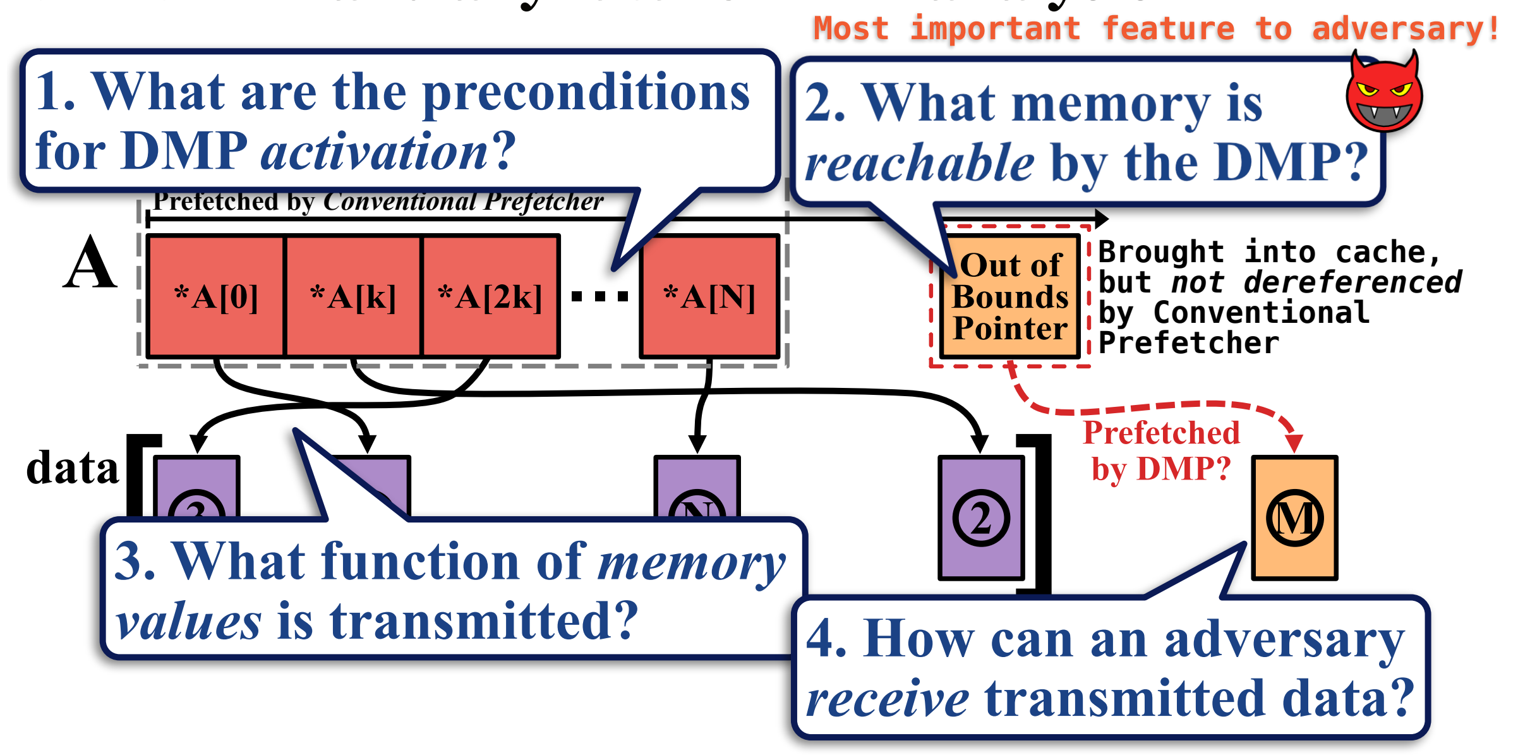

Memory Access Pattern
A
*A[0]
*A[k]
*A[2k]
…
*A[N]
Prefetched by *Conventional Prefetcher*
data
3
1
N
2
Prefetched by *AOP Prefetcher*

## Contributions

1. **Discover the first microarchitecture (in the wild) capable of leaking data-at-rest.**
2. **Provide guidance for the reverse engineering and security analysis of *any* DMP system.**
3. **Prove existence** and **reverse engineer** Apples's DMP to determine **opportunities for and restrictions** on attackers.

## DMP: Data Memory Dependent Prefetcher

As real DMPs have not previously been evaluated for security impact, **this is an unexplored area** useful for framing both the M1's DMP **and any future DMP analysis**.


Most important feature to adversary!
Prefetched by *Conventional Prefetcher*
A
*A[0]
*A[k]
*A[2k]
…
*A[N]
Out of Bounds Pointer
Brought into cache, but *not dereferenced* by Conventional Prefetcher
Prefetched by DMP?
data
2
N
M

1. What are the preconditions for DMP *activation*?

2. What memory is *reachable* by the DMP?

3. What function of *memory values* is transmitted?

4. How can an adversary *receive* transmitted data?

Adversary: I need to answer these questions to understand how to leak data!

Defender: I need to answer these questions to create defenses against DMP attacks!

Designer: This will help me understand the security implications of my designs!

The paper also describes *possible DMP access patterns*, and the **security implications of each**.

ON THE M1 — **1-Level Pointer-Chasing**

```
for (i=0; i<N; i++):
    *A[k*i];
```

**L-Level Pointer-Chasing**

```
for (i=0; i<N; i++):
    **(...)*A[k*i];
```

**1-Level Indirection-Based**

```
for (i=0; i<N; i++):
    B[A[k*i]];
```

**L-Level Indirection-Based**

```
for (i=0; i<N; i++):
    Z[Y[(...)A[k*i]]];
```

## Existence of the M1 DMP

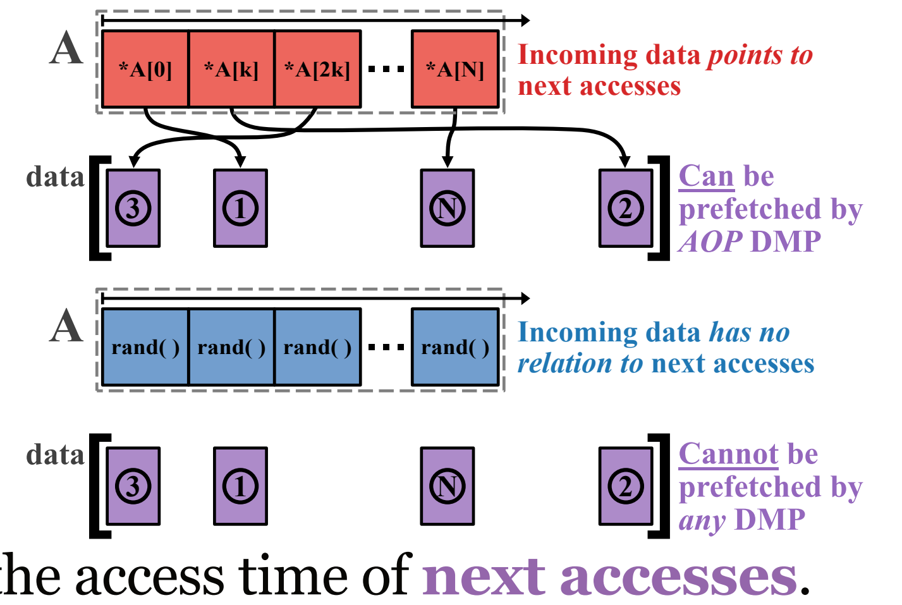
DMPs try to detect associations between addresses outgoing *from* the core and data incoming *to* the core. Starting from an access pattern which **should be prefetchable via the DMP**. We construct **a baseline** which has the same outgoing access pattern, but *removes the incoming data*.

**AOP: Pointer DMP Prefetchable**

```
for (i=0; i<N; i++):
    *A[k*i];
```

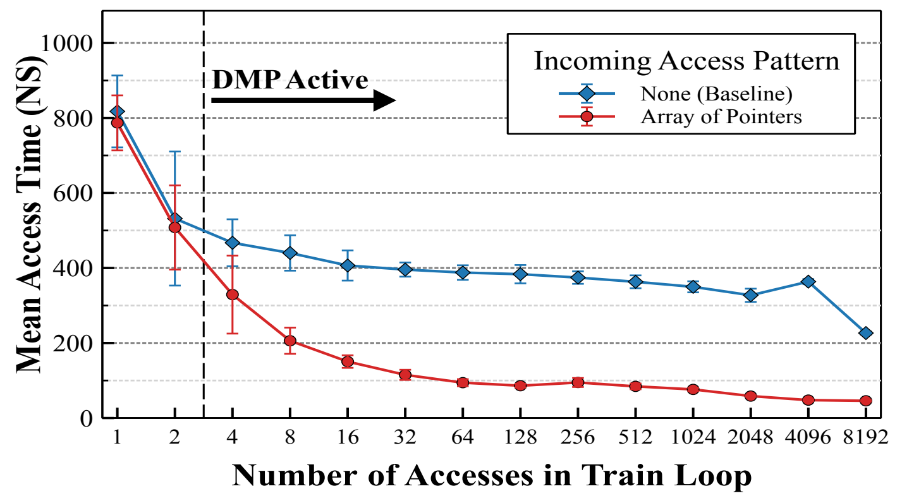
**Baseline: Not DMP Prefetchable**

```
for (i=0; i<N; i++):
    A[k*i];
    idx = prng(idx)
    B[idx]
```


A
*A[0]
*A[k]
*A[2k]
…
*A[N]
Incoming data *points to* next accesses
data
3
1
N
2
Can be prefetched by *AOP* DMP
A
rand()
rand()
rand()
…
rand()
Incoming data *has no relation to* next accesses
data
3
1
N
2
Cannot be prefetched by *any* DMP

In all cases, we measure the access time of **next accesses**.

Only the **Pointer Based** pattern has speedup. We conclude that an AOP DMP must be present on the M1 and A14.



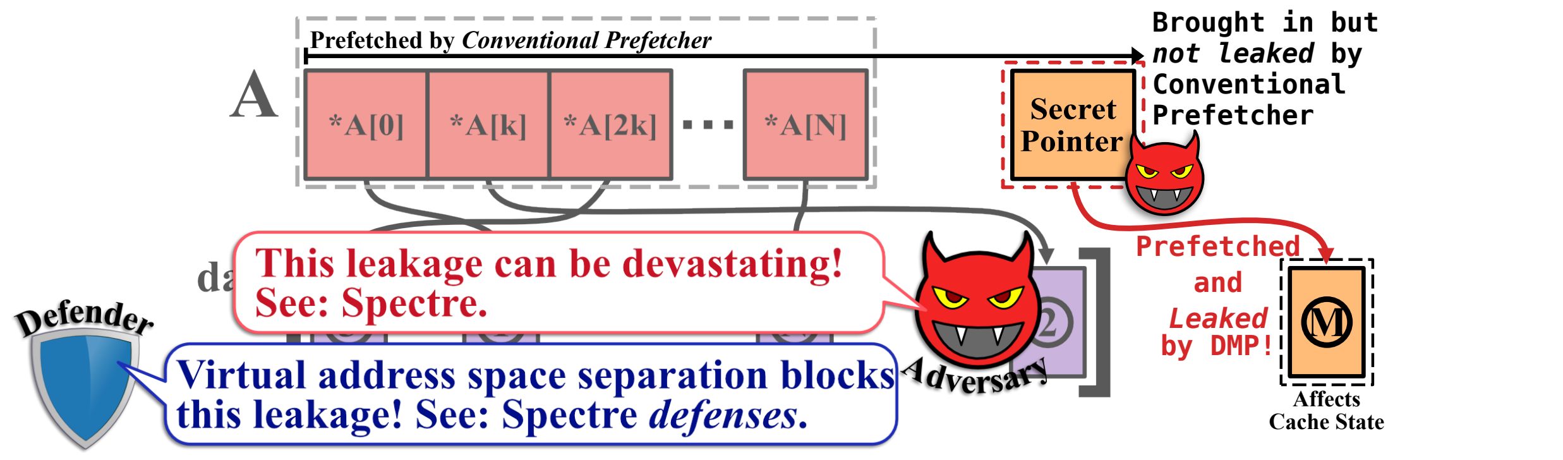
# We discover, in the wild, the first microarchitecture capable of leaking data at rest (in Apple's M1)

prefetchers.info

## Examples of Augury Techniques

### Out Of Bounds Reads

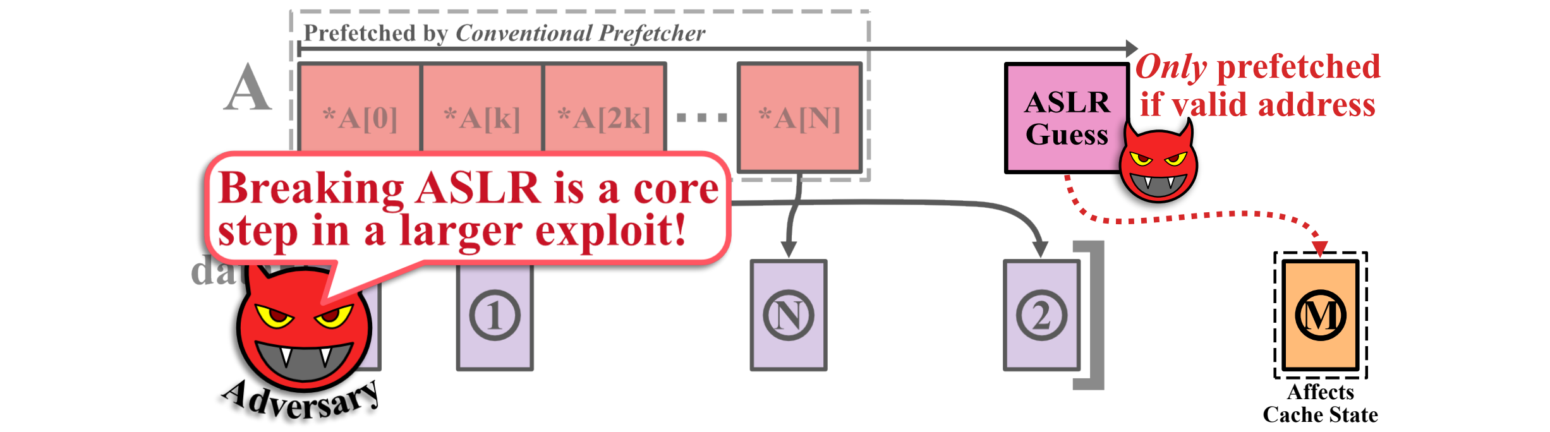
The DMP can be used to read past the end of a buffer, because it will overshoot the bounds of A and prefetch a pointer *which would not have been otherwise accessed.*


Prefetched by *Conventional Prefetcher*
A
*A[0]
*A[k]
*A[2k]
…
*A[N]
Secret Pointer
Brought in but *not leaked* by Conventional Prefetcher
Prefetched and *Leaked* by DMP!
M
Affects Cache State
2

Adversary: This leakage can be devastating! See: Spectre.

Defender: Virtual address space separation blocks this leakage! See: Spectre *defenses.*

### Breaking ASLR by Testing Virtual Addresses


Prefetched by *Conventional Prefetcher*
A
*A[0]
*A[k]
*A[2k]
…
*A[N]
ASLR Guess
*Only* prefetched if valid address
1
N
2
M
Affects Cache State

Adversary: Breaking ASLR is a core step in a larger exploit!

### Beating Speculative Load Hardening

Speculative load hardening (SLH) is a defense against conditional branch-based speculative execution attacks, known by the name of Spectre Variant #1.

**Spectre Vulnerable:**

```
for (i=0; i<N; i++):
    *A[k*i];
```

**Spectre Safe:**

```
for (i=0; i<N; i++):
    mask = (i>=N) ? 0 : ALL_ONES_BITMASK
    *A[k*(i & mask)];
```

**Masked in the core**

Since the DMP only ever sees cache misses, the (non-speculative) *memory access pattern* observed in both cases above is *identical.*

**Spectre Safe... What the Memory System Sees:**

```
for (i=0; i<N; i++):
    *A[k*i];   // DMP Vulnerable
```

Adversary: **It's unsurprising, but important to note: Some code vulnerable to Spectre V1, but protected by SLH, *continues to be vulnerable to the same receive side-channel as before!***

## Mitigating the Threat of DMPs

The paper details various strategies to protect secret data **from the M1's DMP.** These are:

1. Removing Secrets (Sandbox threat model only)
2. Preventing M1 DMP Interaction
3. Protecting Non-Pointer values (limited leakage is likely possible through page-walks or the TLB)

### General DMP Mitigations:

**Remove Secrets**

The only generalized, *but incomplete*, mitigation is to **remove secrets from the virtual address spaces accessible to adversaries** (like many Spectre mitigations).

Unfortunately, *there are many possible DMP designs* that could reach beyond a Spectre attack.

**Remove Gadgets**

We should also consider cases where a **privileged non-malicious program contains latent DMP gadgets** that must be detected and removed.

With aggressive DMPs (like the M1's), *a program can accidentally leak secret values* without any intervention.

## Conclusion

**Exotic microarchitectural optimizations that leak data never accessed by the core have arrived** in mainstream processors and are unlikely to disappear any time soon. While exceptional now, *we expect that this AoP DMP is only the first of many DMPs to be deployed across all architectures and manufacturers.*

## Acknowledgements

We thank Andrei Frumusanu for their exceptionally insightful remark "[...] we might believe they're using some sort of pointer-chase prefetching mechanism."[22]. We thank Dean Tullsen for seeding this idea. This work was funded partially by NSF grants 1954521 and 1942888, as well as by an Intel RARE grant.

The Grainger College of Engineering
UNIVERSITY OF ILLINOIS URBANA-CHAMPAIGN

UNIVERSITY of WASHINGTON

TEL AVIV UNIVERSITY