Binoculars: Contention-Based Side-Channel Attacks Exploiting the Page Walker

Zirui Neil Zhao, Adam Morrison, Christopher W. Fletcher, Josep Torrellas

University of Illinois       Tel Aviv University

USENIX Security’22
Microarchitectural Side Channel Attacks

**Root Cause:** shared hardware resource between the victim and the attacker

Persistent State Change?

- **Stateful**
  - Cached-based attacks *(e.g., Flush+Reload, Prime+Probe)*
  - Address translation-based attacks *(e.g., TLBleed, AnC)*

- **Stateless**
  - Port contention attacks *(e.g., PortSmash, SMoTherSpectre)*

Direct Result of Victim uOps?

- **Direct**
- **Indirect**

- Easy to exploit
- Hard to exploit

- Easily overlooked
Microarchitectural Side Channel Attacks

**Root Cause:** shared hardware resource between the victim and the attacker

Persistent State Change?

<table>
<thead>
<tr>
<th>Stateful</th>
<th>Stateless</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cached-based attacks</td>
<td>Port contention attacks</td>
</tr>
<tr>
<td>(e.g., Flush+Reload, Prime+Probe)</td>
<td>(e.g., PortSmash, SMoTherSpectre)</td>
</tr>
</tbody>
</table>

Address translation-based attacks (e.g., TLBleed, AnC)

Binoculares Attack
(This Work)
The 1st Stateless-Indirect Channel: Binoculars

**Binoculars Attack**  
(This Work)

Hyper-Thread 1  
(HT1)

Sibling Hyper-Thread 2  
(HT2)

Page Walk  
Normal Memory Accesses

*The contention is NOT due to cache footprint

**Highlights:**

+ Easy to observe, up to 20K-cycle contention (with a single dynamic instruction)
+ Leak a wide range of virtual address bits
Virtual Address Translation & Page Walk

Core → Load VA1

Page Walker

L1 Cache

Page Walk

L2 Cache

L3 Cache

Main Memory

Page Table

TLB

TLB Miss!
Virtual Address Translation & Page Walk (x64)

Virtual Page Number (VPN) – Bits 47:12

Offset – Bits 11:0

Virtual Address (VA)

PL4 Index  PL3 Index  PL2 Index  PL1 Index

47  12  11  0

CR3

PGD*  PUD*  PMD*  PTE*

PA

To TLB

Page Walker Load

Addr[11:3] = PL4 Index

L1 Cache

*Each page-table entry is 8-byte long
Three key takeaways:

- Page walker issues multiple (e.g., four) *page walker loads*
- Address bits 11-3 of a page walker load *is determined by its corresponding PL Index*
- Page walker loads go through the cache hierarchy and are *subject to resource contention*
The Binoculars Attack

<table>
<thead>
<tr>
<th>Virtual Address (VA)</th>
<th>PL4 Index</th>
<th>PL3 Index</th>
<th>PL2 Index</th>
<th>PL1 Index</th>
<th>Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>47</td>
<td></td>
<td></td>
<td>12</td>
<td>11</td>
</tr>
<tr>
<td></td>
<td></td>
<td>11</td>
<td>11</td>
<td></td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>RA4</td>
<td>RA3</td>
<td>RA2</td>
<td>RA1</td>
<td></td>
</tr>
</tbody>
</table>

Strong resource contention if WA is “4K-aliasing” with any RA\(_i\) (i.e., share bits 11-0)

Security Implications:
- A high signal-to-noise channel
- Address-dependent contention

+ Up to 20K cycle delay!
+ Occurs regardless of what level in the cache hierarchy the page walker loads read from

```
while (true) {
    store 0 -> WA;
}
```

Repeated Writes

+ Sibling HT2 (Writer)

+ HT1 (Reader)
Primitive 1: Store→Load Channel

Intuition: the attacker triggers page walker loads while the victim writes. If the attacker observes strong contention:
⇒ learn victim stores’ offset

HT1 (Reader)  
TLB-missing access to VA

HT2 (Writer)

Page Walker Loads for translating VA

Information Flow

Sub-cacheline resolution
Demo: Store→Load Channel

HT1 (Reader)  
HT2 (Writer)

$RA_1$ Offset*  
Page 0: 0x000  
Page 1: 0x008  
Page 2: 0x010  
Page 165: 0x528  
Page 511: 0xff8

TLB-missing access
Repeat Reads

Store Offset
Repeated Writes

4K-Aliasing!
Attacker access latency spikes

$RA_1$ Offset = $PL_1$ Index × 8

Attacker-probed latency  
(100 measurements on a Skylake-X machine)

*Latency (cycles)
Offset Under Probing

$R_A^*$ $A$ $O$ $f$ $f$ $t$ $h$ $e$ $s$ $t$ $= P_L × 8$
Primitive 2: Load→Store Channel

**Intuition:** the attacker writes while the victim performs a page walk. If the attacker observes a victim slowdown:

⇒ learn the set of $PL$ indexes of the page that the victim accesses

**Can reconstruct the entire victim VPN**

**Demos can be found in Section 4.2**
Root Cause of Strong Contention

**Intel’s Patent**: issue page walker loads as “stuffed loads”, which bypass the instruction scheduler to avoid any scheduling latency

More details in Section 5 and Appendix A
- Detailed reverse engineering
- Types of resource conflicts in L1 cache
- ...

Attack Montgomery Ladder and ECDSA

**ECDSA**: a digital signature algorithm. One step during an ECDSA signing is to compute the point $k \times G$, where $k$ is the nonce and $G$ is the base point

Knowing the nonce $k$ and the corresponding signature
⇒ derive the private key used for signing

**Goal**: learn the nonce $k$

**Challenge**: the nonce $k$ changes at every victim run and never repeats
⇒ requires a channel with high signal-to-noise ratio to extract $k$ with
   a single victim execution

**Target**: implementation uses *Montgomery ladder* to speed up the signing (OpenSSL 1.0.1e)
Attack Montgomery Ladder and ECDSA

A simplified Montgomery ladder iteration

```c
BIGNUM *x1, *z1, *x2, *z2;
if (k_i) { // checks ith bit of k
  Madd(x1, z1, x2, z2);
  Mdouble(x2, z2);
} else {
  Madd(x2, z2, x1, z1);
  Mdouble(x1, z1);
}
```

Data-oblivious to the sequence of operations and end-to-end timing

Secret-dependent reordering of stores

Detect it with the Store→Load Channel
Attack Montgomery Ladder and ECDSA

Latency Trace of Probing Stores to $x_2$

$k_i = 1$
- $st \rightarrow (x_1, z_1)$
- $st \rightarrow (x_2, z_2)$

$k_i = 0$
- $st \rightarrow (x_2, z_2)$
- $st \rightarrow (x_1, z_1)$
Attack Montgomery Ladder and ECDSA

Oracle Montgomery ladder iteration boundaries

Signal Processing on Raw Traces:
1) Recover iteration boundaries
2) Predict $k_i$

Evaluated on Skylake-X and Cascade Lake-X (100 traces):
+ Average $k_i$ prediction accuracy: 98.4%~98.5%
+ Median brute force attempts: $\approx 2^{24}$
Conclusions

**Binoculars:** the first stateless-indirect channel

1. Easy to Observe
   + Up to 20K-cycle contention
   + High signal-to-noise ratio
   + Extract nonce $k$ with a single victim execution

2. Wide VA Bits Coverage
   
   ![Diagram showing Load→Store and Store→Load with 36, 9, and 3 bits]

3. Open Source
   
   [https://github.com/zzrcxb/binoculars](https://github.com/zzrcxb/binoculars)