Replicating Side Channel Attacks on RISC-V

Ben Chen, Shuwei Zhang, Jikun Liao

Dept of Computer Science and Technology, SUSTech

December 26, 2024

Introduction

In this project, we

- spent 5 weeks to survey the microarchitectural vulnerabilities

Introduction

In this project, we

- spent 5 weeks to survey the microarchitectural vulnerabilities
- replicated Spectre, Phantom and Downfall on x86-64 CPU

Introduction

In this project, we

- spent 5 weeks to survey the microarchitectural vulnerabilities
- replicated Spectre, Phantom and Downfall on x86-64 CPU
- analyzed the attack surfaces on XiangShan

Introduction

In this project, we

- spent 5 weeks to survey the microarchitectural vulnerabilities
- replicated Spectre, Phantom and Downfall on x86-64 CPU
- analyzed the attack surfaces on XiangShan
- set up the workflow for running XiangShan on simulator

Introduction

In this project, we

- spent 5 weeks to survey the microarchitectural vulnerabilities
- replicated Spectre, Phantom and Downfall on x86-64 CPU
- analyzed the attack surfaces on XiangShan
- set up the workflow for running XiangShan on simulator
- migrated the attacks on XiangShan

Background

Side-Channel Attacks

Side channel attacks exploit unintended information leakage in

- energy: voltage, power, radiation...

Background

Side-Channel Attacks

Side channel attacks exploit unintended information leakage in

- energy: voltage, power, radiation...
- timing: execution cost $\rightarrow$ branching result

Background

Side-Channel Attacks

Side channel attacks exploit unintended information leakage in

- energy: voltage, power, radiation...
- timing: execution cost $\rightarrow$ branching result
- cache: timing difference $\rightarrow$ data present or not in cache

Side channel attacks exploit unintended information leakage in

- energy: voltage, power, radiation...
- timing: execution cost $\rightarrow$ branching result
- cache: timing difference $\rightarrow$ data present or not in cache
- contention: cross-core interrupts, bus, and other resource contention $\rightarrow$ network fingerprints

Background

Side-Channel Attacks

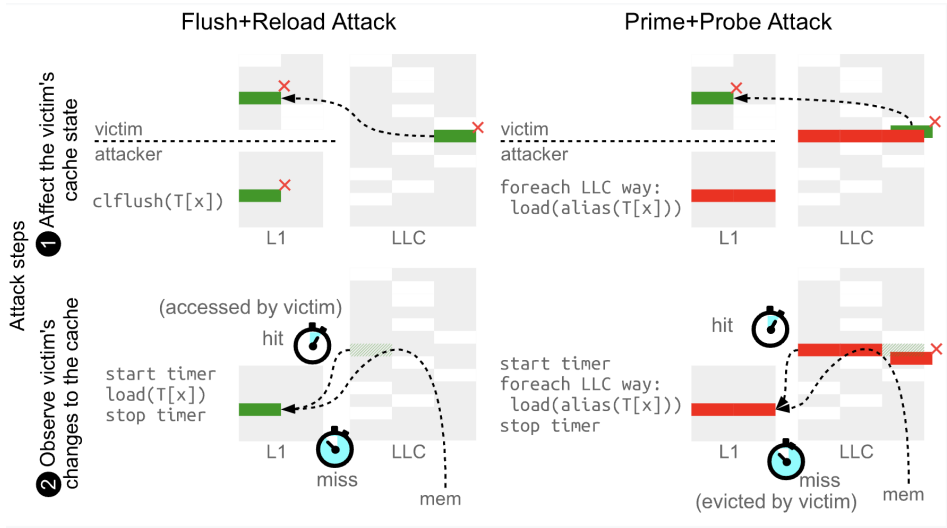
Side channel attacks exploit unintended information leakage in

- energy: voltage, power, radiation...
- timing: execution cost $\rightarrow$ branching result
- cache: timing difference $\rightarrow$ data present or not in cache
- contention: cross-core interrupts, bus, and other resource contention $\rightarrow$ network fingerprints
- other architecture: MWAIT[1] status change encoded in user-readable register

Background

Cache Side-Channel

- Flush+Reload: Flush all cache of attacker's probe array, encode victim's secret in offset of probe array and reload to time the difference
- Prime+Probe: Fill the cache with attacker's probe array, and victim's cache evict a element of probe array in cache, so attacker can tell which one is evicted



Motivation

We've seen plentiful defense on RISC-V

- SafeSpec[2]: Blocking unsafe loads from altering the data cache
- SpectreGuard[3]/SpecTerminator[4]: Marking the unsafe load to prevent speculative load

but few attack on RISC-V and especially on XiangShan

- A Secure RISC[5]: Attack I$ on C906 & U74

Motivation

XiangShan is a open-source RISC-V core IP

- developed by ICT of CAS, led by Yungang Bao

Motivation

XiangShan is a open-source RISC-V core IP

- developed by ICT of CAS, led by Yungang Bao
- out of order, superscalar

Motivation

XiangShan is a open-source RISC-V core IP

- developed by ICT of CAS, led by Yungang Bao
- out of order, superscalar
- runs RISC-V RV64GCV, Linux-capable, FPGA-capable

Motivation

XiangShan is a open-source RISC-V core IP

- developed by ICT of CAS, led by Yungang Bao
- out of order, superscalar
- runs RISC-V RV64GCV, Linux-capable, FPGA-capable
- written in Chisel, agile development and verification

Motivation

XiangShan is a open-source RISC-V core IP

- developed by ICT of CAS, led by Yungang Bao
- out of order, superscalar
- runs RISC-V RV64GCV, Linux-capable, FPGA-capable
- written in Chisel, agile development and verification

Is XiangShan vulnerable to Spectre or so?

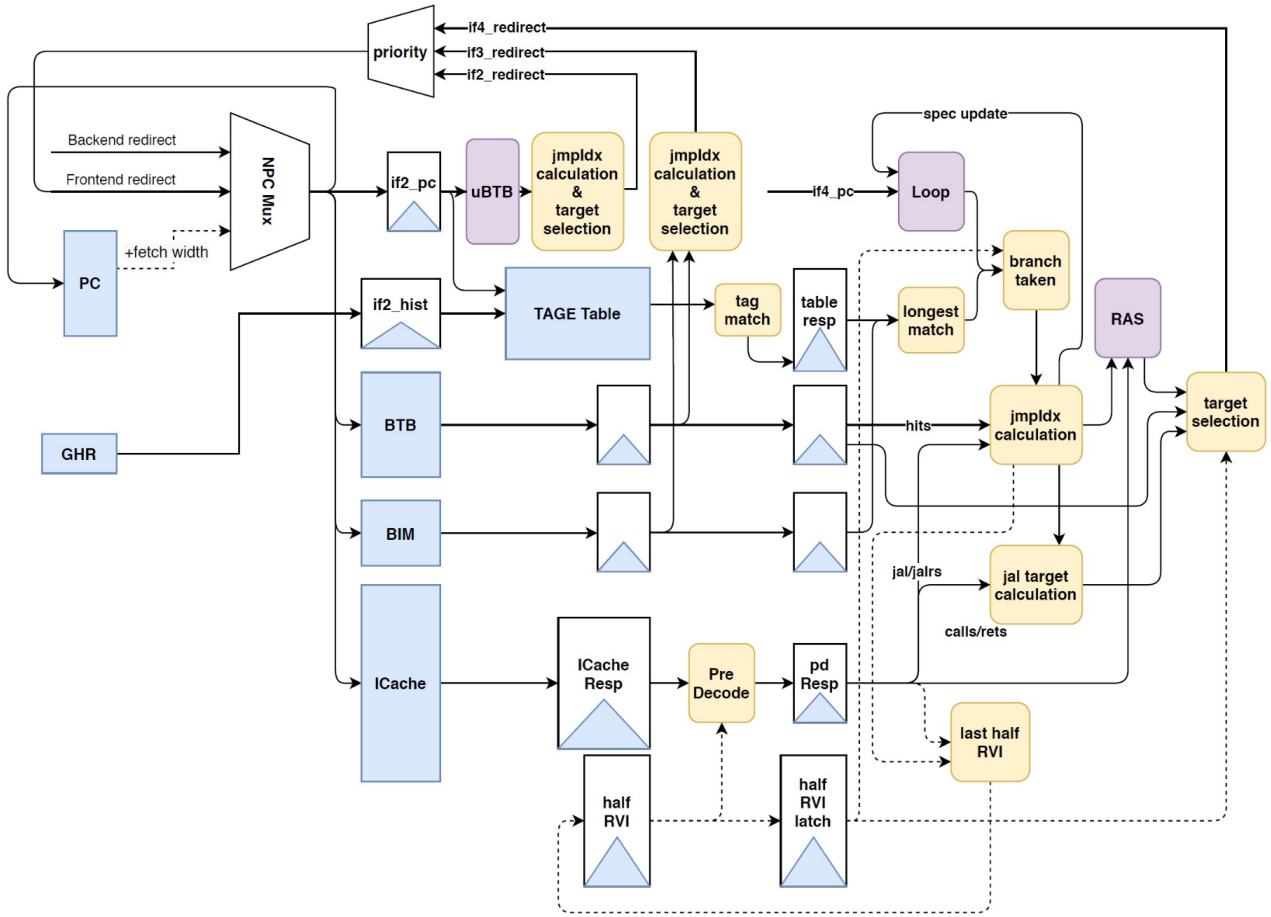
Vulnerable code in victim

```
uint8_t array[160] = {1, 2, ..., 16};
char* secret = "Secret goes here!";

void victim_function(size_t x) {
   if (x < array1_size) { // array1_size = 16
       temp &= array2[array1[x] * CACHELINE_SIZE];
   }
}
```

Training the branch predictor

```
for (int x = 0; x < ENTRY_SIZE; ++x) {
    victim_function(x);
}
flush_cache(array2);
victim_function(secret[i++]);

for (int i = 0; i < 256; i++) {
    addr = &array2[i * CACHELINE_SIZE];
    time1 = __rdtscp(&junk); /* READ TIMER */
    junk = * addr; /* MEMORY ACCESS TO TIME */
    time2 = __rdtscp(&junk) - time1;
    /* READ TIMER & COMPUTE ELAPSED TIME */
    if (time2 <= CACHE_HIT_THRESHOLD)
        results[i]++;
}
```

XiangShan Branch Predictor



Spectre

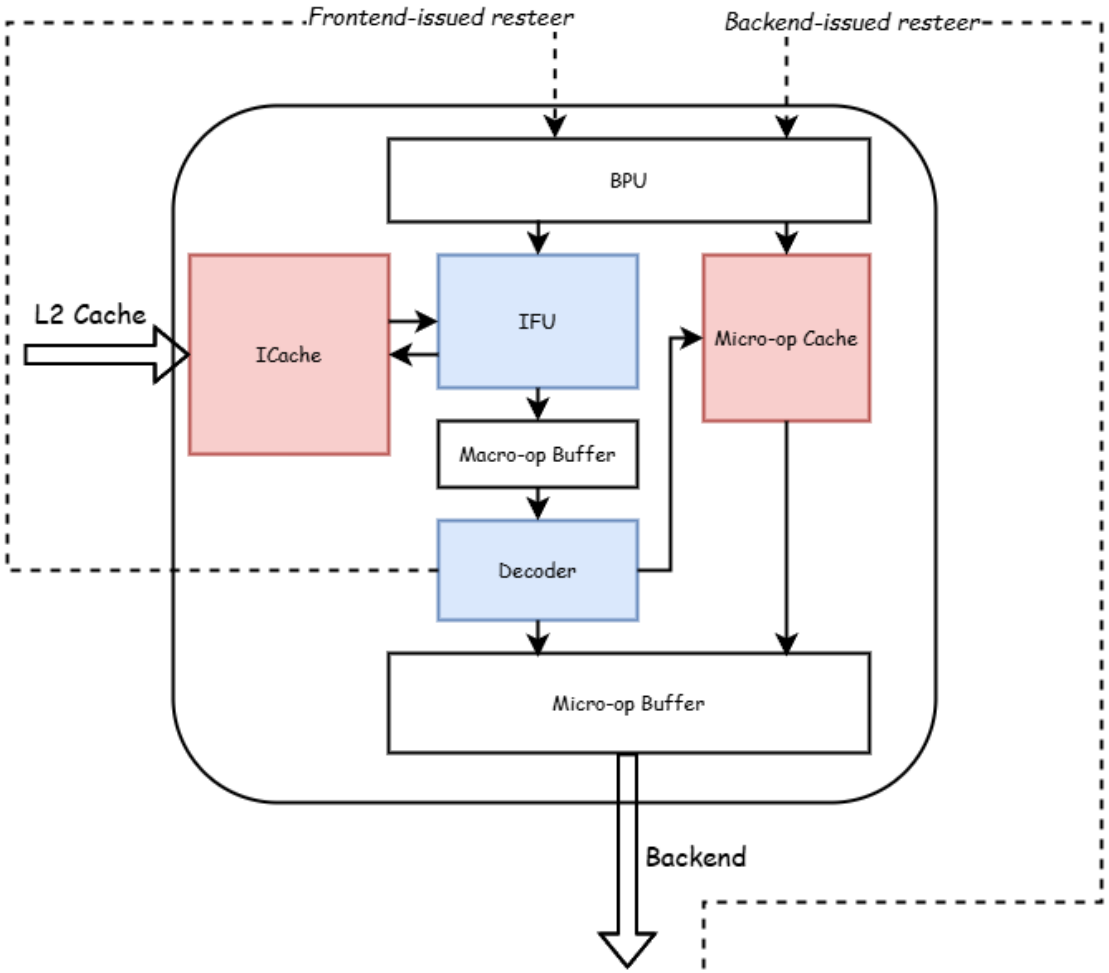
Spectre on XiangShan

- Branch predictors: μBTB, BTB, TAGE, RAS and loop predictor
  TAGE + loop predictor $\rightarrow$ disable the original attack
- Speculative Execution
  Out-of-order, multiple issue $\rightarrow$ Spectre-vulnerable
- Cache: L1 \$ L2
  8 ways, 256 sets, 64B each line $\rightarrow$ enough to encode a character
- Timer and cache manipulation
  `rdcycle` cycle-level timer, `fence.i` memory barrier

Spectre on Nanhu

```
for (int i = 0; i <= ENTRY_SIZE; ++i) {
    int x = i < ENTRY_SIZE ? i : secret[i++]
    \\ optimize branch to jump table
    flush_cache(array2);
    victim_function(x);
}

for (int i = 0; i < 256; i++) {
    addr = &array2[i * CACHELINE_SIZE];
    time1 = rdcycle(); /* READ TIMER */
    junk = * addr; /* MEMORY ACCESS TO TIME */
    time2 = rdcycle() - time1;
    /* READ TIMER & COMPUTE ELAPSED TIME */
    if (time2 <= CACHE_HIT_THRESHOLD)
        results[i]++;
}
```

Decode Pipeline



Branch Prediction

- Prediction on instruction type.
- Prediction on branch target.

Misprediction Resteer

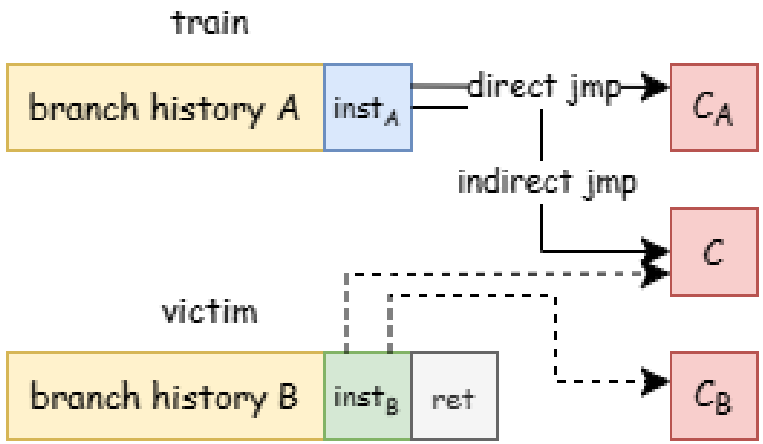
- Frontend Resteer:
  - Mismatch of predicted instruction types.
  - Incorrect branch prediction address (Direct branch).
- Backend Resteer:
  - Taken/Not-taken conditional branch.
  - Incorrect branch prediction address (Indirect branch).

Phantom

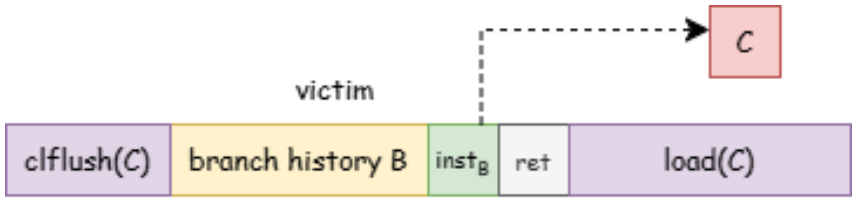
- Exploit transient window caused by frontend resteer.

Phantom Workflow

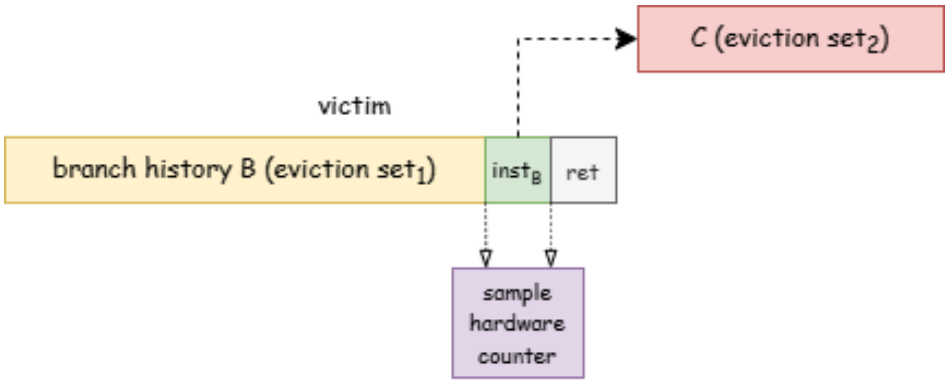
- Train A with direct / indirect branch to C.
- Execute B at aliased address to trigger misprediction to C.
- Set up observation channel to monitor C's advance in pipeline.



IF Channel



ID Channel



EX Channel

Example: IF Channel

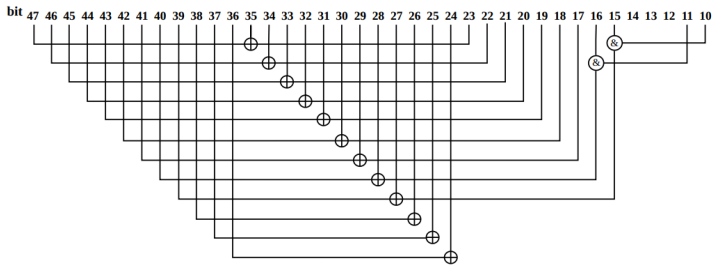
```
for (int j = 0; j < 8; j++)
    train_func();
memory_barrier //asm volatile("sfence;\nmfence;\nlfence")
clflush((void*)monitor_addr);
memory_barrier
victim_func();
delayloop(100000);
memory_barrier
uint64_t tim = memaccesstime((void*)monitor_addr);
```

Hardware Performance Counter used in ID Channel

- Zen2
  - *de_dis_uops_from_decoder.de_dis_uops_from_opcache*
  - *de_dis_uops_from_decoder.de_dis_uops_from_both*
- Zen3
  - *op_cache_hit_miss.op_cache_miss*

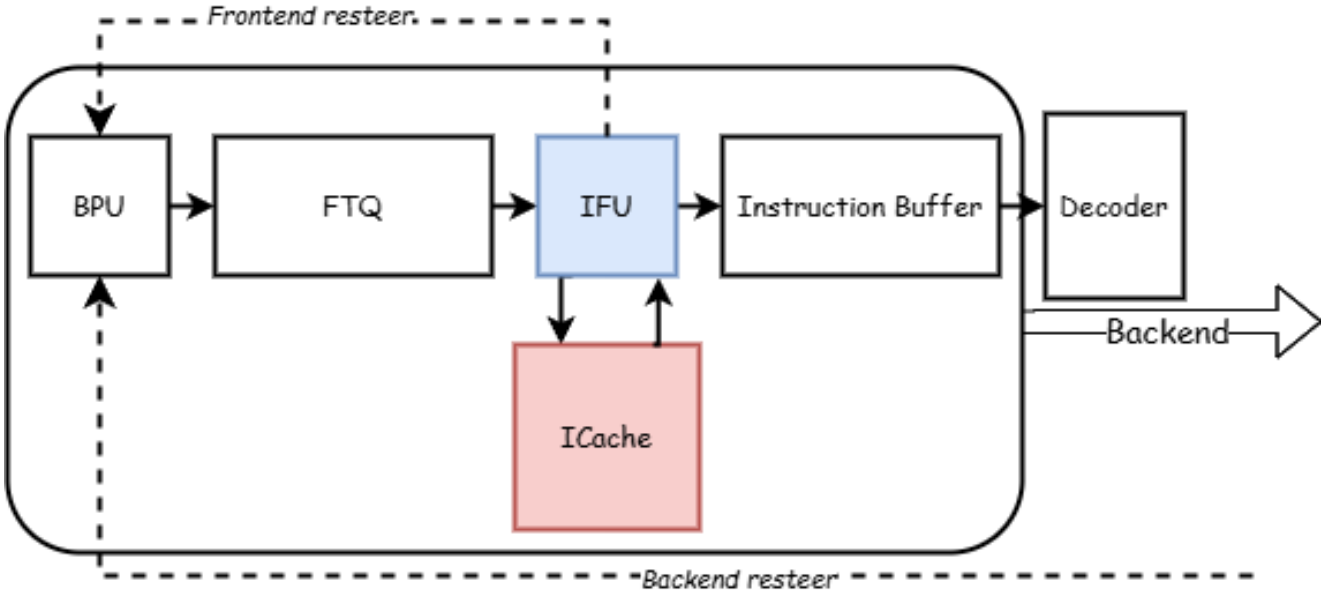
Trigger misprediction

- Create aliased address:
  Flip 19th bit and 31st bit on Zen2, 21st bit and 33rd bit on Zen3.
- Set up branch history:
  4∼8 direct jumps separated by 128 bytes.



$$
\begin{aligned}
f_0 &= b_{47} \oplus b_{35} \oplus b_{23} & \qquad f_1 &= b_{47} \oplus b_{36} \oplus b_{24} \oplus b_{12} \\
f_2 &= b_{47} \oplus b_{37} \oplus b_{25} \oplus b_{13} & f_3 &= b_{47} \oplus b_{38} \oplus b_{26} \oplus b_{14} \\
f_4 &= b_{47} \oplus b_{39} \oplus b_{26} \oplus b_{13} & f_5 &= b_{47} \oplus b_{39} \oplus b_{27} \oplus b_{15} \\
f_6 &= b_{47} \oplus b_{40} \oplus b_{28} \oplus b_{16} & f_7 &= b_{47} \oplus b_{41} \oplus b_{29} \oplus b_{17} \\
f_8 &= b_{47} \oplus b_{42} \oplus b_{30} \oplus b_{18} & f_9 &= b_{47} \oplus b_{43} \oplus b_{31} \oplus b_{19} \\
f_{10} &= b_{47} \oplus b_{44} \oplus b_{32} \oplus b_{20} & f_{11} &= b_{47} \oplus b_{45} \oplus b_{33} \oplus b_{21}
\end{aligned}
$$

Nanhu Frontend



Main Different:

- No μop cache.
- Integrated Instruction Fetch and Prediction Check.

Observation Channel

- IFU *frontend_icache_miss_cnt*, *frontend_flush*
- Decoder *ctrlblock_decoder_utilization*

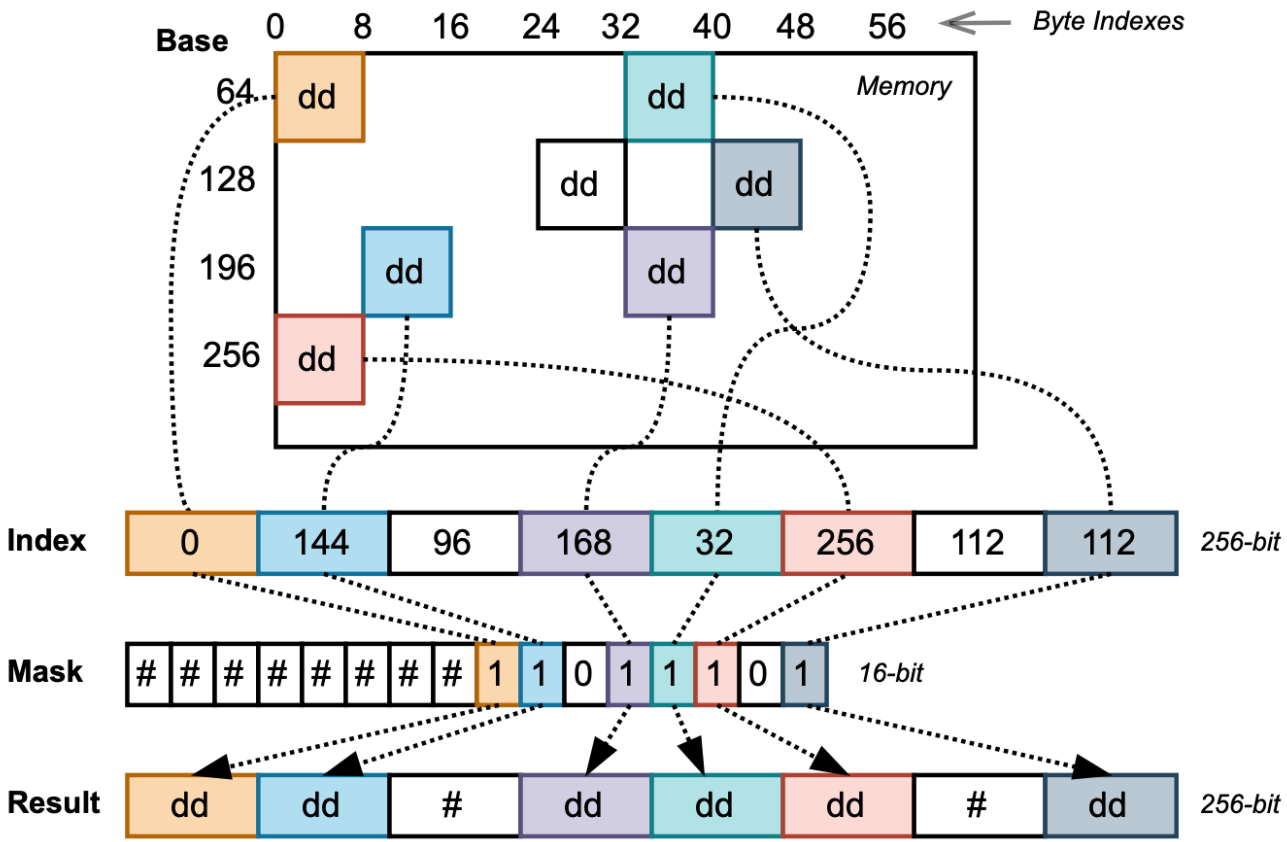
Trigger misprediction

- Create aliased address:

  Using same lower 29 bits
- Set up branch history:

  32 direct jumps separated by 64 bytes.

Execution of gather instruction[6]



Downfall

Gather Data Sampling

Microarchitectural data sampling exploits

- buffers inside microarchitectural components like load buffer, store-commit buffer

Downfall

Gather Data Sampling

Microarchitectural data sampling exploits

- buffers inside microarchitectural components like load buffer, store-commit buffer
- x86 has hyperthreading tech, allowing two threads run on the same core sharing the same resource $\rightarrow$ potential data stealing

Downfall

Gather Data Sampling

Microarchitectural data sampling exploits

- buffers inside microarchitectural components like load buffer, store-commit buffer
- x86 has hyperthreading tech, allowing two threads run on the same core sharing the same resource $\rightarrow$ potential data stealing
- hard to conduct, since data in buffer vanished quickly

Downfall

Gather Data Sampling

Microarchitectural data sampling exploits

- buffers inside microarchitectural components like load buffer, store-commit buffer
- x86 has hyperthreading tech, allowing two threads run on the same core sharing the same resource $\rightarrow$ potential data stealing
- hard to conduct, since data in buffer vanished quickly
- `gather` magnifies the attack by filling up the buffer with vector load

Downfall

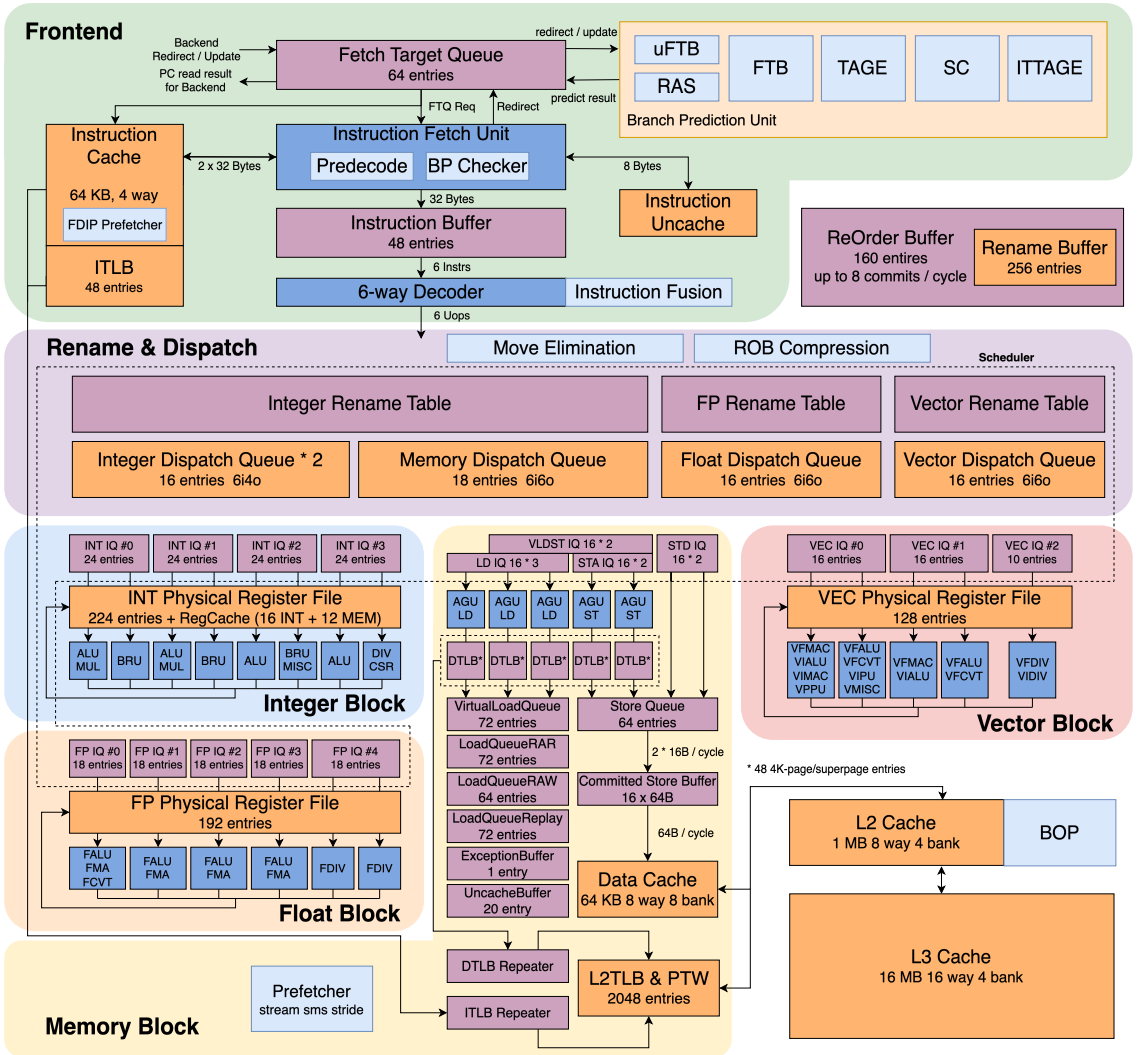
Gather Data Sampling

Microarchitectural data sampling exploits

- buffers inside microarchitectural components like load buffer, store-commit buffer
- x86 has hyperthreading tech, allowing two threads run on the same core sharing the same resource $\rightarrow$ potential data stealing
- hard to conduct, since data in buffer vanished quickly
- `gather` magnifies the attack by filling up the buffer with vector load
- then encoding

Downfall

Kunminghu Architecture



Downfall

RISC-V Vector Extension

Similar to Intel's AVX and ARM's SVE

- XiangShan adds on 32 128-bit vector registers and 7 vector CSR

Downfall

RISC-V Vector Extension

Similar to Intel's AVX and ARM's SVE

- XiangShan adds on 32 128-bit vector registers and 7 vector CSR
- setting vtype and vlen before computing vectors

Downfall

RISC-V Vector Extension

Similar to Intel's AVX and ARM's SVE

- XiangShan adds on 32 128-bit vector registers and 7 vector CSR
- setting vtype and vlen before computing vectors
- Vector load, indexed (gather): vlxb, vlxbu, vlxh, vlxhu, vlxw, vlxwu, vlxd, vflxh, vflxw, vflxd

Downfall

RISC-V Vector Extension

Similar to Intel's AVX and ARM's SVE

- XiangShan adds on 32 128-bit vector registers and 7 vector CSR
- setting vtype and vlen before computing vectors
- Vector load, indexed (gather): vlxb, vlxbu, vlxh, vlxhu, vlxw, vlxwu, vlxd, vflxh, vflxw, vflxd
- XiangShan reuse the Load-Store Unit in execution of vector instruction $\rightarrow$ GDS attack

Downfall with RVV

```
fence.i
// increase the transient window
vsetvli t0, %[vl], e64, m1
// Set vector length and element width to 64 bits
vmv.v.x v0, %[mask]
// Move mask to vector register v0
vle32.v v2, (%[indices])
// Load indices into vector register v2
vluxei64.v v1, (%[src]), v2, v0.t
// Load 64-bit elements using indices and mask
vse64.v v1, (%[dst]) Store loaded elements to dst

encode_secret
flush_and_reload
```

Experiment

CPU	Generation	Memory
Xeon(R) Silver 4210	Cascade Lake	DDR4 128GB
AMD R9-3900X	Zen2	DDR3 32GB
XiangShan	Nanhu (FPGA)	DDR3 16GB
XiangShan	Kunminghu (Verilator & GEM5)	DDR3 8GB

Table: Tested machines

Experiment

CPU	Generation	Memory
Xeon(R) Silver 4210	Cascade Lake	DDR4 128GB
AMD R9-3900X	Zen2	DDR3 32GB
XiangShan	Nanhu (FPGA)	DDR3 16GB
XiangShan	Kunminghu (Verilator & GEM5)	DDR3 8GB

Table: Tested machines

CPU	Spectre	Phantom	Downfall
Xeon(R) Silver 4210	✓		✓
AMD R9-3900X		✓	
XiangShan (Nanhu)	✓	Ongoing	No RVV
XiangShan (Kunminghu)			Ongoing

Table: Result

Future Work

We plan to work on..

- continue to implement the remaining attacks on XiangShan
- propose defense techniques and implement on XiangShan
- explore more attack surfaces on XiangShan and RISC-V

Conclusion

- Validated side-channel attacks on x86
  Spectre v1, Phantom, and Downfall
- Theoretically proved that side-channel attacks on XiangShan and partly implement them
  trying to overcome the inconvenience in the ecosystem
- Continue implementing attacks on RISC-V
- Plan to mitigate the side-channels on RISC-V

References I

[1] Ruiyi Zhang et al. "(M)WAIT for It: Bridging the Gap between Microarchitectural and Architectural Side Channels". In: *32nd USENIX Security Symposium (USENIX Security 23)*. Anaheim, CA: USENIX Association, Aug. 2023, pp. 7267–7284. ISBN: 978-1-939133-37-3. URL: https://www.usenix.org/conference/usenixsecurity23/presentation/zhang-ruiyi.

[2] Khaled N. Khasawneh et al. "SafeSpec: Banishing the Spectre of a Meltdown with Leakage-Free Speculation". In: *2019 56th ACM/IEEE Design Automation Conference (DAC)*. 2019, pp. 1–6.

[3] Jacob Fustos, Farzad Farshchi, and Heechul Yun. "SpectreGuard: An Efficient Data-centric Defense Mechanism against Spectre Attacks". In: *2019 56th ACM/IEEE Design Automation Conference (DAC)*. 2019, pp. 1–6.

References II

[4] Hai Jin, Zhuo He, and Weizhong Qiang. "SpecTerminator: Blocking Speculative Side Channels Based on Instruction Classes on RISC-V". In: 20.1 (Feb. 2023). ISSN: 1544-3566. DOI: 10.1145/3566053. URL: https://doi.org/10.1145/3566053.

[5] Lukas Gerlach et al. "A Security RISC: Microarchitectural Attacks on Hardware RISC-V CPUs". In: *2023 IEEE Symposium on Security and Privacy (SP)*. 2023, pp. 2321–2338. DOI: 10.1109/SP46215.2023.10179399.

[6] Daniel Moghimi. "Downfall: Exploiting Speculative Data Gathering". In: *32th USENIX Security Symposium (USENIX Security 2023)*. 2023.