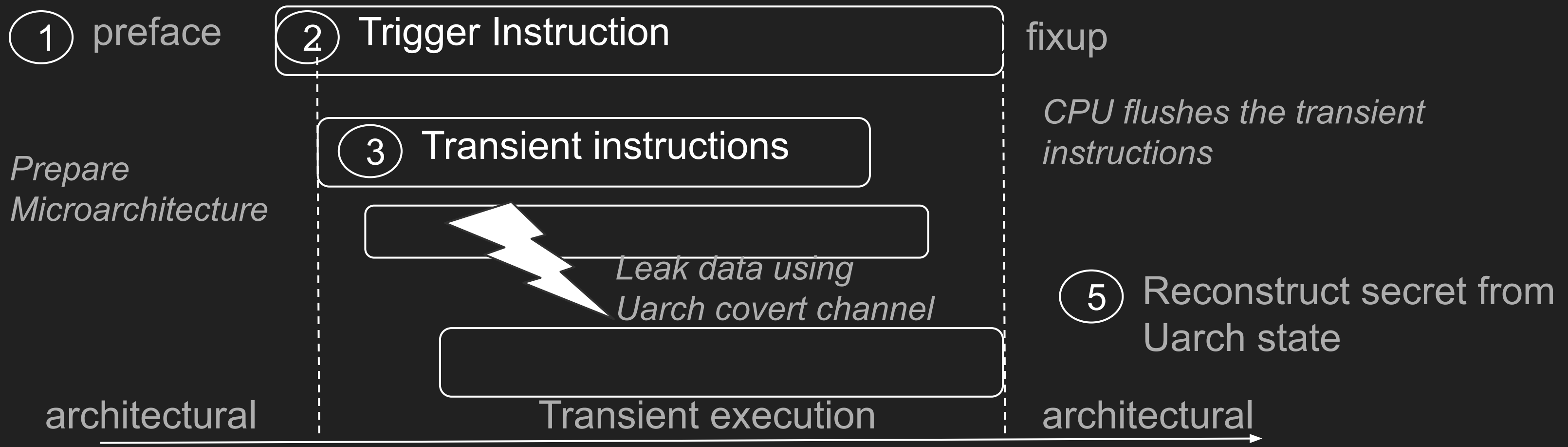
Transient Execution and Speculation Barriers

Ved Shanbhogue - ved@rivosinc.com

Overview of a Transient Execution


1 preface
Prepare Microarchitecture
2 Trigger Instruction
3 Transient instructions
Leak data using Uarch covert channel
fixup
CPU flushes the transient instructions
5 Reconstruct secret from Uarch state
architectural
Transient execution
architectural

Arch. state - x/f/v registers, pc, CSRs.
Uarch state - TLBs, caches, etc.
Transient instructions - instructions whose results are not committed to architectural state
- However uarch state may change - e.g., cache blocks, TLB entries, cache/TLB LRU, etc.

Speculative execution - predict outcome of a branch or data dependency to avoid stalling
- Instructions on mispredicted paths are transient instructions

Attacker prepare uarch as precursor - training predictors, flushing caches, etc.

Ref: A Systematic Evaluation of Transient Execution Attacks and Defenses - Canella et al

Spectre vs. Meltdown

Spectre: Exploits transient execution following control or data flow misprediction

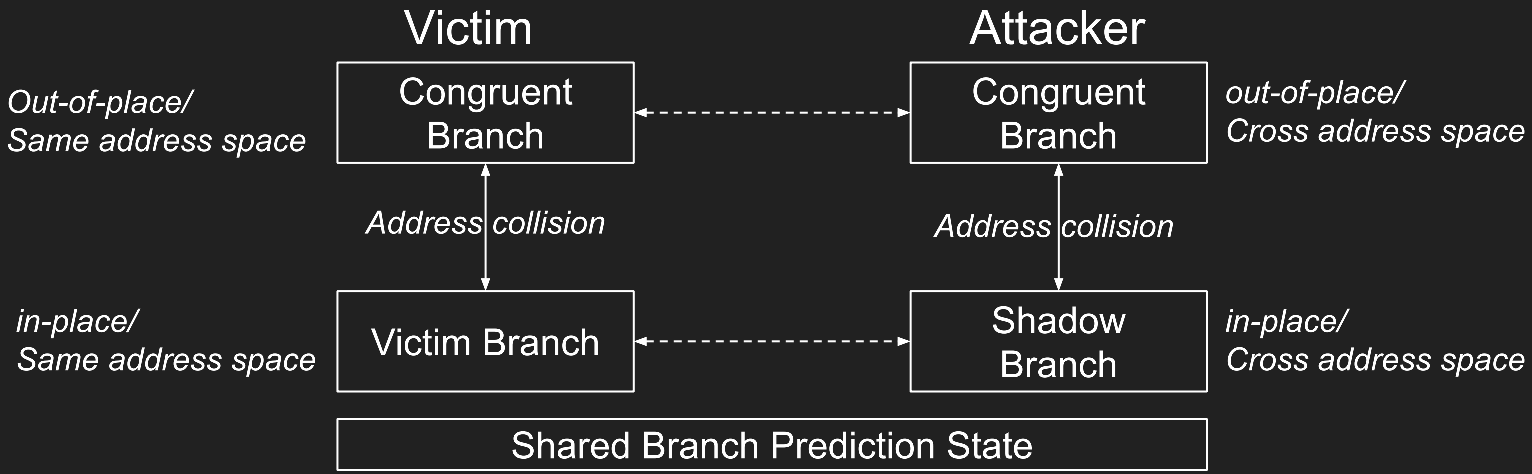
- Relies on the control flow, data dependency, or data value predictions
- Mostly applicable to out-of-order processors
- Bypass SW defined security policies (e.g., bound checks, call/return, store to load ordering, etc.)
- Trick the victim into computing on memory locations that the victim can access but the attacker cannot
- Defenses may involve non-trivial effort and in many cases HW/SW co-design

Meltdown: Exploits transient execution following a faulting instruction

- Use data forwarded by a faulting instruction
- Also applicable to in-order processors
- Bypass hardware enforced security policies (User/Supervisor, privilege levels, etc.)
- Really a hardware bug - defense usually as simple as faulting instructions providing fixed/dummy data

Spectre variants

	Predictors exploited	Description
Spectre-PHT	PHT, BHB	Pattern History Table - predicts outcome of conditional branches
Spectre-BTB	BTB, BHB, PHT	Branch Target Buffer - predicts branch destination address
Spectre-RSB	RSB, BTB	Predict return addresses.
Spectre-STL	STL, Value prediction	Memory disambiguation to predict store to load F/W dependencies.


Victim
Attacker
Out-of-place/ Same address space
Congruent Branch
Congruent Branch
out-of-place/ Cross address space
Address collision
Address collision
in-place/ Same address space
Victim Branch
Shadow Branch
in-place/ Cross address space
Shared Branch Prediction State

Some principles that can help with defences

1. Predictions learned in one privilege domain or one address space must not be used to make predictions in a different privilege domain
2. Predictions must not be learned speculatively.
3. Loads - from memory and registers - must not provide data till the privilege to access that memory or register is determined.

Defence strategies

Disable predictors => feasible; unacceptable performance cost

Fixup transient uarch state => unacceptable perf./impl. cost

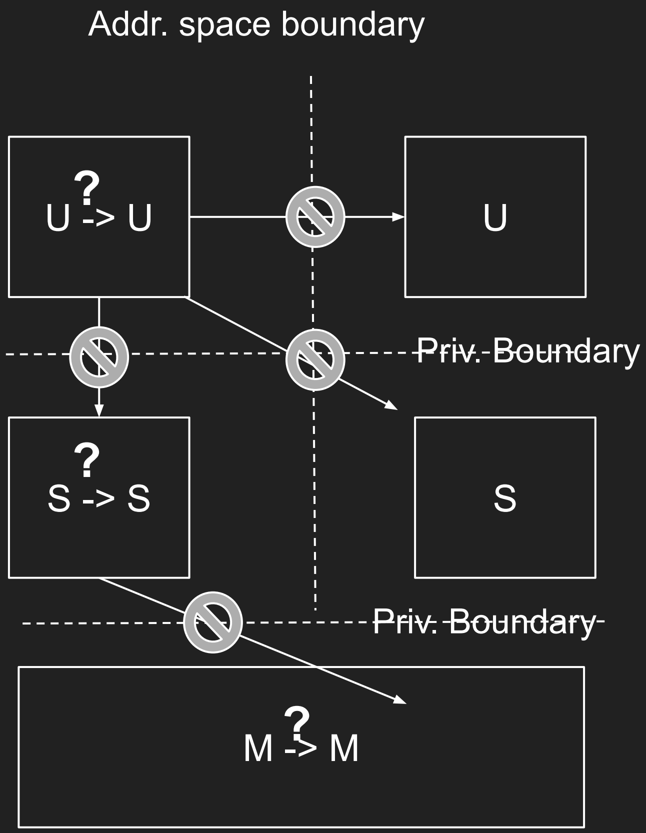
Tag Predictors => feasible; priv. levels, ASID/VMID

Partition Predictors => feasible; priv. levels, ASID/VMID

Cryptographic defenses => feasible; priv. levels, ASID/VMID

Flush Predictors => feasible; perf. Cost may be unacceptable

Speculation Barriers => needed where barriers are not known;
- HW/SW co-design
- Address in-place/same-address space



Proposed Speculation Barriers - Zisb

The Zisb extension introduces three barriers - an execution barrier and two memory barriers - to control speculation. These instructions are encoded using ZImop code points.

TEB - Transient Execution Barrier

Execution of instructions following the TEB in program order must not be observed through side channels until the TEB completes. Instructions after the TEB cannot be speculatively executed as a result of control-flow or data-value speculation.

MDB - Memory Disambiguation Barrier

Loads occurring after the MDB must not bypass stores before the MDB that target the same memory location. Similarly, loads preceding the MDB must not speculatively read data from stores to the same memory location that occur after the MDB.

DPB - Data-value Prediction Barrier

Instructions, excluding branch instructions, that appear after the DPB must not execute speculatively based on the results of any data-value speculation before the DPB.

Example: Transient Execution Barrier

```c
// Exploited C-code example
struct array {
    unsigned long length;
    unsigned char data[];
};
struct array *arr1 = ...; /* small array */
struct array *arr2 = ...; /* array of size 0x400 */
unsigned long untrusted_offset_from_user = ...;
if (untrusted_offset_from_user < arr1->length) {
    unsigned char value;
    value = arr1->data[untrusted_offset_from_user];
    unsigned long index2 = ((value & 1)*0x100)+0x200;
    if (index2 < arr2->length) {
        unsigned char value2 = arr2->data[index2];
    }
}
```

```
// Generated assembly code with problematic branch
// and mitigation
if (untrusted_offset_from_user < arr1->length) {
        ld a4,0(zero)
        ld a5,8(sp)
        bgeu a5,a4,548 <main+0x28>
// If the branch mispredicted then the following load
// could be used to speculatively load from array1
// using an out of bound index. The TEB below prevents
// execution of that younger load till the branch
// resolves
        teb
unsigned char value;
value = arr1->data[untrusted_offset_from_user];
        ld a5,8(sp)
        lbu a5,8(a5)
unsigned long index2 = ((value&1)*0x100)+0x200;
if (index2 < arr2->length) {
        ld a4,0(zero)
unsigned long index2 = ((value&1)*0x100)+0x200;
        andi a5,a5,1
        addiw a5,a5,2
        slliw a5,a5,0x8
if (index2 < arr2->length) {
    unsigned char value2 = arr2->data[index2];
        lbu a5,8(a5)
```

Example: Memory Disambiguation Barrier

```
// Store a function pointer on the stack
sd a5,8(sp)
   :
   :
// If the load bypasses the store then it may provide
// the old contents of the stack to the jalr. The old
// contents may be controlled by untrusted entities
// mbd ensures that the ld cannot bypass sd and must
// receive its data from the previous store that location
mdb
ld a5,8(sp)
jalr ra, (x5)
```