Speculation at Fault: 
Modeling and Testing Microarchitectural 
Leakage of CPU Exceptions 

Jana Hofmann, Emanuele Vannacci, Cédric Fournet, Boris Köpf, and 
Oleksii Oleksenko 

USENIX Security '23
Teaser

• Automatic discovery of information leakages on exceptions

• 13 exceptions variants on four µarch
  • Two new leaks variants and CVE-2023-20588
  • We confirm and corroborate findings from previous work

• The first formal description of exceptions leakage
Software

Instruction Set Architecture (ISA)
Software

Instruction Set Architecture (ISA)
Software

Instruction Set Architecture (ISA) + Contract

5
Black-box Fuzzing with Revizor

Test Case generator
(Program + Input)

Hardware trace

Compare

Contract trace
Black-box Fuzzing with Revizor

Test Case generator (Program + Input)

Hardware trace

Compare

Violation!

Contract trace
Black-box Fuzzing with Revizor

Test Case generator
(Program + Input)

Exception!

Hardware trace

Compare

Contract trace
Challenges

- Design a consistent and deterministic environment
- Analyze and infer CPU behavior
- Model the leakage in a contract
Mapping the Landscape

• **Goal**: Find the right contract for each exception and CPU

• We run 124 fuzzing campaigns, for 24h each (or until violation)

• Intel KabyLake, Intel CoffeeLake, AMD Zen+, AMD Zen3

• **Synchronous Exceptions**
  • Memory errors - (e.g., Page fault #PF)
  • Computation errors (e.g., Divide error #DE)
  • Opcode-based errors (e.g., Undefined instruction #UD)
A Family of Contracts

- **SEQ**: No transient execution
- **DH (Delayed Handling)**: Out-of-Order CPU
- **VS**: The faulty instruction produces a transient value
  - E.g., value forwarding and Null injection
- **VS-Uknown**: express the dependency from the arch state
  - from the source operands of the faulty instruction
  - or from the whole architectural state
Observations

• Some exceptions satisfy the SEQ and DH contracts
  • E.g., Op-code based errors act as serializing events

• We found already reported value speculation
  • E.g., from L1 cache and µarch buffers

• Recovering the returned transient value is not always feasible
New Leaks Found

• Read-Modify-Write Speculation (Kaby Lake and Coffee Lake)
  • New way to trigger MDS

• Non-Canonical Store Forwarding (Coffee Lake)
  • New variant of store forwarding on non-canonical accesses

• Divider State Sampling (Zen+)
Divider State Sampling

.test_case_enter:
1. DIV rbx  # result in rdx:rax
   ...
2. MOV rcx, 0  # rcx <= 0
3. DIV rcx     # divide-by-zero
4. MOV rdi,[rdx] # leak remainder
.test_case_exit:
Divider State Sampling

.test_case_enter:
1. **DIV** rbx    # result in rdx:rax
...
2. **MOV** rcx, 0    # rcx <= 0 
3. **DIV** rcx    # divide-by-zero
4. **MOV** rdi,[rdx]    # leak remainder
.test_case_exit:
Divider State Sampling

.test_case_enter:
1. **DIV** rbx       # result in rdx:rax
   ...
2. **MOV** rcx, 0   # rcx <= 0
3. **DIV** rcx      # divide-by-zero
4. **MOV** rdi,[rdx] # leak remainder

.test_case_exit:

Hypothesis:

\[ rdx = 0 \]
Divider State Sampling

Hypothesis: 
\[ rdx = 0 \]

.test_case_enter:
1. DIV rbx       # result in rdx:rax
   ...
2. MOV rcx, 0   # rcx <= 0
3. DIV rcx      # divide-by-zero
4. MOV rdi,[rdx] # leak remainder

.test_case_exit:

Violations detected
Divider State Sampling

.test_case_enter:
1. DIV rbx        # result in rdx:rax
   ...
2. MOV rcx, 0    # rcx <= 0
3. DIV rcx       # divide-by-zero
4. MOV rdi,[rdx] # leak remainder

.test_case_exit:

Hypothesis:
rdx depends on the faulty instruction
Divider State Sampling

```
.test_case_enter:
1. DIV rbx  # result in rdx:rax
2. MOV rcx, 0  # rcx <= 0
3. DIV rcx  # divide-by-zero
4. MOV rdi,[rdx]  # leak remainder

Hypothesis:
rdx depends on the faulty instruction
```

INFO: [fuzzer] Starting at 14:03:43
80172 ( 1%) Stats: Cls:98.7/101.0,In:200.0,Cv:0,SpF:22587,ObF:0,Prm:51,Flk:1,Vlo:0> Prime 37

---

Contract trace:
872847031572684841 (hash)
Hardware traces:
Inputs [0]:
```
`...
```
Inputs [100]:
```
`...
```
Divider State Sampling

.test_case_enter:
1. DIV rbx      # result in rdx:rax
...
2. MOV rcx, 0   # rcx <= 0
3. DIV rcx      # divide-by-zero
4. MOV rdi,[rdx] # leak remainder

.test_case_exit:

Hypothesis:
rdx = any value
Divider State Sampling

.test_case_enter:
1. `DIV rbx` # result in rdx:rax
   ...
2. `MOV rcx, 0` # rcx <= 0
3. `DIV rcx` # divide-by-zero
4. `MOV rdi,[rdx]` # leak remainder

.test_case_exit:

Hypothesis:
rdx = any value
Divider State Sampling

.test_case_enter:
1. `DIV rbx` # result in rdx:rax

... 
2. `MOV rcx, 0` # rcx <= 0
3. `DIV rcx` # divide-by-zero
4. `MOV rdi,[rdx]` # leak remainder

.test_case_exit:
Hemd Leak Data After A Bug That Could Result in rdx

1. DIV rbx
2. MOV rcx, 0
3. DIV rcx
4. MOV rdi, [rdx]

test_case_exit:

test_case_enter:

---

divider State Sampling

divider State Sampling

Linux Lands Fix For AMD Zen 1 Bug
Summary

• We built several speculation contracts for exceptions
• We built a tool to test them against real CPUs
• We found and analyzed violations to refine our models
• We demonstrated our approach by finding known and new leaks
Questions?

Paper