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

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

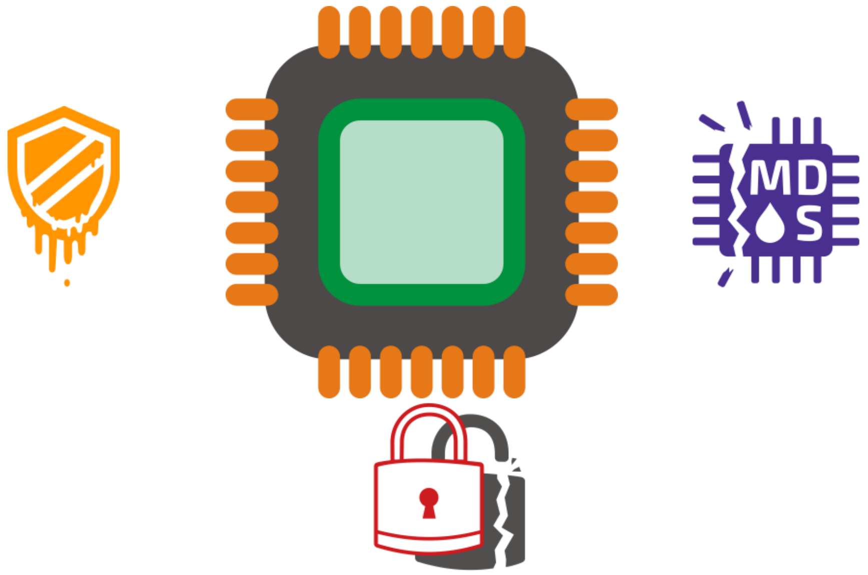
USENIX Security '23

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

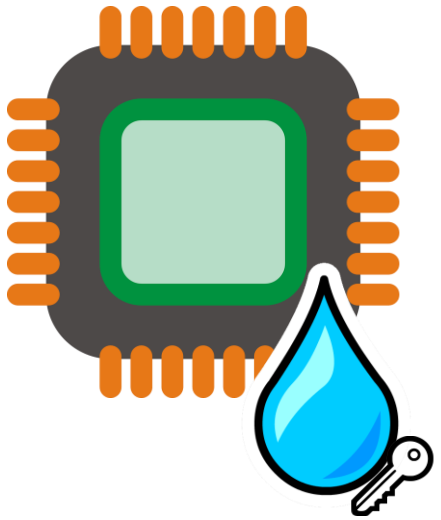
Software

Instruction Set Architecture (ISA)



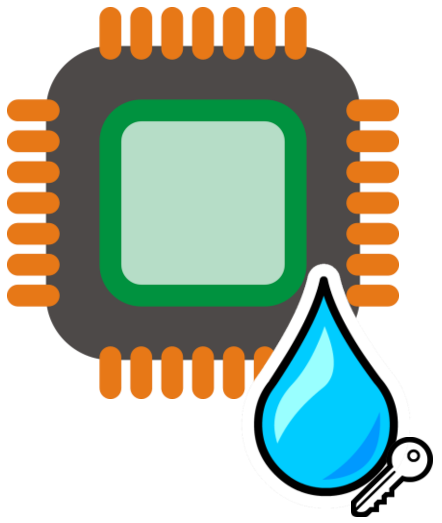
Software

Instruction Set Architecture (ISA)

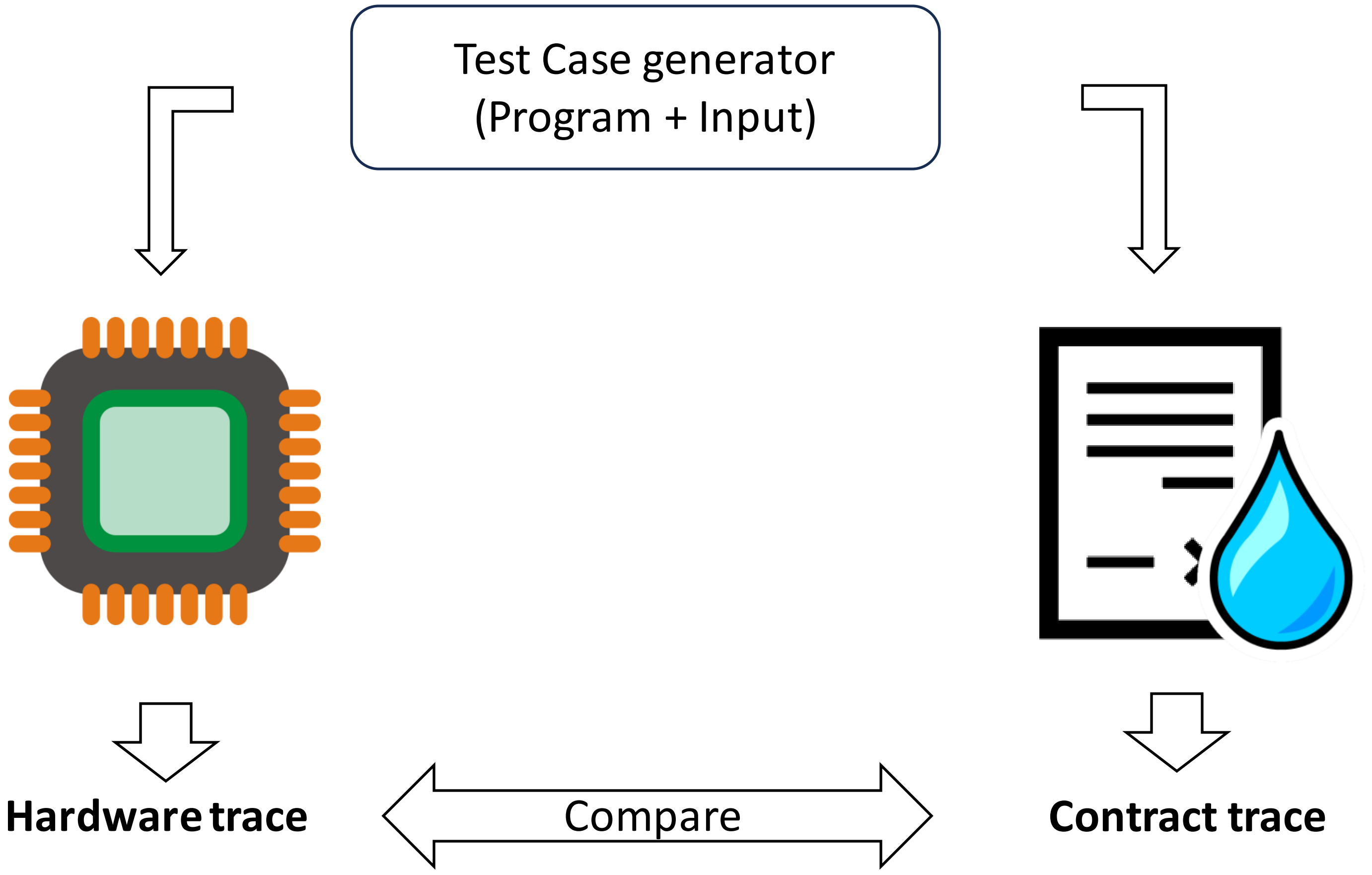


Software

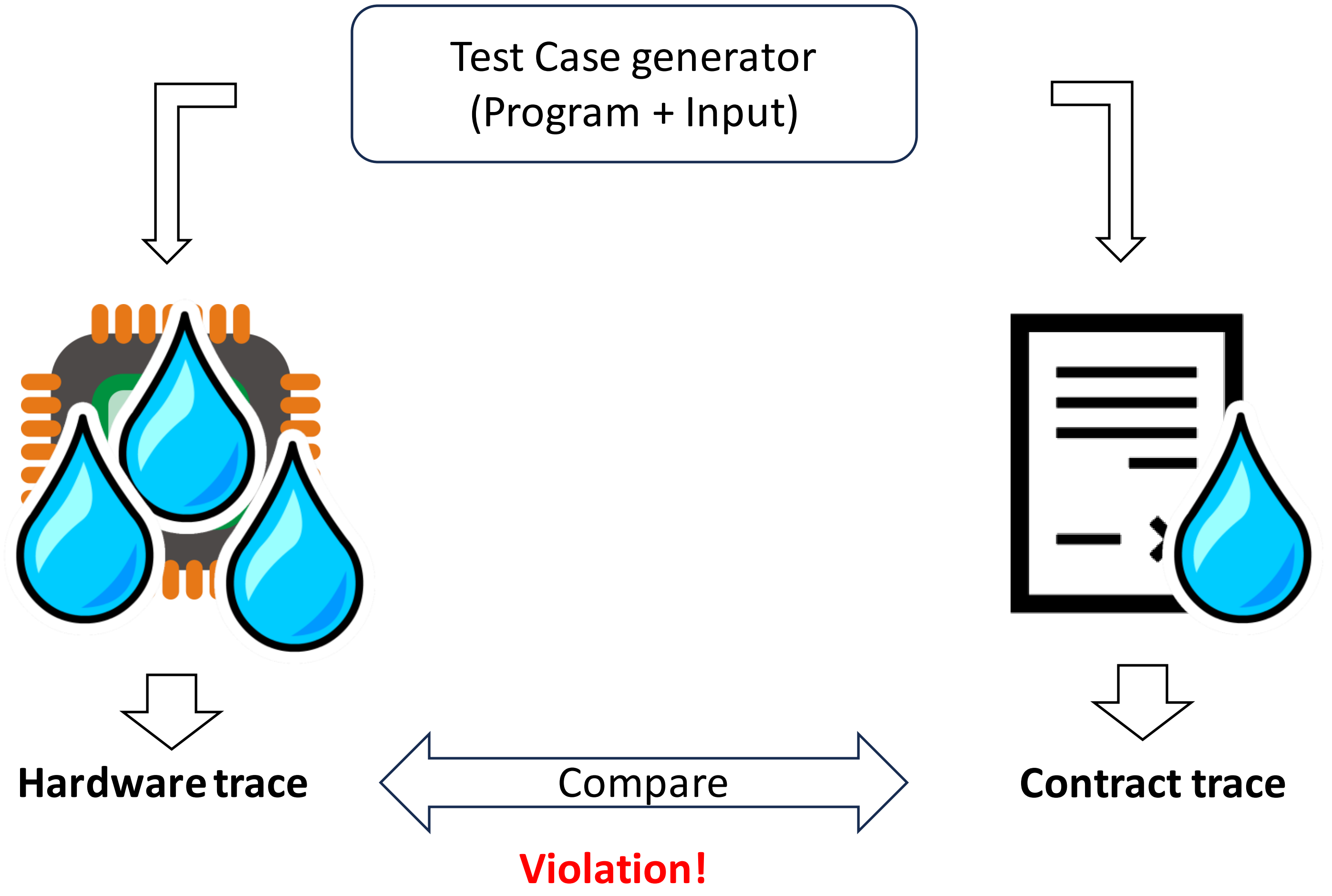
Instruction Set Architecture (ISA) + Contract



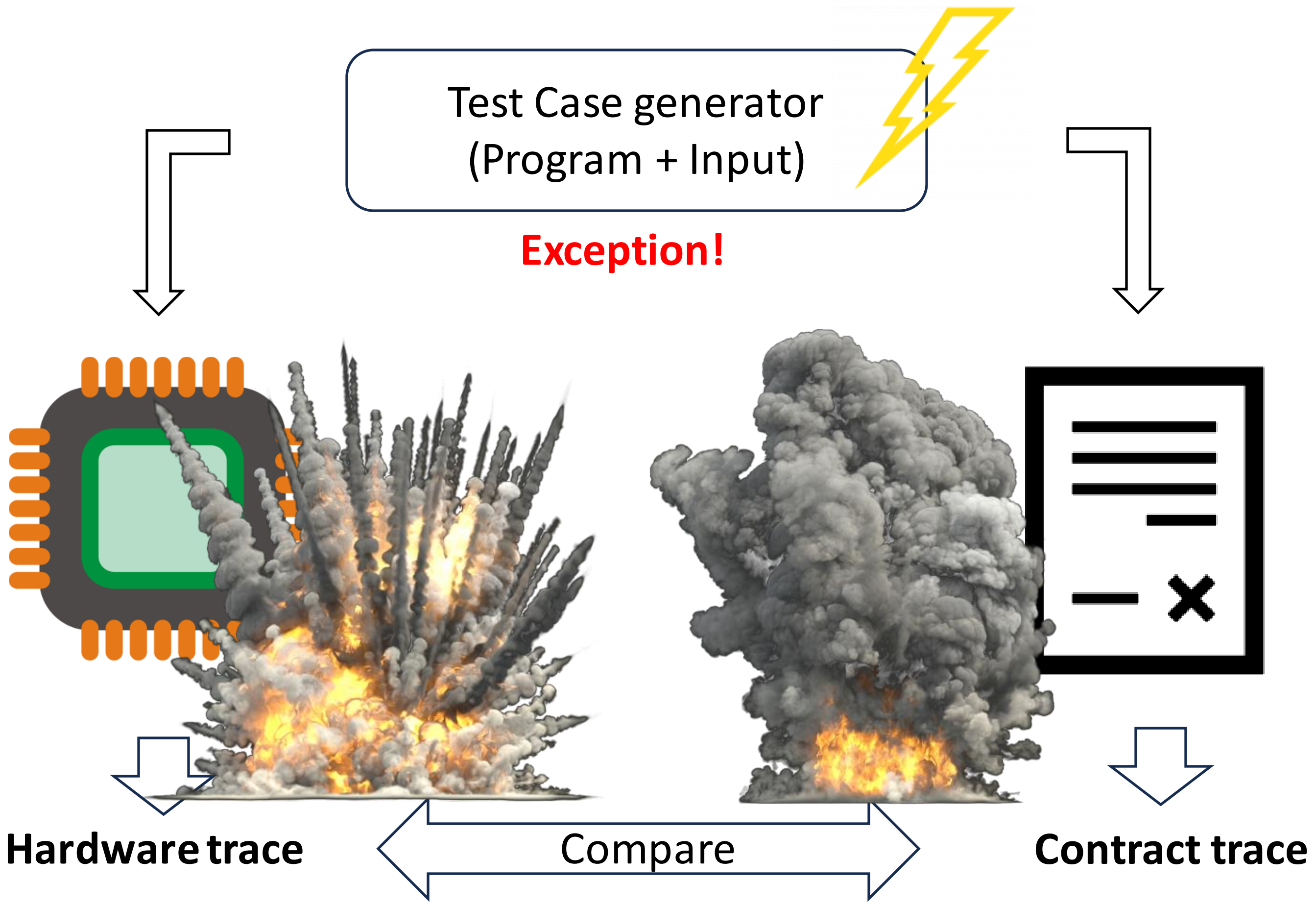
Black-box Fuzzing with Revizor



Black-box Fuzzing with Revizor



Black-box Fuzzing with Revizor



Challenges

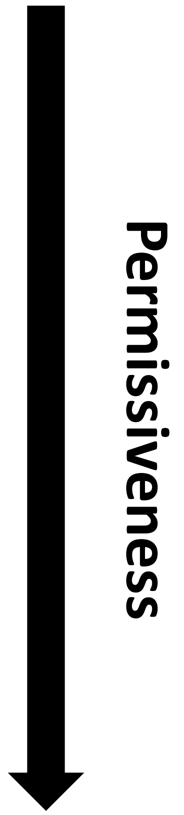
Design	a consistent and deterministic environment
Analyze	and infer CPU behavior
Model	the leakage in a contract

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 CofeeLake, 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

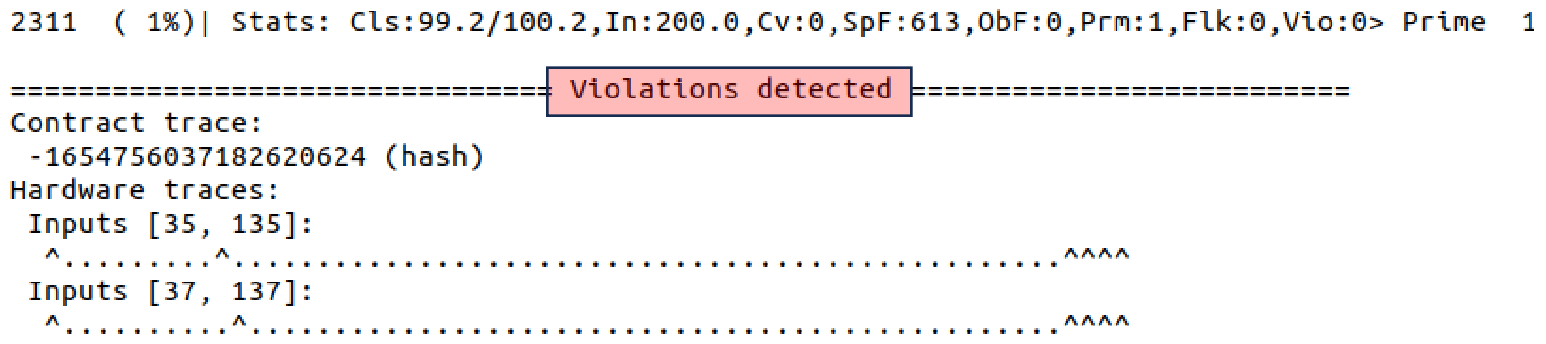
```
.test_case_enter:
1. DIV rbx          # result in rdx:rax
...
2. MOV rcx, 0       # rcx <= 0
3. DIV rcx          # divide-by-zero
4. MOV rdi,[rdx]    # leak remainder
.test_case_exit:
```

Hypothesis:
rdx = 0

Divider State Sampling

```
.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



```
2311  ( 1%)| Stats: Cls:99.2/100.2,In:200.0,Cv:0,SpF:613,ObF:0,Prm:1,Flk:0,Vio:0> Prime  1

=============================== Violations detected =========================
Contract trace:
 -1654756037182620624 (hash)
Hardware traces:
 Inputs [35, 135]:
  ^.........^................................................^^^^
 Inputs [37, 137]:
  ^..........^...............................................^^^^
```

Divider State Sampling

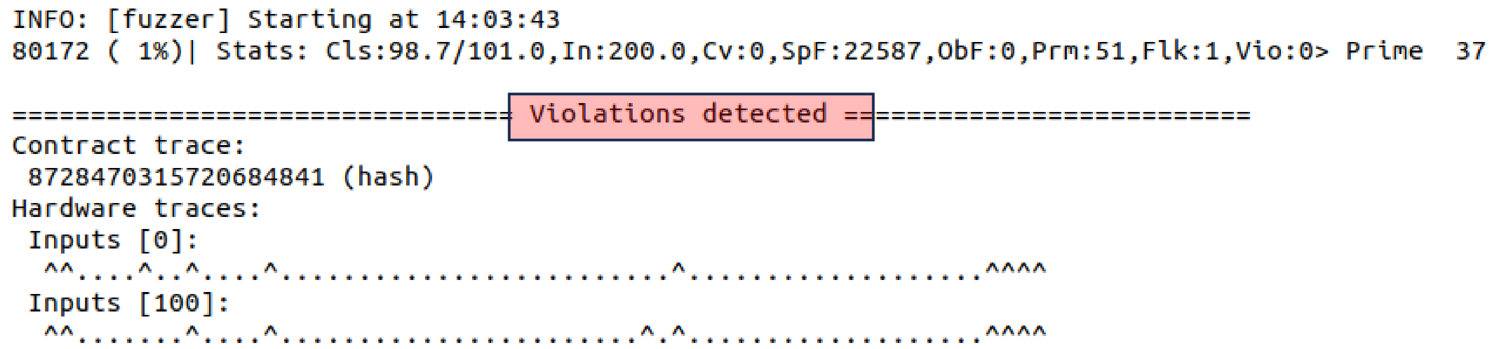
```
.test_case_enter:
1. DIV rbx          # result in rdx:rax
...
2. MOV rcx, 0       # rcx <= 0
3. DIV rcx          # divide-by-zero
4. MOV rdi,[rdx]    # leak remainder
.test_case_exit:
```

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
.test_case_exit:
```

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,Vio:0> Prime  37

================================ Violations detected ==========================
Contract trace:
 8728470315720684841 (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

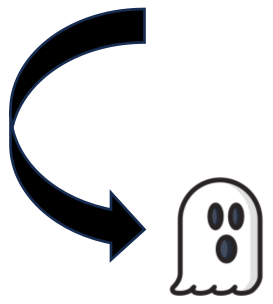
Divider State Sampling

```
.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

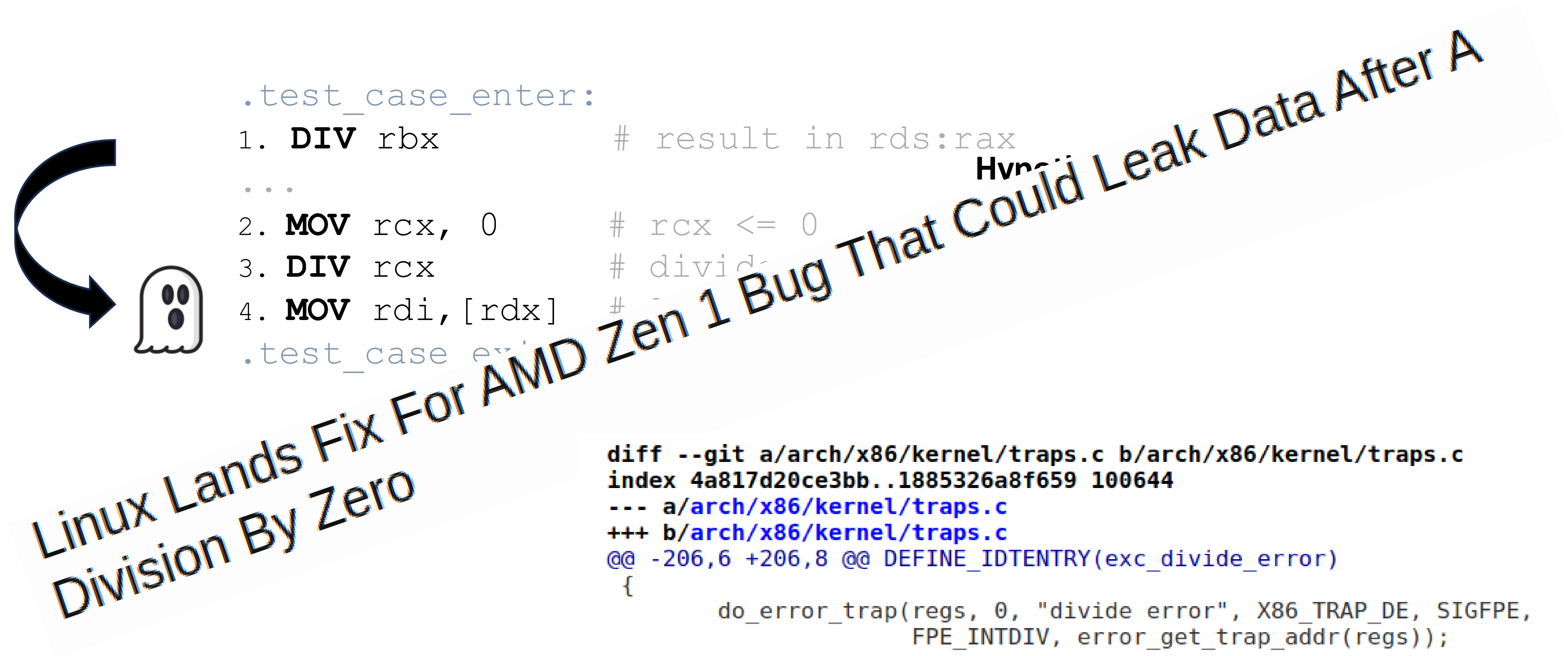


```
.test_case_enter:
1. DIV rbx          # result in rdx:rax
...
2. MOV rcx, 0       # rcx <= 0
3. DIV rcx          # divide-by-zero
4. MOV rdi,[rdx]    # leak remainder
.test_case_exit:
```

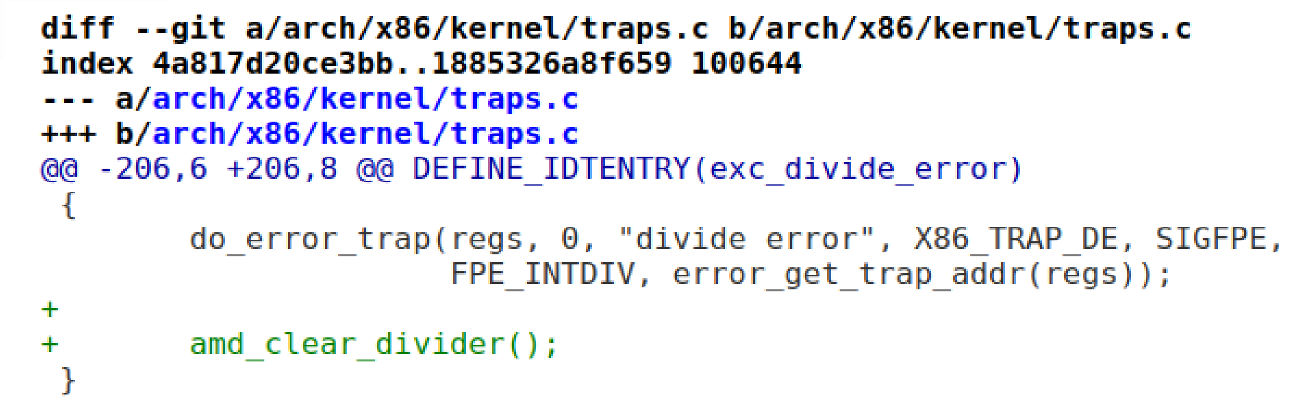
Divider State Sampling

```
.test_case_enter:
1. DIV rbx          # result in rds:rax
...
2. MOV rcx,  0      # rcx <= 0
3. DIV rcx          # divid
4. MOV rdi,[rdx]    #
.test_case_ex
```

Hyp



Linux Lands Fix For AMD Zen 1 Bug That Could Leak Data After A Division By Zero



```
diff --git a/arch/x86/kernel/traps.c b/arch/x86/kernel/traps.c
index 4a817d20ce3bb..1885326a8f659 100644
--- a/arch/x86/kernel/traps.c
+++ b/arch/x86/kernel/traps.c
@@ -206,6 +206,8 @@ DEFINE_IDTENTRY(exc_divide_error)
 {
        do_error_trap(regs, 0, "divide error", X86_TRAP_DE, SIGFPE,
                      FPE_INTDIV, error_get_trap_addr(regs));
+
+       amd_clear_divider();
 }
```

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