中国科学院软件研究所
Institute of Software Chinese Academy of Sciences

智能软件研究中心
Intelligent Software Research Center

Security Strategies in V8: Part 1
Looking into the Spectre Mitigation

qiuji@iscas.ac.cn

2021/04/16

Outline

- Background and reference
- What’s Spectre
- Variants and How they work
- Key factors for the attack
- Hardware solution
- W3C’s mitigation
- V8’s mitigation

Background knowledge

- CPU ISA
    - how load/branch/call work
- Performance enhancement method on Modern CPU
    - out-of-order execution
    - speculative and branch prediction
    - memory hierarchy

V8 Reference from PLCT

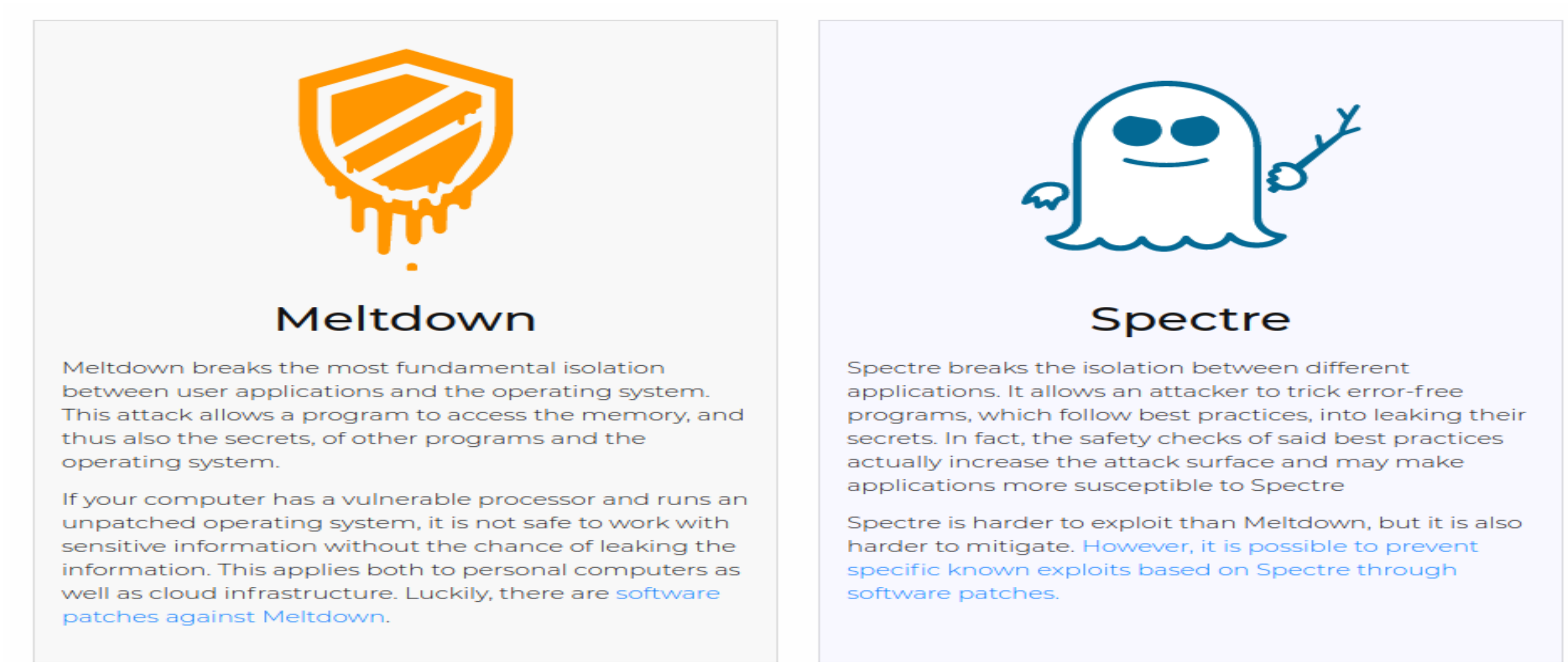
- PLCT contribute slides： https://github.com/isrc-cas/PLCT-Open-Reports
  - V8移植简介
  - V8测试流程介绍以及指令选择单元测试源码分析
  - V8指令选择过程中的优化
  - Dive-into-Torque
- PLCT contribute report videos：
  - 深入V8引擎-第01课：上手开始看 V8 Ignition 解释器的字节码（Bytecodes）
  - 深入V8引擎-第02~04课：从零开始分析V8的构建系统构成part1/part2/part3
  - 深入V8引擎：V8 Call Interface Descriptors
  - V8 Assembler 学习小结
  - V8源码学习：CSA、Builtins、中断等
  - V8单元测试框架
  - V8指令选择中的优化
  - V8移植简介

Reference

- overall introduction: https://v8.dev/blog/spectre
- white paper: https://arxiv.org/pdf/1902.05178.pdf
- white paper for Retpoline: https://support.google.com/faqs/answer/7625886
- Intro of Spectre and Meltdown:
  - https://spectreattack.com/
  - https://spectreattack.com/spectre.pdf
- Intro of HR timer: https://gruss.cc/files/fantastictimers.pdf
- related code:
  - https://github.com/riscv/v8/blob/riscv64/src/compiler/backend/riscv64/instruction-scheduler-riscv64.cc
  - https://github.com/riscv/v8/blob/riscv64/src/compiler/backend/riscv64/code-generator-riscv64.cc
  - https://github.com/riscv/v8/blob/riscv64/src/codegen/x64/macro-assembler-x64.cc

What’s Spectre (and Meltdown)

- They are vulnerabilities in modern computers leak passwords and sensitive data.



Meltdown

Meltdown breaks the most fundamental isolation between user applications and the operating system. This attack allows a program to access the memory, and thus also the secrets, of other programs and the operating system.

If your computer has a vulnerable processor and runs an unpatched operating system, it is not safe to work with sensitive information without the chance of leaking the information. This applies both to personal computers as well as cloud infrastructure. Luckily, there are software patches against Meltdown.

Spectre

Spectre breaks the isolation between different applications. It allows an attacker to trick error-free programs, which follow best practices, into leaking their secrets. In fact, the safety checks of said best practices actually increase the attack surface and may make applications more susceptible to Spectre

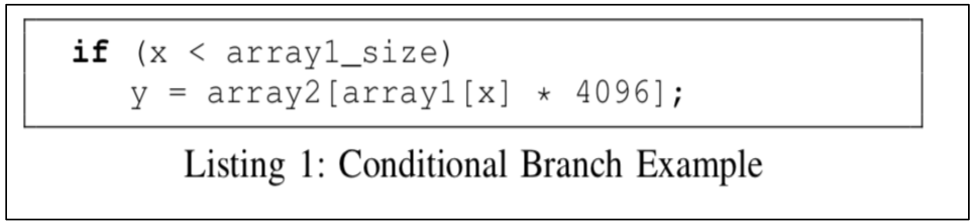
Spectre is harder to exploit than Meltdown, but it is also harder to mitigate. However, it is possible to prevent specific known exploits based on Spectre through software patches.

https://meltdownattack.com

中国科学院软件研究所 Institute of Software Chinese Academy of Sciences | 智能软件研究中心 Intelligent Software Research Center

Spectre： variant NO.1

- exploit **conditional branch** misprediction



```
if (x < array1_size)
    y = array2[array1[x] * 4096];
```

Listing 1: Conditional Branch Example

1. x is the offset of a security info from the array1 start address
2. array2 should be flushed out of cache for prepare

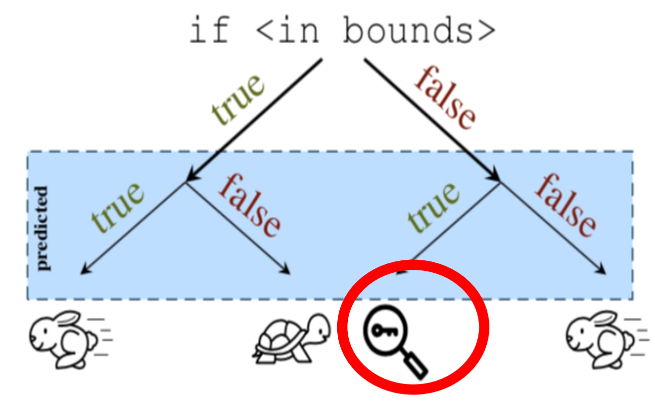


Fig. 1: Before the correct outcome of the bounds check is known, the branch predictor continues with the most likely branch target, leading to an overall execution speed-up if the outcome was correctly predicted. However, if the bounds check is incorrectly predicted as true, an attacker can leak secret information in certain scenarios.

-- “Spectre Attacks: Exploiting Speculative Execution”

Spectre： variant NO.2

- exploit **Indirect Branches** misprediction

victim:
pc with virtual address A: jump good-register (or call good-register)
attacker:
pc with virtual address equal or alias to A: jump gadget-register(or call gadget-register)
gadget code:
transiently read security info and code it into side-channel

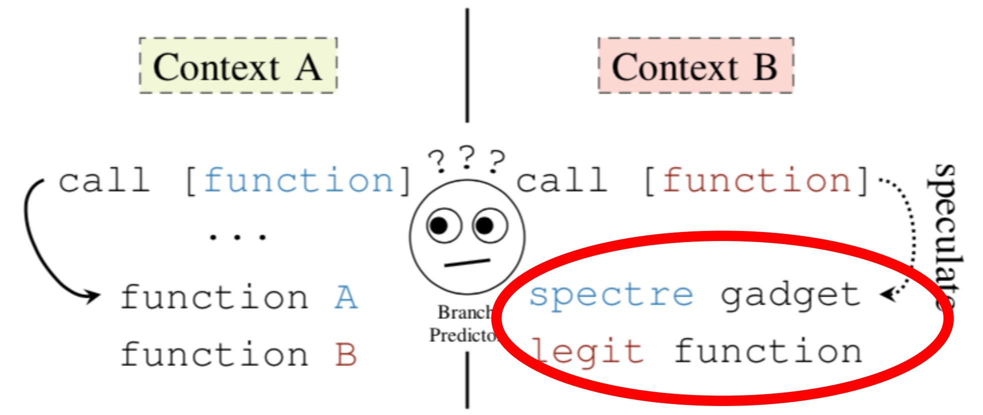


Fig. 2: The branch predictor is (mis-)trained in the attacker-controlled context A. In context B, the branch predictor makes its prediction on the basis of training data from context A, leading to speculative execution at an attacker-chosen address which corresponds to the location of the Spectre gadget in the victim's address space.

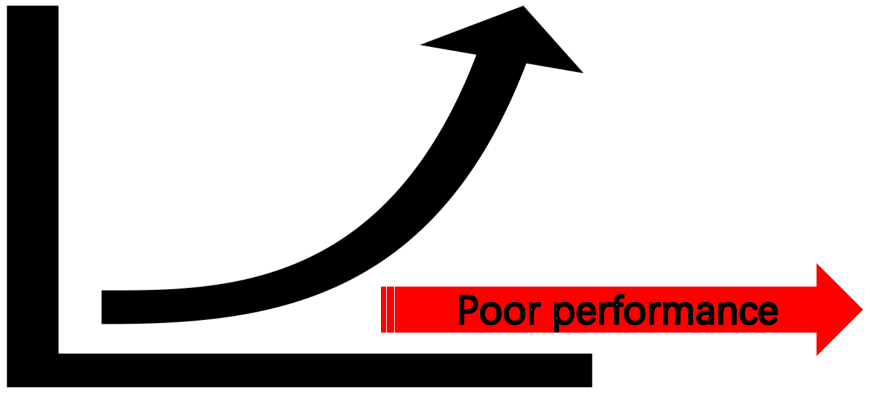
-- "Spectre Attacks: Exploiting Speculative Execution"

Three keys for the attack

- Key1: Conditional Branch or Indirect Branch prediction which will lead to false speculation execution parts
- Key2: Gadgets that transiently load security info then code it into side-channel
- Key3: Timer with enough resolution to measure L1 Cache hit(nanosecond for a Ghz CPU)

Hardware resolution: for Key1

- disable speculation and branch prediction(almost not practical at all)
- using barrier instructions to avoid speculatively executing load
    - degrade performance by 2x~3x
    - not acceptable



W3C's mitigation: for Key2

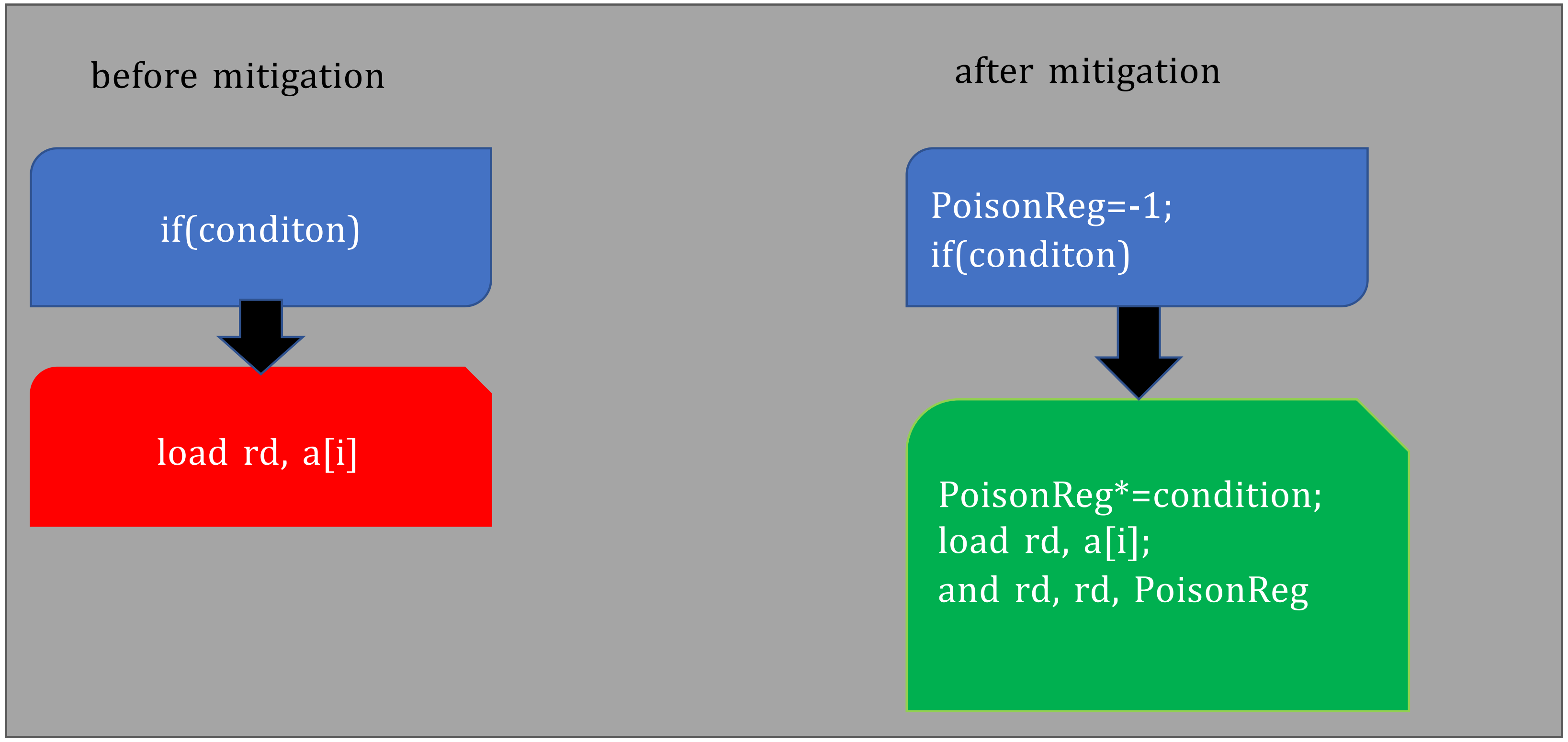
- decrease the resolution of timer API:
    - performance.now() from 5us to 100ms
- disable the SharedArrayBuffer API
    - to prevent construct of high-resolution timer (by measure the clock edge)

V8’s mitigation:

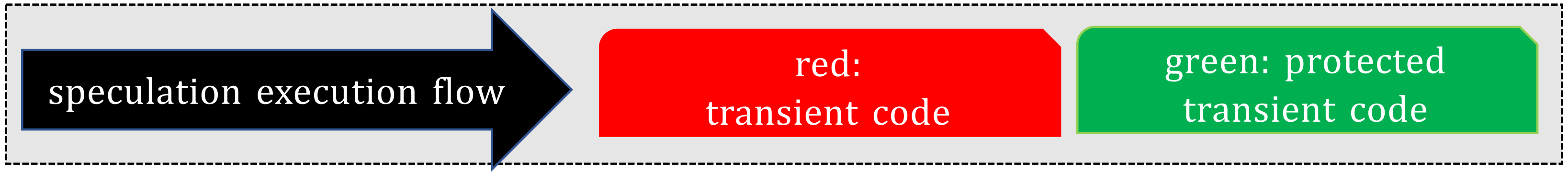
- in JITed code:
  - PoisonReg for Key2 (both variant 1&2)
  - Retpoline for Key1 (variant 2)
- more strict array access bound check
- for 32bit:
  - pad all the array to next power of 2
  - disable the upper bits in user access address (higher half part is privileged address space, e.g., 0x8000000-0xffffffff is kernel space for MIPS32)
- for 64bit:
  - using ALSR

What’s the TurboFan developers must learn.

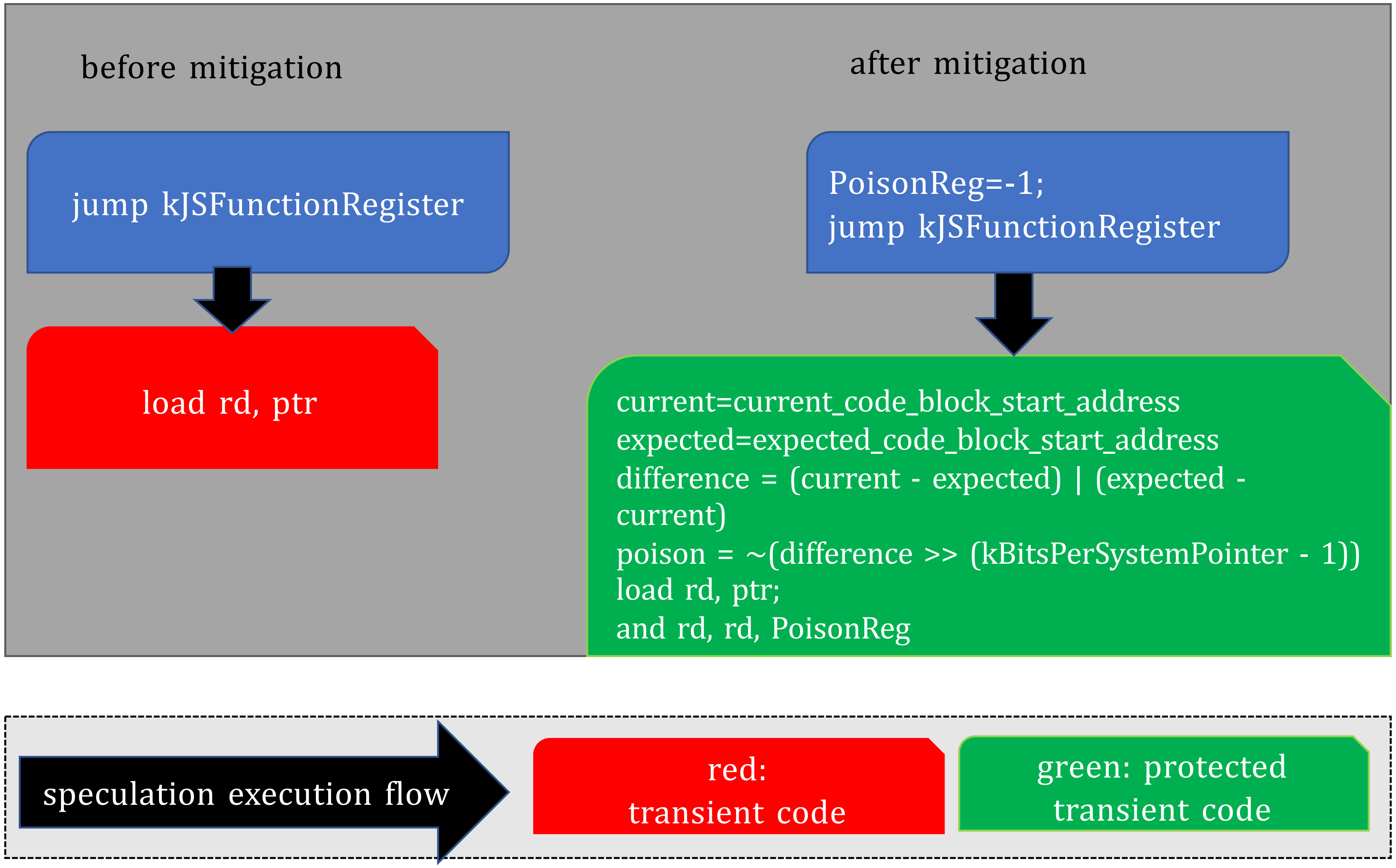
PoisonReg Masking-for Conditional Branch



If speculation is false, PoisonReg will be 0. Then the load result will be masked out.



PoisonReg Masking-for Indirect Branch(Jump register/Call register)



```
jump kJSFunctionRegister
```

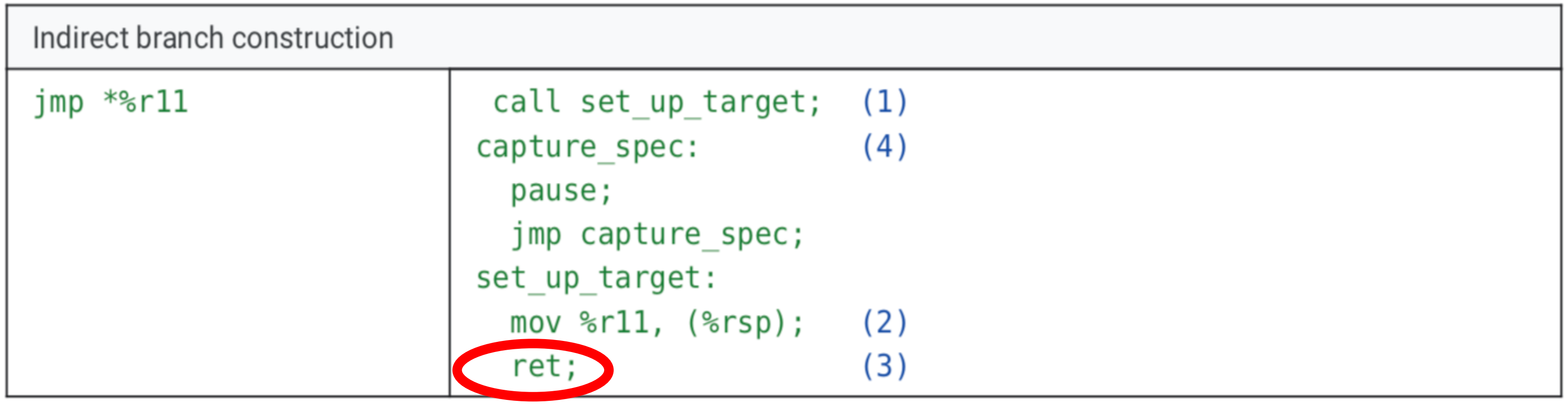
```
load rd, ptr
```

```
PoisonReg=-1;
jump kJSFunctionRegister
```

```
current=current_code_block_start_address
expected=expected_code_block_start_address
difference = (current - expected) | (expected -
current)
poison = ~(difference >> (kBitsPerSystemPointer - 1))
load rd, ptr;
and rd, rd, PoisonReg
```

If speculation is false, PoisonReg will be 0. Then the load result will be masked out.

Retpoline-for Jump register



Indirect branch construction

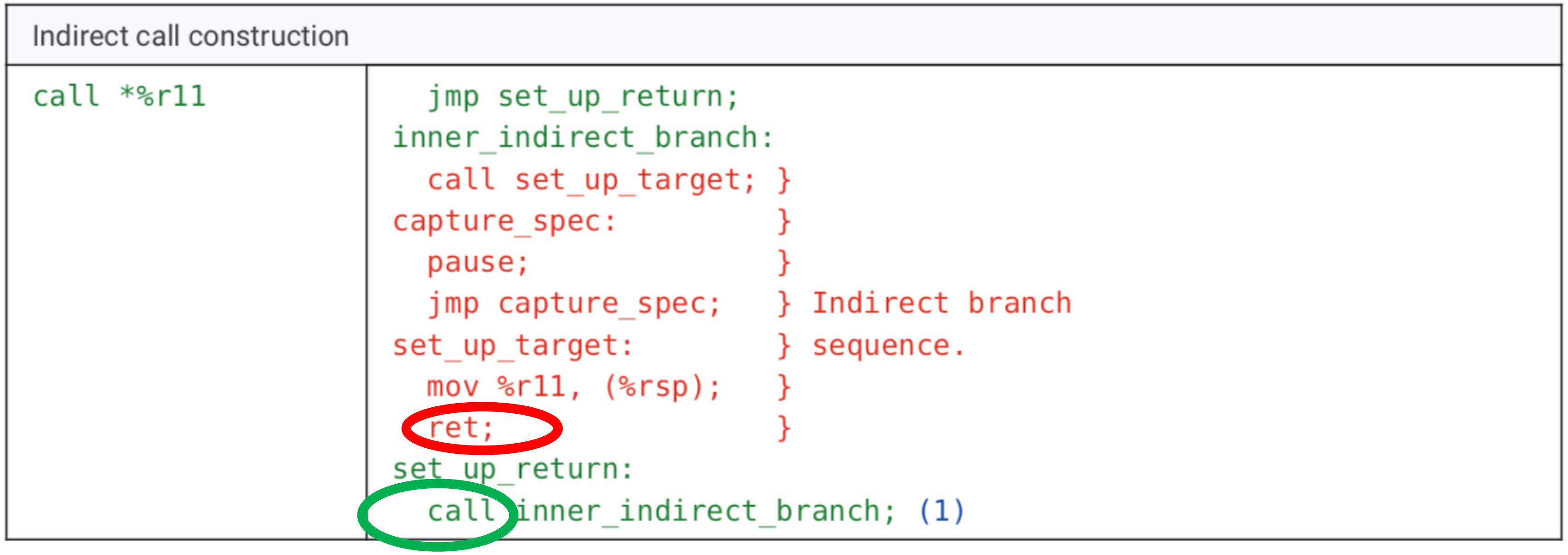
```
jmp *%r11
```

```
 call set_up_target;  (1)
capture_spec:         (4)
  pause;
  jmp capture_spec;
set_up_target:
  mov %r11, (%rsp);   (2)
  ret;                (3)
```

- has no more indirect branch
- the ret is predicted to go to (4) by RSB/RAS, there waiting for truth（ready of %r11）

-- “Retpoline: a soware construct for preventing branch- target-injection"

Retpoline-for Call register



Indirect call construction

call *%r11	see code below

```
call *%r11

  jmp set_up_return;
inner_indirect_branch:
  call set_up_target; }
capture_spec:         }
  pause;              }
  jmp capture_spec;   } Indirect branch
set_up_target:        } sequence.
  mov %r11, (%rsp);   }
  ret;                }
set_up_return:
  call inner_indirect_branch; (1)
```

- has no more indirect branch
- the green call places the right RSB/RAS info for the original call’s return as well as transfers control
- the ret is predicted to go to ”capture_spec” by RSB/RAS, there waiting for truth（ready of %r11）

-- “Retpoline: a soware construct for preventing branch- target-injection"

Summary

- Spectre is severe and has 2 variants
- There are 3 key factors for Spectre attack and JavaScript is vulnerable
- Mitigations are applied for each key factor in V8
- TurboFan takes PoisonReg Masking and Retpoline

中国科学院软件研究所
Institute of Software Chinese Academy of Sciences
智能软件研究中心
Intelligent Software Research Center

Q&A

Thanks

2020/04/16