Week 3 Report

Ben Chen

Dept of Computer Science and Engineering, SUSTech

September 26, 2024

TOC

Title	Conference	Institute	Authors	Idea
Branch History Injection: On the Effectiveness of Hardware Mitigations Against Cross-Privilege Spectre-v2 Attacks	USENIX '22	VUSec	Enrico Barberis Pietro Frigo Marius Muench Herbert Bos Cristiano Giuffrida	Context-based branch prediction is not isolated, which can be polluted by attacker.
TIKTAG: Breaking ARM's Memory Tagging Extension with Speculative Execution	Black Hat '24	UOS	Juhee Kim et al	Use speculative check of tag to leak the check result without causing fault
PACMAN: Attacking ARM Pointer Authentication with Speculative Execution	DEFCON 30 ICCA '22	MIT	Joseph Ravichandran Weon Taek Na Jay Lang Mengjia Yan	Speculative check to leak correctness of forging a PAC without causing fault

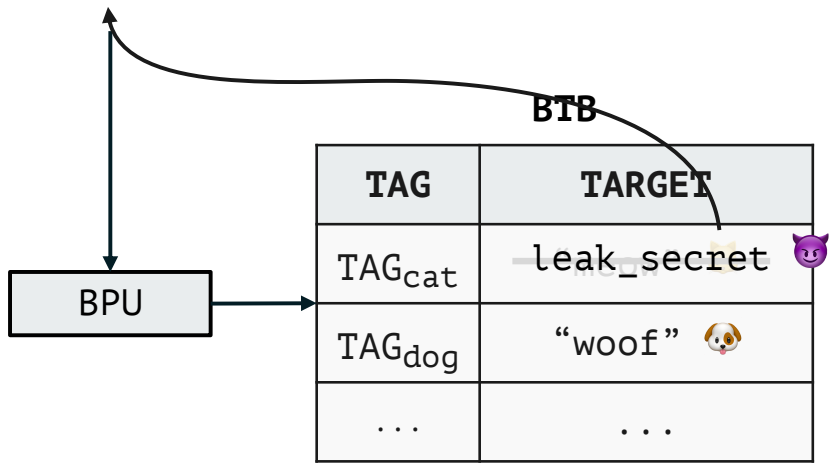
Branch History Injection[1]

Spectre-v2

```
// Cat
Cat kitten = new Cat();
speak(kitten);

//Dog
Dog puppy = new Dog();
speak(puppy);
```

```
void speak(Animal a) {
    a.talk();
}
```

leak_secret 😈



Branch History Injection[1]

Software mitigation: Retpoline. Change the victim jump instruction

```
jmp *%r11
```

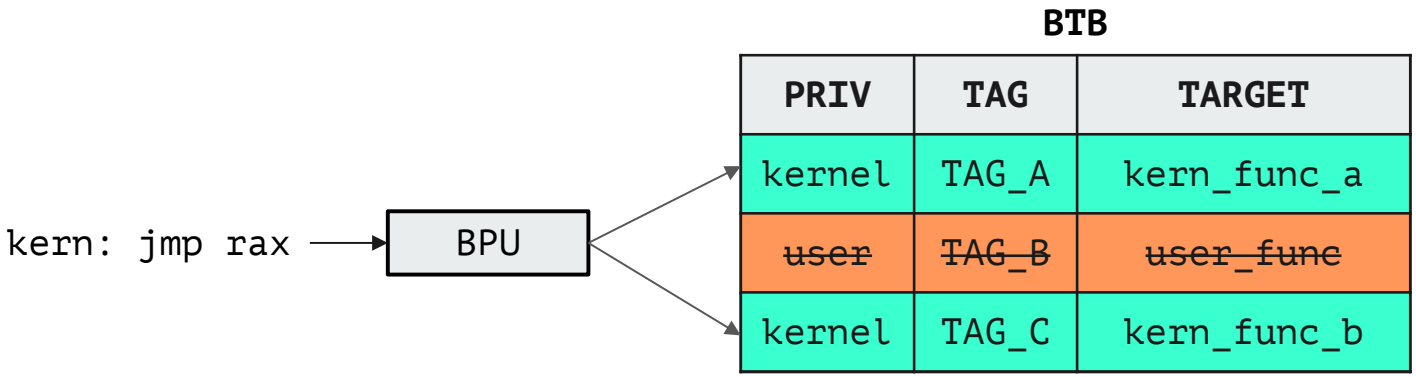
to

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

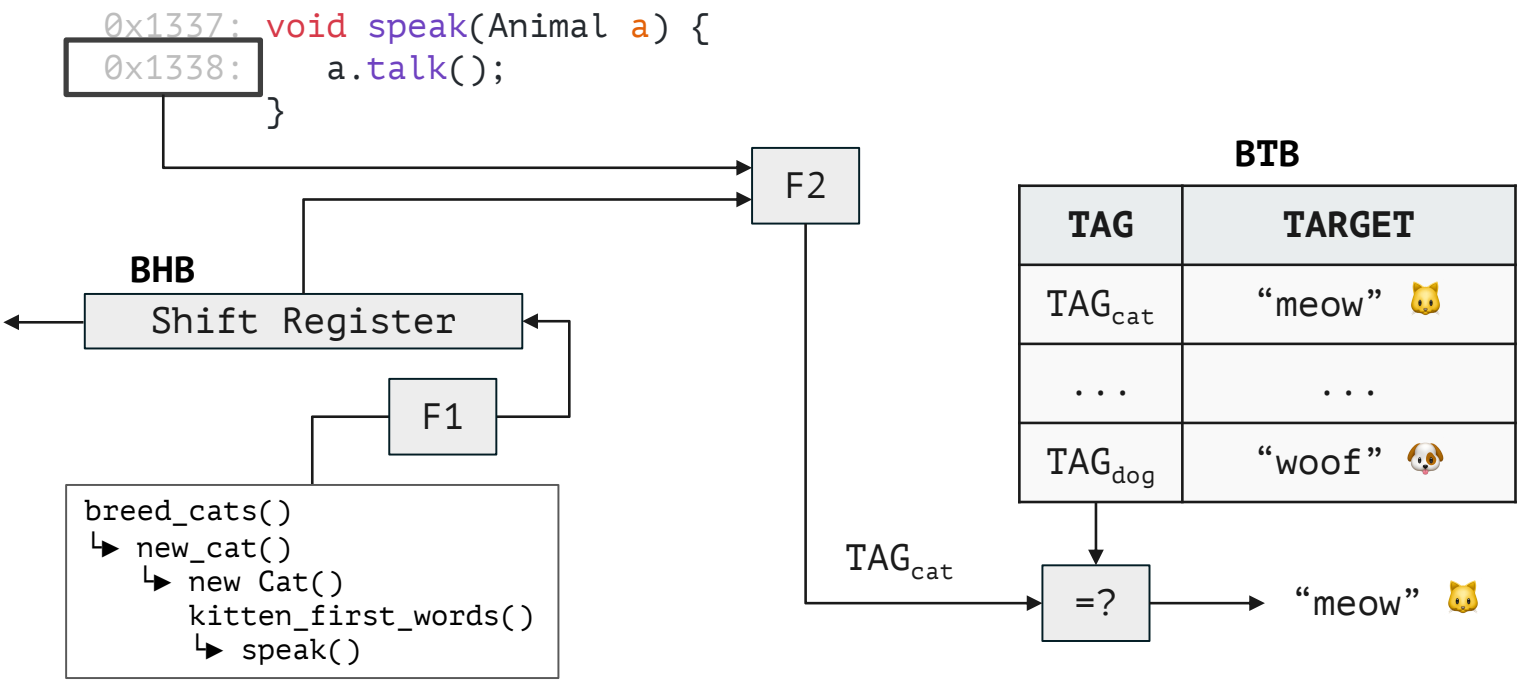
Replace attacker's target with innocuous code.

Intel eIBRS & Arm CSV2

Idea: tag BTB entries by security domain

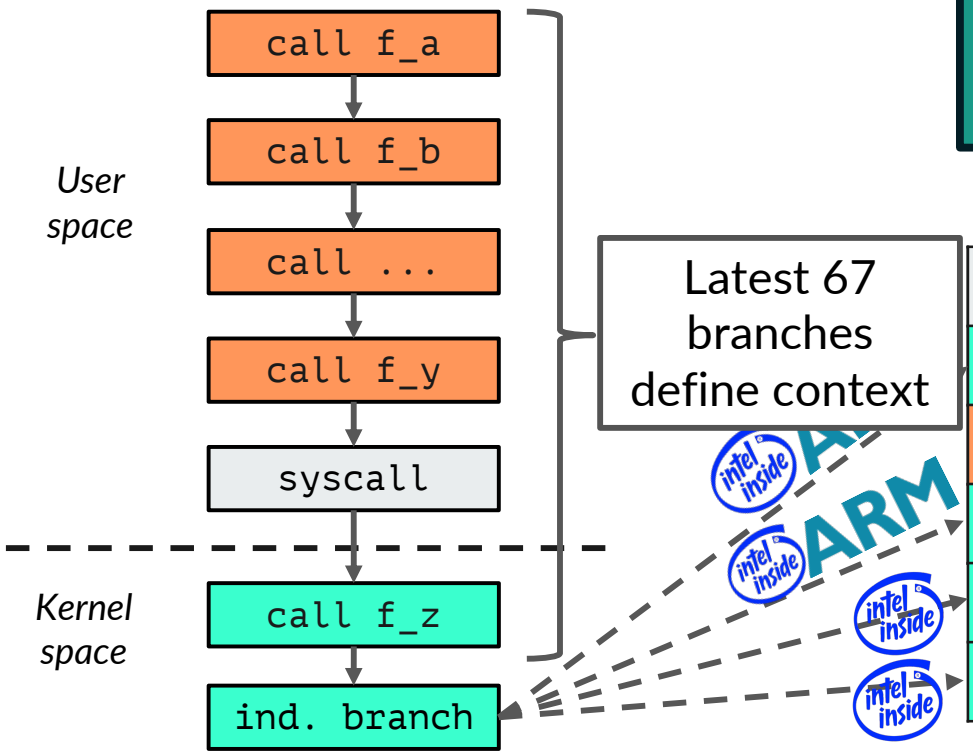


Context-based prediction



Branch History Injection[1]

BHI Capabilities



Latest 67 branches define context

Attacker can mistrain to any indirect branch target

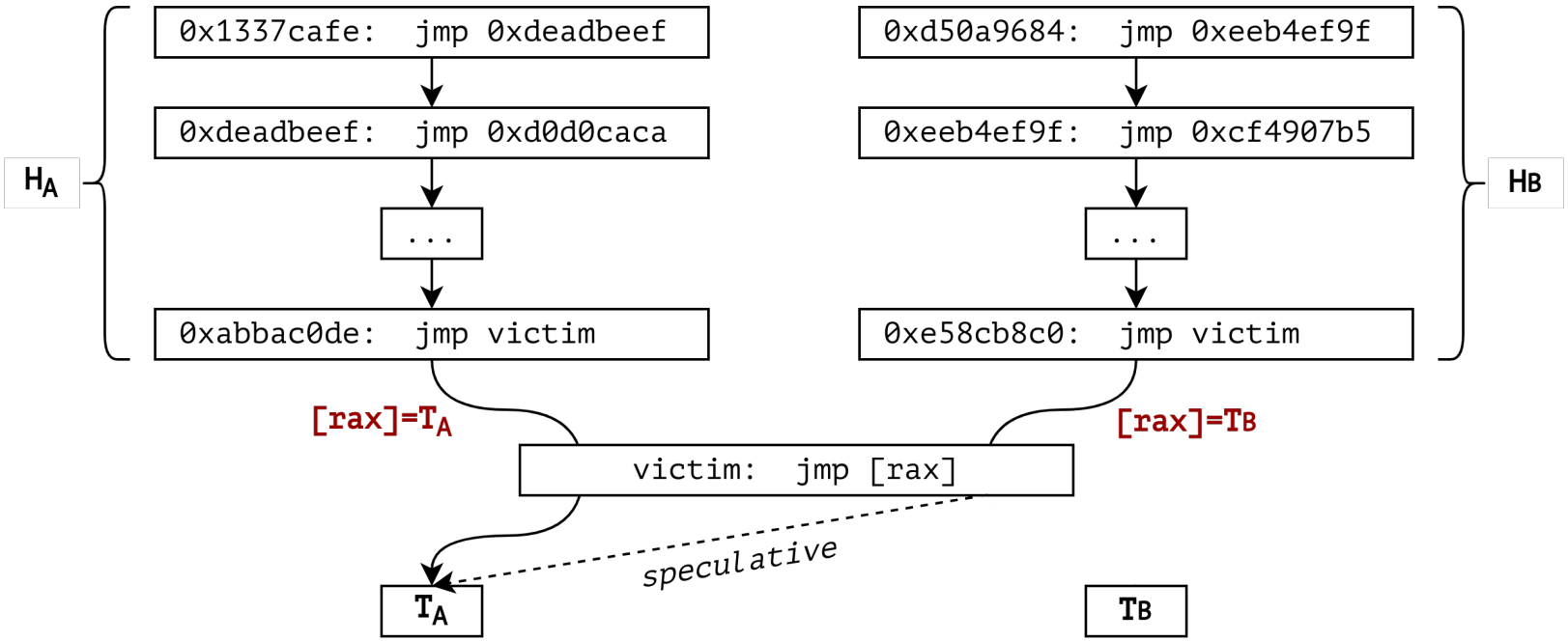
BTB

PRIV	TAG	TARGET
kernel	TAG_A	valid_target_a
~~user~~	~~TAG_B~~	~~user_func~~
kernel	TAG_C	valid_target_b
kernel	TAG_D	other_target_1
kernel	TAG_E	other_target_2

35

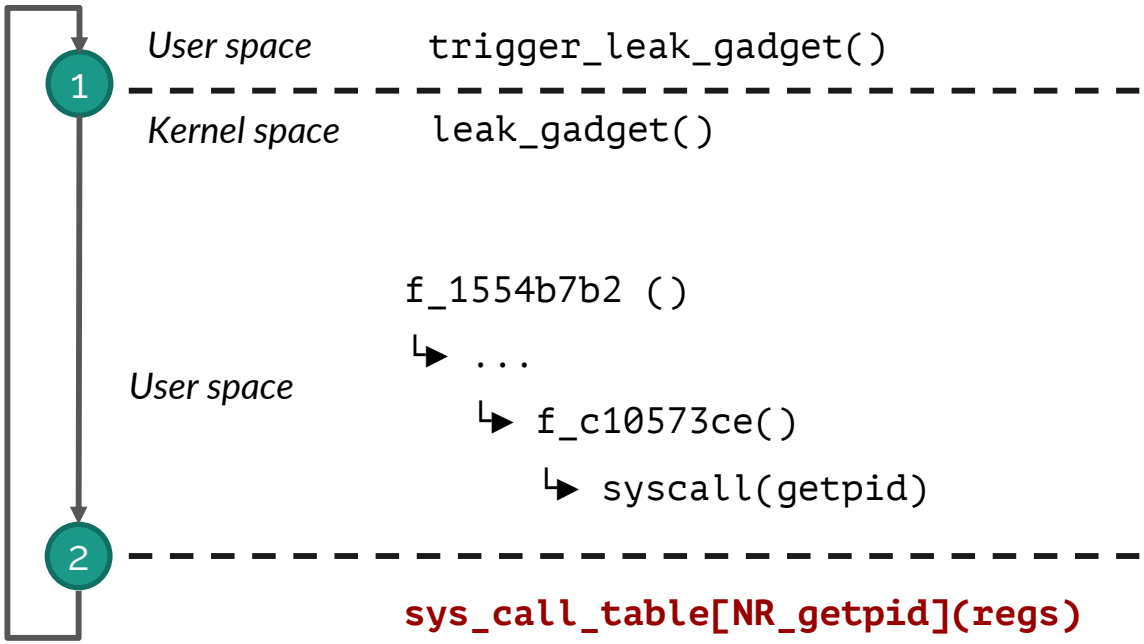
Branch History Injection[1]

BPU Reverse Engineering – Brute Force



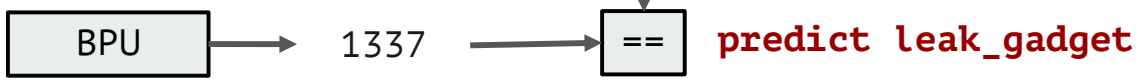
❌ Always misprediction! The BPU is **unable** to distinguish H_A from H_B

Exploitation – The Plan



BTB

TAG	PRIV	TARGET
1337	kernel	leak_gadget
2B04	kernel	sys_getpid
5F33	kernel	sys_getpid



37

Exploitation – Leak Gadget

- We need to find a leak gadget in the kernel code
- Why don't we JIT it with unprivileged eBPF ?

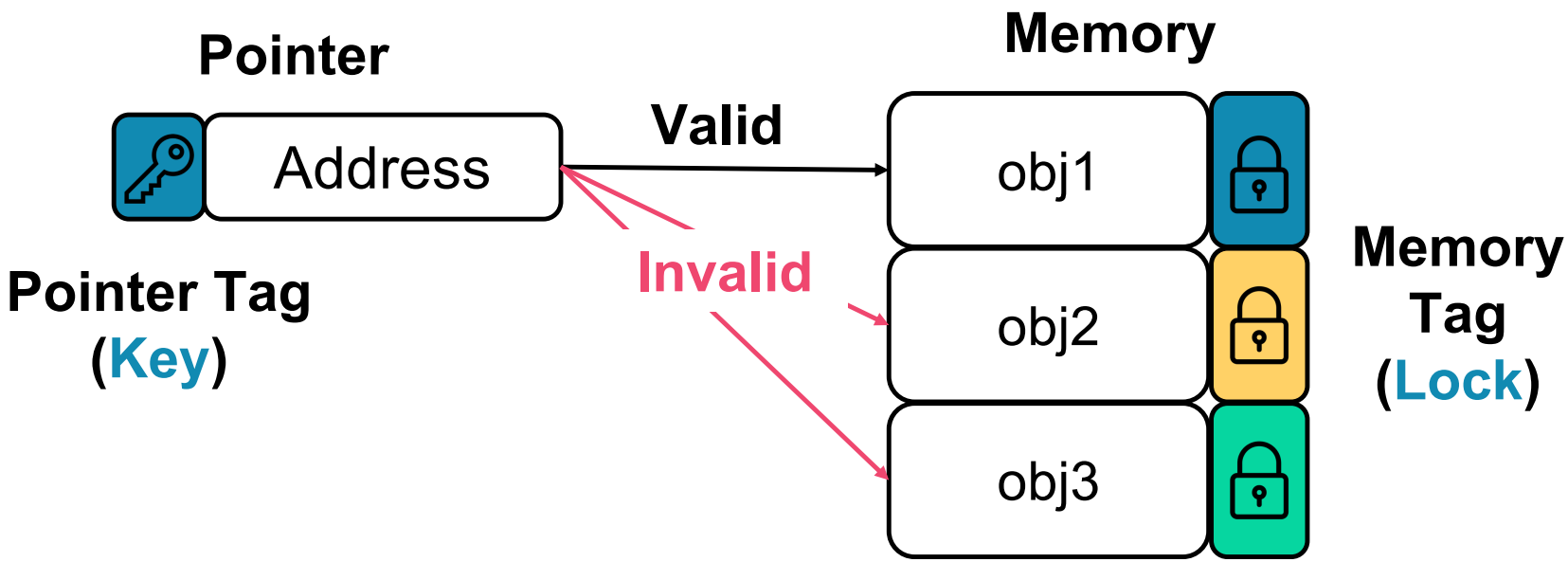
(Yep, there is a JIT engine in the Linux kernel)

```
struct bpf_insn insns_gadget_leak[] = {
    BPF_LDX_MEM(BPF_DW, BPF_REG_0, BPF_REG_1, 168),
    BPF_JMP_IMM(BPF_JEQ, BPF_REG_0, 0, 9),

    BPF_LDX_MEM(BPF_W, BPF_REG_0, BPF_REG_0, 0),
    BPF_LDX_MEM(BPF_W,  BPF_REG_4, BPF_REG_1, 0),
    BPF_ALU64_REG(BPF_RSH, BPF_REG_0, BPF_REG_4),
    BPF_ALU64_IMM(BPF_AND, BPF_REG_0, FR_MASK),
    BPF_ALU64_IMM(BPF_LSH, BPF_REG_0, FR_STRIDE_LOG),

    BPF_LD_IMM64_RAW_FULL(BPF_REG_2, 2, 0, 0, map_array_fd_fr_buf, 0),
    BPF_ALU64_REG(BPF_ADD, BPF_REG_2, BPF_REG_0),
    BPF_LDX_MEM(BPF_DW, BPF_REG_2, BPF_REG_2, 0),

    BPF_MOV64_IMM(BPF_REG_0, 0),
    BPF_EXIT_INSN(),
};
```

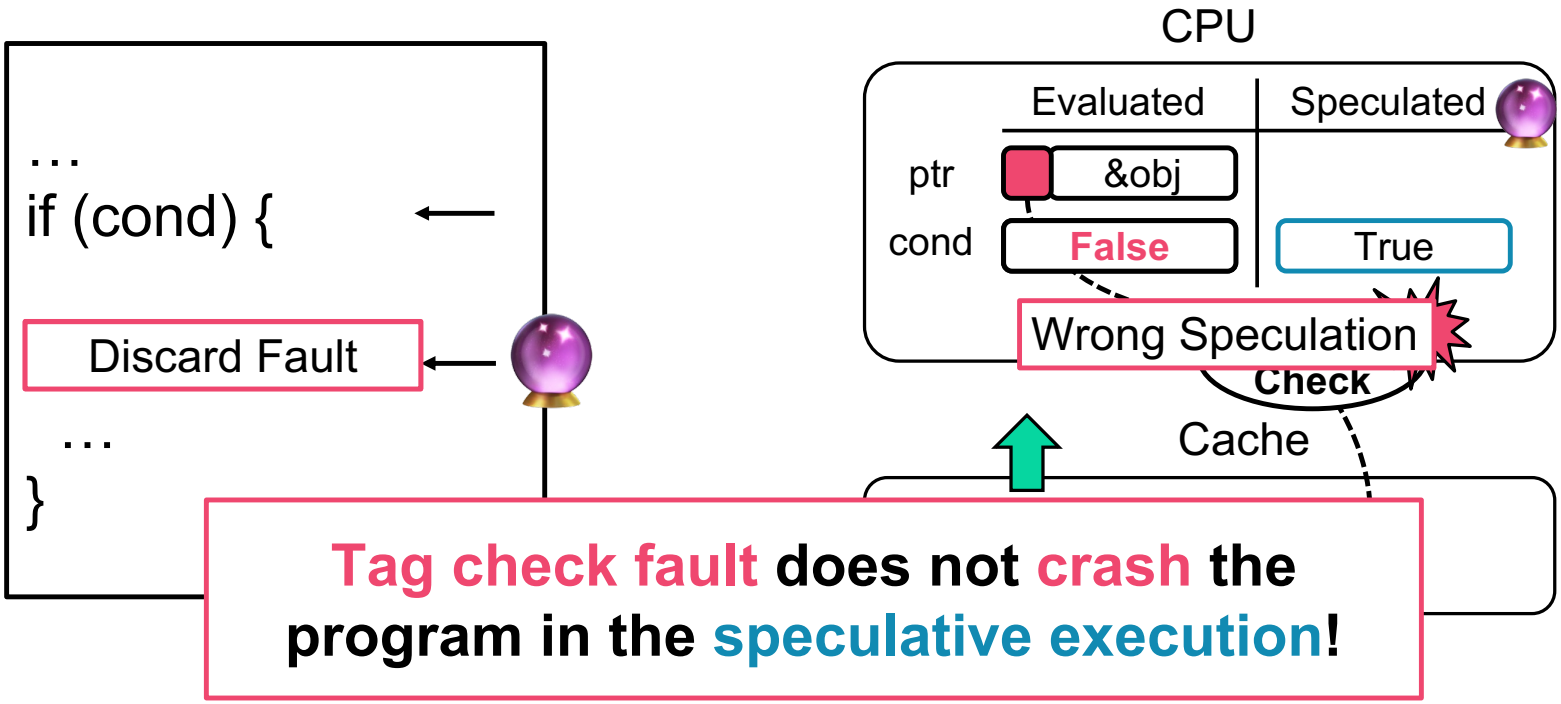
JIT

```
    push   rbp
    mov    rbp,rsp
    ;load er_buf base address
    movabs rsi,0xffffc900028ff110
    ;rdi+0x18 = &pt_regs.r12 transiently
    ;         = &bpf_sock architecturally
    mov    rax,QWORD PTR [rdi+0x18]
    test   rax,rax
    je     fail
    ;Dereference of user r12 value transiently
    mov    eax,DWORD PTR [rax+0x14]
    ;extract the byte to leak
    and    rax,0xff
    shl    rax,0xc
    add    rsi,rax
    ;maccess(er_buf[byte_to_leak*0x1000])
    mov    rsi,QWORD PTR [rsi+0x0]
fail:
    xor    eax,eax
    leave
    ret
```

ARM Memory Tagging Extensions



Tag check fault on Speculative Execution?



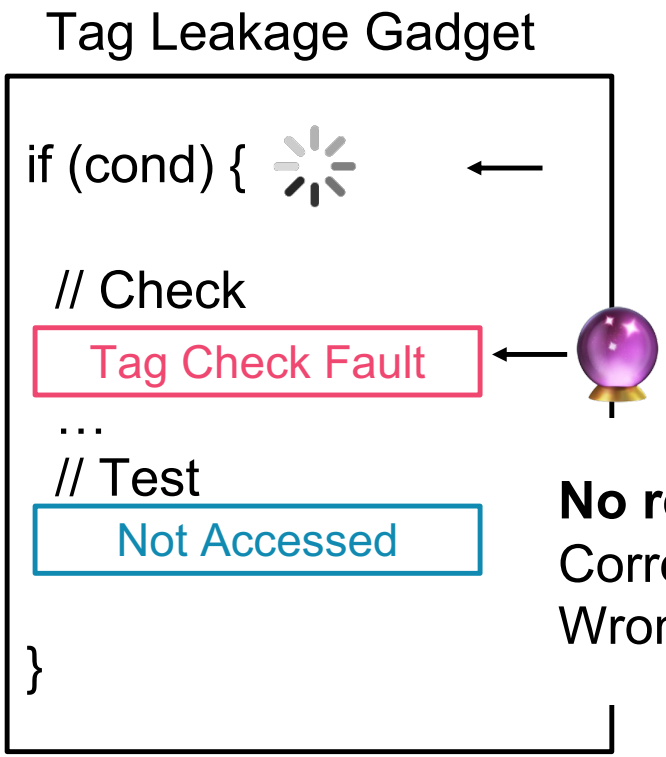
36

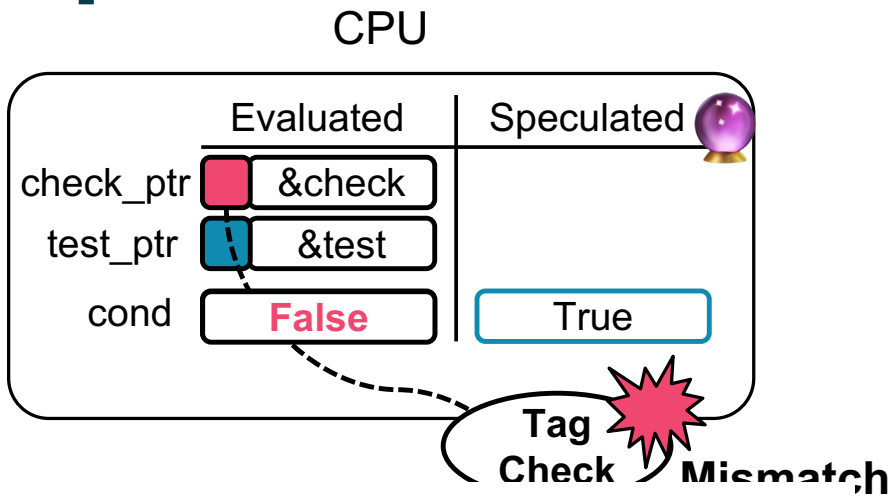
TIKTAG[2]

Speculative prefetch of instructions
$\Rightarrow$ **Avoid** segmentation fault to leak KASLR

Exploit speculative execution
$\Rightarrow$ **Bypass** memory tag check fault

B. Invalid tag in check_ptr





No reason to continue speculative execution

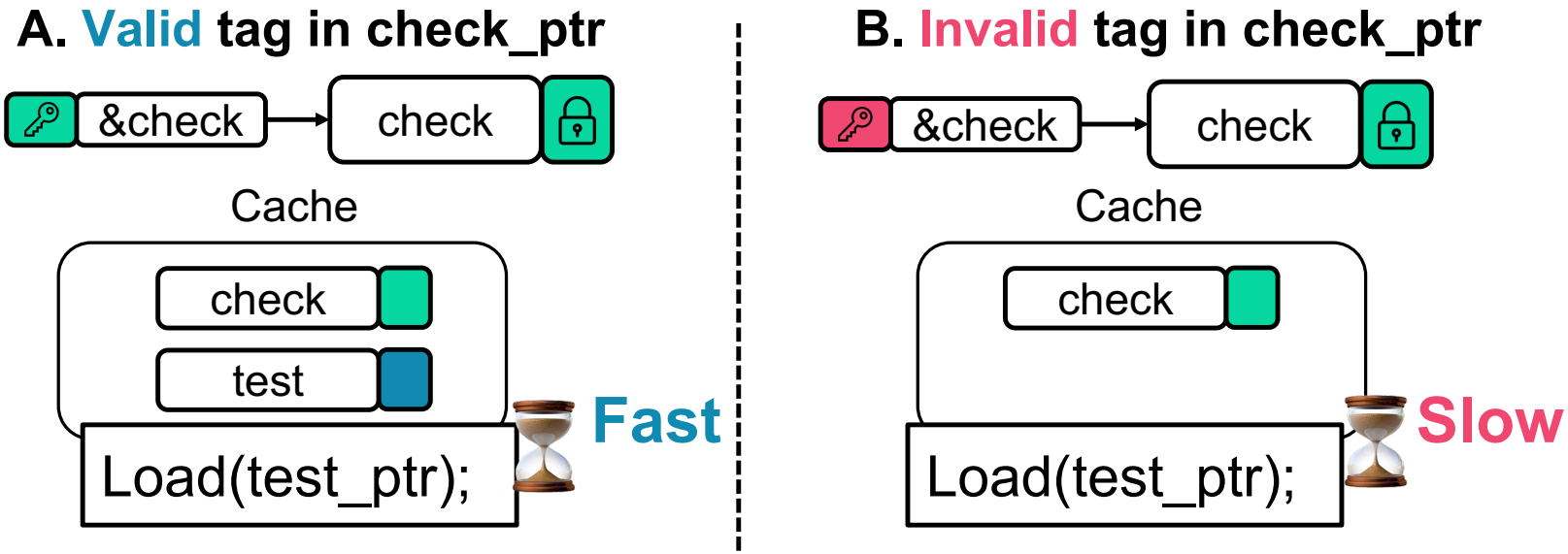
Correct spec→ (synchronous) tag check fault

Wrong spec → Revert execution

check, not test

43

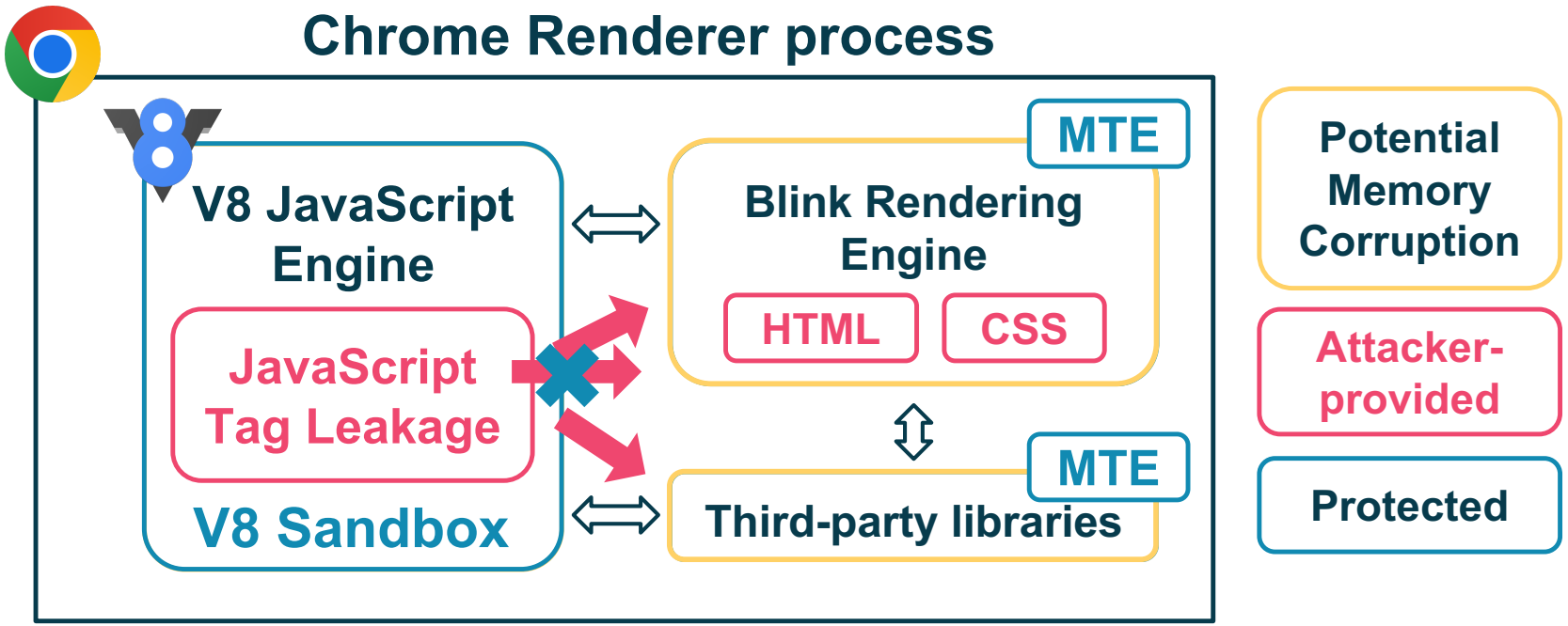
Leak by Cache Side-Channel



Leak whether the tag is Valid/Invalid by test_ptr access latency!

44

Google Chrome Threat Model



52

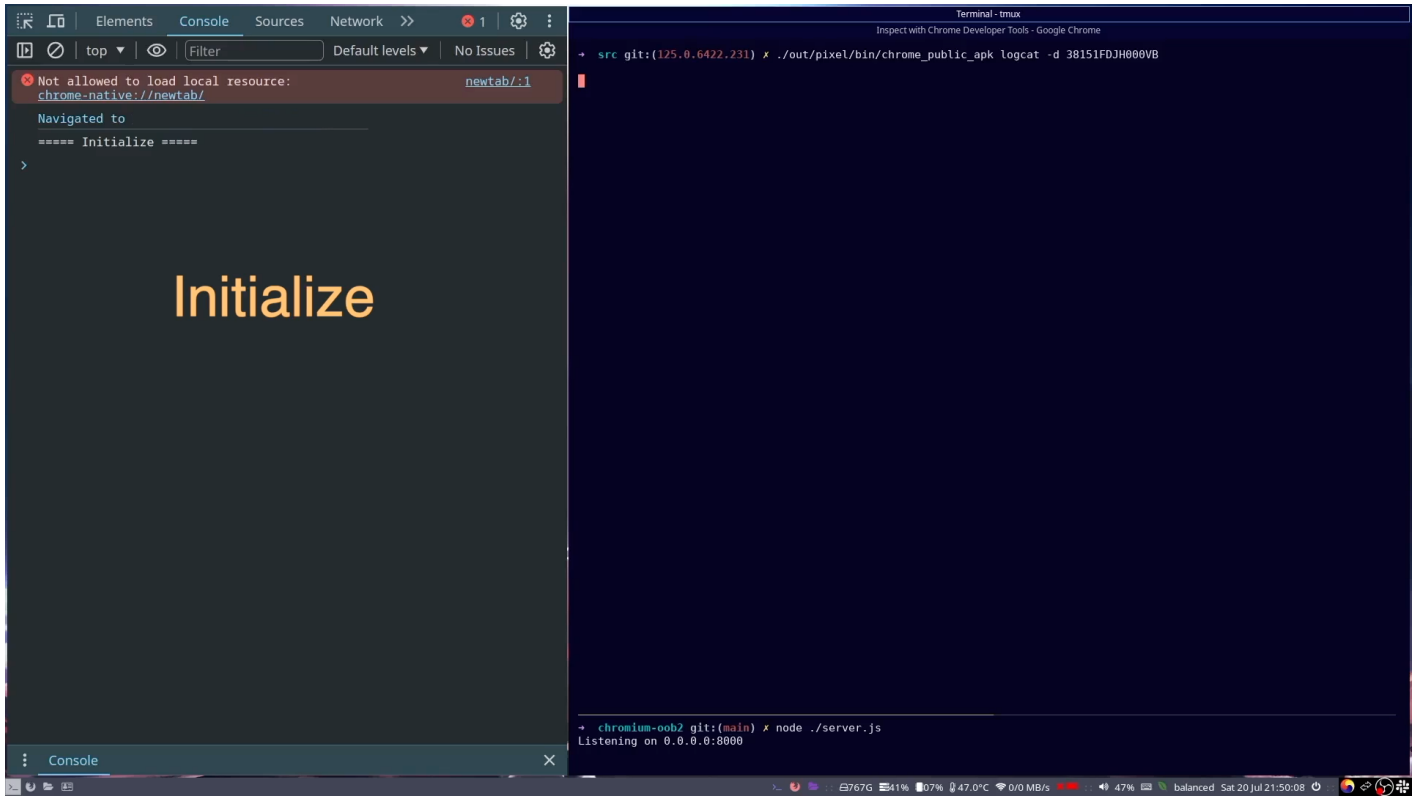
TIKTAG[2]

Case Study: Chrome MTE Bypass Attack

- Leak MTE tag of vulnerable object
- Leak MTE tag of target object
- Keep reallocating target if the tags are different
- Access target object with forged pointer

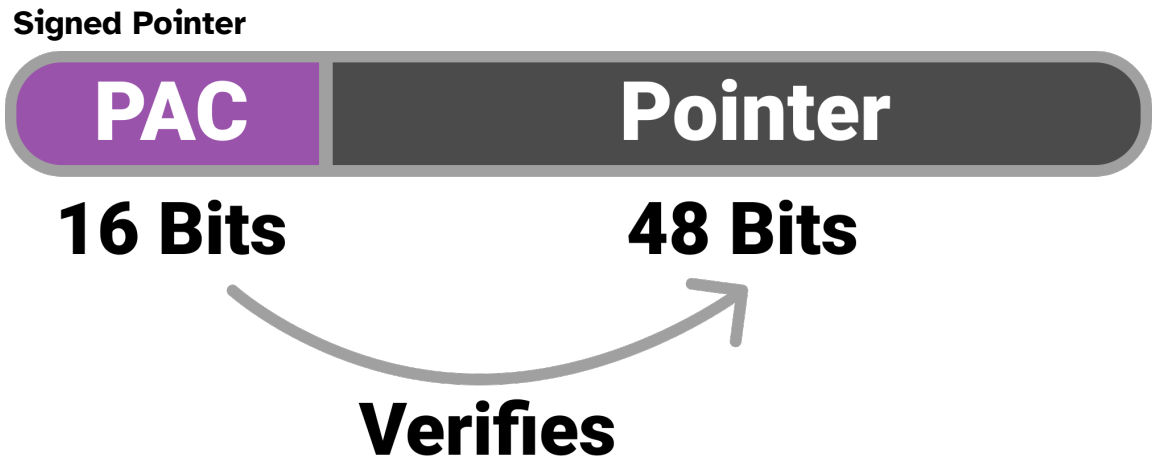
CVE-2023-5217 Chrome libvpx heap overlfow

With MTE Tag Leakage → Attack Success



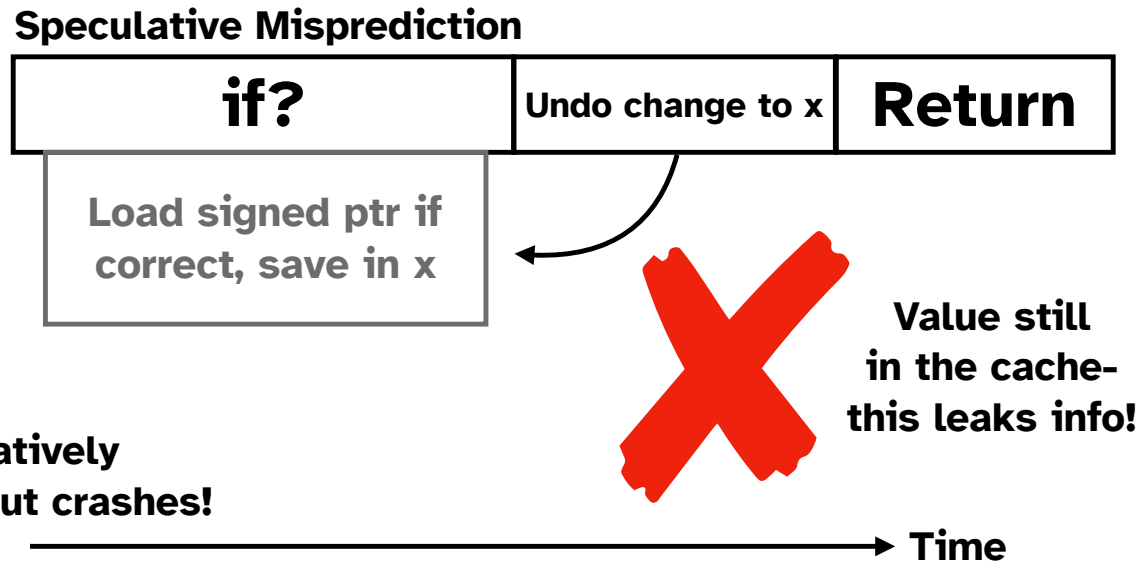
Pointer Authentication

```
PAC = hash(pointer, salt, key)
```



Speculative PACs

```
if (true)
    return
else
    check signed ptr
    x = load signed ptr
```



Operating on PACs speculatively can leak PAC correctness without crashes!

References I

[1] Enrico Barberis et al. "Branch History Injection: On the Effectiveness of Hardware Mitigations Against Cross-Privilege Spectre-v2 Attacks". In: *31st USENIX Security Symposium (USENIX Security 22)*. Boston, MA: USENIX Association, Aug. 2022, pp. 971–988. ISBN: 978-1-939133-31-1. URL: https://www.usenix.org/conference/usenixsecurity22/presentation/barberis.

[2] Juhee Kim et al. *TikTag: Breaking ARM's Memory Tagging Extension with Speculative Execution*. 2024. arXiv: 2406.08719 [cs.CR]. URL: https://arxiv.org/abs/2406.08719.

References II

[3] Joseph Ravichandran et al. "PACMAN: Attacking ARM Pointer Authentication with Speculative Execution". In: *Proceedings of the 49th Annual International Symposium on Computer Architecture*. ISCA '22. New York, New York: Association for Computing Machinery, 2022. ISBN: 9781450386104. DOI: 10.1145/3470496.3527429. URL: https://doi.org/10.1145/3470496.3527429.