CSE 610 Special Topics: System Security - Attack and Defense for Binaries

Instructor: Dr. Ziming Zhao

Location: Frnczk 408, North campus

Time: Monday, 5:20 PM - 8:10 PM

Announcement

Final Exam: 12/14 2020 7:15PM-10:15PM

Same format as the mid-term.

There will be 4 challenges labelled with the vulnerability type.

No in-class CTF. Instead there will be a ***take-home exam***. It will have 2 offline challenges and multiple choices questions.

Today’s Agenda

1. Meltdown

Meltdown and Spectre

https://meltdownattack.com/

https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2017-5754

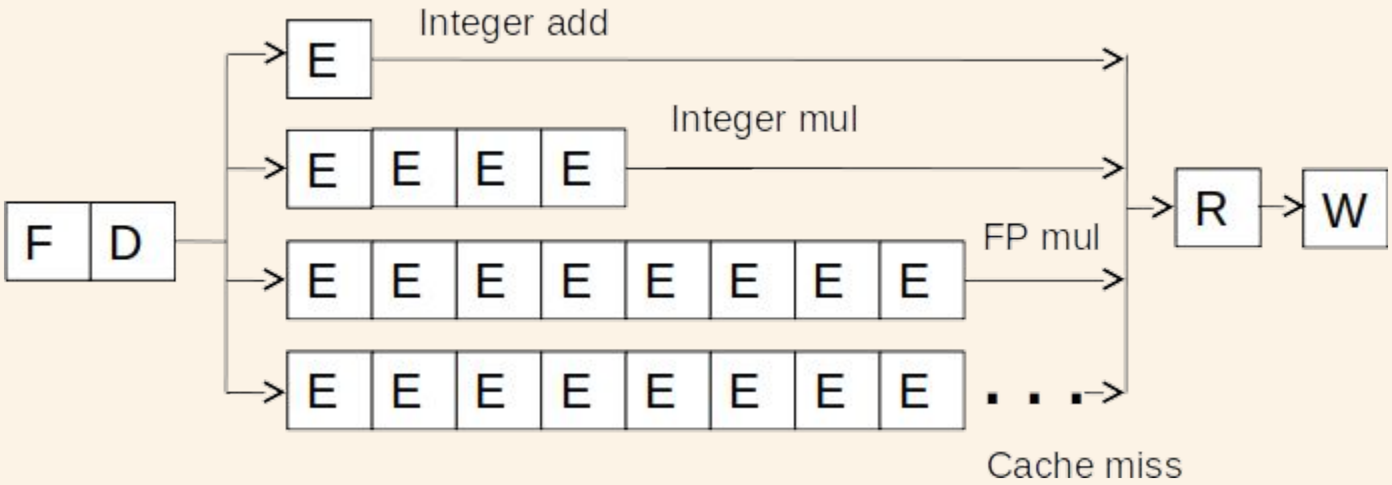
Meltdown Basics

Meltdown allows attackers to read arbitrary physical memory (including kernel memory) from an unprivileged user process

Meltdown uses ***out of order instruction execution*** to leak data via a processor covert channel (cache lines)

Meltdown was patched (in Linux) with KAISER/KPTI

An In-order Pipeline

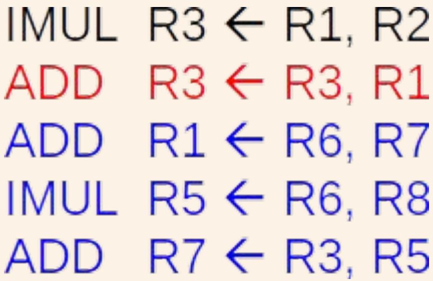


Problem: A true data dependency stalls dispatch of younger instructions into functional (execution) units

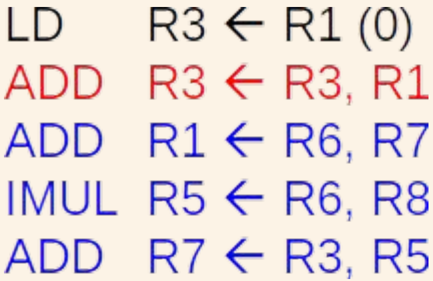
Dispatch: Act of sending an instruction to a functional unit

Can We Do Better?

What do the following two pieces of code have in common (with respect to execution in the previous design)?



```
IMUL  R3 ← R1, R2
ADD   R3 ← R3, R1
ADD   R1 ← R6, R7
IMUL  R5 ← R6, R8
ADD   R7 ← R3, R5
```



```
LD    R3 ← R1 (0)
ADD   R3 ← R3, R1
ADD   R1 ← R6, R7
IMUL  R5 ← R6, R8
ADD   R7 ← R3, R5
```

Answer: First ADD stalls the whole pipeline!
ADD cannot dispatch because its source registers unavailable
Later independent instructions cannot get executed

Out-of-Order Execution
(Dynamic Instruction Scheduling)

Idea: Move the dependent instructions out of the way of independent ones; Rest areas for dependent instructions: Reservation stations

Monitor the source “values” of each instruction in the resting area. When all source “values” of an instruction are available, “fire” (i.e. dispatch) the instruction. Instructions dispatched in dataflow (not control-flow) order

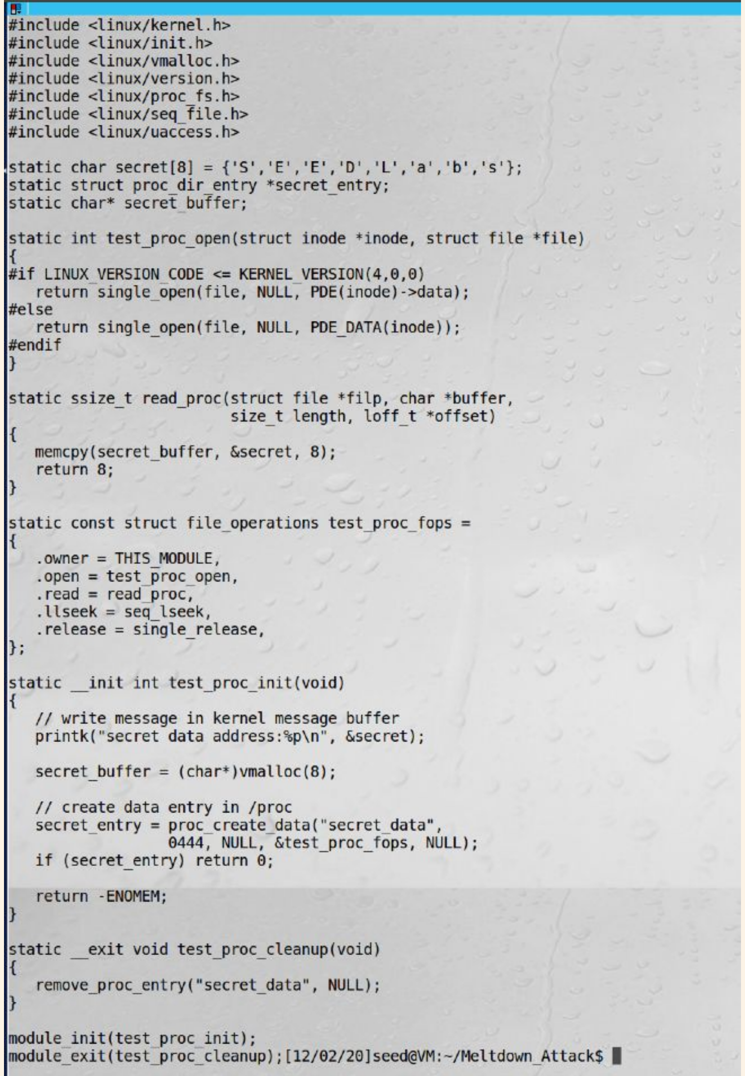
Benefit: Latency tolerance: Allows independent instructions to execute and complete in the presence of a long latency operation

In-order vs. Out-of-order Dispatch

F	D	E	E	E	E	R	W								
	F	D	STALL			E	R	W							
		F	STALL			D	E	R	W						
						F	D	E	E	E	E	E	R	W	
							F	D	STALL				E	R	W

```
IMUL  R3 ← R1, R2
ADD   R3 ← R3, R1
ADD   R1 ← R6, R7
IMUL  R5 ← R6, R8
ADD   R7 ← R3, R5
```

F	D	E	E	E	E	R	W				
	F	D	WAIT			E	R	W			
		F	D	E	R				W		
			F	D	E	E	E	E	R	W	
				F	D	WAIT			E	R	W



```
#include <linux/kernel.h>
#include <linux/init.h>
#include <linux/vmalloc.h>
#include <linux/version.h>
#include <linux/proc_fs.h>
#include <linux/seq_file.h>
#include <linux/uaccess.h>

static char secret[8] = {'S','E','E','D','L','a','b','s'};
static struct proc_dir_entry *secret_entry;
static char* secret_buffer;

static int test_proc_open(struct inode *inode, struct file *file)
{
#if LINUX_VERSION_CODE <= KERNEL_VERSION(4,0,0)
   return single_open(file, NULL, PDE(inode)->data);
#else
   return single_open(file, NULL, PDE_DATA(inode));
#endif
}

static ssize_t read_proc(struct file *filp, char *buffer,
                         size_t length, loff_t *offset)
{
   memcpy(secret_buffer, &secret, 8);
   return 8;
}

static const struct file_operations test_proc_fops =
{
   .owner = THIS_MODULE,
   .open = test_proc_open,
   .read = read_proc,
   .llseek = seq_lseek,
   .release = single_release,
};

static __init int test_proc_init(void)
{
   // write message in kernel message buffer
   printk("secret data address:%p\n", &secret);

   secret_buffer = (char*)vmalloc(8);

   // create data entry in /proc
   secret_entry = proc_create_data("secret_data",
                  0444, NULL, &test_proc_fops, NULL);
   if (secret_entry) return 0;

   return -ENOMEM;
}

static __exit void test_proc_cleanup(void)
{
   remove_proc_entry("secret_data", NULL);
}

module_init(test_proc_init);
module_exit(test_proc_cleanup);[12/02/20]seed@VM:~/Meltdown_Attack$ █
```