

Meltdown and Spectre Samples

Written in Assembly

U. Plonus

March 7, 2018

Contents

1	Introduction	5
1.1	Overview	5
1.2	Nasm	5
2	Cache Access Timing	7
3	Detect Cache Access Time	9
4	Read Array via Cache Access Time	11
5	Signals	13
5.1	Detecting Signals	13
5.2	Handling Signals	13
6	Utilities	15
6.1	Introduction	15
6.2	Quit Program	15
6.3	Printing Text	15
6.4	Printing Numbers	16

1 Introduction

1.1 Overview

TBD

1.2 Nasm

TBD

5 $\langle preamble\ 5 \rangle \equiv$ (9a)
 bits 64
 global _start

2 Cache Access Timing

3 Detect Cache Access Time

TBD

9a $\langle cachetiming.asm \ 9a \rangle \equiv$
 $\langle preamble \ 5 \rangle$

$\langle cachetiming-data \ 9b \rangle$

```
section .text
_start:
 $\langle quit \ 15b \rangle$ 
```

$\langle utilities \ 15a \rangle$

TBD

9b $\langle cachetiming-data \ 9b \rangle \equiv$ (9a)

```
section .bss
    measures: resq 2048
    padding:  resb 4096
    align 4096
    data:      times 257 resb 4096
```


4 Read Array via Cache Access Time

TBD

5 Signals

5.1 Detecting Signals

TBD

5.2 Handling Signals

TBD

6 Utilities

6.1 Introduction

TBD

15a $\langle utilities\ 15a \rangle \equiv$ (9a)
 $\langle print\ 15c \rangle$

6.2 Quit Program

TBD

15b $\langle quit\ 15b \rangle \equiv$ (9a)
 xor RDI,RDI
 mov RAX,60
 syscall
Uses RAX and RDI.

6.3 Printing Text

The routine `_print` prints a null-terminated string to the terminal (stdout). The only argument passed in to the routine (in RDI) is the address of the string to print.

So first we start with clearing AL (setting it to null) and saving the address of the string to RSI. We're using RSI because we later need the address to calculate the length of the string and also RSI is the register that we need to use for the string address in the systemcall.

15c $\langle print\ 15c \rangle \equiv$ (15a) 16a▷
 _print:
 xor AL,AL
 mov RSI,RDI
Uses AL, RDI, and RSI.

Next we search for the terminating `null` (`'\0'`) character. For this we use the instruction `scasb` (scan string byte) which compares the byte at the pointer `[RDI]` with the value in `AL` and sets the flags accordingly. When the byte at `[RDI]` is the value of `AL` the the next instruction (`je`) jumps to the given target (`.found0` in this case). Else the next instruction will be executed (jumping back to the start of this fragment).

`scasb` additionally increments `RDI` so that we go through the string until `\0` is found.

```
16a  <print 15c>+≡ (15a) <15c 16b>
      .next_char:
      scasb
      je      .found0
      jmp     .next_char
      .found0:
```

After we have found the string termination we calculate the number of bytes that the string has. For this we copy the value of the last byte read (which is in `RDI`) to `RDX` and subtract the start of the string (which we saved to `RSI`).

Now we have the address of the string in `RSI` and the length of the string in `RDX` which are the 2nd and 3rd argument in a `syscall`. The 1st argument (in `RDI`) to the `syscall` is the file descriptor (1 is `stdout`). Additionally the number of the `syscall` (1) is passed in `RAX`. The `syscall` (`syscall`) now prints `RDX` bytes from `RSI` to the file descriptor `RDI`.

```
16b  <print 15c>+≡ (15a) <16a 16c>
      mov     RDX,RDI
      sub     RDX,RSI
      mov     RAX,1
      mov     RDI,1
      syscall
```

Uses `RAX`, `RDI`, `RDX`, and `RSI`.

Now that we are done and can return to the caller.

```
16c  <print 15c>+≡ (15a) <16b
      ret
```

6.4 Printing Numbers

TBD