Meltdown and Spectre Samples

Written in Assembly

U. Plonus

March 7, 2018

Contents

1 Introduction

1.1 Overview

TBD

1.2 Nasm

TBD

5 ⟨*preamble* 5⟩≡ (7a)

```
bits 64
      global _start
```

2 Detect Cache Access Time

TBD

7a ⟨*cachetiming.asm* 7a⟩≡

```
⟨preamble 5⟩

⟨cachetiming-data 7b⟩

section .text
_start:
⟨quit 13b⟩

⟨utilities 13a⟩
```

TBD

7b ⟨*cachetiming-data* 7b⟩≡ (7a)

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

3 Read Array via Cache Access Time

TBD

4 Handling Signals

4.1 Detecting Signals

TBD

5 Utilities

5.1 Introduction

TBD

13a ⟨*utilities* 13a⟩≡ (7a)

```
⟨print 13c⟩
```

5.2 Quit Program

TBD

13b ⟨*quit* 13b⟩≡ (7a)

```
        xor     RDI,RDI
        mov     RAX,60
        syscall
```

Uses RAX and RDI.

5.3 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.

13c ⟨*print* 13c⟩≡ (13a) 14a ▷

```
  _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.

14a ⟨*print* 13c⟩+≡ (13a) ◁13c 14b▷

```
.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 systemcall. The 1st argument (in `RDI`) to the systemcall is the file descriptor (`1` is stdout). Additionally the number of the systemcall (`1`) is passed in `RAX`. The systemcall (`syscall`) now prints `RDX` bytes from `RSI` to the file descriptor `RDI`.

14b ⟨*print* 13c⟩+≡ (13a) ◁14a 14c▷

```
     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.

14c ⟨*print* 13c⟩+≡ (13a) ◁14b

```
     ret
```

5.4 Printing Numbers

TBD