

Meltdown and Spectre Samples

Written in Assembly

U. Plonus

March 8, 2018

Contents

1	Introduction	5
1.1	Overview	5
1.2	Nasm	5
2	Cache Access Timing	7
2.1	Detect Cache Access Time	7
2.2	Read Array via Cache Access Time	7
3	Signals	9
3.1	Detecting Signals	9
3.2	Handling Signals	9
4	Utilities	11
4.1	Introduction	11
4.2	Exit Program	11
4.3	Printing Text	11
4.4	Printing Numbers	12

1 Introduction

1.1 Overview

TBD

1.2 Nasm

TBD

5 $\langle preamble\ 5 \rangle \equiv$ (7a)
 bits 64
 global _start

2 Cache Access Timing

2.1 Detect Cache Access Time

TBD

7a $\langle cachetiming.asm\ 7a \rangle \equiv$
 $\langle preamble\ 5 \rangle$

 $\langle cachetiming-data\ 7b \rangle$

```
section .text
_start:
 $\langle exitProgram\ 11b \rangle$ 
```

$\langle utilities\ 11a \rangle$

TBD

7b $\langle cachetiming-data\ 7b \rangle \equiv$ (7a)
section .bss
measures: resq 2048
padding: resb 4096
align 4096
data: times 257 resb 4096

2.2 Read Array via Cache Access Time

TBD

3 Signals

3.1 Detecting Signals

TBD

3.2 Handling Signals

TBD

4 Utilities

4.1 Introduction

TBD

11a $\langle utilities\ 11a \rangle \equiv$ (7a)
 $\langle print\ 11c \rangle$

4.2 Exit Program

TBD

11b $\langle exitProgram\ 11b \rangle \equiv$ (7a)
 xor RDI,RDI
 mov RAX,60
 syscall
Uses RAX and RDI.

4.3 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.

11c $\langle print\ 11c \rangle \equiv$ (11a) 12a▷
 _print:
 xor AL,AL
 mov RSI,RDI
Uses AL, RDI, and RSI.

4 Utilities

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 address `[RDI]` with the value in `AL` and sets the flags accordingly. When the byte at `[RDI]` is not the value of `AL` the the next instruction (`jne`) jumps to the given label (`.next_char` in this case).

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

```
12a  <print 11c>+≡ (11a) <11c 12b>
      .next_char:
      scasb
      jne      .next_char
```

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`.

```
12b  <print 11c>+≡ (11a) <12a 12c>
      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.

```
12c  <print 11c>+≡ (11a) <12b>
      ret
```

4.4 Printing Numbers

TBD