НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
ФІЗИКО-ТЕХНІЧНИЙ ІНСТИТУТ
Кафедра інформаційної безпеки

«До захисту допущено»
В.о. завідувача кафедри

_____ М.В.Грайворонський
(підпис)

“_____”_____2019 р.

Дипломна робота

на здобуття ступеня бакалавра

з напряму підготовки 6.170101 «Безпека інформаційних і комунікаційних систем»

на тему: Дослідження атак, пов’язаних зі спекулятивним виконанням коду на сучасних архітектурах процесорів

Виконав: студент 4 курсу, групи ФБ-52

Дегтярьов Андрій Дмитрович _____ _____
(прізвище, ім’я, по батькові) (підпис)

Керівник к.ф.-м.н., доц. Грайворонський Микола Владленович _____ _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Консультант _____ _____ _____
(назва розділу) (посада, вчене звання, науковий ступінь, прізвище, ініціали) (підпис)

Рецензент _____ _____
(посада, науковий ступінь, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ - 2019 року

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
ФІЗИКО-ТЕХНІЧНИЙ ІНСТИТУТ
Кафедра інформаційної безпеки

Рівень вищої освіти – перший (бакалаврський)

Напрям підготовки 6.170101 «Безпека інформаційних і комунікаційних систем»

ЗАТВЕРДЖУЮ
В.о. завідувача кафедри
_____ М.В.Грайворонський
(підпис)
«___»_____2019 р.

ЗАВДАННЯ
на дипломну роботу студенту

_____
(прізвище, ім'я, по батькові)

1. Тема роботи _____
_____
_____ ,
науковий керівник роботи_____
_____,
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від «27» травня 2019 р. № _1414-с_

2. Термін подання студентом роботи 10 червня 2019 р.

3. Вихідні дані до роботи _____
_____
_____

4. Зміст роботи _____
_____
_____
_____
_____

5. Перелік ілюстративного матеріалу (із зазначенням плакатів, презентацій тощо)
_____

_____

_____

6. Дата видачі завдання _____

Календарний план

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів дипломної роботи	Примітка

Студент _____ _____

(підпис) (ініціали, прізвище)

Науковий керівник роботи _____ _____

(підпис) (ініціали, прізвище)

Реферат

Обсяг роботи 50 сторінок, 13 ілюстрації, 1 таблиця, 8 додатків, 22 джерела літератури.

Об’єктом дослідження є вразливості архітектур сучасних процесорів, пов’язані зі спекулятивним виконанням коду.

Предметом дослідження є техніки очищення процесорного кешу, що унеможливлюють утворення стороннього каналу витоку інформації за часом доступу до данних.

Дослідження атак, пов’язаних зі спекулятивним виконанням коду на сучасних архітектурах процессорів.

Abstract

Publication size: 50 pages with 13 illustrations, 1 table, 8 appendixes and 22 references.

Research objects are vulnerabilities of modern processors architecture related with speculative code execution.

Research subjects are cache freeing techniques that mitigating side channel attacks.

Researching the Attacks Related to Speculative Code. Execution on the Modern Processors Architecture

Зміст

Реферат 4

Зміст 6

Перелік умовних позначень, символів, скорочень і термінів 8

Вступ 9

1 Вразливості пов’язані зі спекулятивним виконанням коду 11

1.1 Основи вразливостей спекулятивного виконання коду та їх експлуатація 11

1.2 Процесори, яких торкнулась проблема вразливостей спекулятивного виконання коду 15

1.3 Ризик’и 16

1.4 Існуючі методи захисту 19

1.5 Retpoline 20

1.6 Вплив існуючих методів захисту на швидкодію сучасних обчислювальних систем 21

Висновки до розділу 1 25

2 Збір статистичних данних 27

2.1 Методика отримання емпіричних данних 27

2.2 Збір статистичних данних про середнє значення відгуку данних з ОЗП 28

2.3 Збір статистичних данних про час відгуку ОЗП в залежності від частоти очищення кешу 29

Висновик до розділу 2 31

3 Обробка статистичних данних 32

3.1 Запропоновані моделі апроксимації отриманих експериментальних данних 32

3.2 Лінійна модель 32

3.3 Гіперболічна модель 35
3.4 Полніноміальна модель 38
Висновки до розділу 3 43
4 Практичне застосування отриманих результатів 44
4.1 Знаходження оптимальних частот очищення кешу 44
4.2 Практичне застосування отриманих результатів 45
4.3 Висновки до розділу 46
Висновки 47
Перелік джерел посилань 49
Додаток A 52
Додаток B 52
Додаток C 53
Додаток D 54
Додаток E 55
Додаток F 55

Перелік умовних позначень, символів, скорочень і термінів

ОЗП – опертивний запам’ятовуючий пристрій.

x86 – 32-бітна архітектура процесорів

Spectre – сімейство вразливостей, пов’язаних зі спекулятичним виконанням інструкцій на сучасних процесорах, що дозволяє зловнмисникам зчитувати пам’ять іншим процесів

x86_64 – 64-бітна архітектура процесорів

rdtsc - (Read Time Stamp Counter) асемблерна інструкція для процесорів архітектури x86 та x68_64

SSE (англ. Streaming SIMD Extensions, потокове SIMD-розширення процесора) — це SIMD (англ. Single Instruction, Multiple Data, Одна інструкція — багато даних) набір інструкцій, розроблених Intel, і вперше представлених у процесорах серії Pentium III як відповідь на аналогічний набір інструкцій 3DNow! від AMD, який був представлений роком раніше. Початкова назва цих інструкцій була KIN, що розшифровувалася як Katmai New Instructions (Katmai — назва першої версії ядра процесора Pentium III).

mm_cflush – інструкція для очищення кешу процесору

Експлойт – програма, що доводить можливість експлуатації відомої вразливості

Суперскалярна архітектура – архітектура обчислювального ядра, що використовує кілька декодерів команд, які можуть навантажувати роботою декілька виконавчих блоків. Планування виконання потоку команд є динамічним і здійснюється самим обчислювальним ядром.

Вступ

Актуальність роботи. Ще до недавнього часу сторонні канали витоку інформації в процесорах сімейства x86 вважалися суто теоретичними. Але дослідники проекту Google Project Zero показали на практиці, що організація таких каналів та їх успішна експлуатація є не просто можливою, а до того ж відносно легкою. Проблема криється в мікроархітектурі сучасних процесорів, тому просто позбутися стороннього каналу за часом, не змінюючи архітектуру не виявляється можливим. До того ж з вищеописаного випливає, що до проблем витоку інформації через спекулятивне виконання інструкцій можуть бути вразливі ледь не усі процесори, випущені за останні 10 років. Враховуючи це, а також той факт, що проведення атаки стороннім каналом неможливо виявити чи відслідкувати проблеми Meltdown та Spectre мають ледь не найбільшу актуальність, в порівнянні з будь-якими іншими вразливостями програмного чи апаратного забезпечення.

Мета дослідження:

Мета дослідження - дослідити особливості організації сторонніх каналів витоку інформації у сучасних процесорах та способи їх нівелювання за допомогою запропонованих технік очищення кешу.

Завдання дослідження:

1. проаналізувати взаємозалежність частоти очищення кешу, швидкість доступу до данних з ОЗП та можливості несанкціонованого отримання данних по сторонньому каналу.
2. На основі отриманих данних побудувати математичну модель залежності швидкості відгуку ОЗП від частоти очищення кешу.
3. За допомогою математичної моделі запропонувати оптимальні техніки очищення процесорного кешу, що дають змогу з однієї сторони зруйнувати сторонній канал витоку інформації, а з іншого боку зберегти швидкодію процесору.

Об’єктом дослідження є вразливості архітектур сучасних процесорів, пов’язані зі спекулятивним виконанням коду.

Предметом дослідження є техніки очищення процесорного кешу, що унеможливлюють утворення стороннього каналу витоку інформації за часом доступу до данних.

Методи дослідження. Для дослідження були використані наукові методи, а саме: емпіричний експеримент – збір величини кількості процесорних тактів в залежності від періоду очищення кешу процесору, та наукове моделювання – побудова математичної моделі данних, отриманих в результаті експерименту.

Наукова новизна. В результаті дослідження було отримано функцію залежності швидкості отримання данних з ОЗП від частоти оновлення кешу, а також оптимальні, з точки зору швидкодії системи, техніки очищення кешу, що руйнують сторонній канал витоку інформації.

Практичне значення. З практичної точки зору такі техніки можуть бути використані для захисту критичного коду від атак, що базуються на вразливостях типу Spectre на будь-яких, навіть вразливих, платформах без суттєвої втрати швидкодії. Такі техніки можуть бути корисними до використання в імплементаціях криптографічних алгоритмів, де з одного боку важливий захист проміжних обчислень, а з іншого боку – швидкодія. Також, такий підхід дає змогу виокремити ті программи та частини програм, де можливість отримання данних стороннім каналом не є небезпечною. Наприклад, в багатьох прикладних застосунках немає обробки критичних данних, тому захист від атак за стороннім каналом витоку інформаціх не є актуальним, проте швидкодія цих програм все ж залишається в пріоритеті для розробників таких продуктів та для кінцевих споживачів.

Використовуючи результати, отримані в ході данного дослідження можна розробити інструменти статичного аналізу та автоматичного доповнення існуючої кодової бази оптимальними алгоритмами очищення кешу.Таким чином розробка інструментів для відносно легкого доповнення необхідних частин вже існуючої кодової бази методами очистки кешу могла б вирішити одразу цілий ряд проблем: захист від стороннього каналу витоку інформації на будь-яких вразливих платформах з мінімально-можливими втратами швидкодії.

1 Вразливості пов’язані зі спекулятивним виконанням коду

1.1 Основи вразливостей спекулятивного виконання коду та їх експлуатація

Одним із слабких місць мікроархітектури сучасних процесорів стали відомі вразливості спекулятивного виконання коду. Суть спекулятивного виконання коду полягає у тому, що branch predictor намагається завчасно передбачити, який код буде виконаний наступним. Якщо деяка гілка коду була обрана, як така, що може бути виконана наступною з високою ймовірністю, тоді ця гілка виконується завчасно на паралельному ядрі процесору, а результати зберігаються у кеш процесору. Потім, якщо передбачення було вірним – код цієї гілки не виконується, а одразу повертається результат виконання з кешу. Якщо branch predictor зробив помилкове припущення, тоді запускається необхідна гілка коду і інструкції виконуються так, ніби не було ніякого спекулятивного виконання. Це доволі суттєво економить час виконання інструкцій, особливо враховуючи, що branch predictor рідко помиляється. (У циклі for на 10 ітерацій, наприклад, branch predictor зробить 9 вірних припущень з 10).

За рахунок таких оптимізацій вже котрий рік гіганти індустрії можуть пишатися все більш і більш продуктивними чіпами.

В свою чергу це створює неявні канали витоку інформації. Так, аналізуючи час отримання доступу до змінних, дослідники з Google Project Zero змогли проаналізувати, які змінні кешуються, в результаті того, що над ними були спекулятивно виконані інструкції. Очевидно, що час доступу до таких змінних був настільки меншим, що дозволяв з непоганою точністю відокремити такі змінні.

А це – ніщо інше, як канал витоку інформації за часом. Враховуючи те, що закешованими можуть бути, наприклад, проміжні обчислення криптографічних алгоритмів – можна сказати що ця проблема є дуже суттєвою.

На данний момент можна виділити 2 підкласи атак на спекулятивне виконання коду:

Meltdown (CVE-2017-5754) та Spectre (CVE-2017-5753 та CVE-2017-5715) Варіацій атак meltdown та Spectre значно більше.

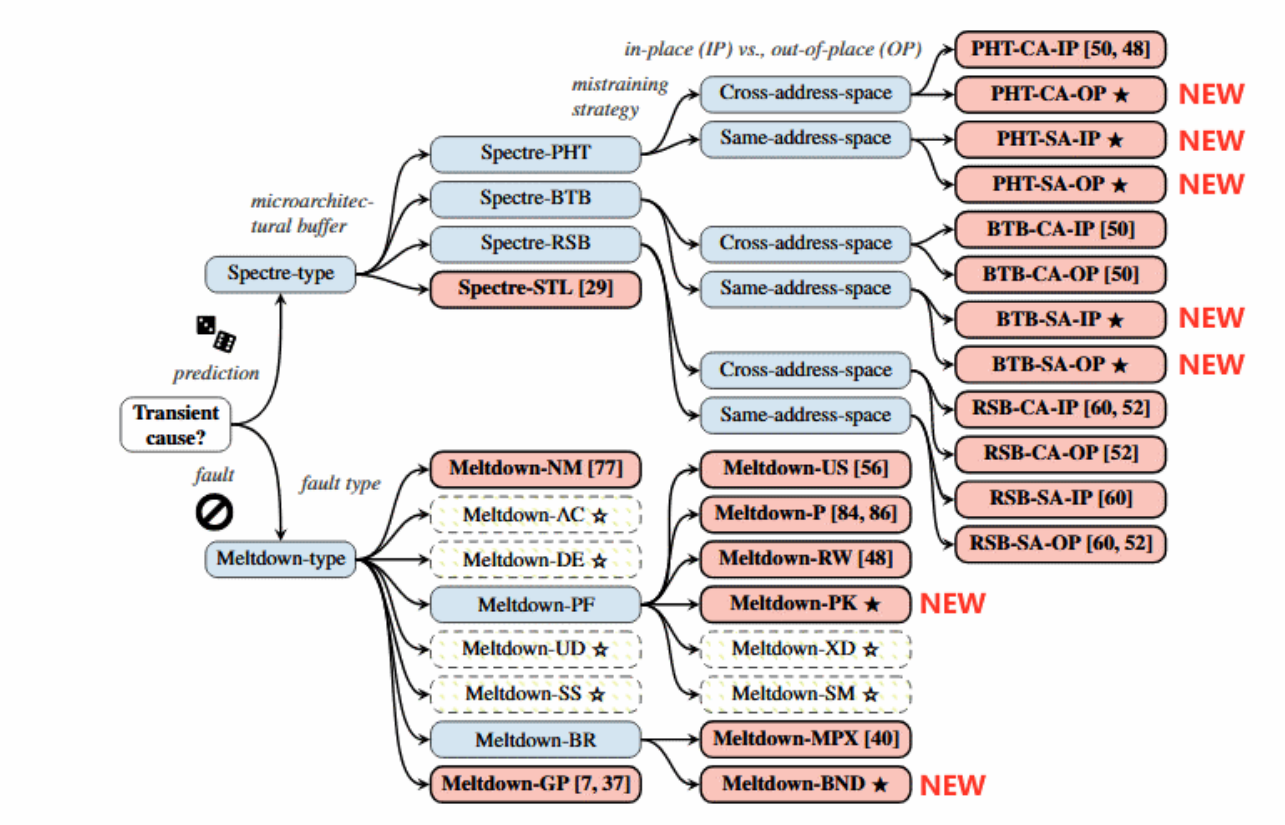

Рисунок 1.1 - Відомі вразливості спекулятивного виконання інструкцій [1]

Дослідниками Google Project Zero, що знайшли оригінальну вразливість (Jann Horn) були опублікавоні доказові експлойти для наведених вразливостей. Розглянемо практичне втілення Bounds check bypass (Spectre variant 1) на прикладі цього експлойту детальніше:

```
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];
```

array1 – спільна пам’ять для жертви та атакуючого. Що в ній – не так важливо. Масиви unused1 та unused2 – не використовуються та існують тільки для того, щоб попасти в різні секції кешу. На багатьох процесорах сімейства Intel Core I – довжина секції – 64 байти, що і визначає розмір цих масивів.

У рядку 25 змінна secret зберігає деяке секретне значення, доступне тільки жертві.

В цьому прикладі код жертви та атакуючого виконується в контексті одного процесу тільки для спрощення. В реальних умовах ми б мали справу з двома окремими процесами зі спільною пам’яттю, а атакуючий мав би змогу викликати код жертви.

unsigned int junk = **0**;

цей рядок потрібний тільки для того, щоб впевнитися, що оптимізації компілятору не будуть застосовані в цьому місці.

Коли змінна x стає рівною array1_size вразливий процесор робить наступну послідовність дій:

1. читає з ОЗП незакешоване array1_size
2. Поки array1_size зчитується з відносно повільного ОЗП, branch predictor невірно передбачає, що x < array1_size і потрібно знову виконувати тіло циклу
3. Спекулятивне зчитування array1[x]. Виконується швидко, адже це значення закешоване.
4. Зчитування array2[array1[x] * 512]. Це виконується довго, тому, що ці значення не закешовані.
5. Поки виконується 4 крок вже завантажилася з ОЗП змінна array1_size. Процесор реєструє невірне передбачення у кроці 2 та відкидає результати обчислень.

```
void victim_function(size_t x)
{
    if (x < array1_size)
    {
        temp &= array2[array1[x] * 512];
    }
}
```

Значення array1[x] множиться на число 512 = 64*8, де 64 – довжина секції кешу в байтах – 8 – кількість байтів для специфічного процесору (Intel Core I). Тож ці 2 (а отже і їх добуток) – залежать від конкретної моделі чіпу.

Функція readMemoryByte() намагається вгадати значення за заданим адресом. Для всіх можливих значень байту (0x0 .. 0xFF) виконується атака flush + reload на кеш. Всі значення зберігаються в масив results. Функція повертає тільки 2 найменших значення. В наступних рядках ми бачимо очищення кешу від сторонніх значень на початку атаки. В циклі використовується інструкція __mm_cflush з intrinsic.h що очищує кеш. Саме ця функція і буде широко використовуватись у нашому дослідженні для блокування стороннього каналу витоку інформації [4].

```
for (i = 0; i < 256; i++)
    _mm_clflush(&array2[i * 512]);
```

Наступні рядки коду виконують задачу тренування алгоритму branch predictor.

```
x = ((j % 6) - 1) & ~0xFFFF;
x = (x | (x >> 16));
x = training_x ^ (x & (malicious_x ^ training_x));
```

Ці рядки генерують невелике значення x 5 разів, щоб виконувалась умова в циклі, на 6 раз генерується велике значення x, branch predictor помиляється.
Після того, як був спекулятивно викликаний код жертви ми робимо часові заміри.

```
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 ((int)time2 <= cache_hit_threshold &&
       mix_i != array1[tries % array1_size])
       results[mix_i]++;
 }
```

Це і є головна частина атаки на кеш.
mix_i = ((i * 167) + 13) & 255; Тут ми не просто вимірюємо доступ до кожного байту в послідовності, а перемішуємо значення, щоб процесор не оптимізовував доступ

до байтів. Далі після цього рядка ми робимо замір того, як швидко ми отримали доступ до значення у кеші. Якщо ми отримали доступ швидко – ми потрапили (вгадали) в кеш, а отже це значення нещодавно було використане (саме тому воно і опинилося у кеші) під час останнього визову коду жертви. Зверніть увагу на те, що поперед цим ми викликали код жертви спекулятивно (з великим значенням x) спеціально. Під час того виконання і був виконаний доступ до array1[x], так це значення і потрапило до кешу. Але так як x – велике число, то ми при такому спекулятивному виконанні отримали доступ набагато далі в пам’яті ніж планувалось. Так як процесор отримував доступ до значення array2[mix_i * 512], було закешовано номер рядка: mix_i, яке ми і відновили з кешу за допомогою атаки на кеш, описаної вище. Таким чином правильно підібравши зміщення, ми отримали значення секрету жертви.

1.2 Процесори, яких торкнулась проблема вразливостей спекулятивного виконання коду

На сьогодні майже усі сучасні мікропроцесори вразливі до атак спекулятивного виконання коду. Дослідники цих вразливостей розробили та успішно здійснили демонстраційні атаки на чіпи виробництва компаній Intel, AMD, ARM, тощо.

Також вразливі процесори IBM Power: Power7, Power8, Power9;
Fujitsu SPARC64 XII та SPARC64 X+, MIPS P5600 та P660. [2]

Вразливість мікропроцесорів сімейства ARM залежить від конкретної версії архітектури та реалізації ядра конкретним виробником. Відомо та підтверджено наявність уразливості в мікропроцесорах з ядрами: Cortex-R7, Cortex-R8, Cortex-A8, Cortex-A9, Cortex-A15, Cortex-A17, Cortex-A57, Cortex-A72, Cortex-A73 та ARM Cortex-A75. [2]

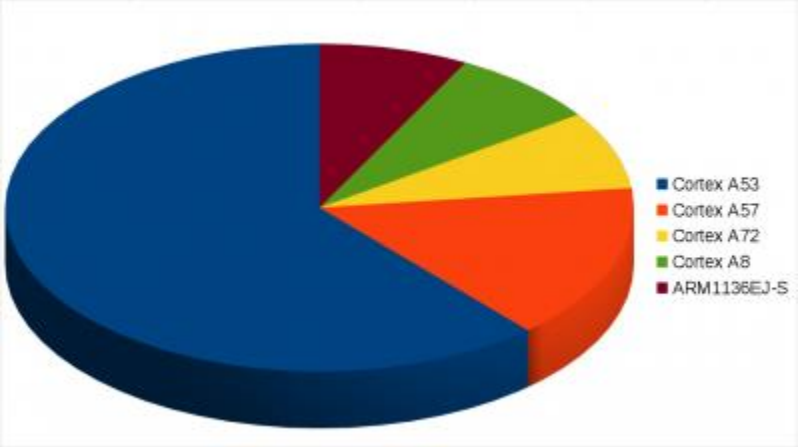

Рисунок 1.2 - Співвідношення найпопулярніших мобільних процесорів на ринку [2]

З графіку зрозуміло, що більшість смартфонів працює на процесорах Cortex A53. На офіційному сайті ARM цей чіп не включено у список вразливих процесорів, тому можна вважати, що на данний момент проблема Spectre не особливо торкнулась сегменту мобільних пристроїв.

Також проблеми типу Spectre присутні у архітектурах багатьох чіпах команії Apple. Виходячи з інформації на офіціному сайті Apple [3] про виправлення та патчі відомих вразливостей, можна зробити висновок, що вразливі всі мобільні процесори, серії A.

З іншої сторони деякі смартфони та ком’ютери з примітивнішими та дешевишими чіпами непідвласні вразливості Spectre, адже в них команди виконуються послідовно, без спроб вгадати наступну гілку виконання.

Наприклад, чіп Raspberry Pi не вразливий ні до Meltdown, ні до Spectre.[2]

1.3 Ризики

Єдиного способу захисту від атак на вразливості типу Spectre на данний час не існує.

Проте, було розроблено механізми захисту (патчі ядер операційних систем, мікрокоду мікропроцесорів, браузерів тощо) від відомих які експлуатують данну

вразливість. Проте щоб повністю позбутися вразливостей класу Spectre виправлень програмного забезпечення недостатньо. Потребуються зміни на рівні мікроархітектури процесорів.

Розробники операційних систем та прикладного пз випустили виправлення своїх продуктів, які покликані зменшити ризик атак, направлених на експлуатацію Spectre. Деякі виробники мікропроцесорів також випустили оновлені версії мікрокоду, покликані захистити від відомих варіантів реалізації, або ж зменшити ризик успішної атаки.

Для експлуатації данної вразливості зловмисник має виконати свої інструкції на стороні жертви. Отже найбільшу небезпеку вразливості типу Spectre несуть для постачальників хмарних обчислень, що дають доступ до багатьох віртуальних, машин, що виконується на одному сервері, окремим клієнтам. Використовуючи вразливість Spectre зловмисники можуть обійти ізоляцію віртуальних машин та отримувати цінні данні з інших віртуальних машин.

Звичайним користувачам найбільшу небезпеку несе вразиливість CVE-2017-5753 (доступ за межі масиву), яка може бути реалізована черз зловмисний код JavaScript у браузері.

Якщо зловмисник буде здатен примусити жертву завантажити свій код JavaScript, він теоретично зможе зчитати з пам’яті жертви будь-яку інформацію, зокрема паролі, що можуть зберігатися у незашифрованому вигляді на диску жертви у вигляді файлів, кешу, в погано організованих менеджерах паролів.

Проте, як виявилось, від варіанту CVE-2017-5753 найпростіше захиститись із майже непомітними наслідками для швидкодії. Відповідні оновлення для своїх веб браузерів уже підготували розробники Mozilla Firefox та WebKit (на основі якого побудовано браузер Chromium, Google Chrome, Opera, тощо).

Дослідники шкідливого програмного забезпечення продовжують реєструвати нові приклади показових експлойтів, що використовують проблему спекулятивного виконання коду.

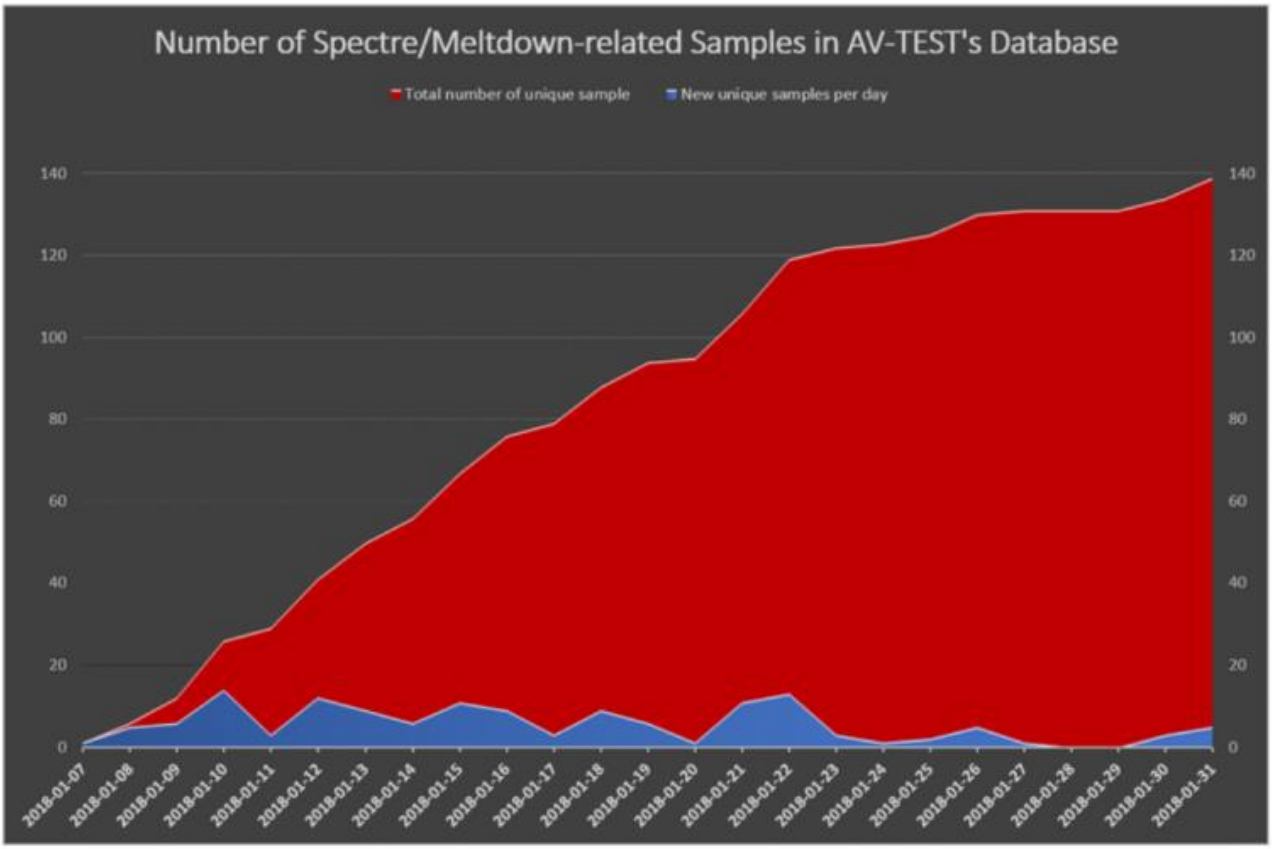

Рисунок 1.3 - Кількість нових зразків шкідливого коду, знайдих компанією AV-TEST. Синій - кількість нових зразків за день. Червоний - кількість нових зразків всього [2]

Отже легко бачити, що розробники шкідливого програмного пз не поспішають використовувати side-channel атаки за часом у реальних проектах. Це пояснюється тим, що для використання таких атак потрібна дуже висока кваліфікація у сфері мікроархітектури сучасних процесорів та низькорівневого програмування. Все додатково ускладнюється тим, що виправлення від вендорів процесорів та розробників операційних систем значно ускладнюють використання вразливостей типу Spectre у реальному житті. Тому, ймовірно, зловмисники наразі перебувають на етапі вивчення існуючої інформації, відлагоджування та тестування експлойтів відомої вразливості.

Але все ж, не дивлячись на складність експлуатації, вразливість існує і її серйозність доведена численними прикладами використання. Тому, в решті-решт, не дивлячись на усю складність цієї предметної області, зловмисники все ж з

відлагодять свої інструменти експлуатації та почнуть їх в повну силу використовувати.

Отже, можна зробити висновок, що атакуюча сторона проявляє неабияку зацікавленість до можливості практичного застосування атаки на вразливості типу Spectre. Хоча, потрібно відмітити, що є велика різниця між показовими експлойтами та повноцінними діючими зразками шкідливого програмного забезпечення. Сравжніх відомих діючих зразків шкідливого пз, що базуються на експлуатації Spectre досі спеціалістами знайдено не було. Це пояснюється тим, що експлуатація вразливостей спекулятивного виконання інструкцій потребує знань мікроархітектури процесорів та низькорівневого програмування. Більш того, є велика різниця між показовим експлойтом, який виконується у лабораторних умовах та являє собою більш академічний інтерес, та реально працюючим практичним шкідливим пз. Перетворити ідею у практичну реалізацію в цій області надзвичайно непросто. Також, програмні та апаратні виправлення від софтверних гігантів та лідерів індустрії значно ускладнюють використання сторонніх каналів.

1.4 Існуючі методи захисту

З вказанами вразливостями ведеться активна боротьба з різних боків:

- Розробники компіляторів GCC додали механізм Retpoline в релізи версії компілятору 8 та новіше.
- Розробники компанії Microsoft, в свою чергу, наділили свій фірмовий компілятор «MSVC» новою опцією «/Qspectre», яка додає в компільовані програми захист від вразливості Spectre variant 1 (CVE-2017-5753). Якщо дана опція ввімкнута, компілятор під час зборки проекту шукатиме потенційно вразливі фрагменти коду та додає інструкцію, що виконує функцію бар'єра для спекулятивного виконання команд.
- Для x86-сумісних мікропроцесорів (Intel та AMD) це інструкція «lfence», для ARM — «csdb».

- У червні 2018 року проект OpenBSD повідомив про рішення учасників проекту відмовитися від функції Hyper-threading при роботі цієї операційної системи на мікропроцесорах виробництва Intel.
- Компанії Intel та AMD мають намір додати три нові інструкції: Branch Restricted Speculation (IBRS), Single Thread Indirect Branch Predictors (STIBP) та Indirect Branch Predictor Barrier (IBPB) до набору інструкцій своїх мікропроцесорів. Нові інструкції будуть додані шляхом завантаження оновлень мікрокоду для процесорів Intel виробництва 2013 року і новіше, а також в нових процесорах AMD. Інформація про ці інструкції оприлюднена в документі з ідентифікатором 336996-001.
- Виправлення для нових мікропроцесорів ARM64 Qualcomm Centriq 2400 (Falkor) та Cavium ThunderX2 будуть додані в нових релізах ядра Linux.
- Компанія AMD, натомість, стверджує, що реалізація атаки сімейства Spectre CVE-2017-5715 (Branch Target Injection/BTI) надзвичайно ускладнена на мікропроцесорах її виробництва. Тому компанія має великий сумнів у потребі щось виправляти. Проте, заради збереження сумісності будуть випущені виправлення, в яких буде додано підтримку інструкцій Indirect Branch Control. Також компанія оприлюднила докладну доповідь про вразливості Spectre у її процесорах.

1.5 Retpoline

Retpoline – це механізм захисту від вразливості Spectre variant 2 (CVE-2017-5715). Назва «Retpoline» походить від двох англійських слів «return» та «trampoline». Цей механізм імітує виконання нескінченного циклу, який ніколи насправді не виконується, але все ж не дає змогу зробити непрямий (спекулятивний) виклик (jump). Така стратегія перемішує непослідовні потоки виконання коду при поверненні значення, таким чином вимикаючи механізм передбачення наступного повернутого значення.

Проте, не дивлячись на усі ці міри, група дослідників компанії Google дійшла висновку, що лише програмними заходами повністю захиститись від витоків інформації через сторонні канали при спекулятивному виконанні програм неможливо, а відмова від спекулятивного виконання недоцільна з точки зору

Більш того, у деяких випадках оновлення спричиняли більше проблем, ніж вирішували. Так, наприклад, оновлення мікрокоду випущеного компанією Intel спричиняли випадкові перезавантаження системи. Тому 23 січня 2018 року компанія порадила відтермінувати оновлення мікрокоду для систем на мікропроцесорах сімейства Broadwell, Haswell, Coffee Lake, Kaby Lake, Skylake і Ivy Bridge. Сервери на багатьох моделях Xeon і Ivy Bridge також мали цю проблему.

1.6 Вплив існуючих методів захисту на швидкодію сучасних обчислювальних систем

Тепер розглянемо вплив існуючих методів захисту на швидкодію обчислювальних систем на практичних прикладах. Для цього було розглянуто ряд проуесорів, що вразливі до атак на спекулятивне виконання інструкцій.
За замовчуванням ядро Linux 4.19-rc1, на якому проводилися тести, не забезпечує «повний» захист від вразливостей шляхом відключення підтримки Intel Hyper Threading / SMT.

Спочатку тести було проведено на стоковому ядрі Linux 4.19-rc1 без будь-якого захисту від атак типу Spectre. Потім, тести повторили із застосуванням різних опцій захисту в реальному часі. Всі системи тестувалися з Ubuntu 18.04.1 LTS x86_64 з ядром Linux 4.19-rc1 через Ubuntu Mainline Kernel PPA, останніми мікрокоді / BIOS, GCC 7.3 і файлової системою EXT4.
Конфігурації обчислювальних систем в тесті:

- Intel Xeon E3-1280 v5 Skylake на материнській платі MSI Z170A SLI PLUS, 16 ГБ DDR4 і 256 ГБ Toshiba RD400 NVMe SSD.

- Intel Xeon E5-2687W v3 Haswell на материнській платі MSI X299 SLI PLUS, 32 ГБ DDR4 і 80 ГБ Intel 530 SATA 3.0 SSD.
- Два Intel Xeon Gold 6138 в стійці Tyan 1U з 96 ГБ RAM і Samsung 970 EVO NVMe SSD 256 ГБ.
- Віртуальна машина KVM на вищезгаданому двопроцесорним сервері Xeon Gold. Ця VM була єдиним активним процесом на машині і була налаштована на доступ до 80% ядер / потоків CPU (64 потоку), 48 ГБ RAM і віртуального диску 118 ГБ. Під час тестування захист від вразливостей відключалася і на хості, і в VM.
- AMD EPYC 7601 на сервері Tyan 2U з 128 ГБ RAM і 280 ГБ Intel Optane 900p NVMe SSD.
- Віртуальна машина KVM на вищезгаданому сервері AMD EPYC 7601. У неї доступ до 80% ядер / потоків CPU (52 потоку), 48 ГБ RAM і віртуального диску 120 ГБ.
- Сервер AMD EPYC 7551 на материнській платі Gigabyte MZ31-AR0 з 32 ГБ RAM і Samsung 960 EVO 256GB NVMe SSD.

Всі тести з набору Phoronix Test Suite.

Були обрані тести, які мають відношення до Spectre / Meltdown, тобто з інтенсивним введенням-виведенням або взаємодіями ядра. Навантаження йде просто на CPU і не сильно залежить від процесорного кешу.

Тестовий профіль CompileBench - ймовірно, найпростіший спосіб показати вплив виправлень для Spectre / Meltdown. На ядрі Linux 4.19 при активації захисту процесори Intel демонструють зниження продуктивності на 7-16%., А процесори AMD – на 3-4%:

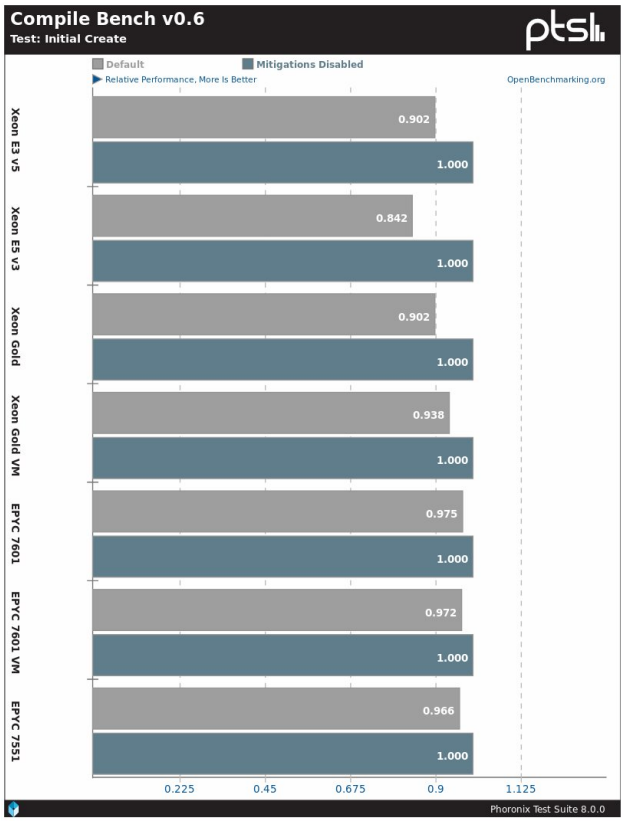

Рисунок 1.5 Результати тестів Phoronix Test Suite з профілелем CompileBench на різних процесорах [7]

Схожа ситуація у підтестах з читанням скомпільованого синтаксичного дерева. (AST). У процесорів Intel зниження продуктивності на 14-15%, у AMD - на 4-5%, що і можна побачити на рисунку 1.5, де показані тестові результати.

В реальних задачах, таких як компіляція ядра Linux різниця буде складати приблизно 2%.

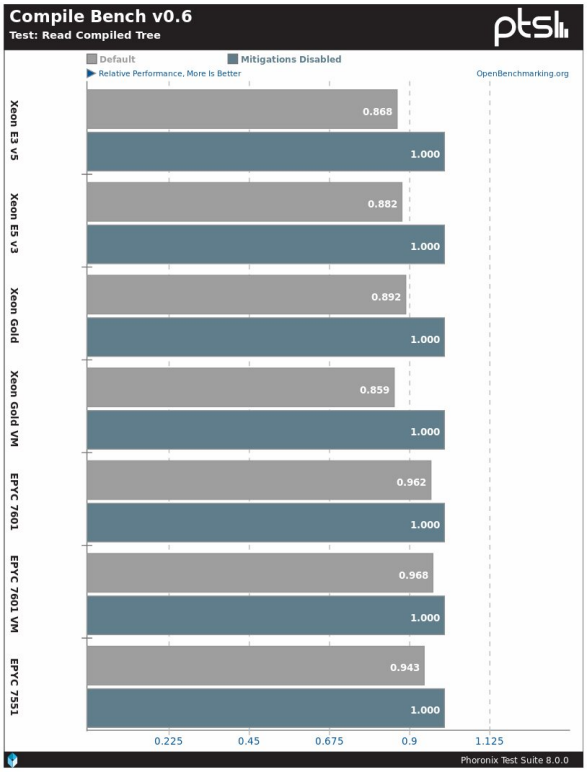

Рисунок 1.6 - Результати тестів Phoronix Test Suite при компіляції абстрактного синтаксичного дерева на різних процесорах [7]

В реальних задачах, таких як компіляція ядра Linux різниця буде складати приблизно 2%.

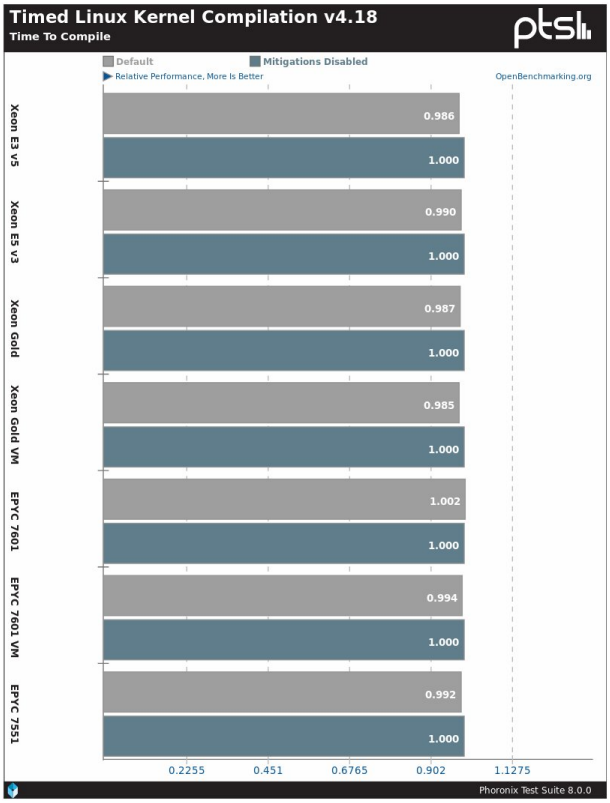

Рисунок 1.7 - Результати тестів Phoronix Test Suite при компіляції ядра Linux на різних процесорах [7]

Отже видно, що існуючі результати дійсно частково вирішують проблему Spectre. Платою за це є втрати у швидкодії системи. Саме тому є сенс дослідити закон зміни часу відгуку данних від частоти оновлення кешу, щоб отримати максимальну швидкодію на захищеній обчислювальній системі.

Висновки до розділу 1

В данному розділі було розглянуто механіку експлуатування side-channel атак, класи вразливих процесорів, існуючі методи захисту від атак на вразливості сімейства Spectre та вплив таких атак на загальну швидкодію обчислювальних систем.

Вразливості класу Spectre виникли через мікроархітектурні прорахунки, тому охоплює значну кількість сучасних процесорів. Виходячи з цього на данний момент неможливо позбутися стороннього каналу витоку інформації, не втративши при цьому швидкодію.

Позбутися стороннього каналу витоку інформації за часом без втрати швидкодії можливо лише зі змінами в мікроархітектурі.

2 Збір статистичних данних

2.1 Методика отримання емпіричних данних

В ході роботи були отримані емпіричні данні, а саме кількість тактів процесору в залежності від частоти очищення кешу.

1. Інструкція «rdtscp»

Ці данні вдалось зібрати за допомогою інструкції rdtscp (Додаток А). Інструкція «rdtscp» – це асемблерна інтсрукція, що зчитує лічильник маркеру часу (TSC - Time Stamp Counter) та повертає у регістрах «EDX» - «EAX» 64-бітну кількість тактів з моменту останнього збросу процесору. Таким чином, можна порахувати час роботи будь-якого набору інструкцій, зчитавши значення лічильника маркеру часу до та після виконання цього набору та віднявши значення, отримане за допомогою «tdtscp» до виконання від значення лічильника після виконання. Такі вимірювання є більш точними та менш ресурсоємними на відміну від функцій, що надаються інтерфейсами різних операційний систем.

За допомогою цієї інструкції вдалось зібрати данні для різних частот очищення кешу по кожній літері з фрази, «The Magic Words are Squeamish Ossifrage.» яка є ключем та на яку власне і проводилась атака експлойту. По кожній букві з цієї фрази було отримано кількість процесорних тактів, що знадобились для атаки при різних частотах очищення кешу. Далі процес збору експериментальних данних буде описано більш детально.

2. Інструкція «_mm_clflush»

SSE інструкції «_mm_clflush» з бібліотеки «initrinsic.h», що поставляється комнанією Intel. Ця бібліоткека дає доступ до всіх можливих низькорівневих операцій, що підтримують процесори виробництва Intel. В данному випадку інструкція “_mm_clflush” дає змогу обчистити певне значення, що можливо було закешоване з кешу усіх рівнів.

SSE – це частина технології SIMD інструкцій. SIMD інструкції дозволяють забезпечити паралелізм на основі данних. SIMD обчислювальні системи

складаються з одного процесору (керуючого модулю), що називається контроллером та декількох модулей обробки данних, що називаються процесорними елементами. Керуючий модуль приймає, аналізує та виконує команди. Якщо в команді зустрічаються данні, контроллер розсилає на усі процесорні елементи команду, та ця команда виконується на декількох чи на усіх процесорних елементах. Кожен процесорний елемент має свою власну модель пам'яті для зберігання данних. Однією з таких переваг вважається те, що в такому випадку більш ефективно реалізована логіка обчислень. До половини логічних інструкцій звичайного процесору пов'язано з керуванням виконання машинних інструкцій, а інша частина відноситься до роботи з внутрішньою пам'яттю процесору та виконання арифметичних операцій. У SIMD процессорах керування виконується контроллером, а «арифметика» - процесорним елементам.
Таким чином, спеціальний набір SSE інструкцій дає змогу зручно та автоматично працювати з кешами процесору.

2.2 Збір статистичних данних про середнє значення відгуку данних з ОЗП

Для коректної оцінки подальших експериментальних данних в нашому дослідженні необхідно отримати середнє значення кількості тактів процесору що йдуть на отримання наступного байту данних. Ми будемо використовувати це значення як опорне, адже воно є оптимальним, хоча і не найкращим, з точки зору швидкодії та коректним з точки зору унеможливлення побудов стороннього каналу, адже якщо при будь-яких послідовностях інструкцій швидкість відгуку ОЗП буде завжди середньою, ми не зможемо сказати точно, які саме данні були закешовані.

Для збору данних була використана вищеописана інструкція «rdtscp». Було виконано 100000 ітерацій на кожній з яких було заміряно середній отримання випадкового елементу масиву. Випадково обирались ітерації, на яких основний потік виконання зупинявся на 1 секунду. Усі ці дії для уникнення різних

оптимізацій та кешування. Отримавши статистичні данні, ми побудували графік розподілу кількості наших тактів, що були задіяні на кожній ітерації (Додаток В):

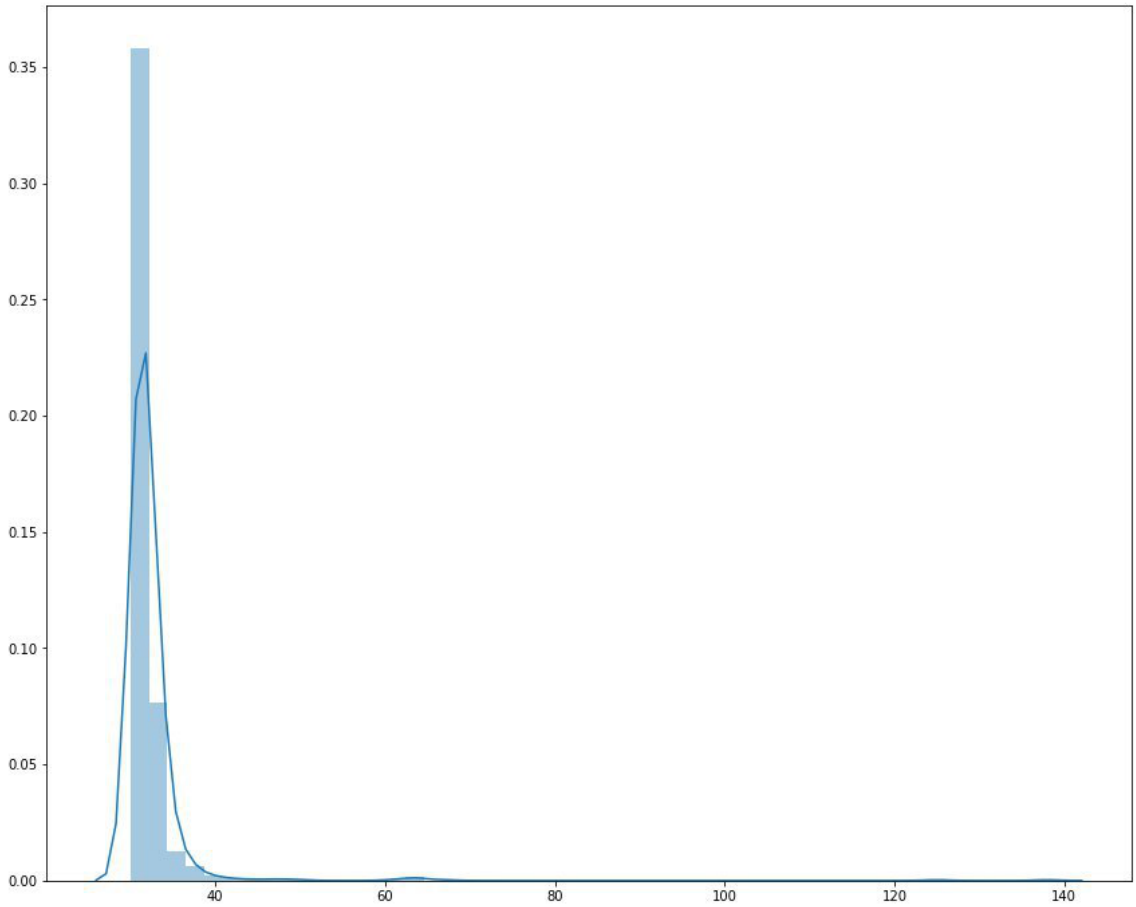

Рисунок 2.2 - Розподіл кількості тактів процесору, що йдуть на отримання данних з ОЗП. Вісь x - такти процесору, вісь y - густина ймовірності

Графік має форму нормального розподілу. В середньому за статистикою маточікування велечини тактів склало 30 тактів за один доступ до випадкового елеманта масиву. Таким чином можна взяти це значення за опорне у подальшому дослідженні.Дисперсія ск

2.3 Збір статистичних данних про час відгуку ОЗП в залежності від частоти очищення кешу

Отже, для дослідження був обраний проміжок частот очищення кешу від 1 до 100. Тобто, при частоті очищення кешу 1 кожна комірка пам’яті з данними

очищквалася, при частоті 2 - очищувалася кожна друга комірка з данними і так далі, в решті-решт очищувалася лише кожна 1000-на комірка з данними. Кількість процесорних тактів записувалася в окремий JSON файл.

Далі, для коджного окремої частоти було підраховано середню кількість процесорних тактів, що були витрачені на отримання конкретної літери з фрази, на яку було виконано атаку. Отже було отримано певну статистику, на основі якої було побудовано наступний графік:

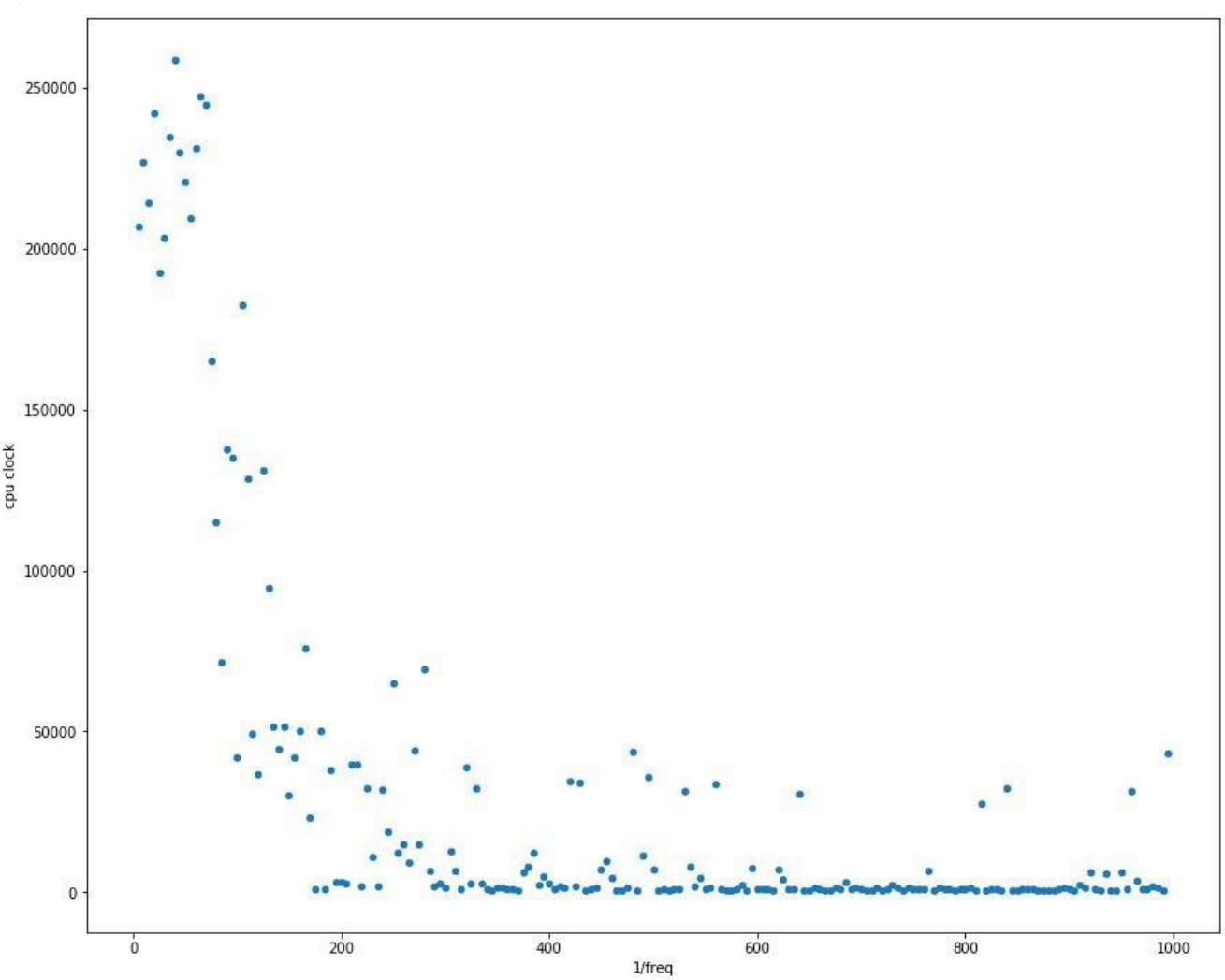

Рисунок 2.1 - Візуалізація отриманої статистики Вісь x - період оновлення кешу, вісь y - кількість процесорних тактів

На малюнку видно, що данні підпорядковуются певній закономірності. Отже, використовуючи данну статистику, можна створити апроксимацію, за допомогою

якої можна провести аналіз та розробити оптимальні методики очищення кешу задля запобігання витоку данних сторонніми каналами за часом.

Висновик до розділу 2

Отже у цьому розділі було отримано статичтичні данні середнього часу відгуку ОЗП та данні про час відгуку ОЗП при очистці кешу з різною частотою.

Статичтичні данні середнього часу відгуку ОЗП нормально розподілені та мають маточікування 30 тактів за один доступ до випадкового елемента масиву.

3 Обробка статистичних данних

3.1 Запропоновані моделі апроксимації отриманих експериментальних данних

Для формування апроксимації отриманих експериментальних данних було запропоновано декілька математичних моделей:

1. Лінійна модель
2. Гіперболічна модель
3. Поліноміальна модель

Опишемо детальніше математичний апарат кожної окремої моделі та отримані результати його використання на наших данних.

3.2 Лінійна модель

У статистиці лінійна регресія — це метод моделювання залежності між скаляром y та векторною (у загальному випадку) змінною X. У разі, якщо змінна X також є скаляром, регресію називають простою.

При використанні лінійної регресії взаємозв'язок між даними моделюється за допомогою лінійних функцій, а невідомі параметри моделі оцінюються за вхідними даними. Подібно до інших методів регресійного аналізу лінійна регресія повертає розподіл умовної імовірності y в залежності від X, а не розподіл спільної імовірності y та X, що стосується області мультиваріативного аналізу.

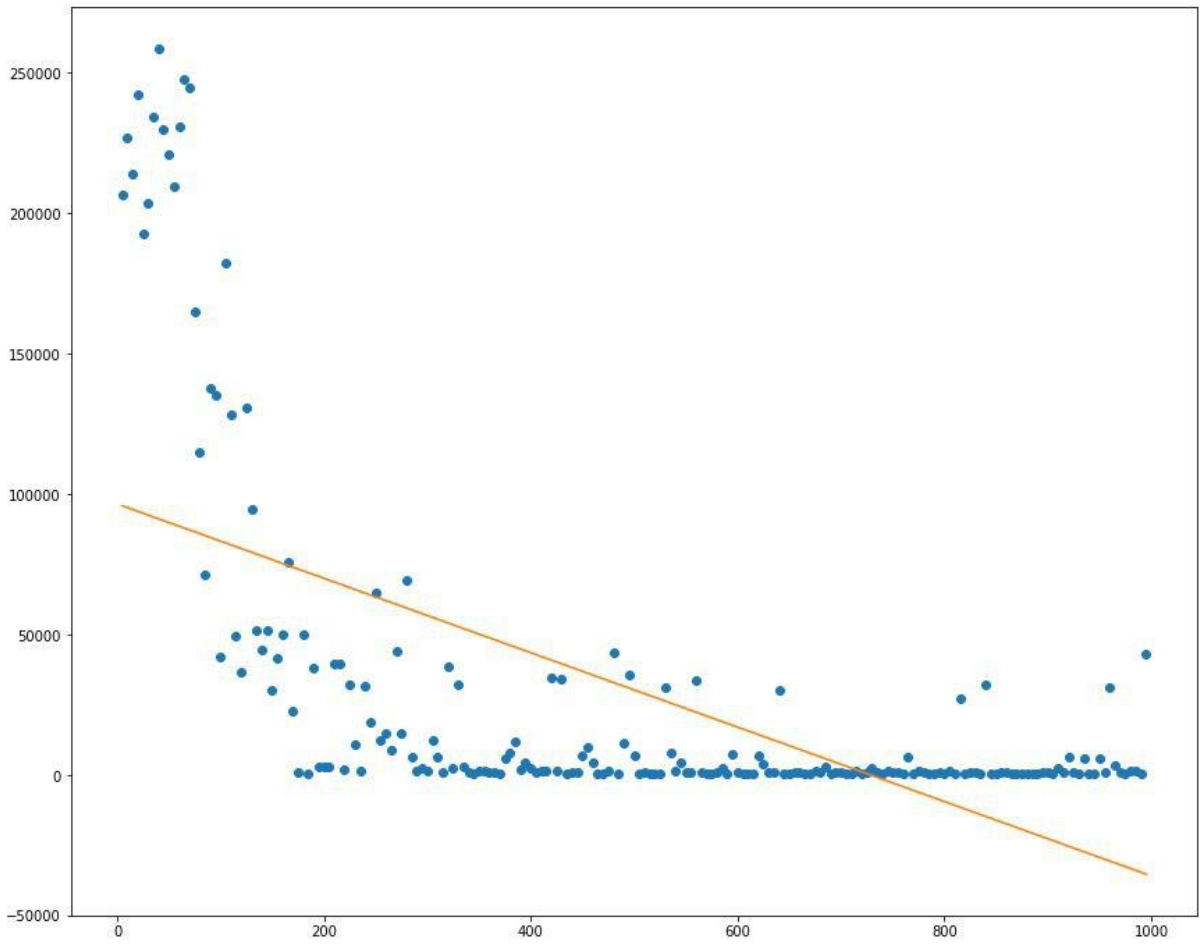

Рисунок 3.2 - Графік апроксимації лінійною функцією. Вісь x - період оновлення кешу, вісь y - кількість процесорних тактів

При розрахунках параметрів моделі лінійної регресії зазвичай застосовується метод найменших квадратів (МНК), але також можуть бути використані інші методи. Так само метод найменших квадратів може бути використаний і для нелінійних моделей. Тому МНК та лінійна регресія хоч і є тісно пов'язаними, але не є синонімами.

Загалом лінійна регресійна модель визначається у вигляді:

$$y = a_0 + a_1 * x + \cdots + a_k * x_k + \varepsilon_i \quad (3.1)$$

де a_0 - математичне очікування залежної змінної y_i, коли змінна x_i дорівнює нулю; a_1 - очікувана зміна залежної змінної y_i при зміні x_i на одиницю (цей коефіцієнт підбирають таким чином, щоб величина $½\Sigma(y_i - \hat{y}_i)^2$ була мінімальна - це так звана

«функція нев'язки»); ε_i - випадкова помилка. При цьому коефіцієнти a_1 і a_0 можна виразити через коефіцієнт кореляції Пірсона, стандартні відхилення і середні значення змінних x і y:

$$\hat{a}_1 = \frac{cor(y,x)\sigma_y}{\sigma_x} \tag{3.2}$$

$$\hat{a}_0 = \bar{y} — \hat{a}_1\bar{x} \tag{3.3}$$

Щоб модель була коректною, необхідно виконання умов Гаусса-Маркова, тобто

1. мають виконуватись наступні умови:
2. Модель данних правильно специфіковано
3. Усі X детерміновані, але не всі рівні між собою
4. Помилки не носять систематичного характеру
5. Дисперсія помилок однакова та рівна деякій $б^2$
6. Помилки не корельовані

Таким чином для коректного використання данної моделі помилки мають бути гомоскедастичні з нульовим математичним очікуванням.

Корректність лінійної моделі в нашому випадку можна перевірити за допомогою лінійного коефіцієнту кореляції Пірсона, що розраховується за такою формулою:

$$r_{xy} = \frac{cov_{xy}}{\sigma_y\sigma_x} = \frac{\Sigma(x-\bar{x})(y-\bar{y})}{\sqrt{\Sigma(x-\bar{x})^2\Sigma(y-\bar{y})^2}} \tag{3.4}$$

де $\bar{x}$ та $\bar{y}$ - середні значення виборок. Таким чином, чим ближче величина коефіцієнту лінійної кореляції до значення 1 тим більш ймовірно, що лінійна

модель коректно апроксимує емпіричні данні. Розглянемо отримані значення коефіцієнтів Пірсона для наших данних (Додаток С):

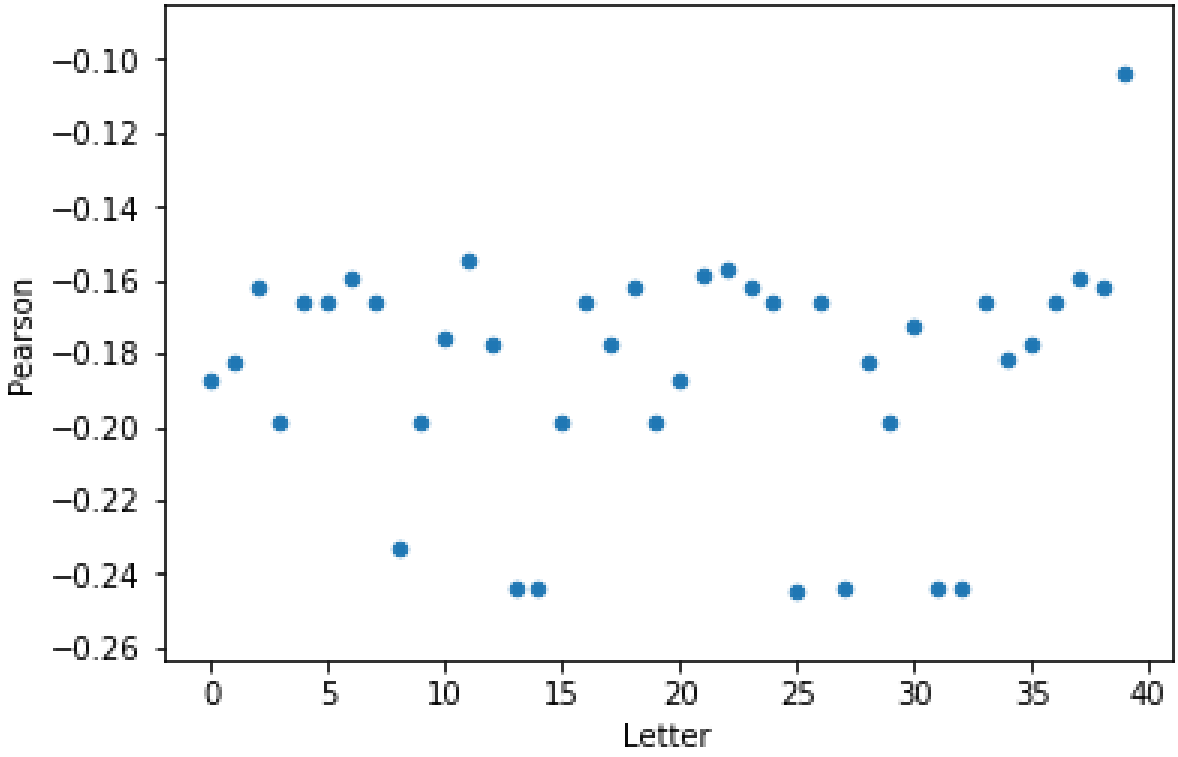

Рисунок 3.2 - Графік лінійної кореляції між буквами ключової фрази "the magic words are squeamish ossifrage" та середнею кількістю процесорних тактів для їх отримання.

Як бачимо, значення коефіцієнтів в середньому знаходяться в межах -0.26 та -0.1, що свідчить про те, що лінійна модель малопридатна для апроксимації наших данних.

Оскільки графік отриманих данних нагадує гіперболічний закон — було вирішено спробувати апроксимувати отримані емпіричні данні за допомогою нелінійної регресії.

3.3 Гіперболічна модель

Першою спробою апроксимувати данні нелінійною регресією була гіперболічна регресія. Опишемо механізм побулови гіперболічнох регресії.

Як відомо, гіперболічний закон описується формулою:

$$y = a + \frac{b}{x} \tag{3.5}$$

Де a та b — шукані коефіцієнти. Для отримання коефіцієнтів скористаємося формулою для визначення коефіцієнтів a та b:

$$F(a,b) = \sum_{i=0}^{n} \left(y_i - \left(a + \frac{b}{x_i} \right) \right)^2 \tag{3.6}$$

Звідки маємо:

$$\begin{cases} \dfrac{\partial F(a,b)}{\partial a} = 0 \\ \dfrac{\partial F(a,b)}{\partial b} = 0 \end{cases} \tag{3.7}$$

Таку сисетму можна розв`язати наступним чином:

$$a = \frac{\Sigma y_i * \Sigma \left(\frac{1}{x_i}\right)^2 - \Sigma \left(\frac{1}{x_i}\right) * \Sigma \left(\frac{y_i}{x_i}\right)}{n\Sigma \left(\frac{1}{x_i}\right)^2 - \Sigma \left(\frac{1}{x_i}\right) * \Sigma \left(\frac{1}{x_i}\right)}, i = [1, n] \tag{3.8}$$

Використовуючи вище наведені формули ми отримали наступні результати апроксимації гіперболою отриманих данних (Додаток D):

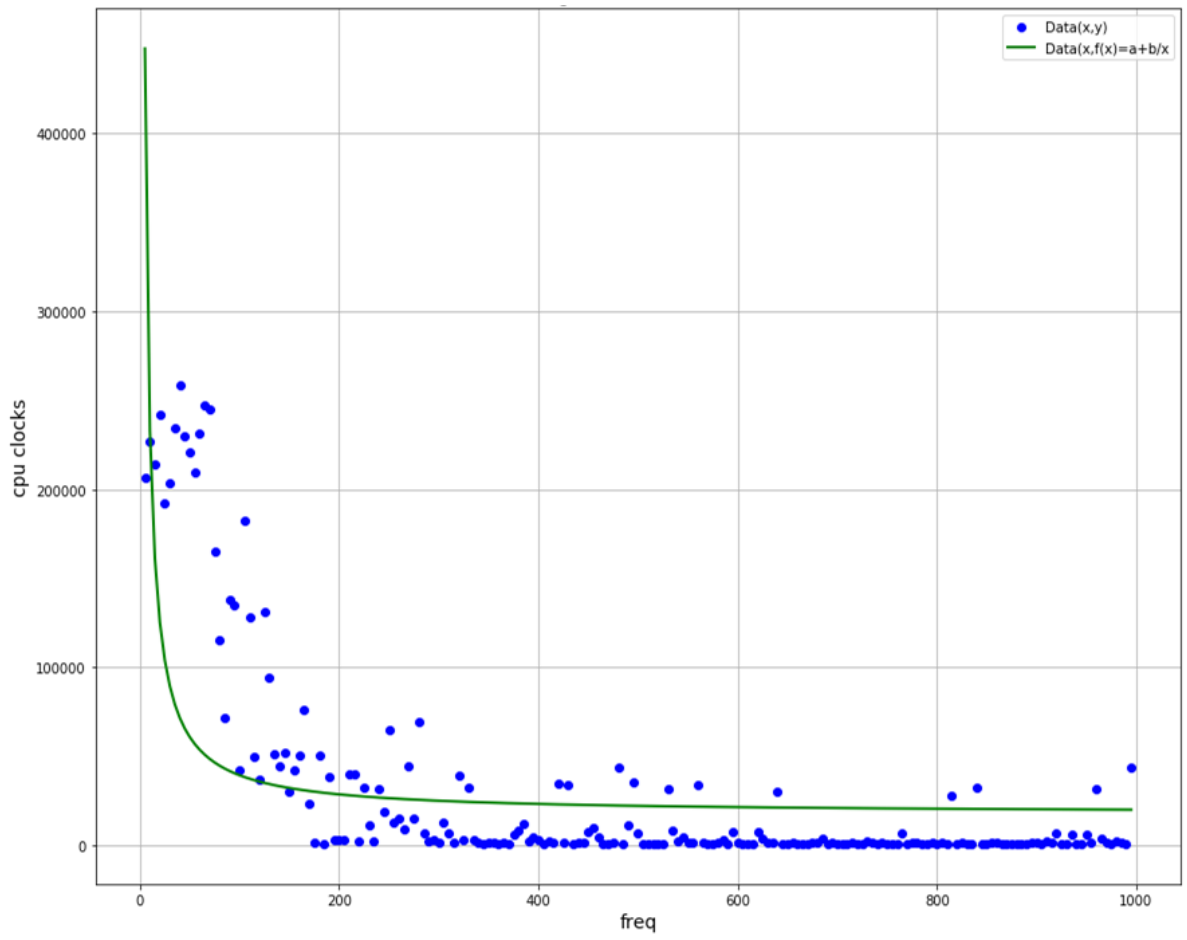

Рисунок 3.3 - Графік апроксимації гіперболічною функцією. Вісь x - період оновлення кешу, вісь y - кількість процесорних тактів

В результаті обчислень ми отримали гіперболу з наступними параметрами:

a = 17754.029

b = 2150170.074

Для оцінки отриманих значень можна скористатись середньоквадратичною похибкою апроксимації, яка обчислюється за формулою:

$$\delta = \frac{\Sigma\left(y_i - a - \frac{b}{x_i}\right)}{n * \Sigma y_i} * 100, i = [1, n] \quad (3.9)$$

В нашому випадку середньоквадратична девіація склала:

δ = 0.5031

З данного результату видно, що гіперболічна апроксимація дала непогану точність.

3.4 Полініноміальна модель

Нарешті спробуємо застосувати більш універсальну — поліноміальну апроксимацію.

Розглянему задачу поліноміальної апроксимації у загальному вигляді. Рівняння поліному у загальному вигляді:

$$y = a_{n-1}x^n + a_{n-2}x^{n-1} + \cdots + a_1 x + \zeta \tag{3.10}$$

де a_i – шукані параметри, ζ −вільний член. Знайдемо за методом найменших квадратів шукані параметри данної регресії:

$$S = \sum_{i=1}^{n} (\bar{y}_i - y_i)^2 \to min \tag{3.11}$$

Де y_i – теоретичні значення полінома () в точках x_i . Підставивши одну формулу в іншу – маємо:

$$S = \sum_{i=1}^{n} \left(\sum_{j=0}^{k} b_j x_i^j \ - y_i)\text{^}2 \right)^2 \to min \tag{3.12}$$

На основі необхідної умови екстремуму функції (k + 1) від змінних
$S = S(a_0, a_1 , .., b_{n-1})$ прирівнюємо до 0 її часткові похідні, таким чином:

Поділивши ліву та праву частину рівності на 2 отримаємо:

$$\sum_{i=1}^{n} x_i^p\left(a_0 + a_1 x_i + a_2 x_i^2 + \cdots + a_k x_i^k\right) - \sum_{i=1}^{n} x^p y_i = 0, p = [0, k] \qquad (3.13)$$

Розкриваючи отриманий вираз, перенесемо у кожному p-тому вирузі останній доданок вправо та поділимо обидві частини на n. В результаті у нас вийшло

(k + 1) виразів, що утворють систему лінійних рівнянь відносно a_p

$$f(x) = \begin{cases} a_o + a_1\bar{x} + a_2\bar{x}^2 + \cdots + a_k\bar{x}^k = y \\ a_o\bar{x} + a_1\bar{x}^2 + a_2\bar{x}^3 + \cdots + a_k\bar{x}^{k+1} = xy \\ a_o\overline{x^2} + a_1\bar{x}^3 + a_2\bar{x}^4 + \cdots + a_k\bar{x}^{k+2} = x^2y \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots. \\ a_0\bar{x}^k + a_1\bar{x}^{k+1} + a_3^{k+2} + \cdots + b_k x^{2k} = \bar{x}^k y \end{cases} \qquad (3.14)$$

Систему (3.14) можна представити у матричному вигляді

$$A = \begin{pmatrix} 1 & \bar{x} & \bar{x}^2 & \ldots & \bar{x}^k \\ \bar{x} & \bar{x}^2 & \bar{x}^3 & \ldots & \bar{x}^{k+1} \\ \bar{x}^2 & \bar{x}^3 & \bar{x}^4 & \ldots & \bar{x}^{k+2} \\ \ldots & \ldots & \ldots & \ldots & \ldots \\ \bar{x}^k & \bar{x}^{k+1} & \bar{x}^{k+2} & \ldots & \bar{x}^{2k} \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ \ldots \\ a_k \end{pmatrix} = \begin{pmatrix} y \\ \bar{x}y \\ \bar{x}^2y \\ \ldots \\ \overline{x^k y} \end{pmatrix} \qquad (3.15)$$

Таким чином, вирішивши данне матричне рівняння відносно вектору коефіцієнтів $S(a_0, a_1, .., a_k)$

Степінь точності поліноміальної апроксимації можна оцінювати за двома критеріями: середньоквадратична похибка та коефіцієнт детермінації.

Коефіцієнт детермінації (позначається як R^2) — статистичний показник, що використовується в статистичних моделях як міра залежності варіації залежної змінної від варіації незалежних змінних. Вказує наскільки отримані спостереження підтверджують модель.

Коефіцієнт детермінації визначається наступним чином:

$$R^2 = 1 - \frac{V(y|x)}{V(y)} = 1 - \frac{\delta^2}{\delta_y^2} \tag{3.16}$$

Де δ — умовна дисперсія залежної змінної.

Стандартне відхилення або середнє квадратичне відхилення, позначається грецькою літерою сигма σ або латинською літерою S — у теорії ймовірності і статистиці найпоширеніший показник розсіювання значень випадкової величини відносно її математичного сподівання. Має ту ж розмірність, що і випадкова величина.

Середнє квадратичне відхилення так само, як і середнє лінійне відхилення, показує, на скільки в середньому відхиляються конкретні значення ознаки від середнього їх значення. Середнє квадратичне відхилення завжди більше середнього лінійного відхилення. Мале значення стандартного відхилення вказує, що дані точоки скупчені ближче до середнього значення (математичного сподівання) вибірки, в той час як великі значення стандартного відхилення вказують, що точки розподілені в більш широкому діапазоні значень.

Наприклад, межа похибки для даних опитування визначається за допомогою розрахунку очікуваного стандартного відхилення в результатах за умови, якби те саме опитування було проведене декілька разів. Таке виведення стандартного відхилення часто називають «стандартною похибкою» оцінювання або «стандартною похибкою середнього», якщо мова йде про середнє. Вона визначає стандартне відхилення усіх середніх значень, отримані для даної генеральної сукупності на основі вибірки.

Стандартне відхилення вибірки може бути обчислено за формулою:

$$\sigma = \sqrt{\sigma^2} = \sqrt{\frac{1}{N}\Sigma(x_i - \bar{x})^2} \tag{3.17}$$

Ці два показники неможливо максимізувати одночасно, оскільки при більш точній апроксимації на вже існуючих данних можна втратити можливість математичної моделі вдало апроксимувати данні в загальному випадку, і навпаки: ми спроби узагальнити модель призводить до втрати точності моделі. Цю закономірність можна побачити з графіку:

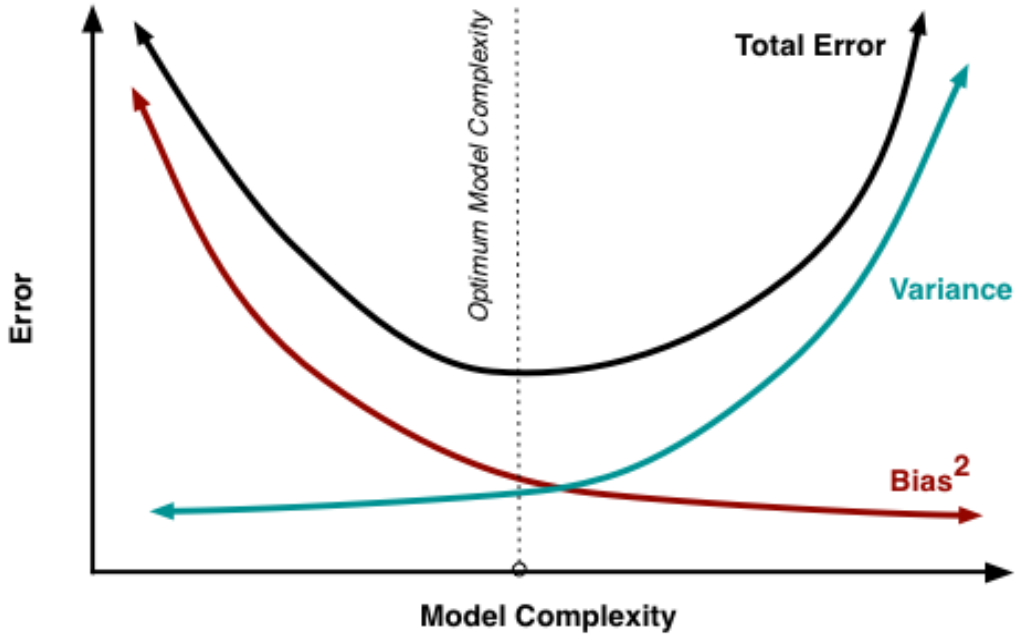

Рисунок 3.4 – графік залежності кількості помилок від складності математичної моделі. Червоний графік – величина R^2, синій – графік варіцій, чорний – сукупна величина помилок

Тому потрібно притримуватись оптимально співвідношення між цими двома параметрами.

В результаті обробки данних та спроб апроксимації поліномами різних степеней було отримано наступну таблицю

Таблиця 3.1 – Коефіцієнти стандарної похибки та детермінації для різних поліноміальний апроксимацій

Степінь поліному	Коефіцієнт детермінації	Стандартна похибка
1	0.3803609827384328	48534.00770735291
2	0.6905313377408699	34299.28882665958
3	0.8272926696697069	25623.128939409802
4	0.8748	21814.49812443201
5	0,876	21671,57
6	0.855757017063888	23416.615452559472

Отже було знайдено поліном з оптимальним співвідношенням стандартного відхилення та коефіцієнтом детермінації. Таким поліномом виявився поліном п’ятої степені.

Отриманий поліном має наступний вигляд:

$$y = 2{,}135 * 10^{-9}x^5 + 8{,}217 * 10^{-6}x^4 - 1{,}173 * 10^2x^3 + 7{,}798x^2 - 2{,}4221 * 10^3x \quad (3.19)$$

Стандартне відхилення склало: 21671.57

Коефіцієнт детермінації: 0.876

На рисунку зображено графік знайденої функції, що відображає матетматичну модель отриманих данних (Додаток Е).

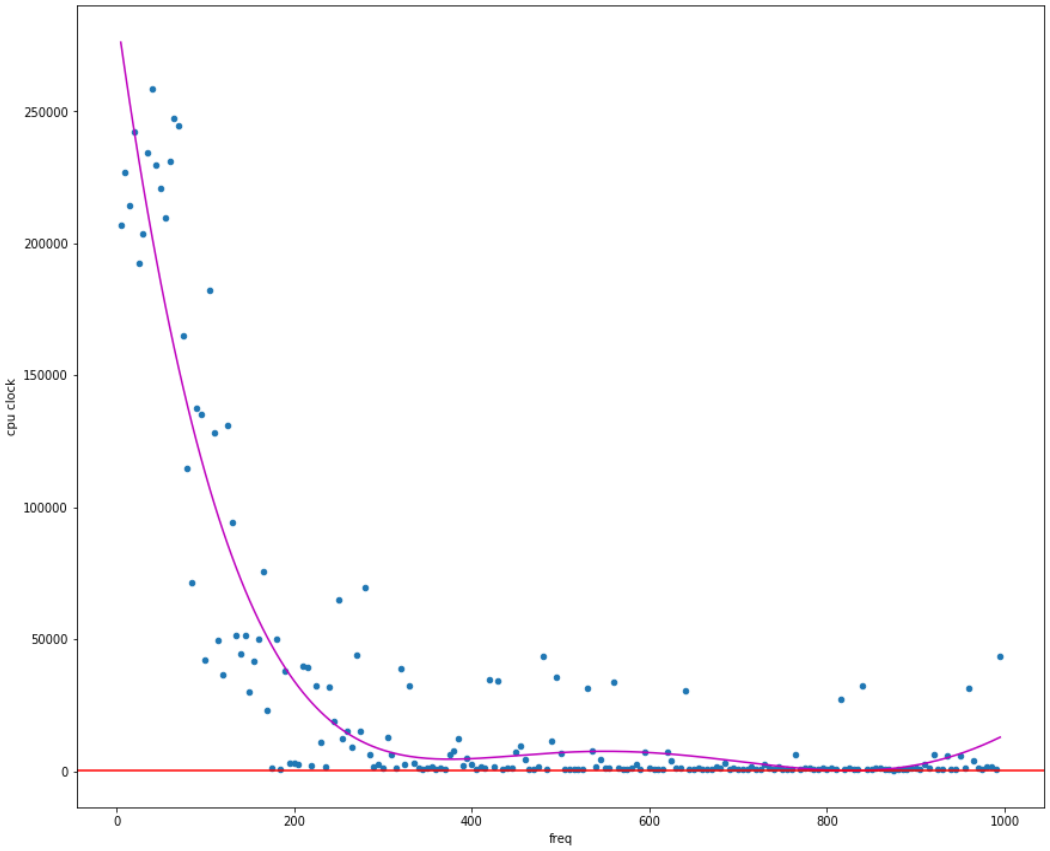

Рисунок 3.5 - Графік апроксимації поліноміальною функцією. Вісь x - період оновлення кешу, вісь y - кількість процесорних тактів

Червноною лінією зображено на графіку середню кількість тактів, що була знайдена у розділі 2.

Висновки до розділу 3

У цьому розділі було розглянуто 3 математичні моделі отриманих експериментальних данних: лінійна, шіперболічна та поліноміальна.
Найбільш ефективною з точки зору мінімізації помилок виявилася поліноміальна модель.

В свою чергу серед інших поліномів – поліном 5 степеня виявив найкращі показники співвідношення стандартної похибки та коефіцієнта детермінації

4 Практичне застосування отриманих результатів

4.1 Знаходження оптимальних частот очищення кешу

Для знаходження оптимальних частот очищення кешу було обрано певний задовільний проміжок часу відгуку ОЗП. Цей проміжок було знайдено на основі результатів дослідження середніх значень відгуку ОЗП.

Інакше кажучи за ідеальне значення часу відгуку ми обрали з маточікування цієї величини, а за припустипе відхилення – значення, близьке до дисперсії цієї величини, а саме ± 100 тактів. Ми розраховуємо, що при середньому відгуку ОЗП в межах цього часу ми матимемо оптимальну швидкодію та захист від стороннього каналу витоку данних.

Розглянемо отриманий проміжок частот, що відповідає нашим вимогам (Додаток F):

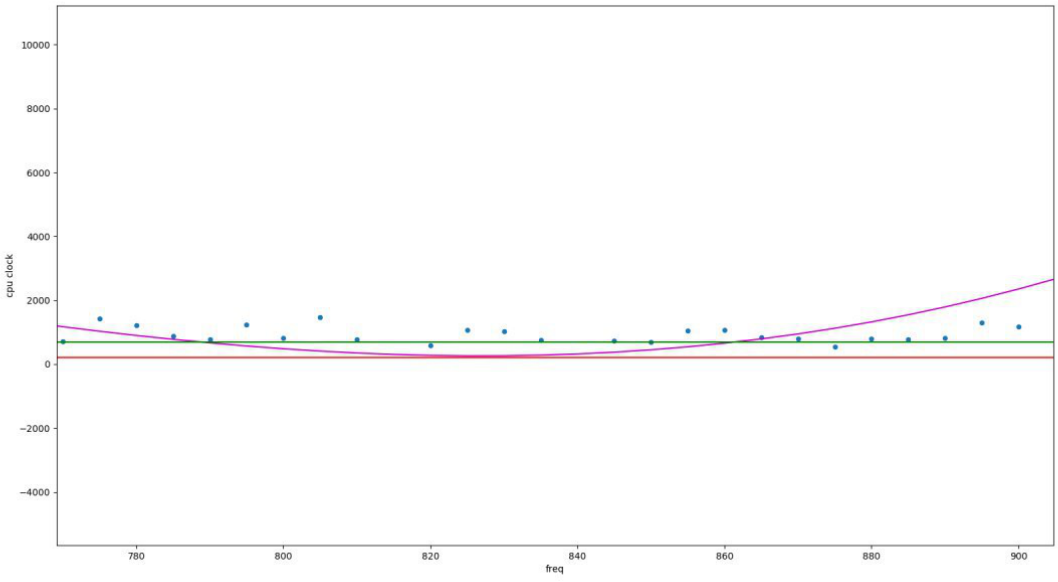

Рисунок 4.1 - Залежність швидкості відгуку ОЗП від періоду очищення кешу

Аналітично ми отримали результат, що наш проміжок лежить у межах 790 тактів та 865 тактів, що також видно з вищенаведеного графіку. Отже при очищенні

кешу один раз на 790 – 865 разів доступу до пам’яті ми матимемо оптимальні параметри

4.2 Практичне застосування отриманих результатів

Хоча на сьогодні докладається значних зусиль задля програмного та апаратного захисту існуючих обчислювальних систем, все ж не можна сказати що вдалося повністю позбутися усіх векторів атак на спекулятивне виконання коду. Протягом усього 2018 та початку 2019 року постійно з’являютсья публікації про нові методи експлуатації вразливостей сімейства Spectre. З іншого боку, далеко не всі данні, що обробляються на вразливих процесорах є критичними. З іншого боку всі способи захисту від атак на спекулятивне виконання коду, в тому числі, запропоновані у цій роботі, призводять до сповільнень в роботі чіпів. Таким чином було б зручно отримати можливість відокремити обробку критичних данних, тобто захищати критичні данні та мати звичну швидкодію при роботі обчислювальної системи, оскільки відсоток саме критичних данних, що обробляються в системі, таких як раундові ключі шифрування, деякі персональні данні, дескриптори об’єктів ядра операційної системи та адреси комірок пам’яті, досить невеликий.

Для досягнення цієї цілі можна за допомогою препроцесору компіляторів мов C та C++ додавати очищення кешу в місцях оборки критичних данних.
Наприклад такий код:

```
#pragma spectre_mitigation
for (int i = 0; i < 100; ++i) {
   x = array1[i];
}
```

після обробки препроцесором можна перетворити у код такого вигляду:

```
#pragma spectre_mitigation
for (int i = 0; i < 100; ++i) {
   if (!(i % optimal_freq))
         _mm_cflush(array[i]);
   x = array1[i];
}
```

Задля ефективної роботи запропонованої техніки захисту потрібно завчасно визначити параметри системи, та виходячи з них обчислити задовільний проміжок частот очищення кешу так, як це було описано вище.

Список параметрів які необхідно завчасно визначити при компіляції для роботи на певній визначеній системі:

1. Середній час відгуку системи
2. Задовільний проміжок відхилення від середнього часу

Саме за допомогою цих двох параметрів можливо визначити задовільний інтервал частоти. Після отримання необхідних данних потрібно перерахувати коефіцієнти регресії та отримати новий проміжок допустимих частот.

Також потрібно визначити за допомогою якої інструкції можна очищувати кеш, адже наведена та описана у цій роботі інструкція «rtdscp» є платформоспецифічною, хоча її аналоги є і в інших поколіннях процесорів Intel та у чіпах виробництва AMD чи інших вендорів.

4.3 Висновки до розділу

У цьому розділі за допомогою математичної моделі, знайденої у розділі 3, було отримано оптимальний проміжок частот очищення кешу.

Використовуючи отриману техніку очищення кешу було запропоновано метод практичної реалізації такої техніки на прикладі мови C.

Висновки

Вразливості спекулятивного виконання коду стали серйозним приводом до занепокоєння багатьох фахівців та ентузіастів в індустрії кібербезпеки. Особливу небезпеку такі вразливості становлять через те, що їх фундамент лежить у площині мікроархітектурних рішень. Також саме з цієї причини вищезгаданих вразливостей не так просто позбутись.

Ймовірніше за все, так чи інакше, канали стороннього витоку інформації так і залишаться існувати у всіх сучасних архітектурах та, ймовірно, у всіх наступних, які будуть логічним продовженням існуючих напрацювань.

Хоча повністю захиститись від атак класу Spectre і неможливо, все ще можна ускладнити умови експлуатації цих вразливостей. Задля цього такі гіганти індустрії як Intel, ARM, Google, Microsoft продовжують виправляти свої продукти, як зі сторони програмного забезпечення, так зі сторони апаратної частини.

З іншого боку усі запропоновані виправлення Spectre так чи інакше уповільнюють роботу процесорів.

Виходячи з цього доцільність оптимізації технік захисту від атак на вразливості спекулятивного виконання коду є більш ніж очевидною.

У цій роботі були запропоновані оптимізації наступного характеру:

1. Оптимізації швидкодії при використанні механізмів захисту
2. Оптимізації за місцем використання механізмів захисту. Цю концепцію можна виразити дуже просто: використовувати захист від Spectre тільки там, де це справді необхідно.

Запропонований метод захисту від Spectre заснований на очистці кешу. При очистці кешу данні завжди завантажуватимуться з оперативної пам’яті, а не з кешу, що дещо довше, тому неможливо визначити, що ці данні нещодавно оброблялись, адже інакше вони мали б бути у кеші та були б завантажені швидше. З іншого боку, отримання данних з кешу значно пришвидшує роботу обчислювальної системи, тому раціональніше не відмовлятися від кешування вцілому, а використовувати його з певною частотою, яка з однієї сторони не дає змогу визначити, які данні

були закешовані, а з іншого боку дає використовувати кешування для оптимізації швидкодії.

Другий пункт оптимізації полягає у використанні розумних механізмів захисту, які захищають тільки ті данні, які необхідно захищати. Такий механізм можна імплементувати на рівні компіляторів, які застосовуватимуть механізм захисту при компілюванні двійкових файлів під конкретну платформу.

Це ще одна оптимізація, яка дозволяє пришвидшити роботу системи та з іншого боку ускладнити експлуатацію вразливостей типу Spectre.

У данній роботі було проведено тестування такого підходу на невеликому застосунку написаного мовою C.

Доповлення коду необхідними інструкціями здійснювалось за допомогою розширення макровизначенням, яке додавалось в потрібних місцях до коду вразливого застосунку за допомогою препроцесору компілятору GCC.

Данні були зібрані за допомогою допрацьованого оригінального експлойту Spectre variant 2.

Аналіз отриманих данних було зроблено ща допомогою мови Python та бібліотек для аналізу данних Pandas та Numpy. Апроксимація проводилась алгоритмами описаними у розділі 3 з використанням бібліотеки Scikit-learn. Аналіз розподілу величин середнього відгуку данних з ОЗП було зроблено за допомогою бібліотеки Seaborn.

Дослідження проводились на оперційній системі GNU/Linux, версії ядра 5.0.9 (дата випуску: 20 квітня 2019 року), з процесором Intel Core I5 -7200U.

Використовуючи запропоновану техніку очищення кешу, вдалося закрити сторонній канал витоку данних за часом, на основі якого працював доказовий експлойт оригінальних дослідників вразливості при незначних втратах швидкодії. Більш точні данні отриманого приросту в швидкодії в порівнянні зі вже існуючими методами захисту потребують масштабного тестування на різних комбінаціях операційних систем та апартних платформ.

Перелік джерел посилань

[1] Мария Нефедова. “Обнаружены семь новых вариаций атак на Meltdown и Spectre” [Електронний ресурс] – Режим доступу до ресурсу: https://xakep.ru/2018/11/15/7-more-meltdown-and-spectre/

[2] Axelle Apvrille. “Does malware based on Spectre exist?” [Електронний ресурс] – Режим доступу до ресурсу: https://www.virusbulletin.com/virusbulletin/2018/07/does-malware-based-spectre-exist/

[3] “About speculative execution vulnerabilities in ARM-based and Intel CPUs” [Електронний ресурс] – Режим доступу до ресурсу: https://support.apple.com/en-us/HT208394

[4] Paul Kocher, Jann Horn, Anders Fogh, Daniel Genkin, Daniel Gruss, Werner Haas, Mike Hamburg, Moritz Lipp, Stefan Mangard, Thomas Prescher, Michael Schwarz, Yuval Yarom “Spectre Attacks: Exploiting Speculative Execution”. Google Project Zero. 2018. С 1-19.

[5] Paul Kocher, Jann Horn, Anders Fogh, Daniel Genkin, Daniel Gruss, Werner Haas, Mike Hamburg, Moritz Lipp, Stefan Mangard, Thomas Prescher, Michael Schwarz, Yuval Yarom “Meltdown: Reading Kernel Memory from User Space”. Google Project Zero. 2018. С 1-18.

[6] Claudio Canella, Jo Van Bulck, Michael Schwarz, Moritz Lipp, Benjamin von Berg, Philipp Ortner, Frank Piessens, Dmitry Evtyushkin, Daniel Gruss. Graz University of Technology, imec-DistriNet, KU Leuven, College of William and Mary. 2018. С 1-16.

[7] Влияние защиты от Spectre, Meltdown и Foreshadow на производительность Linux 4.19 [Електронний ресурс] – Режим доступу: https://habr.com/ru/post/421931

[8] The Intel Intrinsics Guide [Електронний ресурс] – Режим доступу до ресурсу:

https://software.intel.com/sites/landingpage/IntrinsicsGuide/#techs=SSE,SSE2,SSE3&expand=672

[9] Intel® 64 and IA-32 Architectures Software Developer's Manual [Електронний ресурс] – Режим доступу: https://www.intel.com/content/www/us/en/architecture-and-technology/64-ia-32-architectures-software-developer-vol-2a-manual.html

[10] Модель Полиномиальной Регрессии [Електронний ресурс] – Режим доступу: https://habr.com/ru/post/414245/

[11] Лекция 5. Аппроксимация функций по методу наименьших квадратов [Електронний ресурс] – Режим доступу: http://mvm-math.narod.ru/Lec_PM5.pdf

[12] Айвазян С. А., Мхитарян В. С. Прикладная статистика и основы эконометрики: Учебник для вузов. — М.: ЮНИТИ, 1998. — 1022 с.

[13] Карташов М. В. Імовірність, процеси, статистика — Київ, ВПЦ Київський університет, 2007.

[14] С. Р. Рао, Линейные статистические методы и их применения / Пер. с англ. — М.: Наука,1968

[15] Rao, C. Radhakrishna; Toutenburg, Shalabh, Heumann (2008). Linear Models and Generalizations (3rd ed.). Berlin: Springer. ISBN 978-3-540-74226-5.

[16] Несколько слов о «линейной» регрессии [Електронний ресурс] – Режим доступу: https://habr.com/ru/post/233059/

[17] Гмурман В. Е. Теория вероятностей и математическая статистика: Учебное пособие для вузов. — 10-е издание, стереотипное. — Москва: Высшая школа, 2004. — 479 с. — ISBN 5-06-004214-6.

[18] Елисеева И. И., Юзбашев М. М. Общая теория статистики: Учебник / Под ред. И. И. Елисеевой. — 4-е издание, переработанное и дополненное. — Москва: Финансы и Статистика, 2002. — 480 с. — ISBN 5-279-01956-9.

[19] Общая теория статистики: Учебник / Под ред. Р. А. Шмойловой. — 3-е издание, переработанное. — Москва: Финансы и Статистика, 2002. — 560 с. — ISBN 5-279-01951-8.

[20] Суслов В. И., Ибрагимов Н. М., Талышева Л. П., Цыплаков А. А. Эконометрия. — Новосибирск: СО РАН, 2005. — 744 с. — ISBN 5-7692-0755-8.

[21] Animesh Agarwal. Polynomial Regression [Електронний ресурс] / Animesh Agarwal – 2018 – Режим доступу до ресурсу: https://towardsdatascience.com/polynomial-regression-bbe8b9d97491

[22] Ершов Э.Б. Распространение коэффициента детерминации на общий случай линейной регрессии, оцениваемой с помощью различных версий метода наименьших квадратов (рус., англ.) // ЦЭМИ РАН Экономика и математические методы. — Москва: ЦЭМИ РАН, 2002. — Т. 38, вып. 3. — С. 107-120.

Додаток А

```
import subprocess
import pandas as pd
import json

secret_key = 'The Magic Words are Squeamish Ossifrage.'


correlation_data = {}
for i in range(5, 10000, 5):
    print(i)
    attack = subprocess.Popen(f'./tmp {i}', shell=True, stdout=subprocess.PIPE,
stderr=subprocess.STDOUT)
    attack.communicate()
    points = pd.read_csv('data.csv', sep=',', names = [hex(byte) for byte in range(256)])
    attack = subprocess.Popen('rm -f data.csv', shell=True, stdout=subprocess.PIPE,
stderr=subprocess.STDOUT)
    attack.communicate()
    curr_dataset = {}
    for index in range(len(secret_key)):
        val = int(points.iloc[index][ord(secret_key[index]) - 1])
        curr_dataset[secret_key[index]] = val
    correlation_data[i] = curr_dataset
with open('spectre_data.json', 'w') as dump_file:
    dump_file.write(json.dumps(correlation_data))
```

Додаток В

```
import pandas as pd
import json
import matplotlib.pyplot as plt
import seaborn as sns
import operator

plt.rcParams['figure.figsize'] = [14.5, 12]

secret_key = 'The Magic Words are Squeamish Ossifrage.'

with open('spectre_data.json', 'r') as data_file:
    dataset = json.loads(data_file.read())
    t_stat = []
```

```
    for freq in range(5, 1000, 5):
        vals = []
        for letter in secret_key:
            vals.append(dataset[str(freq)][letter])
        t_stat.append((freq, sum(vals) / max(len(vals), 1)))
    df = pd.DataFrame(t_stat, columns=['1/freq', 'cpu clock'])
    df.plot.scatter(x='1/freq', y='cpu clock')

    plt.show(dataset)
    print(max([item[0] for item in t_stat]))
    sns.distplot([item[1] for item in t_stat])`
```

Додаток С

```
from IPython import get_ipython
get_ipython().run_line_magic('matplotlib', 'inline')
import pandas as pd
import json
import matplotlib.pyplot as plt

secret_key = 'The Magic Words are Squeamish Ossifrage.'

corr_table = []
with open('spectre_data.json', 'r') as data_file:
    dataset = json.loads(data_file.read())
    ind = 0
    for letter in secret_key:
        corr_row = []
        correlation_data = []
        for key, value in dataset.items():
            correlation_data.append((int(key), int(value[letter])))
        df = pd.DataFrame(correlation_data, columns=['freq', 'points'])
        corr_table.append((ind, df.corr(method='pearson').iloc[0][1]))
        ind += 1
corr_dataset = pd.DataFrame(corr_table, columns=['Letter', 'Pearson'])
print(corr_dataset)
corr_dataset.plot.scatter(x='Letter', y='Pearson')
plt.show()
```

Додаток D

```
from scipy.optimize import curve_fit
import matplotlib.pyplot as plt
import matplotlib as mpl

plt.rcParams['figure.figsize'] = [14.5, 12]

def minimal_squares_plot(x, y):
    n  = len(x)
    s  = sum(y)
    s1 = sum([1 / x[i] for i in  range(0, n)])
    s2 = sum([(1 / x[i]) ** 2 for i in  range(0, n)])
    s3 = sum([y[i] / x[i]  for i in range(0, n)])
    a  = round((s * s2 - s1 * s3) / (n * s2 - s1 ** 2), 3)
    b  = round((n * s3 - s1 * s) / (n * s2 - s1 ** 2), 3)
    s4 = [a + b / x[i] for i in range(0, n)]
    so = round(sum([abs(y[i] - s4[i]) for i in range(0, n)]) / (n * sum(y)) * 100, 3)
    plt.title(f'Hyperbolic Appr Y= {str(a)} + {str(b)} / x\n. Avg error ratio:
{str(so)}%',size=14)
    plt.xlabel('freq', size=14)
    plt.ylabel('cpu clocks', size=14)
    plt.plot(x, y, color='b', linestyle=' ', marker='o', label='Data(x,y)')
    plt.plot(x, s4, color='g', linewidth=2, label='Data(x,f(x)=a+b/x')
    plt.legend(loc='best')
    plt.grid(True)
    plt.show()
    so=round(sum([abs(y[i]-s4[i]) for i in range(0,len(x))])/(len(x)*sum(y))*100,4)
    print('a:', a)
    print('b:', b)
    print('quadratic deviation: ', so)

with open('spectre_data.json', 'r') as data_file:
    dataset = json.loads(data_file.read())
    t_stat = []
    for freq in range(5, 1000, 5):
        vals = []
        for letter in secret_key:
            vals.append(dataset[str(freq)][letter])
        t_stat.append((freq, sum(vals) / max(len(vals), 1)))
    df = pd.DataFrame(t_stat, columns=['freq', 'cpu clock'])
    minimal_squares_plot(df['freq'], df['cpu clock'])
```

Додаток E

```
from scipy.optimize import curve_fit
import matplotlib.pyplot as plt

plt.rcParams['figure.figsize'] = [14.5, 12]

def hyperb(x, a, b):
    return  (a * x) + b

with open('spectre_data.json', 'r') as data_file:
    dataset = json.loads(data_file.read())
    t_stat = []
    for freq in range(5, 1000, 5):
        vals = []
        for letter in secret_key:
            vals.append(dataset[str(freq)][letter])
        t_stat.append((freq, sum(vals) / max(len(vals), 1)))
    df = pd.DataFrame(t_stat, columns=['freq', 'cpu clock'])

    popt, _ = curve_fit(hyperb, df['freq'], df['cpu clock'])

    fit_y = [hyperb(clock, popt[0], popt[1]) for clock in df['freq']]
    plt.plot(df['freq'], df['cpu clock'], 'o', df['freq'], fit_y, '-')
    print(f'a: {popt[0]}')
    print(f'b: {popt[1]}')
    r2 = r2_score(y,y_poly_pred)
```

Додаток F

```
from scipy.optimize import curve_fit
import matplotlib.pyplot as plt
import json
import pandas as pd
import numpy as np
import operator

from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.preprocessing import PolynomialFeatures

import seaborn as sns
```

```
%matplotlib inline

plt.rcParams['figure.figsize'] = [14.5, 12]

secret_key = 'The Magic Words are Squeamish Ossifrage.'

with open('spectre_data.json', 'r') as data_file:
    dataset = json.loads(data_file.read())
    t_stat = []
    x = []
    y = []
    for freq in range(5, 1000, 5):
        vals = []
        for letter in secret_key:
            vals.append(dataset[str(freq)][letter])
        t_stat.append((freq, sum(vals) / max(len(vals), 1)))
        x.append(freq)
        y.append(sum(vals) / max(len(vals), 1))
    df = pd.DataFrame(t_stat, columns=['freq', 'cpu clock'])

    df.plot.scatter(x='freq', y='cpu clock')

    x = np.array(x)[:, np.newaxis]
    y = np.array(y)[:, np.newaxis]
    # 4 або 5
    polynomial_features= PolynomialFeatures(degree=6)
    x_poly = polynomial_features.fit_transform(x)

    model = LinearRegression()
    model.fit(x_poly, y)
    y_poly_pred = model.predict(x_poly)

    rmse = np.sqrt(mean_squared_error(y,y_poly_pred))
    r2 = r2_score(y,y_poly_pred)
    # середньоквадратична похибка
    print("rmse:", rmse)
    # відмінність реального значення, та того, що було передбачене ф-цією
    print("r2:", r2)

    sort_axis = operator.itemgetter(0)
    sorted_zip = sorted(zip(x,y_poly_pred), key=sort_axis)

    x, y_poly_pred = zip(*sorted_zip)
```

```
plt.plot(x, y_poly_pred, color='m')
# узято з вимірбвань avg_mem_access.c
plt.axhline(y=200, color='r', linestyle='-')
print("Polynom coefitients: ", model.coef_)
zone_started = False
range_rank = 565
for i in range(len(x)):
    if y_poly_pred[i] < range_rank and not zone_started:
        zone_started = True
        print('x_started: ', x[i])
    elif y_poly_pred[i] > range_rank and zone_started:
        zone_started = False
        print('x_ended: ', x[i])
raw_zone_started = False
for i in range(len(x)):
    if y[i] < range_rank and not raw_zone_started:
        raw_zone_started = True
        print('raw_x_started: ', x[i])
    elif y[i] > range_rank and raw_zone_started:
        raw_zone_started = False
        print('raw_x_ended: ', x[i])
plt.axhline(y=range_rank, color='g', linestyle='-')
plt.show()
```