Politechnika Wrocławska

ORGANIZACJA I ARCHITEKTURA KOMPUTERÓW
SPRAWOZDANIE PROJEKTOWE

SYMULOWANIE ATAKÓW SPECTRE

WOJCIECH GRZYBOWSKI, 273212
KACPER WRÓBLEWSKI, 272952

PROWADZĄCY - PROF. DR HAB. INŻ. STANISŁAW PIESTRAK

14 CZERWCA 2024

Spis treści

1 Wprowadzenie

Celem projektu jest przeprowadzenie symulacji ataku Spectre

1.1 Specyfikacja sprzętu fizycznego, na którym operowaliśmy

Laptop: Legion 5 15ACH6

System operacyjny: Ubuntu 22.04.4 LTS x86_64

CPU: AMD Ryzen 5 5600H with Radeon Graphics

Zintegrowane GPU: AMD ATI 05:00.0 Cezanne

Dedykowane GPU: NVIDIA GeForce RTX 3050 Mobile

RAM: 13817MiB

1.2 Oprogramowanie użyte do badania

Otwarty symulator systemów komputerowych gem5.

2 Wstęp teoretyczny

2.1 Potokowanie (ang. Pipelining)

Potokowanie to technika implementacji, w której wiele instrukcji nakłada się podczas wykonywania, wykorzystuje ona równoległość istniejącą pomiędzy działaniami niezbędnymi do wykonania instrukcji. Potokowanie to aktywność w której działania wymagane do wykonania instrukcji są podzielone (ulegają segmentacji) na różne etapy, a sprzęt procesora jest zorganizowany jako seria etapów, z których każdy wykonuje jeden z tych kroków. Etapy mogą działać równolegle, pracując nad różnymi częściami różnych instrukcji. W Pipeliningu, instrukcja jest przetwarzana poprzez sekwencję etapów, z których każdy wykonuje jedną małą część wymaganych operacji (np. pobranie instrukcji z pamięci, określenie typu instrukcji, odczyt z pamięci, wykonanie operacji arytmetycznej, zapis do pamięci i aktualizacja licznika programów). Podejście to zapewnia wysoką wydajność poprzez nakładanie się kroków kolejnych instrukcji, na przykład pobieranie jednej

instrukcji podczas wykonywania operacji arytmetycznych dla poprzedniej instrukcji.

2.2 Predyktor gałęzi (ang. Branch predictor)

Układ cyfrowy, próbujący odgadnąć która gałąź zostanie wybrana przy odkodowaniu – wykonaniu polecenia (w celu zwiększenia wydajności), przy prawidłowym zgadnięciu umożliwia zaoszczędzenie zasobów oraz wykonanie procedur bez opóźnienia. Instrukcje przestaną być wykonywane dopiero po określeniu prawidłowej ścieżki (gałęzi). Zmiany zachodzą w procesorze, a pozostała mikro-architektura nie zostaje przywrócona do poprzedniego stanu, co daje atakującemu możliwość zebrania danych wynikowych (wykonanych poleceń). Przykłady gałąź If-Else, predykcja adresu w pamięci z kutego ma zostać pobrana wartość, itp.

2.3 Wykonanie poza kolejnością (ang. Out – of – order execution, OoOE)

Wykonywanie poza kolejnością to technika optymalizacji, która pozwala na maksymalne wykorzystanie wszystkich jednostek wykonawczych rdzenia procesora. Zamiast przetwarzać instrukcje ściśle według kolejności programu, procesor wykonuje je, gdy tylko dostępne będą wszystkie wymagane zasoby. Podczas gdy jednostka wykonawcza bieżącej operacji jest zajęta, inne jednostki wykonawcze mogą działać dalej. Dlatego instrukcje mogą być wykonywane równolegle, o ile ich wyniki są zgodne z definicją architektury. W praktyce procesory obsługujące wykonywanie poza kolejnością pozwalają na prowadzenie operacji spekulacyjnych w zakresie magistrali (CDB). Jeżeli operand nie jest dostępny, jednostka (lub stacja) rezerwująca (ang. reservation station) może nasłuchiwać na magistrali CDB, dopóki operand nie będzie dostępny, a następnie bezpośrednio rozpocząć wykonywanie instrukcji. Wykonywanie poza kolejnością jest realizowane w silniku wykonawczym.

2.4 Wykonanie spekulacyjne (ang. Speculative Execution)

Technika optymalizacyjna używająca niewykorzystane cykle przetwarzania, w której procesor (ang. Central Processing Unit, CPU) wykonuje szereg zadań

zanim zostanie o nie poproszony, w celu przygotowania informacji, kiedy będą faktycznie potrzebne. Programy są układane w stosy w celu wykonania operacji wejścia – wyjścia, więc rozpoczyna się wstępne pobieranie danych na potrzeby przyszłych potrzeb, wykonując je przed ich wykonaniem (co zwiększa wydajność). Wykonanie spekulacyjne używa predykcji gałęzi (branch prediction), żeby zgadnąć, które instrukcje najprawdopodobniej będą (zostaną wykonane) potrzebna w niedalekiej przyszłości. Wykonanie spekulacyjne korzysta również z analizy przepływu danych w celu zorganizowania pod kątem optymalnego wykonania (zamiast wykonywania ich w kolejności, w jakiej przyszły). Celem tych rozwiązań jest redukcja całkowitego czasu wykonania oraz poprawa ogólnej wydajności procesora.

2.5 Ataki boczno-kanałowe (side - channel attacks)

W kryptografii i bezpieczeństwie komputerowym side-channel to niezamierzona ścieżka przecieku informacji z systemu. Ataki side-channel wykorzystują informacje niezamierzonych przecieków z systemu, takie jak informacje czasowe (np. funkcja sprawdzająca czy hasło jest prawidłowe, sprawdza po każdym znaku. Gdy znak jest prawidłowy wtedy czas wykonania jest dłuższy niż w przypadku „błędnym", co pozwala na odgadnięcie hasła), zużycie energii elektrycznej lub promieniowanie elektromagnetyczne. Ataki te polegają na interakcji przy użyciu złośliwych procesów (maliciuos processes) z ofiarą po przez zasoby, np. pamięć podręczną, predyktor gałęzi (branch – predictor) lub bufory celu rozgałęzionego (branch target buffers).

2.6 Ataki Spectre

Wykorzystują spekulatywne wykonanie w celu wycieku poufnych informacji przez niezamierzone boczne kanały (side channels) po przez oszukanie procesora, aby przyjął złośliwe oprogramowanie. Istnieją różne rodzaje ataków typu Spectre, które są wykorzystywane na odpowiednich platformach np. Intel, AMD oraz ARM. Z powodu wielu wariantów tych ataków nie możliwe jest stworzenie jednego rozwiązania (łatki), które będzie działać na wszystkich platformach (każda platforma znacząco różni się architekturą sprzętu). Dodatkowo należy wspomnieć, że każda łatka (patch), wiąże się ze zmniejszeniem wydajności układu (obciążenie wydajności). Aby całkowicie rozwiązać problem dla danej platformy, byłaby konieczna całkowita przebudowa lub zmiana designu oraz modyfikacja procedur (instruction set architectures – ISAs) procesora. Ataki Spectre wykorzystują

krytyczne wady projektowe w nowoczesnych procesorach, co pozwala atakującym na kradzież danych (informacji) wrażliwych z urządzeń użytkowników wykorzystując do tego mikro architekturę kanałów bocznych (side channels).

2.6.1 Pierwotny Spectre ma dwa warianty:

- Obejście sprawdzania granic (bounds check bypass) – wykorzystuje błędne przewidywania (predykcje) gałęzi warunkowych. Jest to wariant sprawdzany przez nas w ramach projektu.
- Wstrzyknięcie gałęzi docelowej (branch target injection) – ma na celu pośrednie przewidywanie skoku celu (indirect jump target prediction).

Później Intel zidentyfikował osiem nowych podobnych wariantów – „Spectre Next Generation (Spectre - NG)". Jeden z nowych wariantów pozwala na wykonanie złośliwe kodu z wirtualnej maszyny (ang. Virtual Machine - VM), aby zaatakować maszynę hosta lub inne wirtualne maszyny (VMs) na tym samym serwerze. Kolejny wariant wykorzystuje zwracanie buforów stosu (ang. Return Stack Buffers – RSBs), w celu wywołania błędnego przewidywania adresów zwrotnych i wycieków danych wrażliwych w różnych procesach. **Wszystkie warianty trzymają się tej samej reguły – polegają na zmianach w stanach pamięci podręcznej spowodowanych wykonaniem spekulatywnym. Najgroźniejszym i najtrudniejszym do wykrycia wariantem Spectre jest „Spectre Variant 1 (Spectre – V1)".** Warianty wykorzystują wykonania spekulatywne z różną kontrolą oraz błędnym przewidywaniem danych, lecz zazwyczaj ataki Spectre prowadzą do przecieku tajnych informacji poprzez boczny kanał pamięci podręcznej. Zatem monitorowanie zachowania pamięci podręcznej (cache) może pomóc w wykryciu ataków typu Spectre. Spectre – V1 jest aktywowany jako wynik niepoprawnego przewidzenia gałęzi. Możliwe jest „wyuczenie" predyktora gałęzi, po przez np. wywołanie kilka razy tej samej funkcji i podawanie odpowiednich wartości parametrycznie do tejże funkcji. Następnie po „wyuczeniu" podaje się wartość parametru wychodzącą poza zakres (out – of – bound parameter), który wskazuje na sekretny bajt w pamięci ofiary.

2.6.2 Atak Spectre krok po kroku:

1) Przygotowanie bocznego kanału do wykradnięcia danych ofiary, i pozostałych narzędzi, takich jak niepoprawne trenowanie (mis-train) predyktora gałęzi do przyjęcia błędnej ścieżki wykonania oraz załadowanie lokalizację pamięci docelowej do rejestrów.

2) Przekierowanie poufnych informacji z kontekstu ofiary na boczny kanał mikroarchitektury, wykorzystując różne luki w zabezpieczeniach sprzętu, takie jak „Out – Of – Order Execution" czy wykonanie spekulacyjne (speculative execution).
3) Podczas fazy ostatecznej atakujący zyskuje dostęp to sekretnych danych przez przygotowany wcześniej boczny kanał.

2.6.3 Łagodzenie skutków Ataku Spectre

Łagodzenie skutków Ataku Spectre jest trudne, ze względu na ilość wariacji tego ataku. Na przykład, aby zapobiec Spectre – V1 trzeba zatrzymać wykonanie spekulacyjne na wszystkich wrażliwych ścieżkach wykonawczych (sensitive execution paths). Jednak wstawienie takich mechanizmów blokujących we wszystkich gałęziach warunkowych i w ich miejscach docelowych przez kompilator poważnie pogorszyłoby wydajność. Brakuje również wydajnych, niezależnych od architektury sposobów adresowania Spectre-V1 w kodzie przestrzeni użytkownika. Zgłaszano, że opublikowane aktualizacje systemu operacyjnego i oprogramowania układowego dla innych wariantów Spectre spowalniają działanie systemu komputerowego.

2.6.4 Metody wykrywania Ataków Spectre

Pomimo istnienia **wielu wariantów Spectre**, wszystkie polegają na oszukaniu procesora, aby wybrał niewłaściwą ścieżkę wykonawczą i ujawnił poufne dane przez obserwowany boczny kanał mikro architektury. W końcowej fazie Spectre dane wyciekają bocznymi kanałami pamięci podręcznej, atakujący musi stale opróżniać pamięć podręczną, aby upewnić się, że dane nie są buforowane, co oznacza, że współczynnik pomyleń pamięci podręcznej prawdopodobnie wzrośnie. W związku z tym, aby wykryć ataki Spectre, należy monitorować pamięć podręczną pod kątem jej czyszczenia i wycieku danych.

Tradycyjne oprogramowania antywirusowe (ang. AV, Anti Virus) skanują podejrzane instrukcje binarne lub poszukują śladów w raportach systemowych (system logs) w celu statycznego wykrycia złośliwych ataków (malicious attacks). Ataki Spectre zwykle nie zostawiają śladów w raportach systemowych. Stąd ciężko jest wykryć ataki Spectre przy użyciu statycznej analizy. Z przeprowadzonych badań wynika, że złośliwe oprogramowanie może zostać wykryte po przez użycie dynamicznych wzorców

mikroarchitektury wykonania Sprzętowych Liczników Wydajności (ang. Hardware Performance Counters - HPC) pozyskanych z nowoczesnych procesorów. Na oprogramowaniu Android OS zastosowano techniki uczenia maszynowego w celu wykrywania ataków typu przepełnienia bufora (buffer overflow) oraz ataków zorientowanych na zwrot (return – oriented programming).

2.7 GEM 5

Symulator GEM 5 to modułowa platforma do badań architektury systemów komputerowych, obejmująca architekturę na poziomie systemu, a także mikro architekturę procesora. GEM5 to projekt kierowany przez społeczność z otwartym modelem zarządzania (open source software). GEM 5 został pierwotnie stworzony do badań nad architekturą komputerów w środowisku akademickim, ale zaczął być stosowany w projektowaniu systemów komputerowych w środowisku akademickim, w przemyśle do badań i w nauczaniu.

3 Przebieg Symulacji

3.1 Sposób przeprowadzenia badania

3.1.1 Uwagi do symulatora

W symulatorze zmieniony został branch predictor z TimingSimpleCPU() na DerivO3CPU(branchPred=LTAGE()). Z uwagi na jego rzekomo większą podatność na ataki Spectre.

Ponadto zastosowaliśmy narzędzie wbudowane w GEM5 „O3 Pipeline viewer”, oraz kompilator gcc (Ubuntu 7.5.0-6ubuntu2) 7.5.0

3.1.2 Uwagi do kompilatora

~~Na początku próbowaliśmy użyć najnowszego kompilatora gcc, wbudowanego w system ubuntu. Niestety kod się nie kompilował. Według różnych forów internetowych było to spowodowane zabezpieczeniami przed prymitywnymi atakami spectre wbudowanymi w najnowsze kompilatory. W celu kontynuowania badania musieliśmy pobrać starszą wersję gcc. Jako że ludzie już wcześniej odtwarzali ataki spectre kompilatorem gcc 7.5.0, to na niego też się zdecydowaliśmy.~~

~~Niefortunnie natknęliśmy się na nowy problem. Gcc 7.5.0 jest wersją niewspieraną od wielu lat, co za tym idzie nie można pobrać jej z oficjalnej strony gcc. Na szczęście po przekopaniu for internetowych znaleźliśmy stare repozytorium z tym czego szukaliśmy.~~

Po ponownym zainstalowaniu systemu operacyjnego Ubuntu od nowa, okazało się, że powyższe stwierdzenia są fałszywe. Po zainstalowaniu symulatora GEM 5 atak spectre działał poprawnie pomimo użycia najnowszej wersji kompilatora gcc wbudowanej w system.

3.2 Pobieranie i budowanie programu gem5

Na początku sklonowaliśmy repozytorium programu komendą:

git clone https://github.com/gem5/gem5

Następnie zbudowaliśmy folder z programem przy pomocy:

scons build/X86/gem5.opt -j <NUMBER OF CPUs ON YOUR PLATFORM>

3.3 Uruchamianie programu spectre.c w gem5

Po wpisaniu do terminala polecenia:

build/X86/gem5.opt configs/learning_gem5/part1/two_level.py spectre

Powinniśmy uzyskać poprawnie wykonany atak spectre.

```
vivi@vivi-Legion-5-15ACH6:~/gem5$ build/X86/gem5.opt configs/learning_gem5/part1/two_level.py spectre
gem5 Simulator System.  https://www.gem5.org
gem5 is copyrighted software; use the --copyright option for details.

gem5 version 23.1.0.0
gem5 compiled Apr 18 2024 19:09:07
gem5 started May 15 2024 13:43:09
gem5 executing on vivi-Legion-5-15ACH6, pid 13667
command line: build/X86/gem5.opt configs/learning_gem5/part1/two_level.py spectre

Global frequency set at 1000000000000 ticks per second
warn: No dot file generated. Please install pydot to generate the dot file and pdf.
src/mem/dram_interface.cc:690: warn: DRAM device capacity (8192 Mbytes) does not match the address range assigned (512 Mbytes)
src/base/statistics.hh:279: warn: One of the stats is a legacy stat. Legacy stat is a stat that does not belong to any statistics::Group. Legacy stat is deprecated.
system.remote_gdb: Listening for connections on port 7000
Beginning simulation!
src/sim/simulate.cc:199: info: Entering event queue @ 0.  Starting simulation...
src/sim/syscall_emul.cc:74: warn: ignoring syscall set_robust_list(...)
src/sim/syscall_emul.cc:74: warn: ignoring syscall rseq(...)
src/sim/mem_state.cc:448: info: Increasing stack size by one page.
src/sim/syscall_emul.hh:1014: warn: readlink() called on '/proc/self/exe' may yield unexpected results in various settings.
      Returning '/home/vivi/gem5/spectre'
src/sim/syscall_emul.cc:74: warn: ignoring syscall mprotect(...)
Reading 40 bytes:
Reading at malicious_x = 0xfffffffffffd0ee8... Success: 0x54='T' score=2
Reading at malicious_x = 0xfffffffffffd0ee9... Success: 0x68='h' score=2
Reading at malicious_x = 0xfffffffffffd0eea... Success: 0x65='e' score=2
Reading at malicious_x = 0xfffffffffffd0eeb... Success: 0x20=' ' score=2
Reading at malicious_x = 0xfffffffffffd0eec... Success: 0x4D='M' score=2
Reading at malicious_x = 0xfffffffffffd0eed... Success: 0x61='a' score=2
Reading at malicious_x = 0xfffffffffffd0eee... Success: 0x67='g' score=2
Reading at malicious_x = 0xfffffffffffd0eef... Success: 0x69='i' score=2
Reading at malicious_x = 0xfffffffffffd0ef0... Success: 0x63='c' score=2
Reading at malicious_x = 0xfffffffffffd0ef1... Success: 0x20=' ' score=2
Reading at malicious_x = 0xfffffffffffd0ef2... Success: 0x57='W' score=2
Reading at malicious_x = 0xfffffffffffd0ef3... Success: 0x6F='o' score=2
Reading at malicious_x = 0xfffffffffffd0ef4... Success: 0x72='r' score=2
Reading at malicious_x = 0xfffffffffffd0ef5... Success: 0x64='d' score=2
Reading at malicious_x = 0xfffffffffffd0ef6... Success: 0x73='s' score=2
Reading at malicious_x = 0xfffffffffffd0ef7... Success: 0x20=' ' score=2
Reading at malicious_x = 0xfffffffffffd0ef8... Success: 0x61='a' score=2
Reading at malicious_x = 0xfffffffffffd0ef9... Success: 0x72='r' score=2
Reading at malicious_x = 0xfffffffffffd0efa... Success: 0x65='e' score=2
Reading at malicious_x = 0xfffffffffffd0efb... Success: 0x20=' ' score=2
Reading at malicious_x = 0xfffffffffffd0efc... Success: 0x53='S' score=2
Reading at malicious_x = 0xfffffffffffd0efd... Success: 0x71='q' score=2
Reading at malicious_x = 0xfffffffffffd0efe... Success: 0x75='u' score=2
Reading at malicious_x = 0xfffffffffffd0eff... Success: 0x65='e' score=2
Reading at malicious_x = 0xfffffffffffd0f00... Success: 0x61='a' score=2
Reading at malicious_x = 0xfffffffffffd0f01... Success: 0x6D='m' score=2
Reading at malicious_x = 0xfffffffffffd0f02... Success: 0x69='i' score=2
Reading at malicious_x = 0xfffffffffffd0f03... Success: 0x73='s' score=2
Reading at malicious_x = 0xfffffffffffd0f04... Success: 0x68='h' score=2
Reading at malicious_x = 0xfffffffffffd0f05... Success: 0x20=' ' score=2
Reading at malicious_x = 0xfffffffffffd0f06... Success: 0x4F='O' score=2
Reading at malicious_x = 0xfffffffffffd0f07... Success: 0x73='s' score=2
Reading at malicious_x = 0xfffffffffffd0f08... Success: 0x73='s' score=2
Reading at malicious_x = 0xfffffffffffd0f09... Success: 0x69='i' score=2
Reading at malicious_x = 0xfffffffffffd0f0a... Success: 0x66='f' score=2
Reading at malicious_x = 0xfffffffffffd0f0b... Success: 0x72='r' score=2
Reading at malicious_x = 0xfffffffffffd0f0c... Success: 0x61='a' score=2
Reading at malicious_x = 0xfffffffffffd0f0d... Success: 0x67='g' score=2
Reading at malicious_x = 0xfffffffffffd0f0e... Success: 0x65='e' score=2
Reading at malicious_x = 0xfffffffffffd0f0f... Success: 0x2E='.' score=2
Exiting @ tick 153008187000 because exiting with last active thread context
```

Atak poprawnie odczytał tablicę typu char (do której w teorii nigdy się nie odnosił) równą napisowi „The Magic Words are squeamish Ossifrage.”.

3.4 Wizualizacja potokowości

Aby móc podejrzeć potokowość naszego ataku, wpierw musimy wygenerować plik z zapisem wykonanych poleceń assemblera x86 na poziomie pojedynczych cykli zegarowych (ang. trace file). W tym celu uruchamiamy atak jeszcze raz, ale od razu po rozpoczęciu jego działania wciskamy skrót klawiszowy ctrl + c.

Potrzebny będzie nam numer cyklu zegarowego, od którego zaczął wykonywać się program (ang. tick). Przy następnym uruchomieniu ataku ustawimy w tym miejscu flagę, która zacznie sczytywać potrzebne nam informacje do pliku **pipeview.txt**. Zrobimy to komendą:

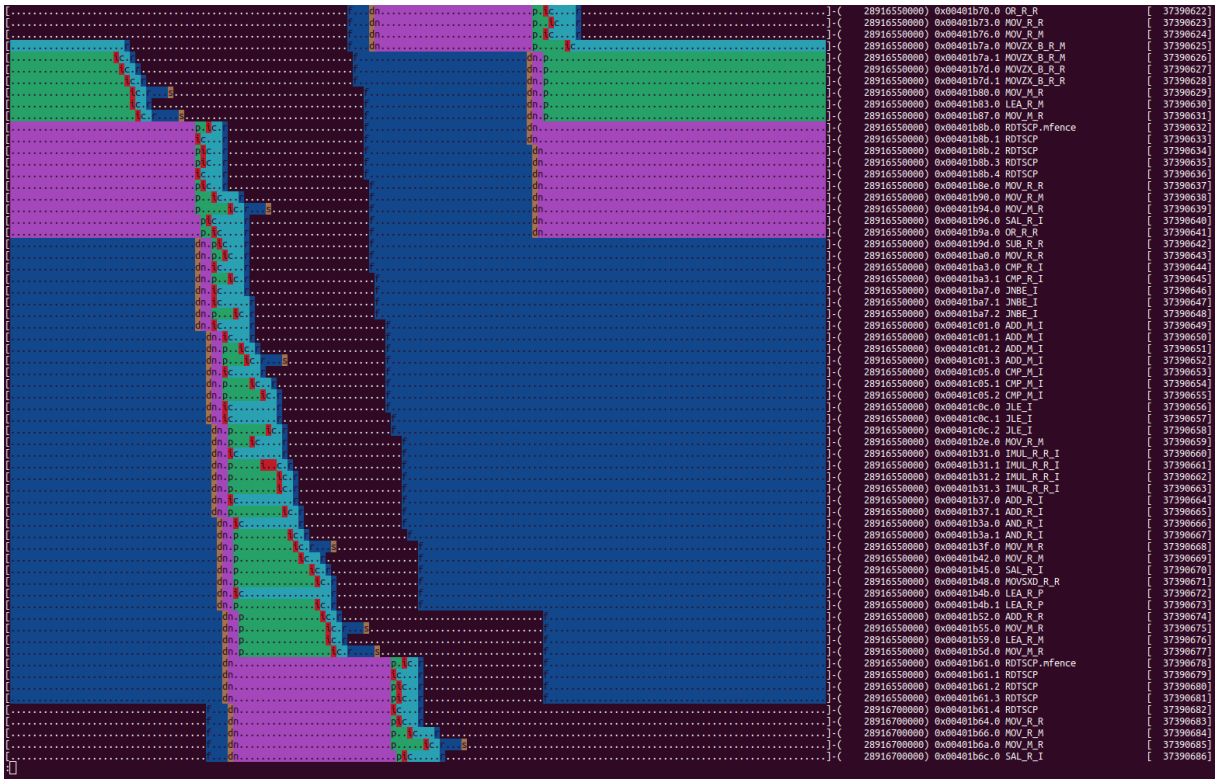
build/X86/gem5.opt --debug-flags=O3PipeView --debug-file=pipeview.txt --debug-start=13062347000 configs/learning_gem5/part1/two_level.py spectre

Należy zwrócić uwagę, że nasz plik śladowy (trace file) jest bardzo dużym plikiem i np. w naszym przypadku, nie mogliśmy wygenerować go dla całego ataku. Z tej uwagi zaczytaliśmy ¾ ataku, co zajęło nam 45GB i zostawiło 300MB wolnego miejsca na dysku. Ponieważ miejsce na dysku było nam potrzebne, powtórzyliśmy tę operację, tym razem generując tylko 8GB pliku trace file.

Ostatnim etapem jest wygenerowanie wizualizacji plikiem **o3-pipeview.py** wbudowanym domyślnie w symulator gem5. Potrzebna jest do tego komenda:

Util/o3-pipeview.py –store_completions m5out/pipeview.txt –color -w 150

W końcu możemy zobaczyć naszą potokowość:



3.5 Odczytywanie wizualizacji potokowości w gem5

Po lewej stronie znajduję się linia czasu naszej potokowości. Każdy znak litery, kropki bądź znaku równości oznacza jeden cykl zegarowy. Konkretne znaczenia liter zamieszczamy w tabeli poniżej:

f	Fetch – Jest to pierwszy etap cyklu wykonywania instrukcji. Polega na pobraniu jej z pamięci komputera i załadowanie do rejestru rozkazów (rejestr instrukcji).
d	Decode – Drugi etap cyklu wykonywania instrukcji. Polega na przeanalizowaniu kodu maszynowego danej instrukcji i rodzieleniu go na kod operacyjny (opcode), argumenty (operands), i rejestry / miesjca w pamięci, które mają w danej instrukcji zostać użyte.
n	Rename – Technika optymalizacji potokowości, mająca na celu poprawienie przetwarzania równoległego na poziomie instrukcji (ang. instruction level parallelism). Polega na przypisaniu aliasów do rejestrów fizycznych będących składowymi rejestrów logicznych.

p	Dispatch – pol. wysłanie polega na przesłaniu rozkodowanych i zmienionych nazw rejestrów w instrukcji do odpowiednich układów mających na celu ich wykonanie.
i	Issue – Oznacza, że konkretne rozgałęzienie napotkało problem: np. próba działania na pamięci wykorzystywanej przez inną gałąź.
c	Commit – Ostatni etap cyklu wykonywania instrukcji, w którym wyniki wykonanych operacji są zapisywane we wskazanych przez instrukcję miejscach.
r	Retire – Ostatni etap cyklu instrukcji. Oznacza, że instrukcja nie jest już potrzebna, w związku z czym jej wykonanie jest przedwcześnie kończone.
l	Load finishes - Ostatni etap cyklu instrukcji. Oznacza, że dane zostały prawidłowo pobrane z pamięci i zapisane w rejestrze .
s	Store finishes - Ostatni etap cyklu instrukcji. Oznacza, że dane zostały prawidłowo pobrane z rejestru i zapisane w pamięci .

Na prawo od widoku potokowości znajduje się tick obecnie wykonywanych cykli zegarowych np. 28916100000. Znowu na prawo od tick'a znajduje się liczba reprezentująca jednocześnie adres wykonywanej instrukcji i jej kod operacyjny oraz kod assemblerowy tejże.

3.6 Odnalezienie miejsca naruszenia pamięci przez atak Spectre w wizualizacji potokowości gem5

Przeszukiwanie wizualizacji ręcznie ma całkiem sporo sensu. Atak zachowuje się powtarzalnie i łatwo „gołym okiem" zobaczyć powtarzający się wzór. Trzeba tylko wiedzieć, jakie instrukcje asemblerowe wykonuje konkretnie atak, w tym celu należy zdekompilować program **spectre.c**.

Za przebieg ataku w pliku **spectre.c** odpowiada funkcja **victim_function** jej kod wygląda następująco w postaci języka C i assemblera x86.

```
1.  void victim_function(size_t x) {
2.    if (x < array1_size) {
3.      temp &= array2[array1[x] * 512];
4.    }
5.  }
```

```
1.  victim_function:
2.  .LFB3924:
3.              .cfi_startproc
4.              pushq       %rbp
5.              .cfi_def_cfa_offset 16
6.              .cfi_offset 6, -16
7.              movq        %rsp, %rbp
8.              .cfi_def_cfa_register 6
9.              movq        %rdi, -8(%rbp)
10.             movl        array1_size(%rip), %eax
11.             movl        %eax, %eax
12.             cmpq        %rax, -8(%rbp)
13.             jnb         .L3
14.             leaq        array1(%rip), %rdx
15.             movq        -8(%rbp), %rax
16.             addq        %rdx, %rax
17.             movzbl      (%rax), %eax
18.             movzbl      %al, %eax
19.             sall        $9, %eax
20.             movslq      %eax, %rdx
21.             leaq        array2(%rip), %rax
22.             movzbl      (%rdx,%rax), %edx
23.             movzbl      temp(%rip), %eax
24.             andl        %edx, %eax
25.             movb        %al, temp(%rip)
```

Nas konkretnie interesuje instrukcja movzbl, która w wcześniej wygenerowanym pliku śladowym przyjmuje postać MOVZX_B_R_M.

4 Wnioski

Symulacja ataku Spectre przy użyciu gem5 pozwoliła nam na dokładne zrozumienie mechanizmów działania tego ataku oraz poznanie metod jego łagodzenia. Nasze wyniki potwierdzają, że spekulatywne wykonanie instrukcji może prowadzić do wycieku poufnych danych, co podkreśla potrzebę dalszych badań i rozwoju technologii zabezpieczeń.

Szczegółowe wnioski:

1. **Zrozumienie mechanizmów ataku:** Symulacja w gem5 umożliwiła nam szczegółowe prześledzenie, jak procesory wykonują spekulatywne instrukcje i jak predyktory gałęzi mogą być manipulowane przez złośliwe oprogramowanie. Zaobserwowaliśmy, że procesory próbują przewidzieć przyszłe instrukcje na podstawie historii wykonania, co pozwala na wykonanie kodu, który normalnie nie powinien być uruchomiony. Wykonanie spekulatywne może prowadzić do naruszenia granic pamięci i odczytu danych przez atakującego.

2. **Identyfikacja podatnych obszarów:** Dzięki wizualizacji potokowości i analizy ścieżek wykonania, udało nam się zidentyfikować konkretne miejsca w kodzie, gdzie spekulatywne wykonanie powoduje naruszenie pamięci. Te miejsca stanowią potencjalne cele ataków i wymagają szczególnej uwagi podczas projektowania zabezpieczeń.

3. **Znaczenie aktualizacji i dobrej praktyki programistycznej:** Nasze badania potwierdziły, że regularne aktualizacje mikrokodu i oprogramowania są kluczowe dla ochrony przed nowymi wariantami ataków. Programiści muszą być świadomi ryzyka związanego ze spekulatywnym wykonaniem, w związku z tym należy implementować odpowiednie techniki zabezpieczeń już na etapie pisania kodu.

4. **Rola edukacji i świadomości:** Zrozumienie ataków takich jak Spectre wymaga obszernej wiedzy z zakresu mikroarchitektury procesorów i technik optymalizacyjnych. Edukacja programistów, inżynierów

bezpieczeństwa i architektów systemów komputerowych jest kluczowa dla skutecznego przeciwdziałania takim zagrożeniom.

5. **Konieczność dalszych badań:** Nasze badania podkreślają, że techniki spekulatywnego wykonania będą nadal stanowić poważne wyzwanie dla bezpieczeństwa komputerowego. Konieczne są dalsze badania nad nowymi metodami wykrywania i łagodzenia skutków ataków boczno-kanałowych. W szczególności, badania nad bardziej zaawansowanymi algorytmami predykcji i technikami izolacji procesów mogą przynieść znaczące korzyści w przyszłości.

Podsumowując, symulacja ataku Spectre w gem5 dostarczyła nam cennych informacji na temat działania tego rodzaju ataków, jego skutków oraz efektywności różnych metod ich łagodzenia. Wnioski te są istotne dla dalszego rozwoju technologii zabezpieczeń i podkreślają potrzebę ciągłej ewolucji w dziedzinie bezpieczeństwa komputerowego.

5 Bibliografia

Artykuł podany przez prowadzącego

Congmiao Li, Member, IEEE and Jean-Luc Gaudiot , Life Fellow, IEEE. (2022, czerwiec 6). Detecting Spectre Attacks Using Hardware Performance Counters.

Pozostałe artykuły i źródła internetowe

Paul Kocher, Jann Horn, Anders Fogh, Daniel Genkin, Daniel Gruss, Werner Haas, Mike Hamburg, Moritz Lipp, Stefan Mangard, Thomas Prescher, Michael Schwarz, Yuval Yarom. (brak daty). Spectre Attacks: Exploiting Speculative Execution.

GEM 5

https://www.gem5.org/