Modern processorarkitektur och Meltdown

Marcus Skrifvars
Datateknik
Åbo Akademi
10.4.2018

Abstrakt

I början av 2018 publicerade en grupp forskare vetenskapliga texter, i vilka de avslöjade två allvarliga säkerhetshål som funnits länge i moderna processorer. Ett av hålen, döpt till Meltdown av forskarna, var riktad till bl.a. Intels processorer på marknaden. Meltdown utnyttjar prediktiv exekvering av instruktioner. Processorn ”gissar” i förväg vilken instruktion kommer att köras till näst, och förkastar resultatet om gissningen visar sig vara fel. Genom att lura datorn att köra dessa spekulativa instruktioner, kan en attackerare komma åt information som är sparad i privata minnet, oåtkomligt för vanliga program. Värdet blir sparat som ett tillstånd i cache-minnet, där det kan avläsas med hjälp av cache-attacker. Till skillnad från det andra upptäckta säkerhetshålet, Spectre, kan processorer skyddas från Meltdown genom olika programfixar. Enligt forskarna är en modifikation till opertivsystemskärnan, kallad **KAISER**, den bästa tillfälliga lösningen. Modifikationen är inte helt säker, och har lett till minskning av prestanda på användarnas datorer.

Innehållsförteckning

1. INLEDNING 1
   1.1 Historia 1
   1.2 Kort översikt 2
      1.2.1 Meltdown 2
      1.2.2 Spectre 2
   1.3 Mål för avhandlingen 3
2. PROCESSORARKITEKTUR 3
   2.1 Pipelines 3
      2.1.1 RISC pipeline 4
      2.1.2 Pipeline-hinder 5
   2.2 Spekulativ exekvering och dess delar 6
      2.2.1 Hoppförutsägelser 6
      2.2.2 Reorder buffer 7
   2.3 CPU Cache 7
   2.4 Adressrymd 8
3. MELTDOWN 8
   3.1 Definition 8
   3.2 Jämförelse med andra skadeprogram 9
4. TRE STEGEN AV EN MELTDOWN-ATTACK 10
   4.1 Steg 1: Avläsning av minnesplats 10
      4.1.1 Undantagshantering 10
      4.1.2 Dämpning av undantag 11
      4.1.3 Race condition 11
   4.2 Steg 2: Det hemliga värdet överförs från minnet 11
      4.2.1 I fall av 0 12
   4.3 Steg 3: Det hemliga värdet mottas 12
      4.3.1 Cache-attacker 12
   4.4 Attacken avslutas 13
   4.5 Skillnader i processorarkitektur 13
5. EXEMPEL PÅ MELTDOWN-ATTACK 14

4.6.1 Exempel på minnesdump 15
6. SKYDDSÅTGÄRDER MOT MELTDOWN 16
6.1 KAISER 16
6.2 Åtgärder inom IT-branschen 17
7. SLUTSATSER 18
6.1 Konsekvenser för industrin 18
6.2 Egna reflektioner 18
8. KÄLLFÖRTECKNING 20

1. INLEDNING

1.1 Historia

En av de största händelserna inom IT-branschen i början av 2018 var upptäckten av Meltdown- och Spectre säkerhetshålen. Dessa offentliggjordes via vetenskapliga texter [1] som publicerades den 3 januari 2018 av en grupp forskare från olika företag. I artikeln ”*Meltdown and Spectre: ‘worst ever’ CPU bugs affect virtually all computers*”, publicerad den 4 januari 2018 [2] behandlar den brittiska dagstidningen The Guardian ämnet. Enligt artikeln anser Daniel Gruss, en av undersökarna som upptäckte Meltdown, att felet troligtvis är ”en av de värsta CPU-buggarna som någonsin har hittats”. Bristen sägs påverka i stort sett alla moderna datorer, medräknat smarttelefoner, surfplattor, servrar och molntjänster. Dock varierar mottagligheten för felet företagsvis [3].

Av marknadens huvudtillverkare av processorer, Amd och Intel, är det den senare vars produkter är drabbade. Enligt forskare påverkar felet i stort sett alla Intels processorer producerade efter 1995 [4]. Amd släppte i sin tur ett pressmeddelande den 11 januari 2018 [5], ca en vecka efter att Meltdown-pappret gavs ut. I meddelandet förnekade företaget att deras processorer är mottagliga för säkerhetshålet. Detta beror på skiljaktigheter i arkitekturen mellan Intels och Amd:s produkter. Bland annat Intel och Microsoft har efterhand gett ut uppdateringar för att minska systems sårbarhet för Meltdown. Uppdateringarna har lett till problem för många användare. Till exempel orsakade Intels första programfix en förminskad prestanda på 0–25 procent beroende på processormodellen. Vissa användare har också råkat ut för slumpmässiga omstarter av deras datorer, en bug som även Intel har bekräftat [6].

Teknikföretaget Apple verifierade även den 4 januari 2018 [7] att alla deras Mac- och iOS-produkter var sårbara för säkerhetshålen. Företaget har sedan dess gett ut uppdateringar för att åtgärda problemen. Spectre och Meltdown har inte endast haft en effekt på hårdvarutillverkare, utan också på andra företag som erbjuder elektroniska

tjänster. Ett exempel är Amazon Web Services, som på grund av sättet som program körs inom tjänsten är särskilt mottaglig för Spectre-attacker [8].

1.2 Kort översikt

Vad är då Meltdown och Spectre? Teamen som upptäckte de två säkerhetshålen har lagt upp en webbsida [1], där man kan läsa artiklarna om ämnet samt hitta svar till de vanligaste frågorna.

1.2.1 Meltdown

Både Spectre och Meltdown utnyttjar det faktum att processorer exekverar deras instruktioner spekulativt, dvs processorn kan i förväg anta olika värden för instruktioner och fortsätta enligt detta. Om antagandet visar sig vara fel, så förkastas den spekulativt exekverade instruktionen. Speciellt Intels processorer använder spekulationer vid åtkomst av minne; program kan spekulativt använda data från *operativsystemkärnan* (*kernel*), och genom att tyda ändringen i processorns cache-minne kan man avläsa värden som är sparade i kärnans minne. Data som potentiellt hämtas kan vara allt från lösenord till foton och dokument. Detta är principen bakom Meltdown. Forskarna som upptäckte Meltdown förklarar namnets ursprung med att ”sårbarheten i grund och botten smälter säkerhetsgränser som vanligtvis upprätthålls av hårdvaran” [1].

1.2.2 Spectre

Spectre är till skillnad från Meltdown en attack med större utsträckning. Attacken exploaterar med hjälp av flera metoder de grundläggande principerna av spekulativ exekvering, och är därför riktad mot alla moderna processorer. I motsats till Meltdown är Spectre svår att förhindra. Namnet har sitt ursprung från att attacken grundar sig på spekulativ exekvering. Problemet är inte lätt att lösa, och kommer därför att ”spöka” en längre tid.

1.3 Mål för avhandlingen

I min avhandling kommer jag att fokusera specifikt på Meltdown. För att förklara exakt hur Meltdown fungerar kommer jag inledningsvis att klargöra relevanta aspekter inom moderna processorer. Därefter tar jag en närmare titt på själva Meltdown-säkerhetshålet: Hur utnyttjar det etablerade principer inom dagens processorarkitekturer, samt hur skiljer det sig från andra skadeprogram? Därefter behandlar jag KAISER, den vanligaste försvarsmekanismen mot Meltdown-anfall. Som avslutning analyserar jag hur företag har hanterat upptäckten av säkerhetshålet, vilken påverkan det haft, och om det har någon effekt på framtida utvecklingen av processorer?

Som huvudsaklig källa kommer jag att använda artikeln om Meltdown som publicerades i början av januari 2018 [9]. Som stöd för att förklara moderna processorarkitekturer och spekulativ exekvering använder jag böckerna *A Practical Introduction to Computer Architecture* av Daniel Page [10] samt *Computer Architecture: A Quantitative Approach* av John L. Hennessy och David A. Patterson [11].

2. PROCESSORARKITEKTUR

I detta kapitel går jag igenom delar av processorn som måste förstås för att kunna förklara hur Meltdown fungerar. För att stöda konceptet om hoppförutsägelser har jag även valt att kort sammanfatta idén och funktionen bakom enkla pipor i dagens processorer. Som primära källor används kapitel 8 och 9 i boken av Daniel Page, samt kapitel 3 och appendix C i ”*Computer Architecture*”.

2.1 Pipelines

Ett sätt att göra processorer snabbare är genom att låta flera instruktioner köras samtidigt. Detta kallas för att *pipa* (*pipelining* på engelska), och är det centrala metoden med vilket dagens processorer görs effektivare. Med hjälp av pipor ökar antalet instruktioner som kan utföras per tidsenhet. Detta görs genom att dela in kretsen i olika

stadier, där varje fas kan hantera en process i taget. För att processorn skall veta vilken instruktion som skall köras till näst finns det ett register kallat *Program Counter*, förkortat **PC**, där adressen till den nuvarande processen är sparad. Själva instruktionerna och värdena är sparade i olika register i minnet. I vilket minne värdet är sparat kan påverka hur länge det tar för faserna att bli färdiga. På värden hämtade från registren görs sedan operationer med en aritmetisk logisk enhet, ALU.

2.1.1 RISC pipeline

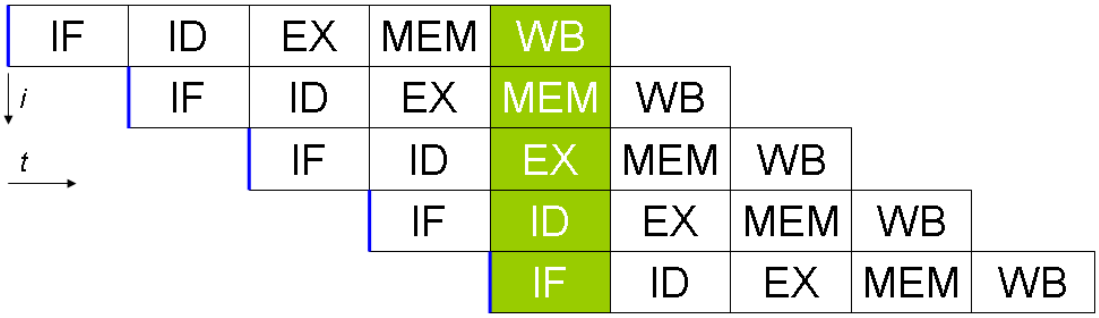
En gammal och välkänd metod för en pipa är indelad in i fem faser, ofta kallad **RISC** (Reduced Instruction Set Computer) pipeline:

1. **IF**, Instruction Fetch - PC hämtar nästa instruktion från minnet
2. **ID**, Instruction Decode - Instruktionen avkodas samtidigt som värden hämtas från register
3. **EX**, Execute - ALU kör den specificerade berä kningen baserat på instruktionen och värden som hämtats
4. **MEM**, Memory access - Om instruktionen är kräver läsning eller sparning, görs det här
5. **WB**, Write-back cycle - Resultatet skrivs i registerminnet

Figur 1 förtydligar hur flera instruktioner körs samtidigt vid samma tidpunkt. Teoretiskt kan en ny instruktion hämtas varje klockcykel. Då är processorns prestanda upp till fem gånger snabbare än en CPU som inte använder pipelines. Mellan varje steg i pipelinen finns ett *pipeline register*, en typ av buffertminne där resultatet från steget sparas. Pipeline-register används för att två eller flera instruktioner som körs parallellt inte ska påverka varandra när de använder samma hårdvaruresurs. Det är via dessa register som data skickas mellan olika faser i en pipeline. Instruktioner behöver inte alltid köras i ordningen förklarad ovan.

Ett sätt som används i moderna processorer för att optimera pipelinen är att ha instruktioner att köras direkt när resurser är användbara. Sekvensen som instruktionerna exekveras i bestäms då av tillgängligheten av data-operander, istället

för ordningen som de är skrivna i programmet. Att utföra instruktioner i denna ordning kallas för *out-of-order execution* eller *dynamisk exekvering*. I Meltdown-artikeln [9] kallar forskarna instruktioner som körs i dynamisk ordning för *obeständiga* (*transient*) instruktioner.



Figur 1: Den klassiska RISC-pipan, bestående av 5 stadier.

2.1.2 Pipeline-hinder

När instruktionerna i pipelinen blir mer avancerade kan det uppstå fel, så kallade *pipeline-hinder* (*pipeline hazards*). Dessa kan ha olika orsaker och indelas därför i tre olika kategorier. På sidan C-11 i *Computer Architecture* indelas de i *resource-* (kan även kallas *structural*), *data-* och *control hazards*. Strukturella hinder uppkommer när två processer försöker komma åt samma del av processorn, och löses oftast genom att uppehålla en av instruktionerna för en klockcykel. Dessa pauser kallas för pipeline-bubblor. Data-hinder uppstår i sin tur då en instruktion är beroende av resultatet från en tidigare process som ännu inte är färdig. En vanlig lösning är *forwarding*, där ett värde skickas framåt från pipeline-registret istället för att hämta det från huvudregistret. Kontroll-hinder sker då fetch-stadiet inte vet från vilken adress som nästa instruktion ska hämtas.

Vid villkorliga hoppinstruktioner kan instruktionerna förgrenas på olika sätt baserat på ett resultat (ha flera så kallade *branches*). Då kan PC:s värde eventuellt ändras till något annat än planerat. Ett sätt att lösa problemet är att i förväg gissa resultatet, och därmed även adressen till nästa instruktion. Detta kallas för *hoppprediktion* (*branch-prediction*),

och använder sig ofta av spekulativ exekvering för att i förväg exekvera den instruktion som mest sannolikt kommer köras till näst.

2.2 Spekulativ exekvering och dess delar

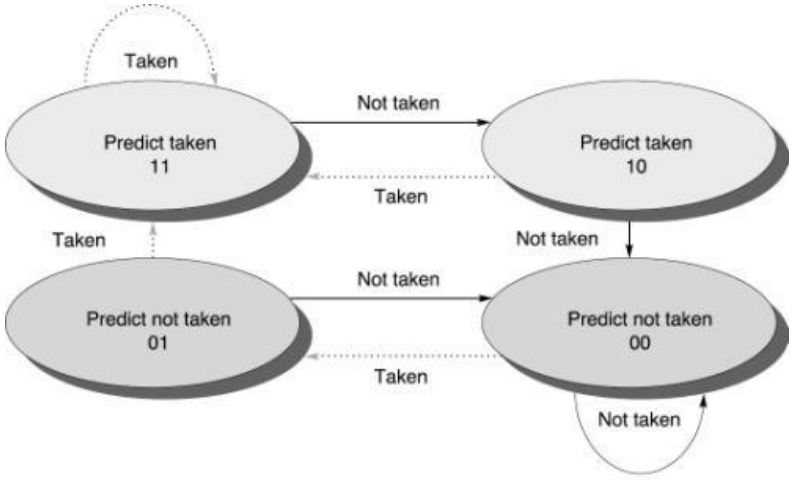
På sidan 183 i *Computer Architecture: A Quantitative Approach* [7] uttrycker författarna att hårdvarubaserad spekulering är en förening av 3 centrala koncept:
Den första är hopprediktion, som används för att förutsäga vilken som blir nästa instruktion. Den andra, spekulation, används istället för att exekvera de instruktioner som hämtats med hjälp av prediktion, fastän data-hinder inte är lösta. Det sista konceptet är dynamisk skedulering, som är nödvändigt för att till sist köra instruktionerna i en passande ordning.

2.2.1 Hoppförutsägelser

Hopprediktion kan indelas i två huvudkategorier: Statisk och dynamisk. Statiska hopprediktorer baserar sig endast på hopptypsinstruktionen och är därför elementära samt billiga att producera. Dynamisk hopprediktion använder sig alternativt av statistik från tidigare programkörningar för att gissa det troligaste instruktionen. För att uppnå detta används ofta någon form av buffertminne eller tabell. En ofta använd hopprediktor är den så kallade 2-bits prediktoren (se figur 2). Den anger om en förgrening ska väljas (Taken) eller inte (Not Taken) med hjälp av en 2-bits räknare i cache-bufferten för prediktoren. En förgrening är ofta ensidig, dvs är antingen ofta vald eller inte vald. 2-bits prediktoren sätts igång med en stark inklination till antingen ”Taken”- eller ”Not Taken”-fasen. Vid fel förutsägelse ändras den högra biten och inklinationen blir svag. Om nästa prediktion är rätt flyttas man tillbaka till stark inklination-fasen, och den högra biten växlas. Om prediktoren gissar fel igen, när det är i svag-stadiet så ändras den vänstra biten och växlar då prediktoren till den motsatta gissningens svaga stadie. Nutida processorer använder sig av mera avancerade, så kallade adaptiva prediktorer, vilka är anpassade till olika resultatmönster som kan uppkomma.

2.2.2 Reorder buffer

Då instruktioner exekveras spekulativt kan det finnas variationer i ordningen som de blir färdiga. Därför måste resultaten sparas på något ställe tills det är dags för instruktionen att köras. För att spara spekulativt skapade resultat används en *Re-order buffer* (**ROB**). Varje objekt i ROB innehåller 4 olika fält: Instruktionstyp, registervärdet för destinationen, värdet för instruktionen, och till sist ”ready”-fältet som anger om instruktionen har exekverat färdigt. Då en instruktion visar sig vara felbedömd; inkorrekt antagande har gjorts och fel förgrening har exekverats i förväg, tömmer ROB alla artiklar som kommer efter objektet som gissats fel. Därefter hämtas nästa instruktion från den rätta förgreningen. Dessa ”missprediktioner” kan ha stora konsekvenser för prestandan av en processor. Av den orsaken har noggrannheten av förutsägelserna stor betydelse.



Figur 2: Diagram för 2-bits prediktoren.

2.3 CPU Cache

Det tar en förhållandevis lång tid för en processor att hämta data från RAM-minnet. För att försnabba minnesåtkomsten finns det ett litet inbyggt minne i processorn kallat *cache*. I cache-minnet sparas värden som anses vara nödvändiga för instruktionen. Detta kan vara värden som snart skall hämtas, eller som har hämtats tidigare. Hämtning av instruktioner i förväg kallas på engelska för ”*prefetching*”. Om data som eftersöks inte finns i cache-minnet, sker en så kallad *cache-miss*. I detta fall hämtas värdet från

huvudminnet, vilket tar en längre tid. Om processorn använder sig av dynamisk exekvering så kan andra instruktioner köras under den tid som data hämtas från huvudminnet. I cacheminnet sparas data inuti fält av bestämd storlek, kallade för *cache-linjer*. Dagens processorer använder sig av flera nivåer av cacheminnen.

2.4 Adressrymd

I cache-minnen sparas även översättningstabeller som används för att kartlägga adresser i det fysiska minnet till *virtuella minnet*. Varje program har sin egen *adressrymd* (*address space*), vilket är omfånget av adresser som är giltiga för processen och tillgängliga i minnet. Tanken med att använda virtuella minnen och översättningstabeller är att man processer kan bli ”skyddade” från varandra genom att ha sina egna tabeller med adresser som pekar till olika sidor i minnet. Program bör alltså vara isolerade från varandra, och inte komma åt eller påverka data som finns utanför deras egen adressrymd. Behörighetsnivåer, kallade för *ringar*, har också introducerats för att ytterligare isolera processer. Det kan finnas upp till 8 nivåer av ringar, där nivå 0 är ämnad för processer med de största rättigheterna [12].

3. MELTDOWN

Det är nu dags att presentera avhandlingens centrala tema, Meltdown.

3.1 Definition

Meltdown är namnet på sårbarheten som avslöjades i början av 2018. Det är inte fråga om ett visst program, men istället tillvägagångssättet med vilket som ett skadeprogram kan hämta data. Med hjälp av en Meltdown-attack kan program bryta mot behörighetsskyddet i adressrymden, och på detta sätt avläsa värden sparade i kärnan eller molntjänster; värden som programmet inte har tillstånd eller rättighet att läsa. Spectre, som offentliggjordes samtidigt som Meltdown, möjliggör också läsning av data som inte är åtkomligt för vanliga program, men utnyttjar istället andra applikationer för att erhålla data sparade i minnet [13]. Det trovärdigaste målet med båda typerna av attacker skulle vara att komma åt personlig information från målpersonens dator, telefon, surfplatta eller datalagringstjänst.

Det är värt att notera att det finns dåligt med bevis på att Meltdown- och Spectre-säkerhetshålen skulle ha utnyttjats på lyckat sätt av andra parter än forskarna som upptäckte felen [14]. Detta beror delvis på att en dessa attacker inte lämnar några spår efter sig, ett faktum som forskarna intygat i frågor och svar-sektionen på hemsidan för Meltdown och Spectre [1]. I början av februari 2018 fanns det emellertid rapporter om att nästan 140 olika skadeprogram har försökt använda sig av sårbarheterna för att nå användarens känsliga data [15].

3.2 Jämförelse med andra skadeprogram

Eftersom lyckade Meltdown-attacker inte har bevisats ”i det verkliga livet” [16] så kan det vara svårt att jämföra dessa potentiella attacker med andra skadeprogram. Ett sannolikt sätt för att få ett program som utnyttjar Meltdown på målpersonens dator skulle vara genom att använda en *trojansk häst*, dvs en fil som uppger sig vara något annat för att installeras på datorn utan användarens godkännande. Ett av förra årets mest sensationella skadeprogram, utpressningstrojanen *WannaCry*, krypterade all data på användarens data varefter det krävde en lösensumma för dekryptering [17]. Som motsats skulle en Meltdown-attack kunna köras utan att användarens kännedom. Till skillnad från många andra spionprogram, som försöker lura användaren att mata in information eller installera ett program som avläser till exempel knapptryckningar, så använder en Meltdown-attack sig av inneboende bristfälligheter i processorns arkitektur. Ett program som utnyttjar Meltdown skulle kunna avläsa privat information utan att den tilldelats speciella rättigheter. Möjliga Meltdown-angrepp är därför mycket svårare att upptäcka för antivirusprogram än ett ”vanligt” spionprogram eller mask. Enligt Meltdown-forskarna är program som använder sig av säkerhetshålet svåra att skilja från normala program, men antivirusprogram kan känna igen själva trojanerna som programmen är gömda inuti [1].

4. TRE STEGEN AV EN MELTDOWN-ATTACK

Meltdown-forskarna har delat in en möjlig attack i tre steg, definierat på följande sätt:

1. Innehållet från en minnesplats som angriparen valt laddas in i ett register. Minnesplatsen är inte tillgänglig för programmet.
2. En obeständig instruktion ändrar en cache-linje baserat på värdet i minnesplatsen.
3. Med hjälp av cache-attacker avläser angriparen vilken cache-linje som ändrades i steg 2. På basen av detta kan det hemliga värdet i minnesplatsen bestämmas.

I detta kapitel kommer vi att ta en noggrannare titt på de olika stegen av en attack, och därefter i nästa kapitel vidare förtydliga dessa med ett exempel. All information är baserad på stegen specificerade i kapitel 5 i Meltdown-artikeln, publicerad den 3 januari 2018 [9].

4.1 Steg 1: Avläsning av minnesplats

Det första steget i en Meltdown attack grundar sig på obeständiga instruktioner, som exekveras i förväg för att spara tid. Med hjälp av dessa spekulativt exekverande instruktioner kan man lura processorn att köra instruktioner som inte borde vara möjliga. Principen är följande: Skapa en obeständig instruktion som hämtar data från en minnesplats som programmet inte har rättighet till. Då en process inte har befogenhet till en minnesplats kommer det att uppstå ett *undantag* (*exception*). Dessa leder oftast till att programmet avslutar sin exekvering, men Meltdown-forskarna har listat två sätt att kringgå detta problem.

4.1.1 Undantagshantering

Ett sätt att hindra undantag är genom att förgrena applikationen före den hämtar värden från den ogiltiga minnesadressen. Då skulle avhämtningen av de förbjudna värdena ske av barnprocessen till den ursprungliga applikationen. Denna barnprocess skulle exekveras färdigt före programmet slutar att köras på grund av undantag.

4.1.2 Dämpning av undantag

Genom att ”dämpa” undantaget försöker man hindra att det uppstår några undantag överhuvudtaget. Ett möjligt sätt är att skapa instruktionen som hämtar data från minnet in i en villkorlig hoppinstruktion (t.ex. en om-sats) med ett villkor som aldrig uppfylls. Eftersom principen bakom spekulativ exekvering är att i förväg exekvera möjliga förgreningar, så kan denna instruktion exekveras spekulativt. Resultatet, innehållande undantaget orsakad av obehöriga minneshämtningen, blir sparat i ROB, och körs endast om hoppinstruktionen körs. Undantaget uppstår alltså inte alls, eftersom villkoren till förgreningen inte någonsin kommer att uppfyllas. Instruktionen körs inte, men dess spekulativt hämtade resultat är fortfarande sparat i ROB. Enligt forskarna kan dämpning av undantag kräva ”övning” av prediktorn, så att prediktorn säkerligen kommer att välja förgreningen som attackeraren har skapat.

4.1.3 Race condition

Då instruktionerna är färdigexekverade, börjar avbrott och undantag som uppstått hanteras i ROB. Meltdown-attacken tävlar nu med att komma till steg 2 förrän undantagen får programmet att stanna.

4.2 Steg 2: Det hemliga värdet överförs från minnet

Om det första steget har lyckats, det vill säga programmet har avläst informationen från minnesplatsen utan att stanna på grund av undantag, så måste värdet överföras till ett ställe var attackeraren kan avläsa det.

Meltdown-forskarna har lyckats med detta genom att i den obeständiga instruktionen även utföra en beräkning utgående från värdet som hämtats från minnesplatsen. Baserat på det värdet som uträknats görs sedan en åtkomst till cache-minnet. I artikeln har man som beräkning valt att multiplicera det gömda värdet från minnesplatsen med storleken på sidorna i cache-minnet, i detta fall 4 KB. Detta värde är relativt stort, och är valt för att åtkomsterna i cache-minnet ska ha stort avstånd från varandra. Stort avstånd mellan åtkomster minskar risken för att närstående minnesplatser i cache-minnet är i användning på grund av ”prefetching”, vilket underlättar steg 3. Det hemliga värdet är

nu inte endast sparat i den spekulativt exekverade koden, utan det finns även nu sparat som ett tillstånd i cache-minnet, som kommer att avläsas i steg 3.

4.2.1 I fall av 0

Den obeständiga instruktionen tävlar fortfarande med att köras färdigt före undantag uppstår. Om ett undantag har skett samtidigt som programmet försöker avläsa värdet från minnesplatsen i steg 1, så kommer registret verka vara nollställt för Meltdown-programmet. Registret ger värdet 0 eftersom undantag som inte hanteras avbryter pågående instruktioner i minnesplatsen. För att Meltdown-applikationen inte ska fortsätta med det felaktiga värdet 0, ska det kontrollera att det hämtade värdet är olika med noll. Som lösning till detta möjliga bekymmer har forskarna infört en kontroll, som startar om programmet från steg 1 ifall värdet som hämtas visar sig vara lika med noll.

4.3 Steg 3: Det hemliga värdet mottas

I början av den tredje och sista fasen i en Meltdown-attack är det gömda värdet sparat i form av ett tillstånd i cache-minnet. För att få värdet tillbaka i ett tillstånd där det kan avläsas har forskarna använt *cache-attacker*. Dessa typer av attacker kan utföras av helt skilda processer, eftersom de inte är beroende av spekulativt exekverade instruktioner.

4.3.1 Cache-attacker

Cache-attacker utnyttjar cache-minnets huvudsakliga funktion, en ökad snabbhet vid hämtning av värden. Det finns olika tekniker som kan tillämpas, men den metod som Meltdown-forskarna har brukat kallas för **Flush+Reload**. Denna form av cache-attack ”spolar” den valda adressen i primärminnet, och mäter därefter tiden det tar för minnet att återigen hämta värdet. Baserat på tiden det tar, kan attackeraren avgöra det är fråga om en cache-träff, eller en cache-miss.

I Meltdown-attacken utnyttjas Flush+Reload för att bestämma vilken adress i cache-minnet som användes i steg 2. Attackeraren kan gå igenom alla sidor av minnet som värdet skickats till för att fastställa vilken minnesplats som använts av den obeständiga

instruktionen. Då minnesplatsen har hittats, kan det hemliga värdet kan härledas utgående av adressen, eftersom dess värde är det gömda värdet multiplicerat med ett annat tal, vilket är bekant för attackeraren (4 KB användes i artikeln).

4.4 Attacken avslutas

Meltdown-programmet har nu hämtat data från en minnesadress som den inte har rättighet till. Säkerhetsgränser som drivs igenom på arkitekturnivå i processorn har förbigåtts. Enligt undersökarna kan dessa tre steg upprepas för att till slut kopiera alla filer, inte enbart från kärnminnet, utan hela fysiska minnet. I artikeln påstår forskarna ha kommit upp till en överföringshastighet på 503 KB/s, med felfrekvensen 0,02%. Dämpning av undantag har en negativ inverkan på hastigheten, medan Flush+Reload-attacken är den största flaskhalsen i attacken, eftersom största tiden av programmet går till dessa cache-riktade attacker.

4.5 Skillnader i processorarkitektur

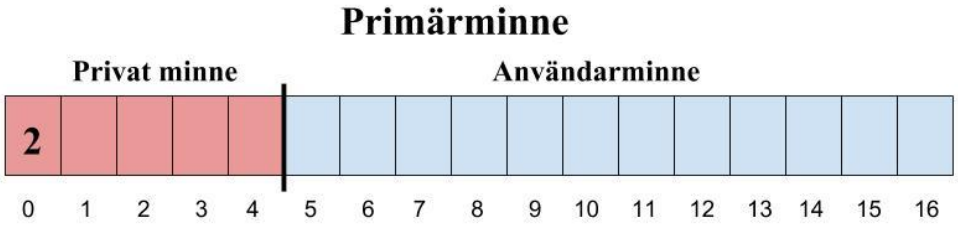
Forskarna har lyckats upprepa Meltdown-attacker på alla datorer med processorer tillverkade av Intel, oberoende av operativsystem [9]. Amd:s processorer har undersökarna i sin tur inte lyckats besegra. Den specifika komponenten som orsakar skyddet har förblivit aningen oklar. Forskarna spekulerar i artikeln att immuniteten kan bero på att tillvägagångssättet för attacken är för långsam för dessa processorer, och att undantagen då hinner uppstå innan koden hunnit köras färdigt. En annan tänkbar orsak som tas fram är att vissa nödvändiga delar, så som ROB, inte finns i alla processorer. Forskarna har dock bevis på att dynamisk exekvering kan användas för att nå minnesadresser i privatminnet i en Amd-processor [9].

Amd-bolaget har själv inte bjudit på något fördjupat svar. I artikeln som företaget publicerade den 11 januari 2018 [18] finns en respons till Spectre och Meltdown. Medan Spectre är ett möjligt hot, så anser Amd att sättet de hanterar skydd av privilegienivåer i sidor av minnet leder till ”noll sårbarhet” för Meltdown i deras produkter.

5. EXEMPEL PÅ MELTDOWN-ATTACK

Som avslutande exempel kommer en simplifierad version av en Meltdown-attack att visas. Det är värt att betona att en attack i verkligheten skulle vara mer avancerad, med betydligt större mellanrum mellan privat-och användarminnet. I en Meltdown-attack får inte närliggande minnesplatser vara sparade i cache-minnet

Vi antar att det fysiska minnet är strukturerat på sättet specificerat i figur 3. Det finns en tydlig gräns i det fysiska minnet mellan användarminnet som är tillgängligt för alla program, och det privata minnet, vilket endast används av processer med rätta privilegier.

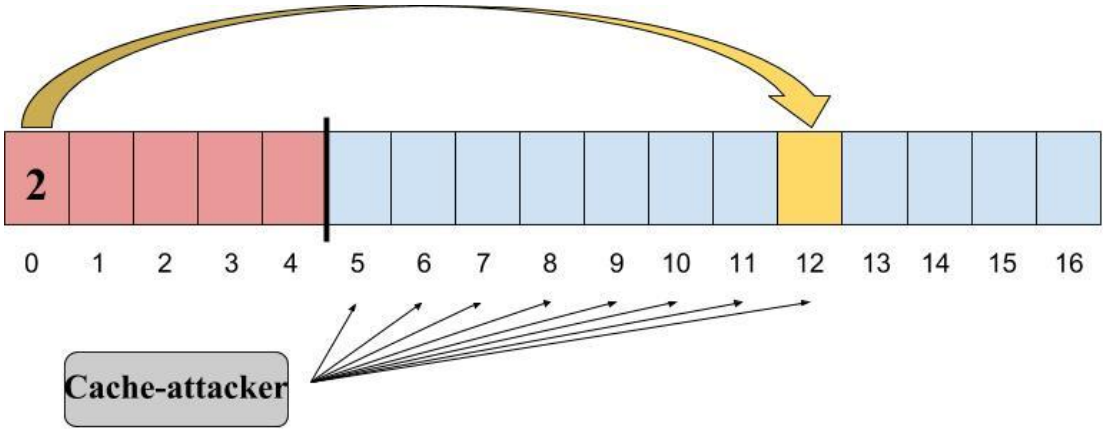


Figur 3: Förenklad version av det fysiska minnet.

Från det privata minnet vill attackeraren komma åt värdet som är sparat i minnesplats 0. För att åstadkomma detta använder Meltdown-programmet kod, som hämtar data från plats 0. Kodsnutten i fråga läggs in i en om-sats, en villkorlig hoppinstruktion som leder till en förgrening. Attackeraren måste se till att prediktorn väljer att exekvera koden spekulativt, det vill säga i förväg. I detta fall känner attackeraren till storleken på **array1**-räckan, och vet därför att värdet av x, som är lika med noll, aldrig kommer att vara mindre än storleken av array1. På detta sätt kommer hämtningen av data endast ske spekulativt. När programmet körs fysiskt kommer den att hoppa över om-satsen, vilket leder till att undantag som skapats av ogiltig hämtning av minen dämpas.

```
//Exempel (pseudokod)
//Attackeraren vet värdet på x och array1
if (x < array1.size){
     array2[array1[x] * 6]
}
```

En aritmetisk beräkning med värdet i minnesplats 0 görs: Talet multipliceras med 6. Resultatet, alltså siffran, blir adressen i primärminnet som kommer att användas, illustrerat i figur 4. Cache-minnet, som sparar värden vilka nyss använts, läser då även in adressen på motsvarande linje som i huvudminnet. I exemplet kalkyleras adressen i primärminnet till 12.



Figur 4: Användarminnet utsätts för cache-attacker.

Användarminnet börjar nu utsättas för cache-attacker av typen Flush+Reload. Dessa kan göras av ett helt annat program. Vid varje minnesplats mäts tiden det tar att hämta data från valda adressen. Då programmet kommer till adress nr.12 märker det att tiden var betydligt snabbare än för de andra adresserna i primärminnet. Detta tyder att värdet för minnesplats 12 också måste vara sparat i cache-minnet. Minnesplats 12 är följaktligen den adress vart värdet har skickats i föregående steg. Då attackeraren vet adressen, kan det hemliga värdet räknas ut genom att adressens nummer divideras med talet man multiplicerade med tidigare. ***12 / 6 = 2***. Det privata värdet måste således vara 2.

4.6.1 Exempel på minnesdump

Genom att iterera igenom hela privata minnet kan en attackerare avläsa lösenord och andra känsliga data. I figur 5, taget från Meltdown-artikeln [9], visar forskarna hur det kan se ut då en Meltdown-attack dumpar minne från *Firefox*-webbläsaren i en dator med operativsystemet *Ubuntu* 16.10. Man kan tydligt se sparade lösenord, så som **Dolphin18** och **insta_0203**, listade i den högra spalten. Raderna med XX-tecken tyder på att Flush+Reload-attacken inte gav något resultat till dessa bytes.

```
f94b76e0: XX XX XX XX XX XX XX XX XX  XX XX XX XX XX XX 81 |...............|
f94b76f0: 12 XX e0 81 19 XX e0 81 44  6f 6c 70 68 69 6e 31 |........Dolphin1|
f94b7700: 38 e5 e5 e5 e5 e5 e5 e5 e5  e5 e5 e5 e5 e5 e5 e5 |8...............|
f94b7710: 70 52 b8 6b 96 7f XX XX XX  XX XX XX XX XX XX XX |pR.k............|
f94b7720: XX XX XX XX XX XX XX XX XX  XX XX XX XX XX XX XX |................|
f94b7730: XX XX XX XX 4a XX XX XX XX  XX XX XX XX XX XX XX |....J...........|
f94b7740: XX XX XX XX XX XX XX XX XX  XX XX XX XX XX XX XX |................|
f94b7750: XX XX XX XX XX XX XX XX XX  XX e0 81 69 6e 73 74 |............inst|
f94b7760: 61 5f 30 32 30 33 e5 e5 e5  e5 e5 e5 e5 e5 e5 e5 |a_0203..........|
f94b7770: 70 52 18 7d 28 7f XX XX XX  XX XX XX XX XX XX XX |pR.}(...........|
f94b7780: XX XX XX XX XX XX XX XX XX  XX XX XX XX XX XX XX |................|
```

Figur 5: Exempel på kopiering från privata minnet med hjälp av Meltdown.

6. SKYDDSÅTGÄRDER MOT MELTDOWN

Det är svårt att skydda sig mot potentiella Meltdown-attacker. Dels för att de är näst intill omöjliga att spåra, dels för att de utnyttjar en svaghet i processorns arkitektur, vilket inte kan åtgärdas på något enkelt sätt med hjälp av mjukvaruuppdateringar. En spartansk lösning skulle vara att helt enkelt inte använda sig av spekulativ exekvering alls. Detta skulle dock leda till en betydande minskning i processorns prestanda. Dynamisk exekvering, något som dagens datorer har utformats till att tillämpa, skulle inte längre vara genomförbart. En fullständig korrigering av säkerhetshålet kräver att processorer designas på nytt med detta i åtanke. Det finns emellertid tillfälliga lösningar. Programfixen som forskarna i Meltdown-artikeln tar fram kallas **KAISER**, och har verkställts av alla stora teknikföretag skyndsamt efter publiceringen av artikeln [19].

6.1 KAISER

Kaiser, som även döpts om till ”*Kernel page-table isolation”* i vissa operativsystem [20], är en programfix riktad till kärnat i operativsystemet. Den ursprungliga tanken med rättelsen var skydda datorer från cache-attacker genom att hindra program från att bryta det slumpmässiga arrangemanget av adresser i kärnan. Forskarna har upptäckt att Kaiser även förhindrar Meltdown-attacker. Programfixen gör det möjligt för kärndata att inte kartläggas i användarminnet där det är tillgängligt för potentiellt skadliga

program. Det finns då inte längre några pekare till kärnminnet, som kan avslöja hur mycket adressen förskjutits av instruktionen i Meltdown-koden.

Det finns likväl svagheter med KAISER-programfixet som inte kan lösas. Moderna processorer är utformade enligt x86-arkitektur. Denna typ av arkitektur funktionerar endast om det i användarens adressrymd finns avbildningar av ett flertal ”privilegierade” minnesplatser. En Meltdown-attack skulle kunna utnyttja dessa kvarblivna adresser; ingen hemlig information finns att hämtas med från adresserna, men de kan innehålla pekare, vilka kan hjälpa att räkna ut siffran som använts i den slumpmässiga ordnandet av adresser. Om attackeraren känner till siffran, fungerar inte randomiseringen längre för att hindra en Meltdown-attack [9].

6.2 Åtgärder inom IT-branschen

Meltdown-forskarna hade redan på sommaren 2017 kontaktat framstående it-företag om deras upptäckt [9]. Därför var responsen till säkerhetshålet snabb när det avslöjades vid årsskiftet. Någon form av KAISER -rättelsen har lanserats av alla företag vars produkter kan utnyttjas av Meltdown. En del av företagen, t.ex. Microsoft och Apple, införde programfixar redan i slutet av 2017. Dessa fixar motiverades inte för allmänheten och orsakade minskad prestanda för användare, vilket gav upphov till spekulationer bland användare [21].

Intel, ett av företagen som förlorat mest på upptäckten av Meltdown säkerhetshålet, publicerade den 15 mars 2018 ett blogginlägg skrivet av verkställande direktören Brian Krzanich [22]. I inlägget redovisar Krzanich företagets framtida hantering av Spectre och Meltdown. Planen är indelad i två huvudsakliga delar: Kontinuerlig uppdatering av nuvarande processorer, och ändringar i designen av hårdvara i framtiden. Krzanich påpekar att programfixar har lanserats till alla produkter som kommit ut dom senaste 5 åren. Både Intels nya Xeon- och framtida åttonde generationens-processorer kommer att använda sig av arkitektur vilket är modifierat med tanke på både Meltdown och Spectre. Produkterna planeras att lanseras på hösten 2018, och kommer att använda sig av ytterligare skyddsåtgärder för att separera program från varandra.

Tillvägagångssättet specificerat av Intel verkar vara det vanligaste sättet att hantera Meltdown: Släpp programfixar till hårdvara som redan används, och se till att säkerhetshålet är täppt i alla nya produkter.

7. SLUTSATSER

6.1 Konsekvenser för industrin

Elektronikföretaget Intel, som av många anses vara den största processortillverkaren på marknaden, har utsatts för negativa efterverkningar på grund av upptäckten av Meltdown och Spectre. Omedelbart efter artiklarnas publicering sjönk värdet på företagets aktier med över 3%, medan motståndaren Amd upplevde en höjning på upp till 5% [23]. Ett misstag av denna kaliber påverkar oundvikligen allmänhetens syn på företaget. Programfixarna som introducerats för att rätta till Meltdown-buggen har haft negativa inverkan på processorernas effektivitet [24]. Det är dessutom oklart om företagets framtida produkter kommer att lida av samma effektförluster. Tiden är nu inne för konkurrenter att ta över marknaden

Attacker som utnyttjar spekulativ exekvering har klassificerats som en ny klass av sårbarheter av Microsoft. I artikeln ”*Speculative Execution Bounty Launch*”, publicerad den 14 mars 2018 [25] lanserade företaget ett belöningsprogram för att uppmuntra vidare forskning i ämnet. Microsoft är redo att betala upp till 250 000 dollar till den som upptäcker ytterligare bristfälligheter gällande spekulationer av instruktioner.

6.2 Egna reflektioner

Meltdown kommer troligtvis i framtiden att ses på som ett intressant kapitel i datorernas historia. Säkerhetshålet rättades hastigt efter att artikeln publicerades tidigt i januari 2018. Fastän hotet är borta, så har upptäckten av faran lett till ett uppvaknande i hela industrin. Det är fråga om ett grovt fel i designen av arkitekturen, som inte upptäckts på över 10 år. Kan det då finnas andra aspekter av dagens datorer; förbiseenden i arkitekturen, som kan utnyttjas för att använda datorer på andra sätt än tillverkarna har avsett? Den snabba takten i utvecklingen av hårdvara och den hårda

konkurrensen inom industrin kan leda till små försummelser som inte upptäcks förrän mycket senare.

Det är värt att påpeka att det är svårt att tala om påverkan av Meltdown utan att även ta i beaktan Spectre.

8. KÄLLFÖRTECKNING

[9] “Meltdown”, Moritz Lipp, Michael Schwarz, Daniel Gruss, Thomas Prescher, Werner Haas, Stefan Mangard, Paul Kocher, Daniel Genkin, Yuval Yarom, Mike Hamburg, 3.1.2018. Tillgänglilg vid:
https://meltdownattack.com/meltdown.pdf
[Åtkomstdatum:
27.2.2018]

[1] https://meltdownattack.com/
[2] https://www.theguardian.com/technology/2018/jan/04/meltdown-spectre-worst-cpu-bugs-ever-found-affect-computers-intel-processors-security-flaw
[3] https://www.marketwatch.com/story/these-companies-are-most-exposed-to-meltdown-spectre-breaches-2018-01-19
[4] https://www.bleepingcomputer.com/news/security/google-almost-all-cpus-since-1995-vulnerable-to-meltdown-and-spectre-flaws/
[5] https://www.amd.com/en/corporate/speculative-execution
[6] https://newsroom.intel.com/news/firmware-updates-and-initial-performance-data-for-data-center-systems/
[7] https://support.apple.com/en-us/HT208394
[8] https://aws.amazon.com/security/security-bulletins/AWS-2018-013/
[9] https://meltdownattack.com/meltdown.pdf
[10] A Practical Introduction to Computer Architecture, Daniel Page, Springer, 2009
[11] Computer Architecture: A Qantitative Approach; Fifth Edition, John L. Hennessy & David A. Patterson, Elsevier, 2012
[12] http://www.brokenthorn.com/Resources/OSDev12.html
[13] https://spectreattack.com/spectre.pdf
[14] http://www.dell.com/support/contents/en/en/fibsdt1/article/product-support/self-support-knowledgebase/software-and-downloads/support-for-meltdown-and-spectre
[15] http://www.eweek.com/security/meltdown-spectre-malware-samples-emerge-though-few-attacks-follow

[16] https://www.virusbulletin.com/blog/2018/02/there-no-evidence-wild-malware-using-meltdown-or-spectre/
[17] https://www.csoonline.com/article/3227906/ransomware/what-is-wannacry-ransomware-how-does-it-infect-and-who-was-responsible.html
[18] https://www.amd.com/en/corporate/speculative-execution-previous-updates#paragraph-337801
[19] https://www.bleepingcomputer.com/news/security/list-of-meltdown-and-spectre-vulnerability-advisories-patches-and-updates/
[20] https://lwn.net/Articles/741878/
[21] https://twitter.com/aionescu/status/930412525111296000
[22] https://newsroom.intel.com/editorials/advancing-security-silicon-level/
[23] https://qz.com/1171391/the-intel-intc-meltdown-bug-is-hitting-the-companys-stock-big-time-while-rival-amd-is-soaring/
[24] https://newsroom.intel.com/editorials/intel-security-issue-update-initial-performance-data-results-client-systems/
[25] https://blogs.technet.microsoft.com/msrc/2018/03/14/speculative-execution-bounty-launch/

Figur 1:
https://upload.wikimedia.org/wikipedia/commons/2/21/Fivestagespipeline.png

Figur 2:
https://people.engr.ncsu.edu/efg/521/s06/common/lectures/notes/lec15.html

Figur 5:
https://meltdownattack.com/meltdown.pdf , Listing 4