Západočeská univerzita v Plzni

Databázové systémy a metody zpracování informací 2

KIV/DBM2

Statistické zpracování dat o léčbě mozkových příhod

Autor
Marek Lovčí

11. listopadu 2019

FAKULTA
APLIKOVANÝCH VĚD
ZÁPADOČESKÉ
UNIVERZITY
V PLZNI

Obsah

1 Zadání **2**

2 Spekulativní exekuce instrukcí **2**

3 Meltdown **2**

4 Spectre **2**
4.1 První varianta . . . 2
4.2 Druhá varianta . . . 3

5 Foreshadow a další varianty Spectre a Meltdown **3**
5.1 Objasnění pojmů a vztažení nové kategorizace na původní . . . 4
5.1.1 Kategorizace *Meltdown* . . . 4
5.1.2 Kategorizace *Spectre* . . . 4
5.2 Negativní výsledky . . . 4
5.2.1 Meltdown-AC . . . 5
5.2.2 Meltdown-DE . . . 5
5.2.3 Meltdown-SS . . . 5
5.3 Ostatní útoky typ Meltdown . . . 5
5.3.1 Meltdown-NM . . . 5
5.3.2 Meltdown-PF . . . 5
5.3.3 Meltdown-BR . . . 6
5.3.4 Meldown-GP . . . 6
5.4 Útoky typu Spectre a pojmy . . . 6
5.4.1 Spectre-PHT . . . 6
5.4.2 Spectre-BTB . . . 7
5.4.3 Spectre-STL . . . 7
5.4.4 Spectre-RSB . . . 7

6 Závěr **8**

1 Zadání

Je dán anonymizovaný dataset o výsledcích léčby pacientů s mozkovou příhodou ve FN Plzeň za 2016-2017. U pacientů je mimo jiné identifikován etiologický typ mozkové mrtvice, klinický výsledek (škála mRS), míra vstupního deficitu (škála NIHSS), přítomnost rizikových faktorů. Data nejsou úplná, často jsou některé hodnoty neznámé.

Prvním úkolem je zhodnotit závislosti dobrého klinického výsledku (mRS-out: hodnota 1 oproti 0), případně mortalitu (mRS – 1Y: hodnota 6 oproti 0-5) jednorozměrnou regresí na věku, pohlaví, vstupního NIHSS (TSS příjem) a rozdílu výstupní-vstupní NIHSS. Druhý úkol spočívá ve vytvoření Kaplan–Meierových křivek přežití pro jednotlivé stanovené diagnózy a podrobně popsat proces vytvoření a její vlastnosti.

2 Spekulativní exekuce instrukcí

Spekulativní provádění instrukcí je optimalizační technika, při které počítač (procesor) vykonává dopředu úkoly, aniž by věděl, že budou v budoucnu opravdu potřeba. To provádí s cílem předejít zpoždění.

Dnešní počítače používají jednu z forem spekulativní exekuce - prediktivní exekuci. Ta za pomoci mechanismů, jako např. „branch predictor“ odhadne větvení programu a danou cestu předpočítá. Jestliže se předpověď ukáže jako pravdivá, je „commitnuta“. V případě že se ukáže, že předpověď byla špatná, je nutné výpočet zahodit a vykonat správné instrukce. V případě, že se předpověď prediktoru ukáže jako správná, tak vede ke zrychlení běhu programu.

Dalšími formami spekulativní exekuce jsou např. „greedy execution“ (při které se předpočítají obě možnosti podmíněného větvení) nebo „lazy evaluation“, která sama o sobě nezahrnuje žádnou spekulaci. Tyto koncepty se využívají ve světě softwaru a o jejich praktické využití pro hardware jsem se nezajímal.

3 Meltdown

Moderní operační systémy zajišťují tzv. paměťovou izolaci - uživatelské programy (procesy) nemohou číst z paměťového bloku vyhrazeného pro jiné procesy, obzvláště ne z bloku vyhrazeného pro jádro operačního systému. *Meltdown* toto omezení zcela překonává a to na hardwarové úrovni. Útočník napíše kód, který se spustí během spekulativního procesu. Kód čte data z paměťových bloků, ze kterých nemá práva číst, jelikož ale vše probíhá během spekulativního okna, tak tyto práva jsou teprve kontrolována. Po dokončení kontroly práv je zjištěno, že program práva na čtení z dané paměti nemá a data jsou zahozena. Útočník následně provede tzv. *Flush+Reload* útok (*cache side-channel attack*), kterým zjistí nejen kam se zahozená data uložila, ale rovnou i obsah daných paměťových buněk.

Pro útočníka by bylo velice výhodné číst data, které zpracovává jádro operačního systému. Před zveřejněním diskutovaných hrozeb využíval *Linux* randomizaci adresního prostoru *kernelu (systémového jádra)* (tzv. *KASLR*), ale tento zabezpečovací mechanismus se dal relativně snadno obejít. Nyní však implementuje vylepšení (tzv. *KAISER* nebo *KPTI*), který zajišťuje, že tabulka indexující uživatelský adresní prostor obsahuje pouze naprosto nezbytné odkazy do adresního prostoru *systémového jádra operačního systému Linux*. Toto opatření se ukázalo jako dostačující ochrana proti útokům typu *Meltdown*. Nyní ale systém musí provádět kroky navíc, vždy, když uživatelská aplikace požádá o akci operačního systému (tzv. systémové volání). Tím se přichází o část výkonu, která je v některých případech až v řádu desítek procent (z nalezených dat se zdá, že snížení výhonu je ve většině případů *cca* 5 %, ale ve výjimečných případech může dosáhnout až 30 %). Problém s tímto nebezpečím se netýká všech procesorů. Nejvíce ovlivněným výrobcem (avšak ne jediným) je *Intel*.

4 Spectre

V původní práci o útoku typu *Spectre* jsou popsány jeho dvě základní varianty. Důležitá informace je ta, že na rozdíl od *Meltdown* neumožňuje *Spectre* číst data z celého adresního prostoru, ale pouze z prostoru vyhrazeného pro danou aplikaci. I tak ale může dojít k odcizení citlivých informací.

4.1 První varianta

První varianta je velice podobná útoku typu *Meltdown*, kdy může útočník při spekulativním vykonávání kódu načíst data na která nemá práva a po jejich zahození je přečíst z *cache* procesoru.

Útočník nechá při trénovací fázi načítat procesor data, která jsou pak použita k lokalizaci jiných dat. Dotazy na data v rozsahu na která má práva to ale nekončí a útočník se dál dotazuje i na data mimo povolený rozsah. Procesor v rámci časových úspor načte i data mimo povolený rozsah. Po zjištění nesrovnalosti procesor data zahodí.

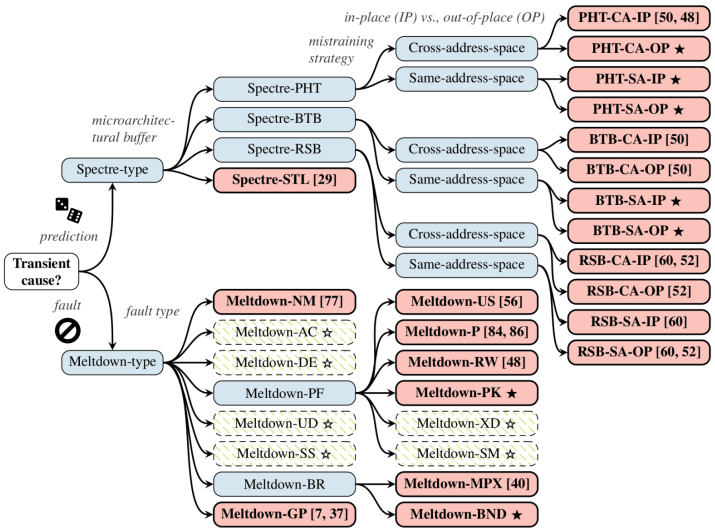
4.2 Druhá varianta

Touto variantou útoku lze natrénovat tzv. „Branch Target Buffer“ tak, že začne provádět kód - *gadget*, jenž bude mít přístup k datům, která jsou předmětem zájmu a která by měla být nepřístupná. Výsledky jsou opět potlačeny, ale vedlejší účinky provedení daných instrukcí zůstávají.

5 Foreshadow a další varianty Spectre a Meltdown

Základními variantami *Spectre* a *Meltdown* problémy s architekturou procesorů bohužel nekončí. V průběhu roku 2018 přibyly další výzkumy a objevy variant a nových typů zranitelností na *CPU*. Takovými případy jsou *Foreshadow*, známý též jako „L1 Terminal Fault (**L1TF**)“, nové varianty *Spetre* a *Meltdown* zmíněné práci „A Systematic Evaluation of Transient Execution Attacks and Defenses“ ale i problémy vzniklé kombinací různých chyb - například *Spectre v2* a *Hyper-Threadingu*.

> Problém, o kterém je nyní řeč, se týká využitelnosti chyby **Spectre v2** v kombinaci s děravým *Hyper-Threadingem* v procesorech *Intel*. Ty totiž mají tu nepříjemnou vlastnost, že *HT* vlákna běžící na jednom fyzickém *CPU* jádru sdílejí datovou *L1 cache*, takže za pomoci **Spectre v2** si může daný proces číst data procesu druhého. Do *linuxového* jádra tak zamířily *patche*, které měly tento problém řešit. Záplatě se obecně říká *STIBP (Single Thread Indirect Branch Predictors)* a najdeme ji v **Linuxu 4.20**.
>
> *Phoronix* tuto verzi přeměřil, přičemž dospěl k hodně smutným číslům: záplata snižuje procesorový výkon v mezních případech až o 50 %, tedy na polovinu. Vzhledem k efektivitě *Hyper-Threadingu* v současných *CPU Intelu* lze říci, že výkonnostní dopad aplikace *STIBP* je přibližně stejně dramatický, jako samotné vypnutí *Hyper-Threadingu*.



Obrázek 1: Klasifikační strom s demonstrovanými (červené, tučné), s negativními výsledky (zelené, přerušované) a nově prezentovanými útoky

13. listopadu 2018 byl vydán článek shrnující dosud existující varianty útoků. Výzkumníci v tomto článku přidali i 8 nových variant útoku *Meltdown* a 5 variant *Spectre* včetně „proofs-of-concept“ pro 7 z nich - 6 variant *Meltdown* se zatím nepodařilo prakticky dokázat, vše je shrnuto v obrázku 1.

5.1 Objasnění pojmů a vztažení nové kategorizace na původní

Následující výčet pojmů je z převážné části přebrán z článku „Seven new Spectre and Meltdown attacks found“, ale je doplněn o některá chybějící data. Tento výčet by měl sloužit jako objasnění, kde se v nové kategorizaci nachází *Spectre-v1*, *Spectre-v2*, *Meltdown* a také jako přehled s vysvětlením použitých zkratek.

5.1.1 Kategorizace *Meltdown*

- Meltdown-NM: Floating Point Unit (FPU) state information leakage flaw; linux kernels that follow the "Lazy FPU Restore"scheme are vulnerable to the FPU state information leakage issue; device-not-available exception known as **#NM** exception is thrown
- Meltdown-AC: tried to exploit memory **alignment check** exceptions
- Meltdown-DE: tried to exploit **division** (by zero) **errors**
- Meltdown-PF: named after **Page Fault #PF** exception
  - Meltdown-US: reads kernel memory from **user space** on pipelined processors that do not transiently enforce the user/supervisor flag
  - Meltdown-P: attacking faults of pages without **present** bit
  - Meltdown-RW: this is in same out-of-order address access of otherwise **read or write** protected pages, where even content of read or write protected pages could be leaked
  - Meltdown-PK: bypasses memory **protection keys** on Intel CPUs
  - Meltdown-XD: tried to exploit non-executable memory
  - Meltdown-SM: tried to exploit the **supervisor mode** access prevention (SMAP) mechanism
- Meltdown-UD: tried to exploit invalid opcode exception - Invalid Opcode, known as **#UD**
- Meltdown-SS: tried to exploit out-of-limit segment accesses - **Stack-Segment** fault
- Meltdown-BR: exploits an x86 bound instruction on Intel and AMD - **Bound Range** Exceeded
  - Meltdown-MPX: MPX stands for **Memory Protection eXtensions**
  - Meltdown-BND
- Meltdown-GP: **General Protection** Fault

5.1.2 Kategorizace *Spectre*

Kategorizace útoků typu *Spectre* je založena na rozlišení toho, který z predikčních mechanismů je využit (*Pattern History Table* (PHT), *Branch Target Buffer* (BTB) nebo *Return Stack Buffer* (RSB)) a který mechanismus se využije k jeho natrénování (*in-place*, nebo *out-of-place*).

- Spectre-PHT: exploits the **Pattern History Table**
- Spectre-BTB: exploits the **Branch Target Buffer**
- Spectre-STL: exploits the CPUs memory disambiguation prediction, specifically **store-to-load** forwarding (STLF)
- Spectre-RSB: exploits the **Return Stack Buffer**

5.2 Negativní výsledky

Autorům článku *A Systematic Evaluation of Transient Execution Attacks and Defenses* se nepodařilo prokázat uskutečnitelnost útoků souvisejících s načítáním instrukcí do CPU - **Meltdown-XD** při kterém se pokusili spustit program (instrukce) nacházející v tzv. *non-executable memory* (část paměti obsahující strojový kód, při pokusu o načtení dat z jejího adresního prostoru je vyvolána výjimka) a **Meltdown-UD** (#UD - *Opcode exception*) pokoušející se procesor „donutit“ vykonat nevalidní strojové instrukce.

Meltdown-SM nesoucí svůj název podle tzv. „supervisor mode access prevention“ (SMAP) je neúspěšným pokusem o zneužití příznaku udávajícího že určitý program má práva využívat všechny, včetně privilegovaných instrukcí procesoru.

5.2.1 Meltdown-AC

Alignment Check Exception (#AC) - v překladu *výjimka kontroly zarovnání* je jedna z výjimek vygenerovaných procesorem při nastání chyby.

Nezarovnaný přístup do paměti (tzv. unaligned memory access) se vyskytne v případě pokusu o čtení N bajtů dat, které začínají na adresní pozici, která není rovnoměrně dělitelná N (tzn. $adresa\%N \neq 0$). To znamená, že například čtení 4 bajtů dat z adresy 0x10004 je v pořádku, ale čtení 4 bajtů z adresy 0x10005 by znamenalo nezarovnaný přístup do paměti a případné vyvolání výjimky.

Bylo zjištěno, že výsledky nezarovnaných přístupů k paměti nezanechávají data ve struktuře procesoru. Důvodem je pravděpodobně to, že #AC výjimka je vygenerována ještě před samotným vykonáváním přístupu do paměti. Tím pádem varianta Meltdown-AC není zranitelností a není zneužitelná.

5.2.2 Meltdown-DE

V článku *A Systematic Evaluation of Transient Execution Attacks and Defenses* byl popsán experiment, při kterém se pokusili získat data, která mohla být teoreticky zanechána v paměti procesoru po vygenerování chyby způsobené dělením nulou. Podezření se však nepotvrdilo - procesory po dělení nulou vytvoří výjimku (#DE) a nastaví příslušný registr (paměťový blok uchovávající výsledek operace) na nulu. Z této operace tedy neexistují žádná zneužitelná data.

5.2.3 Meltdown-SS

Posledním z nepotvrzených typů útoků cílí na zneužití „Stack-Segment Fault“ (#SS). *Stack-Segment* je dle definice oblast paměti vyhrazená pro automatické proměnné. Automatická proměnná je lokální proměnná, která je alokována a dealokována automaticky, když tok programu vstupuje a opouští rozsah proměnné.

Jako v předchozích případech, i pokus o přístup do této paměti vyvolá výjimku, k validaci podmínek pro její vyvolání však dochází před samotným přístupem k datům nacházejících se v paměti, kterou se útočník snaží diskreditovat. Nedochází tedy k žádnému úniku dat.

5.3 Ostatní útoky typ Meltdown

5.3.1 Meltdown-NM

Popis dle seznamu **CVE** (*Common Vulnerabilities and Exposures*): *software* využívají metodu pro obnovu stavu *FPU* (*matematického koprocesoru - floating point unit*), tzv. **Lazy FP** na procesorech od společnosti Intel, může potenciálně umožnit jednomu procesu, aby získal data z jiného procesu přes spekulativní kanál.

Do nedávna byl stav FPU na vyžádání ukládán a obnovován pomocí triku, při kterém použití FPU novým procesem způsobovalo chybu, která spustila logický obvod pro „save/restore“ na samotném matematickém koprocesoru. Při spuštění nového procesu se využíval starý stav FPU, který byl při prvním opravdovém využití matematického koprocesoru nahrazen správným. To se dělo s cílem snížit zatížení obvodu pro přepínání úloh.

Starý stav zanechává jistý otisk na struktuře procesoru, který je možné bočním kanálem extrahovat. Této logiky využívá útok Meltdown-NM. Tento problém byl na operačním systému Linux vyřešen přechodem na přímé ukládání a obnovu stavu matematického koprocesoru.

5.3.2 Meltdown-PF

Meltdown-PF sám spadá do kategorie útoků s *negativními výsledky*, většina jeho podvariant jsou však legitimními útoky. „Supervisor Mode Access Prevention“ (SMAP) je mechanismus, který zabraňuje procesům s vysokými právy (například procesy operačního systému) číst data z adresního prostoru procesů patřících uživateli, či jiné entitě s nižšími právy. Výzkumníci se obávali, že takový přístup mohl zanechat v paměti procesoru data, která by byla přístupná útočníkovi. Tuto domněnku se nepodařilo potvrdit, jelikož nebyla ve struktuře procesoru nalezena žádná data, která mohla být zanechána po vzniku výjimky. Stejně jako v případě *Meltdown-AC*, se výzkumný tým domnívá, že výjimka #PF (page-fault) je vyvolána ještě před vlastní manipulací s daty na fyzických adresních prvcích a tedy neexistují data, která by mohla být ponechána v paměti.

Meltdown-US Tato varianta je originálním útokem *Meltdown*. Pro jeho podrobnější popis viz kapitolu 3.

Meltdown-P Známý též pod názvem Foreshadow nebo L1TF. Princip spočívá v tom, že někde v paměti procesoru se nachází chráněný blok paměti nazývaný *enkláva*. Když se neautorizovaný požadavek pokusí do této paměti přistoupit, tak není vyvolána výjimka a místo dotazovaných dat se předají data fiktivní.

Díky tomuto mechanismu nefunguje pro získání dat z enklávy klasický útok Meltdown-US. Tabulka stránek (page-table) tvoří mapování mezi virtuální a fyzickou adresou paměti. Obsahuje pomocné informace, jako je například „present“ bit, který indikuje které stránky (shluky dat) se nacházejí ve fyzické paměti.

Při útoku *Meltdown-P* se smaže „present“ bit záznamu vedoucího do enklávy, díky tomu lze z tuto paměť načíst aniž by data byla nahrazena fiktivními daty. Data jsou tedy načtena, ale během jejich zpracovávání procesor zjistí, že došlo k neautorizovanému přístupu, je vyvolána #PF výjimka a data jsou zahozena stejným způsobem jako při útoku *Meltdown-US*.

Meltdown-RW Tento typ útoku je mezi *Meltdown* výjimečný tím, že se nesnaží získat data napříč úrovněmi privilegovaného režimu procesoru, ale překonává zabezpečení na dané bezpečnostní úrovni. Bylo zjištěno, že při spekulativním vykonávání instrukcí je ignorován příznak umístěný na *tabulce stránek* určující práva pro čtení a zápis. Útočník může tím pádem přepsat data určená jen pro čtení, čímž je zcela překonán softwarový bezpečnostní mechanismus zvaný „sandbox“, který se spoléhá na hardwarové vymáhání přístupu k datům určeným pouze pro čtení.

Meltdown-PK Tímto útokem lze číst data chráněná technologií Intel Protection Keys. Varianta byla prezentována a dokázána v článku *A Systematic Evaluation of Transient Execution Attacks and Defenses*.

Intel Protection Keys je technologie nacházející se na některých procesorech od společnosti *Intel* umožňující uživatelským procesům měnit práva tomuto procesu náležícím stránkám v tabulce stránek. To má za cíl umožnit vývoj bezpečnějších aplikací. Zajímavé je, že proti tomuto útoku neexistuje softwarová záplata - pokud uživatelský program používá *Intel Protection Keys*, tak je zranitelný. Chybu musí opravit *Intel* v nové verzi procesorů.

5.3.3 Meltdown-BR

Pro efektivní vývoj softwaru procesory poskytují nástroje jako je například výjimka *bound range exceeded (#BR)* upozorňující na iterování přes indexy mimo pole. *Meltdown-BR* poskytuje způsob, jak obejít kontroly na procesoru, které vyvolávají #BR výjimku. Tím je umožněno číst data za povolený rozsah odpovídající velikosti iterovaného pole a získat tak informace, ke kterým za normálních okolností nemá útočník přístup.

O variantách **Meltdown-MPX** a **Meltdown-BND** se mi nepodařilo zjistit dostatek informací, abych o nich mohl podrobněji referovat. O **Meltdown-MPX** pojednává původní článek vydaný Intelem, k **Meltdown-BND** jsem materiál nenalezl.

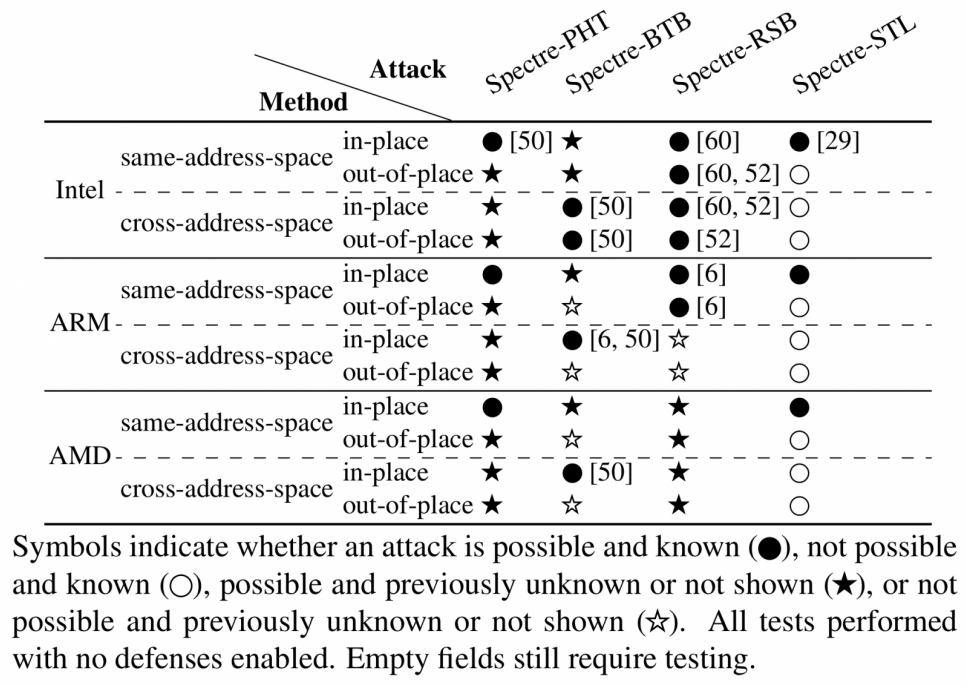
5.3.4 Meldown-GP

Tato varianta umožňuje útočníkovi číst data z registrů procesoru vyhrazených pro operační systém. Bylo zjištěno, že ačkoliv je po pokusu o přístup do takové paměti vyvolána #GP (*general protection fault*) výjimka, jsou data i přesto dostupná v případě, že výjimka vyvstala při spekulativním vykonávání instrukcí. Za normálních okolností nepřístupná data o systému mohou být získána postranními kanály (např. již zmíněným útokem *Flush+Reload*).

5.4 Útoky typu Spectre a pojmy

5.4.1 Spectre-PHT

PHT (Pattern History Table) je struktura sloužící jako prediktor výsledků operací. Tato struktura (tabulka) je předtrénována tak, aby načetla načetla data, na která nemá útočník právo. Tento útok je originální variantou útoku *Spectre-v1*.



	Method	Attack	Spectre-PHT	Spectre-BTB	Spectre-RSB	Spectre-STL
Intel	same-address-space	in-place	● [50]	★	● [60]	● [29]
		out-of-place	★	★	● [60, 52]	○
	cross-address-space	in-place	★	● [50]	● [60, 52]	○
		out-of-place	★	● [50]	● [52]	○
ARM	same-address-space	in-place	●	★	● [6]	●
		out-of-place	★	☆	● [6]	○
	cross-address-space	in-place	★	● [6, 50]	☆	○
		out-of-place	★	☆	☆	○
AMD	same-address-space	in-place	●	★	★	●
		out-of-place	★	☆	★	○
	cross-address-space	in-place	★	● [50]	★	○
		out-of-place	★	☆	★	○

Symbols indicate whether an attack is possible and known (●), not possible and known (○), possible and previously unknown or not shown (★), or not possible and previously unknown or not shown (☆). All tests performed with no defenses enabled. Empty fields still require testing.

Obrázek 2: Shrnutí variant Spectre, jejich známosti, adresního prostoru na který je možno útočit a ovlivněný výrobce/architektuar procesorů.

5.4.2 Spectre-BTB

Originální útok *Spectre-v2*. „Branch Target Predictor“ je část procesoru, která předpovídá výsledek kódu nacházející se v podmínkové větvi programu. „Branch Target Buffer“ (BTB) je vyrovnávací paměť, která uchovává výsledky, na rozdíl od *PHT* je ale separátní pro každou podmíněnou větev. Na rozdíl od Spetre-PHT, kde útočník přetrénuje mechanismus validními hodnotami, Spetre-BTB přetrénuje mechanismus vlastními škodlivými daty. Díky tomu si útočník může navolit data tak, že směřují na paměť obsahující tzv. „gadget“, což je kód, který se spustí a vystaví útočníkovi původně tajné informace (přenese data do kanálu, který je plně v režii útočníka). Aby bylo možné takový útok provést, tak musí být známa virtuální adresa „gadgetu“ v adresním prostoru.

5.4.3 Spectre-STL

Dalším způsobem, jak zlepšit výkon procesoru je zápisu aktualizovaných výsledných hodnot do vyrovnávací paměti namísto zápisu přímo do paměti k tomu určené (např. cache nebo RAM). Bez této nutnosti neustále načítat a ukládat data do pomalejší paměti je umožněno procesoru provádění pokynů v rychlejším tempu. Všechna data uložená v zásobníku se nakonec zapíšou zpět do hlavní paměti. Při takovém procesu vzniká riziko, že je přečten obsah vyrovnávací paměti před tím, než stačí být zapsána aktualizovaná hodnota. Aby se této situaci zabránilo, procesor kontroluje vyrovnávací paměť před každým čtením, které provádí. Při těchto operacích je povolen „unaligned access“ (nezarovnaný přístup) do paměti a tak je celý proces kontroly výpočetně náročný. Tento proces je nazýván „disambiguace paměti“.

Jelikož je proces kontroly vyrovnávací paměti náročný, tak se používá další urychlení - „Speculative Store Buffer Bypass“. V případě, že process disambiguace předpoví, že načítání nebude kolidovat s ukládáním dat do vyrovnávací paměti, tak není kontrola vyrovnávací paměti prováděna.

Optimalizace „Speculative Store Buffer Bypass“ byla v kombinaci s „Flush+Reload“ útokem zneužita k získání hodnot z vyrovnávací paměti. Omezením *Spectre-STL* (*Speculative Store Bypass*) je, že útočník může získat informace pouze z paměti, která se nachází na stejné úrovni privilegovanosti jakou má program využitý k útoku.

5.4.4 Spectre-RSB

Na rozdíl od předešlých útoků, Sperctre-RSB nezneužívá prediktory větvení (branch predictors), ale „Return Stack Buffer“. *RSB* je lokální, při změně kontextu resetovanou, kopií struktury „Call Stack“,

obvykle o velikosti 4 - 16 záznamů. „Call Stack“ je datová struktura uchovávající informace o aktivních subprocesech počítačového programu.

Příkaz *return* v počítačovém programu obvykle vrací vypočtenou hodnotu funkce do místa, odkud byla funkce zavolána. Kam má hodnotu vrátit dohledává ve struktuře *call stack*. Aby bylo nalezení cílové lokace rychlejší, tak se používá menší kopie - *RSB*.

Ve článku *Spectre Returns! Speculation Attacks using the Return Stack Buffer* je prezentováno několik variant útoku na *RSB* s cílem získat data která nemají být přístupná.

První varianta je způsobené přehlcením *RSB*, to způsobí přepsání skutečných užitečných adres, na které by mělo být referováno, ale už nemůže být, protože se v *RSB* nenachází.

Druhá varianta upravuje hodnoty v *RSB* napřímo - ať už je přepisuje nebo rovnou maže. To způsobuje neshodu mezi hodnotami v *RSB* a *call stacku* a tudíž mispredikci procesoru.

Třetí varianta naplňuje *RSB* adresami, které jsou mimo povolený adresní prostor. Ostatními okolnostmi popisovanými v originálním článku je pak možno na tyto adresy přistupovat bez vyvolání výjimek procesorem.

Čtvrtá varianta zahrnuje sdílení *RSB* napříč vlákny CPU bez resetování hodnot při kontextuálním přechodu.

6 Závěr