Želimir Mikulić, dipl.ing.[1]

SIDE-CHANNEL NAPADI I KORIŠTENJEM SLABIH TOČAKA ARHITEKTURA SA SPEKULATIVNIM IZVOĐENJEM

Stručni rad / Professional paper
UDK 004.052.4

U ovom radu prikazani su koncepti na kojima su zasnovani novootkriveni tipovi napada poznati pod nazivom Spectre i Meltdown. Obe vrste napada koriste isključivo sigurnosnu ranjivost samog hardware-a ne slabosti softwarea što ih čini neovisnim o OS-u. Princip, zajednički svim modernim procesorima, po kojem instrukcije koje se izvode spekulativno ne podižu signal prekida u trenutku kad se detektira da je korisnički proces pristupio zaštićenoj/privilegiranoj memoriji drugog procesa već tek kad se za nju izvodi „commit"/"retire" otvara dovoljno velik vremenski prozor koji omogućava izvođenje side-channel napada. Implementacija upravljanja spekulativnim i izvođenjem s poremećenim redoslijedom, priručnom memorijom, dubina cjevovoda i druge karakteristike mogu imati utjecaja na efikasnost side- channel napada do mjere da na nekom platformama izvedljivost pojedinih tipova napada nije mogla biti potvrđena, ali potencijalna prijetnja pronalaženja optimalnijeg koda napada i novih iskoristivih sporednih svojstava ostaje.

Ključne riječi: *side-channel napadi, spekulativno izvođenje, računalna arhitektura, privilegirana memorija.*

1. Uvod

Razvoj novih računalnih arhitektura i novi pristupi u primjeni postojećih arhitektura sa spekulativnim izvođenjem otvorili su mogućnost iskorištavanja slabih točaka njihovih mikro arhitektura za napade korištenjem sporednih svojstava (engl. „side-channel attacks") s ciljem čitanja sadržaja zaštićene odnosno privilegirane memorije iz korisničkog prostora.

„Arhitektura predstavlja ponašanje procesora koje proizvođač obećava, mikro arhitektura predstavlja njenu implementaciju"[2]

Jedan od ključnih koncepata računalne sigurnosti zahtijeva od računalne arhitekture i operacijskog sustava da izolira i zaštiti memorijski prostor svakog pojedinog procesa i osigura da se pri izvođenju instrukcija korisničkog procesa onemogući pristupanje privilegira-

[1] Veleučilište u Šibeniku
[2] Anders Fogh: "Covert shotgun: Automatically finding covert channels in SMT", prezentacija na RUHR UNIVERSITÄT Bohum, https://www.youtube.com/watch?v=oVmPQCT5VkY (pristupio: 12.11.2017)

noj memoriji jezgre („kernel-a“) operacijskog sustava. Kao što ukazuje u [LAMPSON] Butler W. Lampson: vrlo je teško (i skupo) ograničiti informaciju ako dijelite računalo.

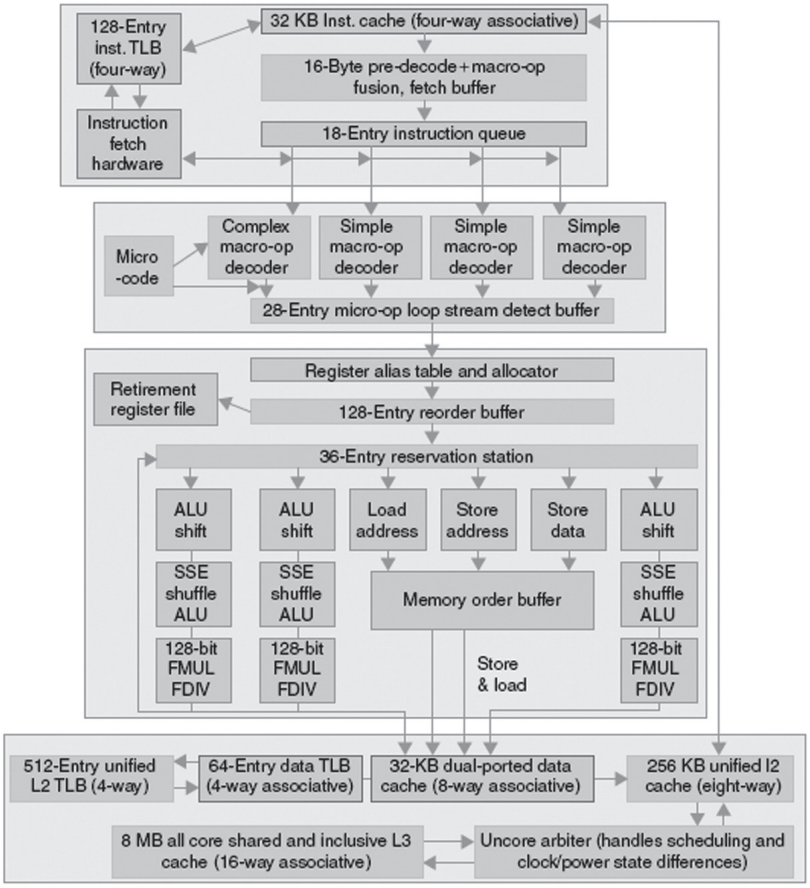
Nedavno objavljeni dokazi mogućnosti iskorištavanja (“proof of concept”, PoC) inovativnih koncepata izvođenja side-channel napada [HORN, KOCHER, LIPP, MAISURADZE] izazvale su izuzetno veliku pažnju među stručnjacima za računalnu sigurnost, ali i u medijima zbog činjenice da mogućnost iskorištavanja te vrste napada za otkrivanje sadržaja privilegirane memorije ne proizlazi iz zlonamjernog software-a kojeg bi trebalo instalirati na napadnutu platformu, odnosno slabosti operacijskih sustava, već se zasniva isključivo na karakteristikama hardware-a i to onima koje su zajedničke svim modrenim arhitekturama procesora i neovisne su o operacijskom sustavu koji upravlja njihovim radom. Prvenstveno se to odnosi na korištenje spekulativnog izvođenja, priručne memorije, predviđanja grananja i sl. Da bi mogli procijeniti potencijal opasnosti koja prijeti od slabih točaka modernih procesora poznatih pod imenima „Meltdown“ i „Spectre“ potrebno je objasniti ključne karakteristike računalnih arhitektura koje ih omogućavaju i tip side-chanell napada koje koriste. Postojanje navedenih prijetnji otkriveno je nezavisno od strane više grupa istraživača, stručnjaka za računalnu sigurnost, a dokaz o mogućnosti njihove eksploatacije javno obznanjen 3. siječnja 2018. Problem je ocijenjen ozbiljnim jer je napad moguć na najzastupljenijim tipovima procesora, moguće ga je obaviti pomoću aplikacija za čije izvođenje se ne traže administratorske ovlasti, pa i javascript koda koji je ugrađen u web stranice i jer se očekuje da bi primjena privremenih rješenja za ublažavanje ili potpuno osiguranje od tih prijetnji moglo dovesti do značajnog smanjenje efikasnosti rada procesora.

2. Preduvjeti

2.1. Spekulativno izvođenje

Spekulativno izvođenje tehnika je kojom se želi poboljšati performanse procesora na način da se instrukcije koje slijede nakon uvjetnog grananja izvode prije nego što se razriješi uvjet grananja. Kontrolna logika procesora spekulirajući o mogućem ishodu grananja pokreće izvođenje jednog ili više sljedova instrukcija prije nego se može utvrditi da li i koji od njih treba obaviti. Kad se uvjet razriješi i utvrdi pravi slijed instrukcija, rezultat njihovog izvođenja već je spreman, uz uvjet da je predviđanje ishoda grananja bilo uspješno. Rezultati izvođenja svih sljedove instrukcija za koje se utvrdi da nisu trebali biti izvedeni biti će jednostavno odbačeni tj. promjene stanja procesorskih registara do kojih oni dovode neće biti napravljene i nastaviti će se s daljnjim izvođenjem onog slijeda na koji nas grananje usmjerava tj. stanje procesora će biti promijenjeno sukladno rezultatu njihovog izvođenja („commit“ ili „retire“). Vremenski prozor u kojem se instrukcije izvode spekulativno ostaje otvoren dok se ne utvrdi rezultat grananja. Kroz to vrijeme se svi operandi i operacije drže u tzv. rezervacijskim stanicama povezanim s izvršnim jedinicama koje paralelno obavljaju mikro operacije i rezultate vraćaju u rezervacijske stanice kao i u jedinicu za upis u registre („register retirement file“) koji će biti obavljen samo za one instrukcije za koje se utvrdi da im izvođenje nije spekulativno nakon čega se one uklanjaju iz spremnika za izmjenu redoslijeda („reorder buffer“). (*Slika 1*.)

Slika 1. Struktura Intel Core i 7 cjevovoda s memorijskim komponentama



Izvor: 1. Patterson D.A., Hennessy J.L. Computer Organisation and Design: The Hardware/Software Interface ARM, 5th Edition, str. 359

Važno je primijetiti da rezultati spekulativnog izvođenja instrukcija koje ostaju spekulativne tj. čije izvođenje neće izazvati promjene u procesorskim registrima, niti će izazivati eventualno zapisivanje u memoriju, u većini slučajeva, ostavljaju traga u mikro arhitekturi, npr. podatci i instrukcije koje su tijekom spekulativnog izvođenja bili učitani u

priručnu memoriju neće iz nje biti obrisani („flushed"). To stvara preduvjete da se korištenjem sporednih svojstava, poput vremena odziva kod dohvaćanja određene memorijske lokacije ili vremena izvođenja indirektno adresiranog programskog skoka, posredno otkrije sadržaj memorijskih lokacija privilegirane memorije koji je dohvaćen u fazi spekulativnog izvođenja prije nego što je utvrđeno da im se iz tog procesa nije smjelo pristupiti. Naime kao što je opisano u [FOGH] kako bi se izbjeglo da instrukcija koja se izvodi spekulativno, i možda će takvom i ostati, izazove prekid („interrupt"), zbog statusa iznimke ((exception status") koja se javlja kod pristupanja privilegiranoj memoriji, podizanje tog prekida se ne obavlja u trenutku kad se mikro operacijski kod instrukcije stvarno izvede, nego tek kad i ako se utvrdi da

ona ne ostaje spekulativna, prilikom njenog „commit"-a. Ključno je pri tome osigurati da trajanje vremenskog okvira u kojem se instrukcije izvode spekulativno bude dovoljno dugo da se izvedu instrukcije koje će ostaviti „trag" izmjenom sporednih svojstava.

2.2. Priručna memorija

Korištenje priručne memorije je tehnika kojom se skraćuje vrijeme pristupa lokacijama u glavnoj memoriji što predstavlja ključno usko grlo svih modernih računala baziranih na von Neumannovom modelu, a to su praktički sva računala. Ideja je da se u sam procesor ugradi manja količina skuplje, ali znatno brže priručne memorije u koju se kopira sadržaj pojedinih lokacija glavne memorije i to uz sadržaj lokacije kojoj se pristupa i onih lokacije za koje algoritmi za upravljanje priručnom memorijom predviđaju da će im u nekom od slijedećih procesorskih ciklusa biti potrebno pristupiti. U slučaju da je lokacija kojoj je potrebno pristupiti već kopirana u priručnu memoriju operacija dohvata se obavlja znatno brže nego u slučajevima kad je dohvat potrebno obaviti iz glavne memorije. Moderni procesori imaju priručnu memoriju realiziranu na dvije ili više razina, pri čemu su na L1 razini razdvojene podatkovna i instrukcijska priručna memorija, dok je priručna memorija na L2 razini zajednička za memorijske lokacije s podatkovnim i instrukcijskim sadržajima, ali i dalje je odvojena po procesorskim jezgrama kod višejezgrenih procesora. Postoje implementacije sa L3 i L4 razinama, pri čemu se u svim arhitekturama pokušava poboljšati efikasnost pristupanju sadržajima lokacija glavne memorije odabiranjem veličine, organizacije, asocijativnosti, politike zapisivanja i zamjenjivanja te veličine bloka na svim razinama priručne memorije. (*Slika 2.*)

Slika 2. Priručna memorija u ARM Cortex-A53 i Intel Core i7 920

Characteristic	ARM Cortex-A8	Intel Nehalem
L1 cache organization	Split instruction and data caches	Split instruction and data caches
L1 cache size	32 KiB each for instructions/data	32 KiB each for instructions/data per core
L1 cache associativity	4-way (I), 4-way (D) set associative	4-way (I), 8-way (D) set associative
L1 replacement	Random	Approximated LRU
L1 block size	64 bytes	64 bytes
L1 write policy	Write-back, Write-allocate(?)	Write-back, No-write-allocate
L1 hit time (load-use)	1 clock cycle	4 clock cycles, pipelined
L2 cache organization	Unified (instruction and data)	Unified (instruction and data) per core
L2 cache size	128 KiB to 1 MiB	256 KiB (0.25 MiB)
L2 cache associativity	8-way set associative	8-way set associative
L2 replacement	Random(?)	Approximated LRU
L2 block size	64 bytes	64 bytes
L2 write policy	Write-back, Write-allocate (?)	Write-back, Write-allocate
L2 hit time	11 clock cycles	10 clock cycles
L3 cache organization	–	Unified (instruction and data)
L3 cache size	–	8 MiB, shared
L3 cache associativity	–	16-way set associative
L3 replacement	–	Approximated LRU
L3 block size	–	64 bytes
L3 write policy	–	Write-back, Write-allocate
L3 hit time	–	35 clock cycles

Izvor: 1. Patterson D.A., Hennessy J.L. Computer Organisation and Design: The Hardware/Software Interface ARM, 5th Edition, str 484

Vrijeme dohvaćanja sadržaja memorijske lokacije ovisi značajno o tome da li je i na kojoj razini njena kopija zatečena u priručnoj memoriji ili nije. Za memorijske lokacije čiji se sadržaj prilikom dohvaćanje ne zatekne u priručnoj memoriji tipično vrijeme pristupa je veće od 100 procesorskih ciklusa naspram 1 do 10 procesorskih ciklusa ovisno o razini priručne memorije u kojoj je sadržaj memorijske lokacije dostupan.

Za ARM Cortex-A53 u slučaju da je sadržaj lokacije već učitan na L1 razini priručne memorije on se dohvaća u 2 takta procesora, 12 ako je tek na L2 razini, a 126 ako uopće nije zatečen u priručnoj memoriji. Mjerenjem brzine izvršavanja instrukcije dohvaćanja pojedine memorijske lokacije moguće je sa sigurnošću utvrditi da li je njen sadržaj zatečen u priručnoj memoriji i na kojoj razini.

2.3. Zaštićena i privilegirana memorija

Temeljni koncept je da je memorijski prostor svakog procesa zaštićen od memorijskih prostora ostalih procesa. To znači da operacijski sustav mora, uz podršku hardwarea, memorijske lokacije koje dodijeli jednom procesu zaštititi od mogućnosti da ih prilikom izvođenja adresiraju instrukcije drugog procesa. Ako prilikom izvođenja instrukcije ona pokušava pristupati memorijskoj lokaciju dodijeljenoj drugom procesu (zaštićenoj memoriji) to će biti onemogućeno generiranjem sistemske greške odnosno signala prekida. Istovjetna je situacija kod pristupanja privilegiranoj (kernel) memoriji. Njoj je moguće pristupati isključivo kad procesor radi u privilegiranom /kernel modu tj. kad izvodi instrukcije jezgre operacijskog sustava. Problem se javlja kad se instrukcija čitanja iz memorije izvodi u spekulativnom modu. Ako se unutar vremenskog okvira, dok procesor radi u spekulativnom režimu, pojavi naredba koja pristupa zaštićenoj/privilegiranoj memoriji njen mikro kod se izvršava, ali ne dolazi do zapisivanja u registre, kod operacije dohvaćanja, odnosno zapisivanja u memoriju, kod operacije pohranjivanja, već se rezultat bilježi u spremnik („reorder buffer queue“) zajedno sa informacijom o registriranoj iznimci. Signal prekida se ne generira, jer se ne zna da li će ta instrukcija kad se uvjet grananja razriješi biti izvedena u cijelosti ili samo spekulativno. On će se generirati tek ako bi se pokazalo da takva instrukcija ne ostaje spekulativna i da je njen rezultat potrebno upisati u registar/memoriju. Ono što predstavlja narušavanje sigurnosti je činjenica da je sadržaj zaštićene memorije bio pročitan i da ga se je, unutar navedenog vremenskog okvira, moglo upotrijebiti i iskoristiti za ostavljanje odgovarajućeg traga u mikro arhitekturi procesora kojemu je moguće naknadno „legalno“ pristupiti i posredno zaključiti na vrijednost koja je pročitana iz zaštićene memorije.

2.4. Side-channel napadi

Side-channel napadi tj. napadi korištenjem sporednih svojstava spadaju u kategoriju implementacijskih napada.

Procesi izvođenjem na fizičkim uređajima djeluju na njihova fizička svojstva, poput potrošnje (npr. procesora u različitim fazama izvođenja algoritma), elektromagnetskih zračenja (npr. s monitora), zvuka (npr. kod tipkanja na tipkovnici) itd. Rezultati mjerenja tih svojstava uspješno su korišteni s ciljem razotkrivanja ključeva šifri i sl.

Svi spomenuti tipovi napada nužno trebaju vanjsku mjernu opremu za razliku od side-channel napada koji su se pojavili u novije vrijeme i koji zahtijevaju samo izvođenje odgovarajućeg koda [ACIICMEZ, LIU]. Oni ne koriste slabe točke software-a, poput „buffer-overflow" ranjivosti, već hardware-a procesora i omogućavaju indirektno „zaključivanje" o sadržaju osjetljivih dijelova memorije i stoga nisu ograničeni na pojedini operacijski sustav, već podjednako pogađaju sve koji se na tim hardware-skim platformama izvode.

Napadi zasnovani na mikro arhitekturi procesora iskorištavaju vremensku usklađenost priručne memorije, povijest predviđanja grananja, analizu među spremnika odredišta grananja („Branch Target Buffers) i sl. da bi iz zaostalih tragova instrukcija izvođenih u spekulativnom radu (odbačenih kad je kontrolna logika utvrdila da se one ne obavljaju tj. da se ne obavlja „commit"/"retire" indirektnim metodama zaključivali o sadržaju lokacija zaštićene memorije. Ideja nije da se „zavara" operacijski sustav koji s hardware-om onemogućava da se izvođenjem koda jednog korisničkog procesa pročita sadržaj zaštićene/privilegirane memorije, već da ga se razotkrije iz tragova koje u mikro arhitekturi procesora ostave instrukcije koje se „ne izvode" tj. koje ostaju u spekulativnom modu, bez pohranjivanja rezultatanjihovog izvođenja.

Napadi, potvrda čije izvodljivosti je javno objavljena 03.siječnja 2018. poznati su pod imenima „Spectre" i „Meltdown" i predstavljaju dva tipa koncepata side-chanell napada koja na različite načine i u različitoj mjeri ugrožavaju tajnost sadržaja zaštićene memorije napadnutih računala.

3. Spectre

Sustavi, s mikroprocesorima koji koriste spekulativno izvođenje (kod uvjetnih grananja i predviđanja odredišta indirektnih skokova) procesu sa standardnim korisničkim pravima pružaju mogućnost razotkrivanja privilegiranih informacija drugih procesa koji se izvode na istom procesoru koristeći se analizom vremena odziva priručne podatkovne memorije a potencijalno i drugim tipovima side-channel napada.

Pod nazivom Spectre kriju se dvije varijante iskorištavanja ranjivih točaka mikro arhitekture gotovo svih modernih procesora s oznakama („Common Vulnerabilities and Exposures) CVE-20107-5753 i CVE-20107-5715 prema standardu za davanje imena otkrivenim povredama informacijske sigurnosti koju održava MITRE kao primarni CNA („CVE Number Authority").

Više grupa istraživača nezavisno je otkrilo i prijavilo vodećim proizvođačima procesora, i OS software-a obje varijante iskorištavanja ranjivosti. Sukladno protokolima proizvođačima je ostavljen rok duži od šest mjeseci prije nego što su rezultati i javno obznanjeni 03. siječnja 2018, do kada je većina već pripremila, a neposredno nakon obznanjivanja i objavila privremene sigurnosne zakrpe za svoje proizvode. Rok je vjerojatno bio značajno duži jer je iz dostupnih podataka vidljivo da je Intel navedene CVE brojeve rezervirao još 01.veljače 2017., čime je proizvođačima dano dodatno vrijeme da pripreme što kvalitetnija privremena rješenja, uzimajući u obzir da su ovi napadi ugrožavali gotovo sve moderne procesore u poslužiteljima, radnim stanicama, prijenosnim računalima i mobilnim uređajima svih najznačajnijih proizvođača.

Spectre napadi zasnivaju se na shemi prema kojoj se napadnuto računalo „navede“ da pogrešno predvidi ishod grananja i spekulativno izvodeći instrukcije pročita sadržaj zaštićene memorije koji se onda iskorištavanjem sporednih kanala učini dostupnim.

U prvoj, pripremnoj, fazi napada potrebno je logiku za predviđanje ishoda BPL („Branch Prediction Logic“) i cilja grananja BTB („Branch Target Buffer“) dovesti u stanje u kojem će pogrešno predvidjeti ishod grananja i omogućiti spekulativno izvođenje instrukcija koje će adresirati privatnu memoriju. Instrukcije koje će poslužiti u tu svrhu u primjerima prikazanim u [KOCHER] i [HORN] iskoristiti će se i za pripremu sadržaja priručne memorije izbacivanjem iz nje vrijednosti koja je nužna za određivanje odredišta grananja, kao i onih vrijednosti vrijeme dohvaćanja kojih će biti sporedno svojstvo („side channel“) preko kojeg će se u posljednjoj fazi rekonstruirati u drugoj – spekulativnoj fazi dohvaćeni sadržaj privatne memorije.

3.1. Varijanta 1

U ovoj varijanti eksploatira se pogrešno predviđanje ishoda uvjetnog grananja koje će dovesti do spekulativnog i po razrješenju uvjeta odbačenog izvođenja instrukcije koja dohvaća lokaciju zaštićene memorije. Prema varijantama prikazanim u [HORN] i []KOCHER] proces koji se izvodi ima pristupa do dva polja byte-ova: **vektor1** koji je veličine **vel_vektor1 i vektor2** koje je veličine 64 KiB ili 128 KiB. Veličina **vektor2** ovisi o veličini linije priručne memorije koja u većini procesora nije veća od 256 B.

Dio koda koji predstavlja napad je slijedeći:

```
if ( x < vel_vektor1 ) {
        temp = vektor2 [vektor1[x] * 256];
}
```

Varijabla **x** je izvan dozvoljene vrijednosti za indeks **vektor1** i iznosi (adresa ciljane lokacije s koje želimo pročitati vrijednost - početna adresa polja). Ključno je, u trenutku kad se krene izvoditi gornji kod, osigurati da je logika predviđanja grananja dovedena u stanje koje će voditi u spekulativno izvođenje, te da **vel_vektor1** i cijelo polje **vektor2** ne budu u priručnoj memoriji, dok lokacija iz privatne memorije koju pokušavamo pročitati **vektor1[x]** mora biti u priručnoj memoriji. Ovaj zadnji uvjet najlakše možemo postići ako za svaku ciljanu lokaciju nekoliko puta ponovimo ciklus: “kondicioniranje“ logike za predviđanje grananja kroz višestruko uzastopno pozivanje gornjeg koda s „legalnom“ vrijednošću za x – poziv gornjeg koda s „nelegalnom“ vrijednošću x koja odgovara ciljanoj lokaciji, čime se osigurava da se učita i zadrži u priručnoj memoriji.

Uz navedene uvjete, zbog promašaja u priručnoj memoriji, nije moguće izračunati uvjet dok se **vel_vektor1** ne učita iz glavne memorije što dovodi do spekulativnog izvođenja instrukcija koje se generiraju iz slijedeće instrukcije C koda. Indeks polja **vektor2** stiže se izračunati dok je procesor još spekulativnom režimu jer je vrijednost **vektor1[x]** u priručnoj memoriji, nakon čega se pokreće učitavanje iz memorije člana polja **vektor2** s indeksom koji je direktno ovisan o vrijednosti učitanog byte-a iz zaštićene memorije nakon još jednog promašaja. Za razliku od lokacije zaštićene memorije, pokušaj čijeg dohvaćanja iz priručne memorije će u normalnom režimu rada generirati sistemsku grešku, program može pristupati svim članovima polja **vektor2** pa i onom koji će nakon razrješavanja promašaja u priruč-

noj memoriji biti u nju učitan kao posljedica spekulativno izvedenog koda iako neće biti učitan u registar procesora i pohranjen u varijablu **temp**. Budući bi u priručnu memoriju trebala biti učitana samo jedna linija s članovima polja **vektor2** moguće je iskoristiti mjerenje brzine odziva kod pristupanja članovima polja koji se učitavaju u različite linije priručne memorije i iz toga zaključiti na vrijednost byte-a iz zaštićene memorije kojeg smo napali.

Iako je proces relativno spor on omogućava čitanje po potrebi i sadržaja kompletnog kernel prostora.

Kao dokaz izvodivosti u [KOCHER] je dan i primjer JavaScript koda koji, kad se izvodi u Google Chrome pregledniku, može privatnu memoriju procesa u kojem se izvodi. Nemogućnost poziva naredbi za pražnjenje („flushing“) priručne memorije **clflush** u JavaScriptu moguće je nadomjestiti učitavanjem velikog broja adresa s istom vrijednosti indeksnih bitova adrese kao i adrese koje želimo ukloniti iz priručne memorije. Činjenica da je Google u Chrome pregledniku kao i drugi isporučitelji preglednika namjerno smanjili točnost mjerača vremena visoke rezolucije (rezolucija u nano sekundama) kako bi odgovorili na side-channel napade korištenjem mjerenja vremena odziva priručne memorije uporabom performance.now, čija je rezolucija sada smanjena na 5μs [CHROME], [SCHWARZ] nije ih spriječila. Korištenjem Web Workers API-ja, relativno je jednostavno kreirati nezavisnu pozadinsku dretvu [WW] koja u petlji dekrementira vrijednost pohranjenu u lokaciju u dijeljenoj memoriji čime se omogućava kreiranje mehanizma mjerenja vremena u visokoj rezoluciji u JavaScriptu [SCHWARZ].

3.2. Varijanta 2

Spectre napadi u ovoj varijanti zasnivaju se na ideji da se, umjesto instrukcije uvjetnog grananja kao u varijanti 1, iskoristi instrukcija indirektnog skoka s odredištem definiranim u memorijskoj lokaciji koja je dio koda napadnutog procesa. Većina modernih arhitektura instrukcijskog skupa (ISA), između ostalih x86, ARM, MIPS ISA itd., podržavaju instrukcije indirektnog skoka, pa tako i one kod kojih je adresa odredišna adresa definirana u memorijskoj lokaciji na koju upućuje neki od procesorskih registara ili pokazivač. Stvaranjem preduvjeta da prilikom izvođenja instrukcije indirektnog skoka dođe do promašaja u priručnoj memoriji kod dohvaćanja lokacije s adresom odredišta skoka, adresa na kojoj će se nastaviti izvođenje će biti spekulativno preuzeta s odgovarajuće lokacije iz spremnika odredišnih adresa („Branch Target Buffer“, skraćeno BTB). BTB predstavlja zapravo „priručnu memoriju“ odredišnih adresa skoka koju indeksiramo podskupom bitova adrese instrukcije skoka. Ključni element ove vrste napada je da je BTB zajednički svim procesima koji se izvode na procesoru. Izvodljivo je, npr. u slučaju primjene virtualizacije kad se kod više virtualnih strojeva (VM) izvodi na istoj jezgri, izvođenjem instrukcija indirektnog skoka u jednom VM-u tako pripremiti BTB-a da prilikom obavljanja instrukcije indirektnog skoka u drugom VM-u spekulativno usmjeriti na lokaciju s odabranim fragmentom koda („gadget“) u vlastitom prostoru. Fragment koda na koji će se usmjeriti napad potrebno je odabrati tako da njegove instrukcije pristupaju ciljanim lokacijama (registrima, memorijskih lokacijama) i da prije nego dođe do razrješenja stvarne lokacije indirektnog skoka i „odbacivanja“ rezultata izvedenog koda ostave trag u mikro arhitekturi koji će omogućiti da proces koji obavlja napad, korištenjem neke od side channel tehnika, otkrije njihov sadržaj. Najbolji mjesto za traženje odgovarajućeg fragmenta

koda koji će poslužiti za napad predstavlja kod nekog od DLL-ova koje napadnuti proces koristi. Ti kodovi su dovoljno veliki da je moguće u njima pronaći sekvencu instrukcija koja će poslužiti za napad i prilikom izvođenja uvezani su u memorijski prostor napadnutog procesa. Ovaj tip napada je složeniji za izvođenje, ali i teži za obranu od njega, prvenstveno što bi rješenja koja bi onemogućavala da izvođenjem jednog procesa utječemo na sadržaj BTB-a kojeg će koristiti drugi proces dovela do značajnog pada efikasnosti procesora.

Obje varijante Spectre napada koriste se izazivanjem izvođenja instrukcija u spekulativnom režimu rada (iz čega vuče porijeklo i naziv ove vrste napada) koji će nakon razrješenja uvjeta ili lokacije grananja biti odbačen što im omogućava samo da pročitaju sadržaj privilegirane memorije, ali ne i da ga mijenjaju. Potvrda mogućnosti izvođenja ove vrste napada napravljena je na recentnim modelima iz više obitelji procesora uključujući one najraširenije zasnovane na x86 i ARM arhitekturi.

4. Meltdown

Za razliku od Spectre napada koji eksploatiraju spekulativno izvođenje na razini instrukcija Meltdown napad se zasniva na „spekulativnom" izvođenju na razini mikro operacija. Moderni mikroprocesori s ciljem da maksimalno iskoriste sve izvršne jedinice svoje arhitekture i optimiraju performanse procesora primjenom tehnike izvođenja instrukcija u poremećenom redoslijedu („out-of-order execution"). Učitane instrukcije se dekodiraju u mikro operacije i umjesto da se one izvode u slijedu zadanom kodom izvode se čim su resursi (izvršna jedinice i podatkovni elementi) na raspolaganju. Ovisno vrsti i raspoloživosti izvršne jedinice koju pojedina mikro operacija zahtjeva može doći do poremećaja redoslijeda u kojem će one biti obavljene, što znači da se zapravo obavljaju spekulativno. Promjene koje rezultati mikro operacije izazivaju na stanje procesora (promjene na registrima) biti će provedene u pravilnom redoslijedu prilikom „commit"/"retire", ali mikro operacije koje čekaju na izvršavanje imaju, u nekim arhitekturama, te rezultate na raspolaganju neposredno nakon što se mikro operacija obavi. Redovito zbog primjene navedene tehnike u radu procesora dolazi do toga da se veći broj mikro operacije obavi prije nego što dođe do „commit"/"retire"-a operacije koja je po programskom slijedu ispred njih, ostavljajući o tome trag u mikro arhitekturi procesora npr. u stanju priručne memorije, TLB-u i slično. U slučajevima kad neka instrukcija izazove javljanje iznimke („exception") npr. kod dohvaćanja memorijske lokacije iz privilegiranog područja ona će biti podignuta, a time i zaustavljeno pohranjivanje rezultata instrukcija koje je slijede, tek prilikom „commit"/"retire". To na nekim arhitekturama ostavlja dovoljno velik vremenski okvir da se u poremećenom redoslijedu izvede dovoljno instrukcija koje je slijede da bi se obavio side-channel napad.

U [LIPP] je detaljno opisan napad koji se zasniva na gornjoj ideji, ispitan na više tipova procesora tvrtke Intel, koji je omogućio čitanje cjelokupnog prostora privilegirane memorije („kernel" prostor) na osobnim računalima i na računalima u oblaku. Za omogućavanje ovog napad bilo je ključan preduvjet je bio što svi testirani operacijski sustavi (Linux, OS X, Windows 10) mapiraju cijeli „kernel" prostor u virtualni korisnički prostor. To omogućava da se iz korisničkog prostora adresiraju lokacije u „kernel" prostoru iako im se može pristupati samo ako procesor radi u privilegiranom režimu, jer se u suprotnom podiže iznimka. S druge strane kako su u „kernel" prostor smješteni podaci o mapiranju kompletne fizičke memorije to

otvara prostor za pristupanju sadržajima kompletne memorije svih procesa koji su pokrenuti na tom procesoru.

Testiranje napada nije bilo uspješno na ARM i AMD procesorima iako je potvrđeno da je iskorištavanje prekorednog izvođenja moguće i na njima i da se dio instrukcija koje slijede instrukciju koja pristupa privilegiranoj memoriji spekulativno izvedu. Razlike u dubini cjevovoda i logici upravljanja prekorednim izvođenjem onemogućile su napad kakav je uspješno izveden na Intel arhitekturi, moguće i zbog toga što se vremenski okvir od dohvaćana privilegirane memorije do podizanja iznimke bio prekratak za izvođenje side-channel napada.

Ova vrsta napada može biti otežana primjenom ASLR („Address Space Layout Rendamisation") tehnike primijenjene u zaštiti jezgre operacijskog sustava nasumičnim odabira lokacije na koju se učitava kernel prostor prilikom svakog pokretanja [EDGE]. Iako se ova zaštita zbog uspješnih side-channel pokazala neefikasnom primjena najnovije verzije KAISER zakrpe [GRUSS] na testiranim operacijskim sustavima uspješno smanjuje ograničava memorijski prostor koji je izložen Meltdown napadu na x86 arhitekturama, pa time i Intel procesorima uz minimalno smanjenje performansi.

Važno je napomenuti da primjena gornje zakrpe nema utjecaja na Spectre napade jer oni ne pristupaju zaštićenoj odnosno privilegiranoj memoriji oni zasnivaju instrukcijom koja se izvodi u korisničkom procesu koji obavlja napad, već instrukcijom u napadnutom procesu, čiji trag se onda side-channel napadom prenosi napadaču.

5. Zaključak

Potvrđivanjem mogućnosti da je iskorištavanjem slabih točaka arhitekture prikazani koncept side-channel napada ostvariv, kod gotovo svih modernih procesora, ukazalo ja na visoku razinu opasnosti od čitanja sadržaja privilegiranih dijelova memorije i u pojedinim slučajevima kompletnog „kernel" prostora. To može omogućiti potencijalnom napadaču da se domogne osjetljivih zaštićenih podataka kojima bi, time što su smješteni u „kernel" prostor, pristup trebao biti onemogućen, iako ne omogućava promjenu sadržaja tj. zapisivanje u zaštićeni/privilegirani dio memorije.

Spectre i Meltdown, napadi iako oba iskorištavaju ranjivosti koje proizlaze isključivo iz mikro arhitekture hardware-a procesora, ne koriste slabosti i neovisni su o software-u operacijskih sustava, koriste se sporednim svojstvima za razotkrivanje napadnutog sadržaja i napadaju direktnu suštinski koncept odvajanja i zaštite privatnosti memorijskog prostora korisničkih i kernel procesa, imaju suštinsku razliku. Spectre napadi predstavljaju napade kojima se informacija „bočnim kanalom" prenosi preko granice memorijskih prostora, dok Meltdown napad zasniva na efikasnom uklanjanju granice između memorijskih prostora.

U radovima [HORN], [KOCHER], [LIPP] itd. potvrđena je praktična izvedivost ovakvih napada na najraširenijem skupu platformi. Dokazana je prihvatljiva efikasnost realiziranih napada i ukazano na dodatna sporedna svojstva koja bi mogla osigurati nove inovativne side-channel napade koji bi iskorištavali još kraće vremenske okvire spekulativnog izvođenja i time zaobišli privremene zakrpe ovih napada.

Neposredno po javnom objavi o uspješno izvedenim napadima većina proizvođača je objavila izvještaje o ugroženosti [AMD], [ARM], [INTEL2] i privremene software-ske i/ili hard-

ware-ske zakrpe za zaštitu od ovih vrsta napada. To uključuje proizvođače hardware-e, operacijskih sustava i preglednika. Neke od tih zakrpa, prvenstveno one koje bi izmjenama u mikro-kodu procesora u pojedinim situacijama blokirale ulazak u spekulativno izvođenje ili prisilno brisale sve razine priručne memorije u slučaju javljanja iznimke/prekida izazivaju značajnu degradaciju performansi procesora.

Iako se objavljena varijanta Meltdown napada uspješno sprječava postojećim zaprekama (KEISER) mogu se očekivati nove varijante primjene tog koncepta za koje će tek trebati osmisliti efikasnu zaštitu. Pouzdana zaštita od Spectre napada, posebno od varijante 2, a koja ne bi značajno narušavala performanse softwarea, vjerojatno neće biti moguća pomoću software-skih zakrpa i izmjenama samo u mikro kodu hardware-a. Iako proizvođači procesora demantiraju izvješća o drastičnoj degradaciji performansi procesora o kojima su izvijestili pojedini nezavisni istraživači, one nisu neočekivane. Kvalitetnija rješenja će biti moguća tek zamjenom hardwarea novim generacijama procesora dizajniranim u startu uz uvažavanje postojanja kanala kojima je moguće indirektno zaključivati o stanju u procesoru.

Do toga će trebati pričekati jer ovi napadi ciljaju direktno koncept i implementaciju spekulativnog izvođenja čija primjena je ključna za efikasnost današnjih procesora pa će iznalaženje kvalitetnih kompromisa između sigurnosti i efikasnosti biti zahtjevan zadatak.

Iako većina korisnika osobnih računala neće biti izrazito pogođena eventualnom degradacijom performansi procesora nakon primjene odgovarajućih zakrpi to bi moglo imati vrlo značajan utjecaj na performanse računalnih oblaka.

LITERATURA

1. Acıiçmez O., Koç Ç.K., Seifert JP. (2006) Predicting Secret Keys Via Branch Prediction, *Abe M. (eds) Topics in Cryptology – CT-RSA 2007. CT-RSA 2007. Lecture Notes in Computer Science, vol 4377. Springer, Berlin, Heidelberg*, vol 4377. Springer, Berlin, Heidelberg [ACIICMEZ]
2. AMD: Software techniques for managing speculation on AMD processors, https://developer.amd.com/wp-content/resources/Managing-Speculation-on-AMD-Processors.pdf (pristup: 30.01.2018) [AMD]
3. ARM: Firmware interfaces for mitigating CVE_2017_5715 System Software on Arm Systems, link na stranici: https://developer.arm.com/support/security-update/latest-news/cache-speculation-issues-update (pristup: 24.01.2018)

Edge, J.: Kernel address space layout randomization, na Linux Security Summit 2013, https://lwn.net/Articles/569635 (pristup: 12.12.2017)[EDGE]

4. Fog, A.: The microarchitecture of Intel, AMD and VIA CPUs, An optimization guide for assembly programmers and compiler makers, Technical University of Denmark, http://www.agner.org/optimize/microarchitecture.pdf (pristup:17.06.2017) [FOG]
5. Fogh, A.: Negative result: Reading kernel memory from user mode, https://cyber.wtf/2017/07/28/negative-result-reading-kernel-memory-from-user-mode (pristup: 17.11.2017) [FOGH]

6. Gruss, D., Lipp, M., Schwarz, M., Fellner, R., Maurice, C., Mangard S.: KASLR is Dead: Long Live KASLR, *International Symposium on Engineering Secure Software and Systems* (2017), Springer, pp. 161–176. [GRUSS]
7. Horn J., at al. Reading privileged memory with side-channel, https://googleprojectzero.blogspot.hr/2018/01/reading-privileged-memory-with-side.html (pristup: 07.01.2018) [HORN]
8. Intel: "Intel 64 and IA-32 Architectures Optimization Reference Manual". January 2016 [INTEL1]
9. Intel: Intel Analysis of Speculative Execution Side Channels, January 2018, https://newsroom.intel.com/wp-content/uploads/sites/11/2018/01/Intel-Analysis-of-Speculative-Execution-Side-Channels.pdf (pristup: 24.01.2018) [INTEL2]
10. Kocher, P., Genkin, D., Gruss, D., Haas, W., Hamburg, M., Lipp, M., Mangard, S., Prescher, T., Schwarz, M., Yarom, Y.: Spectre Attacks: Exploiting Speculative Execution, https://arxiv.org/pdf/1801.01203 (pristup: 12.01.2018) [KOCHER]
11. Lampson B.W.: A Note on the Confinement Problem, Communications of the ACM, 16, 10 (Oct. 1973), pp 613-615 [LAMPSON]
12. Lipp, M., Schwarz, M., Gruss, D., Prescher, T., Haas, W., Mangard, S., Kocher, P., Genkin, D., Yarom, Y., Hamburg, M.: Meltdown, https://arxiv.org/pdf/1801.01207 (objavljeno: 03.01.2018., pristup: 12.01.2018) [LIPP]
13. LIU, F., YAROM, Y., GE, Q., HEISER, G., AND LEE, R. B.: Last-level cache side-channel attacks are practical, *IEEE Symposium on Security and Privacy (S&P) 2015* (2015), IEEE str. 605-622[LIU]
14. Maisuradze, G., Rossow, C.: Speculose: Analyzing the Security Implications of Speculative Execution in CPUs, https://arxiv.org/pdf/1801.04084 (pristup: 15.01.2018) [MAISURADZE]
15. Mowery. K.,, Keelveedhi, S., Shacham, H.: Are AES x86 Cache Timing Attacks Still Feasible?, *CCSW'12, October 19, 2012, Raleigh, North Carolina, USA* [MOWERY]
16. Oren, Y., Kemerlis, V.P., Sethumadhavan, S., Keromyti, A.D.: The Spy in the Sandbox -- Practical Cache Attacks in Javascript, https://arxiv.org/pdf/1502.07373 (pristup: 15.12.2017) [OREN]
17. Patterson D.A., Hennessy J.L. Computer Organisation and Design: The Hardware/Software Interface ARM, 5th Edition , 2017 Elsevier Inc. , ISBN: 978-0-12-801733-3 [PATT]
18. Schwarz M., Maurice C., Gruss D., Mangard S. (2017) Fantastic Timers and Where to Find Them: High-Resolution Microarchitectural Attacks in JavaScript. In: Kiayias A. (eds) *Financial Cryptography and Data Security. FC 2017*. Lecture Notes in Computer Science, vol 10322. Springer, Cham [SCHWARZ]
19. Security: Chrome provides high-res timers whuch allow side channel attacks, https://bugs.chromium.org/p/chromium/issues/detail?id=508166 (pristup: 02.07.2016.) [CHROME]
20. Tomasulo, R. M. An efficient algorithm for exploiting multiple arithmetic units. IBM Journal of research and Development 11, 1 (1967), 25–33. [TOMASULO]
21. Web Workers: Using Web Workers, https://developer.mozilla.org/en-US/docs/Web/API/Web_Workers_API/Using_web_workers (pristup: 12.01.2018) [WW]

Summary

SIDECHANNEL ATTACKS THAT EXPLOIT VULNERABILITIES OF ARCHITECTURES WITH SPECULATIVE EXECUTION

Concepts on which are founded recently discovered Spectre and Meltdown attacks have been described in this paper. Both attacks with their variants are addressing only hardware vulnerabilities and are not exploiting any software weakness which makes them operating system independent. Speculative execution is basic architectural concept of all modern processor designs on various levels. Principle according to which instruction, while executed in speculative mode or during out-of-order execution, does not raise interrupt in case of memory permission access violation instantly, but only then instruction is retired, opens big enough time frame window, which enables information leaking through the sidechannel. Implementation of speculative and out-of-order execution logic, cache control, pipeline depth and other characteristics can influence performance and possibility of the sidechannel attack in the way that on certain microarchitectures some variants of the attack were not currently reproducible, but potential threat from attack code optimization and discovery of the new exploitable covert channels stays.

Keywords: *sidechannell attack, speculative execution, computer architecture, privileged memory.*