



Kandidaatintutkielma

Tietojenkäsittelytieteen osasto

Spectre- ja Meltdown-tyyppiset hyökkäykset

Susanna Rajamäki

31.5.2020

MATEMAATTIS-LUONNONTIETEELLINEN TIEDEKUNTA
HELSINGIN YLIOPISTO

Ohjaaja:

TkT Kimmo Järvinen

Tarkastaja:

Prof. Valtteri Niemi

Yhteystiedot:

PL 68 (Gustaf Hällströmin katu 2a)

00014 Helsingin yliopisto

Sähköpostiosoite: info@cs.helsinki.fi

URL:<http://www.cs.helsinki.fi>

HELSINGIN YLIOPISTO – HELSINGFORS UNIVERSITET – UNIVERSITY OF HELSINKI

Tiedekunta – Fakultet – Faculty Matemaattis-luonnontieteellinen tiedekunta	Koulutusohjelma – Utbildningsprogram – Study programme Tietojenkäsittelytieteen kandiohjelma	
Tekijä – Författare – Author		
Susanna Rajamäki		
Työn nimi – Arbetets titel – Title		
Spectre- ja Meltdown-tyyppiset hyökkäykset		
Ohjaajt – Handledare - Supervisors		
TkT Kimmo Järvinen		
Työn laji – Arbetets art – Level Kandidatkielma	Aika – Datum – Month and year 31.5.2020	Sivumäärä – Sidoantal – Number of pages 32 sivua
Tiivistelmä – Referat – Abstract		
<p>Tietokonejärjestelmien on oltava turvallisia ja luotettavia käyttää. Sivukanava on mikä tahansa tietokonejärjestelmän laskennan havaittava sivuvaikutus, jota voidaan mitata. Sivukanavahyökkäyksiä on perinteisesti käytetty tarkkailemaan vain sellaista laskentaa, joka tapahtuu ohjelman normaalilin suorituksen aikana. Transienttisuorittamiseen kohdistuvat hyökkäykset ovat aivan eri tason sivukanavauhka, joka on vasta viime vuosina löydetty. Transienttisuorittaminen on arkkitehtuurisella tasolla poispyyhittävä suorittamista. Vaikka siitä ei jää tietokoneen käyttäjälle näkyvään tilaan jälkiä, niin mikroarkkitehtuuriseen tilaan jää. Tämä on perusta muun muassa uudenlaisille Spectre- ja Meltdown-tyyppisille hyökkäyksille, jotka hyödyntävät transienttisuorittamisen mikroarkkitehtuurisia sivuvaikutuksia selvittääkseen salaisuuksia.</p> <p>Transienttisuorittamisesta johtuvia haavoittuvuuksia on hoidettu ohjelmistotason ratkaisuilla, jotka paikkaavat tietoturva-aukoja yksi kerrallaan. Haavoittuvuudet johtuvat laitteistotason suunnittelusta. Vaikean korjaamisen lisäksi transienttisuorittamiseen kohdistuvia hyökkäyksiä on vaikea toteuttaa. Ne ovat kuitenkin niin uusi asia, että niiden merkittävyyttä voi päävästi arvioida vasta tulevaisuudessa.</p>		
<p>ACM Computing Classification System (CCS)</p> <p>Security and privacy → Security in hardware → Hardware attacks and countermeasures → Side-channel analysis and countermeasures</p> <p>Security and privacy → Systems security → Operating systems security</p>		
Avainsanat – Nyckelord – Keywords		
Spectre, Meltdown, transienttisuorittaminen, sivukanavahyökkäys		
Säilytyspaikka – Förvaringsställe - Where deposited		
Helsingin yliopiston kirjasto		
Muuta tietoa – Övriga uppgifter – Additional information		

Sisällys

1	Johdanto.....	1
2	Sivukanavahyökkäykset	4
3	Transienttisuorittamiseen kohdistuvat hyökkäykset	6
4	Laitteisto-ominaisuksien haavoittuvuuksia.....	9
4.1	Muistihierarkia	10
4.2	Epäjärjestykssä suorittaminen	11
4.3	Spekulaatiivinen suorittaminen.....	12
4.4	Haaran ennakkointi.....	13
4.5	Keskeytykset ja poikkeukset	14
5	Spectre-tyyppiset hyökkäykset	16
5.1	Pohjustusvaihe	17
5.2	Spekulaatiivinen suorittaminen ja salaisuuden paljastuminen.....	19
5.3	Spectre-hyökkäyksiä.....	21
6	Meltdown-tyyppiset hyökkäykset.....	22
6.1	Meltdown	24
6.2	Foreshadow	26
7	Pohdintaa	28
	Lähteet.....	29

1 Johdanto

Tietokonejärjestelmien on oltava turvallisia ja luotettavia käyttää. Tämä toteutetaan huolehtimalla eheydestä, saatavuudesta ja luottamuksellisuudesta [1]. Eheys tarkoittaa sitä, että tieto on oikeaa eikä se muutu, katoa tai vahingoitu virheiden tai luvattomien toimenpiteiden takia. Saatavuus tarkoittaa sitä, että tiedot ovat helposti oikeutettujen käyttäjien käytössä. Luottamuksellisuus tarkoittaa sitä, ettei tietoa vuoda vahingossa ja että käyttäjä pääsee yksityiseen tietoon käsiksi vain, jos on autorisoitu siihen. Luottamuksellisuutta vahvistetaan muun muassa pääsyoikeuksien tarkastamisilla sekä eristämislaitteilla ja salauksilla. Nykyaisien tietokonejärjestelmien tietoturvan yksi perusta on muistin eristäminen siten, että käyttöjärjestelmämuistin alueet merkitään eisävätettaviksi ja suojataan käyttäjältä [2].

Kryptografista salausta käytetään estämään tietojen vuotamista [3]. Pitkään kryptografisten salausten murtaminen tarkoitti lähinnä sitä, että matemaattisesti selvitettiin salausalgoritmi ja pääteltiin alkuperäinen viesti salatusta viestistä (ciphertext). Jos hyökkääjä saa salausavaimen hallintaansa, niin salatun viestin purkaminen on triviaalia. Sivukanavahyökkäykset pystyvät selvittämään salausavaimen ilman, että tietävät alkuperäisen ja salatun viestin yhteyttä [4].

Sivukanava on mikä tahansa tietokonejärjestelmän laskennan havaittava sivuvaikutus, jota hyökkääjä voi mitata ja johon hyökkäys mahdollisesti voi vaikuttaa [4]. Tällaista ei-toiminnallista, mitattavaa informaatiota ovat esimerkiksi virrankulutus, välimuistin (cache) sisältö tai aika, joka algoritmin laskentaan käytetään [3]. Yksi ensimmäisistä sivukanavahyökkäyksistä kirjoitetuista

tutkimusjulkaisuista näytti, miten RSA:n yksityisen avaimen voi selvittää mittaan kauan aikaa kuluu viestin purkamiseen [5].

Sivukanavahyökkäyksiä on perinteisesti käytetty tarkkailemaan vain sellaista laskentaa, joka tapahtuu ohjelman normaalien suorituksen aikana [6]. Tällöin tietovuodon riskin voi päättää arvioimalla algoritmia ja minkälaisia dataa se käsittelee. Sivukanavariskin on nähty lähinnä koskevan salausalgoritmeja (encryption algorithms) ja muita ohjelmia, jotka tekevät paljon laskentaa arkaluontoisella datalla. Tällaisia sivukanavahyökkäyksiä on mahdollista estää vaikeuttamalla arkaluontoisia algoritmeja niin, ettei niiden vuotamaa tietoa voi hyödyntää. Hyökkääjää voi hämätä esimerkiksi tekemällä suorituksen kannalta ylimääräistä laskentaa. Tietovuodot transientisuurittamisen (transient execution) kautta ovat aivan eri tason sivukanavauhka, joka on vasta viime vuosina löydetty.

Muistihauun hitaus on usein suorituksen pullonaula. Prosessorit käyttävät muun muassa haaran ennustamista ja spekuloivaa suoritusta maksimoidakseen tehokkuuden [6]. Esimerkiksi kun ohjelman suoritus haarautuu moneen eri vaihtoehtoon ja oikea haara riippuu muistihauun tuloksesta, niin prosessori arvaa suunnan ja lähtee suorittamaan joitain ohjelman haaraa jo ennen kuin muistihaku on valmistunut. Jos muistihauun tulos vahvistaa arvauksen menneen oikein, niin ennakoitu suoritus viedään loppuun ja prosessori jatkaa siitä eteenpäin. Tämä nopeuttaa prosessorin toimintaa. Jos arvaus menee väärin, niin prosessori pyyhkii pois suorittamisen tulokset ja siirtyy suorittamaan oikeaa haaraa. Tälläista poispyyhitystä suorittamista kutsutaan transientisuurittamiseksi.

Vaikka suorittamisen vaikutukset hylätään eivätkä ne näy käyttäjälle, niin transientisuurittamisen mikroarkkitehtuurin tekemät sivuvaikutukset jäävät

jäljelle [7]. Tämä on perusta muun muassa Spectre-, Meltdown- ja Foreshadow-hyökkäyksille, jotka hyödyntävät transienttisuorittamisen mikroarkkitehtuurisia sivuvaikutuksia saadakseen selville salaista tai arkaluontoista tietoa.

Tämän kandidaatintutkielman luvussa kaksi kerrotaan sivukanavahyökkäyksistä ja luvussa kolme transienttisuorittamiseen kohdistuvista hyökkäyksistä. Luvussa neljä kerrotaan laitteisto-ominaisuuksista, joita transienttisuorittamiseen kohdistuvat hyökkäykset hyödyntävät. Luvussa viisi kerrotaan Spectre-tyyppisistä hyökkäyksistä ja luvussa kuusi Meltdown-tyyppisistä hyökkäyksistä. Viimeisessä luvussa on omaa pohdintaani kandidaatintutkielmassani käsittelemistäni asioista.

2 Sivukanavahyökkäykset

Sivukanavahyökkäykset eivät ole uusi asia [8]. Varhainen esimerkki sellaisesta tehtiin 1970-luvulla salasanan murtamishyökkäyksessä TENEX-häytöjärjestelmälle. Käyttöjärjestelmän suorittaessa salasanan validointia se teki hakuja muistin tietyille sivulle (pages) [9]. Hyökkäyksessä sijoitettiin käyttäjän antama syöte näiden sivujen rajalle niin, että ensimmäinen ei-tunnettu merkki oli ensimmäisen sivun lopussa. Jos merkki oli oikea, niin käyttöjärjestelmä siirtyi seuraavalle sivulle, joka laukaisi sinne asetetun poikkeuksen. Tämän antoi hyökkääjälle tehokkaan tavan arvata salasana merkki kerrallaan.

Sivukanavahyökkäyksen erityinen käyttötapaus on piilokanava (covert channel) [2]. Piilokanava on kommunikointiväylä, jota ei ole tarkoitettu tai suunniteltu tiedon siirtämiseen lähettilään ja vastaanottajan välillä [10]. Piilokanava eroaa sivukanavasta siinä, että sivukanavassa lähettilä ei tarkoituksella välitä tietoa vastaanottajalle. Sivukanavassa tiedon lähetäminen tai oikeastaan vuotaminen on sivuvaikutus, joka syntyy tietokoneen käytön yhteydessä. Piilokanavassa hyökkääjä kontrolloi sekä sivuvaikutuksen aikaansaavaa osaa että sivuvaikutusta mittaavaa osaa [2]. Esimerkiksi HTTP-otsikkotietoja voi käyttää piilokanavana viestin lähetämiseen [10].

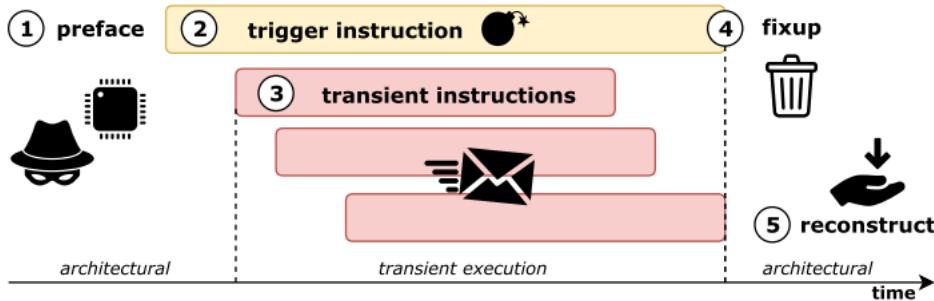
Prosessoriin kohdistuvia sivukanavahyökkäyksiä ovat esimerkiksi differentiaalinen virta-analyysi (differential power analysis) ja välimuistiin kohdistuva ajastushyökkäys (cache timing attack) [11]. Differentiaalinen virta-analyysi tehdään mittaamalla prosessorin virrankulutuksen vaihtelua ohjelman suorituksen aikana. Sen tarkoituksena on tunnistaa tiettyjen käskyjen virrankulutusta. Välimuistiin

kohdistuvat ajastushyökkäykset, kuten Flush+Reload, perustuvat siihen, että datan noutaminen keskusmuistista on hidasta ja välimuistista nopeaa. Flush+Reload on välimuistiin kohdistuva mikroarkkitehtuurinen hyökkäystekniikka, joka havaitsee, kun jaettuun muistipaikkaan tehdään haku [12]. Tekniikka sisältää kaksi pääoperaatiota. Flush-operaatio tyhjentää monitoroidun muistipaikan sisällön pois välimuistista. Reload-operaatio mittaa kauanko kestää päästä käsiksi monitoroituun paikkaan. Hyökkääjä päättelää mitatun ajan perusteella, että vietinkö monitoroidun paikan sisältö välimuistiin operaatioiden välissä.

Sekä virrankulutus että ajastus ovat prosessorien toiminnan sivuvaikutuksia [11]. Viime vuosina prosessorien toiminnallisuuden sivuvaikutukset ovat herättäneet huomiota muun muassa transientisuoittamista hyödyntävien Spectren, Meltdownin ja Foreshadow myötä.

3 Transienttisuorittamiseen kohdistuvat hyökkäykset

Transienttisuorittamiseen kohdistuvat hyökkäykset käyttävät hyväkseen transienttikäskyjä (transient instructions) [7]. Transienttikäskyt ovat käskyjä, jotka suoritetaan spekulatiivisesti tai epäjärjestysessä, mutta joita ei viedä loppuun [13]. Transienttisuorittaminen voi paljastaa tietoa mikroarkkitehtuurisen tilan kautta [7]. Transienttikäskyjä esiintyy jatkuvasti prosessorin suorittaessa edeltä käsin asioita minimoidakseen viiveen ja maksimoidakseen tehokkuuden [2].



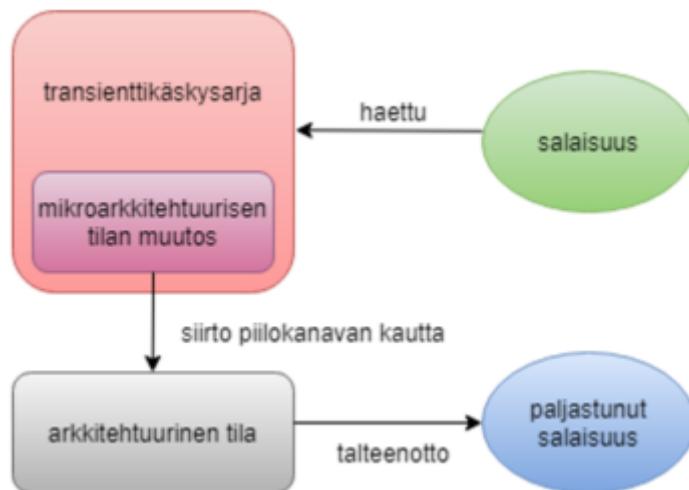
Kuva 3.1 Transienttisuorittamiseen kohdistuvan hyökkäyksen vaiheet [7]

Kuvasta 3.1 näkyy miten transienttisuorittamiseen kohdistuva hyökkäys etenee viidessä vaiheessa [7].

1. Valmistellaan uhrin mikroarkkitehtuuri sopivan tilaan esimerkiksi väärinkouluttamalla haaran ennakkointiyksikköä.
2. Suoritetaan laukaiseva käsky (trigger instruction).
3. Ennen kuin laukaiseva käsky suoritetaan loppuun, niin prosessori alkaa suorittaa transienttikäskysarjaa (transient instruction sequence), joka tekee

muutoksia mikroarkkitehtuurisella tasolla. Hyökkääjä ohjaa transienttikäskyt toimimaan mikroarkkitehtuurisen piilokanavan lähettiläänä.

4. Prosessori tyhjentää suoritusliukuhihnan (execution pipeline) laukaisevan käskyn jäljiltä ja hylkää kaikki arkkitehtuuriset muutokset, jotka transienttisuorittaminen teki.
5. Piilokanavan vastaanottaja lukee ei-autorisoidun transienttilaskennan (transient computation) tulokset arkkitehtuurisella tasolla.



Kuva 3.2 Salaisuuden paljastuminen transienttisuorittamisella

Kuva 3.2 näyttää miten transienttisuorittamiseen kohdistuva hyökkäys urkkii salaisuuden. Transienttikäskysarja hakee salaisuuden, joka näkyy mikroarkkitehtuurisen tilan muutoksesta. Piilokanavaa pitkin tehdään siirto arkkitehtuuriseen tilaan, jossa salaisuus paljastuu talteenoton kautta.

Tunnetut transienttisuorittamiseen kohdistuvat hyökkäykset voidaan tällä hetkellä jakaa kolmeen luokkaan [13]. Spectre-tyyppiset hyökkäykset hyödyntävät väärinennakointia ja Meltdown-tyyppiset hyökkäykset poikkeuksia [7]. Tämän

lisäksi on olemassa mikroarkkitehtuurinen dataotanta (microarchitectural data sampling) eli MDS-hyökkäykset, jotka hyödyntävät Intelin prosessorien optimointia [13]. Tunnettuja MDS-hyökkäyksiä ovat ZombieLoad, Fallout ja RIDL. MDS-hyökkäykset rajataan tutkielman ulkopuolelle.

4 Laitteisto-ominaisuksien haavoittuvuuksia

Käskykanta-arkkitehtuuri (instruction set architecture, ISA) on laitteiston (hardware) ja ohjelmiston (software) välillä oleva rajapinta [7]. Se sisältää tietokoneen toteutuksen mallin sellaisena kuin se näkyy käyttäjälle. Käskykanta-arkkitehtuuri määrittelee muun muassa rekisterit, konekielen käskykannan, muistin osoitusmuodon ja arkkitehtuurin, suorittimen tilan ja keskeytysten (interrupts) käsittelyn. Esimerkiksi x86 ja ARMv8 ovat käskykanta-arkkitehtuureja. Mikroarkkitehtuuri kuvilee, miten käskykanta-arkkitehtuuri on toteutettu prosessorilla. Mikroarkkitehtuuri sisältää muun muassa suoritusyksiköt (execution units) ja välimuistin. Sama käskykanta-arkkitehtuuri voidaan toteuttaa erilaisilla mikroarkkitehtuureilla.

Sekä käskykanta-arkkitehtuuri että mikroarkkitehtuuri ovat tilallisia (stateful) [7]. Käskykanta-arkkitehtuurissa tilaan kuuluu esimerkiksi data, joka on rekistereissä tai keskusmuistissa tuloksellisen laskennan jälkeen. Käskykanta-arkkitehtuurin tila on näkyvä eli ohjelmoija voi tarkastella sitä. Mikroarkkitehtuuriseen tilaan kuuluvat esimerkiksi välimuistissa olevat merkinnät, osoiteenmuunnospuskuri (translation lookaside buffer, TLB) ja suoritusyksikköjen käyttö. Mikroarkkitehtuurinen tila on näkymätöntä eli ohjelmoija ei voi tarkastella mikroarkkitehtuurista tilaa suoraan vaan ainoastaan epäsuorasti.

Tietokoneen prosessori suorittaa laskentaa paljon nopeammin kuin hakee muistista tietoa. Tätä nopeuseroa tasoittamaan on kehitetty erilaisia mikroarkkitehtuurisia

keinoja, kuten muistihierarkia (memory hierarchy), epäjärjestyksessä suorittaminen (out-of-order execution), spekulatiivinen suorittaminen (speculative execution) ja haaran ennakoointi (branch prediction) [6]. Nämä tehostavat prosessorin suoritusta ennakoimalla ohjelman tulevan toiminnan olevan samankaltaista kuin aiempi toiminta.

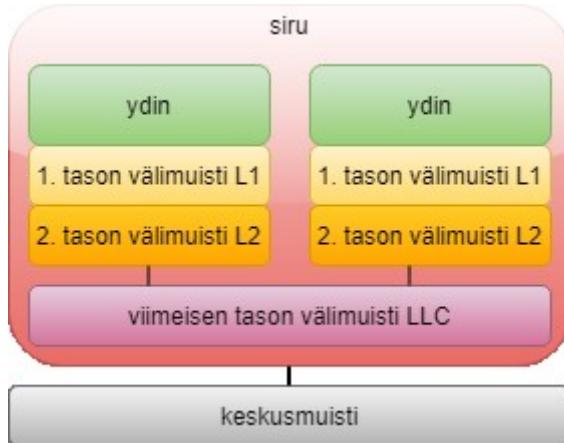
Kun useita ohjelmia suoritetaan samalla laitteistolla joko samanaikaisesti tai ajanjaon kautta, niin yhden ohjelman tekemät mikroarkkitehtuuriset muutokset voivat vaikuttaa toisiin ohjelmiin mikroarkkitehtuurisella tasolla [6]. Tällöin tietoa voi vuotaa epäsuorasti ohjelmasta toiseen mikroarkkitehtuuriseen tilaan kautta. Tätä mikroarkkitehtuurin ominaisuutta voi käyttää erilaisissa hyökkäyksissä.

Seuraavaksi käydään läpi joitakin transientisuorittamisen hyökkäysten hyödyntämiä mikroarkkitehtuurin ominaisuuksia hieman yllä olevaa tarkemmin. Lisäksi kerrotaan keskeytyksistä ja poikkeuksista (exceptions), koska ne ovat oleellisia Meltdown-tyypissä hyökkäyksissä.

4.1 Muistihierarkia

Tiedon hakuaika muistista lyhenee, kun viimeksi käytettyä dataa puskuroidaan eli varastoidaan rekistereihin tai välimuistiin [3]. Välimuistiksi sijaitsevat prosessorin ja keskusmuiston välissä pienentääkseen niiden välistä nopeuseroa. Prosessorit käyttävät monen eri tason välimuisteja nopeuttaakseen tiedon saantiaikaa. Jokaisella prosessorin ytimellä (core) on oma ensimmäisen tason (first level) eli L1-välimuisti, joten se on pienempää, nopeampaa ja kalliimpaa kuin viimeisen tason (last level) eli LL-välimuisti [14]. Viimeisen tason välimuisti on yhteinen kaikille

moniytimisen sirun (chip) prosessoreille. Kuvasta 4.1 näkee muistihierarkian moniytimisellä sirulla.



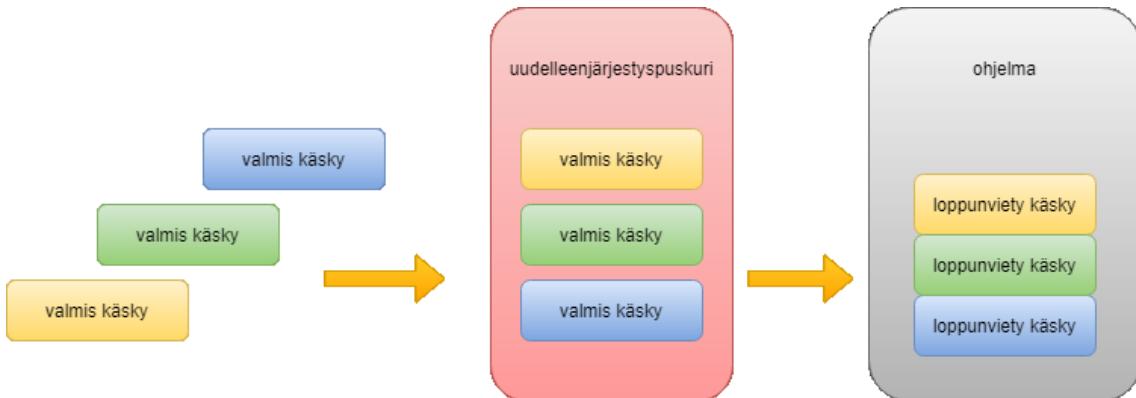
Kuva 4.1 Muistihierarkia moniytimisellä sirulla

4.2 Epäjärjestyksessä suorittaminen

Epäjärjestyksessä suorittaminen on optimointiteknikka, jonka avulla prosessorien erilaiset suoritusyksiköt ovat maksimaalisesti käytössä [2]. Epäjärjestyksessä suorittaessa prosessori ei suorita ohjelman käskyjä järjestyksessä vaan heti, kun kaikki käskyn suorittamiseen tarvittavat resurssit ovat saatavilla. Kun yhden operaation suoritusyksikkö on varattu, niin toiset suoritusyksiköt voivat mennä edelle. Siten käskyjä voi ajaa rinnakkain. Rinnakkain ajaminen lisää tehokkuutta, mutta voi sotkea tiedon oikeellisuuden, kun kaksi tai useampi operaatio tekee haun samaan muistipaikkaan (memory location) [15].

Muistiosoituksen yksinkertaistaminen (memory disambiguation) huolehtii epäjärjestyksessä suorittaessa muistioperaatioiden oikeasta järjestyksestä [15]. Epäjärjestyksessä suorittaessa viitattava muisti noudetaan rekistereihin ja

tallennetaan välimuistiin [2]. Prosessori laittaa valmiit käskyt jonoon uudelleenjärjestyspuskuriin (reorder buffer) [6]. Jonossa olevat käskyt loppuunviedään ohjelman suoritusjärjestyksessä (execution order) eli vasta, kun kaikki edeltävät käskyt ovat loppuunvietyjä (kuva 4.2). Jos epäjärjestyksessä suorittaminen hylätään, niin rekisterit ja muisti pyyhitään, mutta välimuiston sisältö säilyy [2]. Vasta loppuunvienti tekee käskyn näkyväksi arkkitehtuurisella tasolla [6]. Vaikka epäjärjestyksessä suoritettuilla käskyillä ei olisi arkkitehtuurisesti näkyviä vaikutuksia, niin niillä voi olla mikroarkkitehtuurisia sivuvaikutuksia.



Kuva 4.2 Epäjärjestyksessä suoritettujen käskyjen loppuunvienti

4.3 Spekulaatiivinen suorittaminen

Spekulaatiivisessa suorittamisessa prosessori suorittaa käskyjä etukäteen [6]. Tämä voi tapahtua esimerkiksi epäjärjestyksessä suorittamisen aikana silloin, kun prosessori saavuttaa ehdollisen haarautumiskäskyn (branch instruction). Haaran oikea suunta riippuu edeltävistä käskyistä, joiden suorittamista ei ole vielä loppuunviety. Tässä tilanteessa prosessori tallentaa jatkokohdan (checkpoint), joka sisältää nykyisen tilanteen. Sitten prosessori ennakkoi mitä polkua ohjelma kulkee ja spekulaatiivisesti suorittaa käskyjä polun varrelta. Jos ennakointi menee oikein,

niin jatkokohtaa ei tarvita ja spekulatiivisesti suoritetut käskyt loppuunviedään. Jos ennakointi menee väärin, niin prosessori hylkää kaikki spekulatiivisesti suoritetut käskyt, lataa uudelleen ennakointia edeltävän tilan ja suoritusta jatketaan oikeaa polkuuta pitkin. Hylkääminen tehdään niin, että väärällä suorituspolulla (execution path) suoritetut käskyt ja niiden tekemät muutokset eivät tule arkkitehtuurisesti näkyväksi.

Poikkeuksia, jotka tapahtuvat spekuloinnin aikana, ei voi käsitellä ennen kuin tiedetään, että käsky ei ole spekuloiva [8]. Tämä tapahtuu käskyn loppuunviennin yhteydessä. Jotkut prosessorit saattavat jatkaa suorittamista vielä poikkeuksen laukaisevan käskyn, kuten pääsyoikeusrikkomuksen, jälkeen.

Spekulatiiviseen suorittamiseen vaaditaan, että prosessori arvaa haarautumiskäskyn todennäköisen lopputuleman [6]. Oikein osuvat arvaukset nopeuttavat prosessorin suorituskykyä, kun spekulatiivisesti suoritetut käskyt voidaan viedä loppuun. Pahantahtoisilla ohjelmilla on kuitenkin mahdollisuus hyödyntää väärinennakointia, ennen kuin data palautetaan [11].

4.4 Haaran ennakointi

Kun ohjelma sisältää ehdon, niin prosessori ennakoii mitä ehdollista käskyä se lähtee spekuloivasti suorittamaan [11]. Haaran ennakointia on kahdenlaista: haaran suunnan ennakointia ja haaran kohteen ennakointia (branch target prediction) [16]. Haaran suunnan ennakointi on proseduuri, joka ennakoii arvaamalla mihin suuntaan ehdollisessa haarassa mennään. Haaran kohteen ennakointi on proseduuri, joka ennakoii ehdollisen tai ehdottoman haaran seuraavan suoritettavan kohdekäskyn (target instruction). Epäsuorat haarautumiskäskyt (indirect branch instructions)

voivat hypätä useampaan kuin kahteen kohdeosoitteeseen [6]. Prosessorit ennakoivat toimintojen lisäksi myös tietovirtojen riippuvuuksia [7].

Haaran ennakointiyksikkö (branch prediction unit, BPU) ylläpitää eri ehdollisten haarojen välillä keskeytymätöntä käskyjen toimittamista suoritusliukuhihnalille [17]. Kun useaa prosessia suoritetaan samalla fyysisellä ytimellä, niin ne jakavat yhteisen haaran ennakointiyksikön. Jakaminen on käyttöasteen ja kompleksisuuden kannalta hyvä ratkaisu. Samalla se kuitenkin antaa hyökkääjälle mahdollisuuden manipuloida jaetun haaran ennakointiyksikön tilaa ja luoda sivukanavan, jonka avulla voi päättää uhriprosessin haarautumiskäskyn suunnan tai kohteen. Tällainen vuoto voi vaarantaa arkaluontoista dataa.

4.5 Keskeytykset ja poikkeukset

Keskeytykset ja poikkeukset keskeyttävät ohjelman normaalilin suorittamisen ja käynnistävät keskeytyskäsittelijän, joka toimii käyttäjätilan (user mode) sijasta etuoikeutetussa tilassa (kernel mode). Keskeytys on asynkroninen tapahtuma, jonka voi laukaista esimerkiksi I/O-laitte. Poikkeus taas on synkroninen tapahtuma, jonka prosessori luo, kun havaitsee käskyn suorituksen aikana yhden tai useaman edellytyksen täyttyvän.

Käytetään Intelin tapaa jakaa poikkeukset kolmeen luokkaan: vikoihin (faults), ansoihin (trap) ja kumoamisiin (aborts) [18, p. 162]. Vika on poikkeus, joka ei pysäytä suorittamista; ansa on ohjelmallisesti hallittu poikkeus virhetilanteessa ja kumoaminen on suorituksen pysäyttävä poikkeus. Prosessori luo vian, jos havaitsee korjattavan virheen (error) ja ohjelma voi jatkaa suoritusta vian selvittämisen ja uudelleenkäynnistämisen jälkeen [7]. Ansat raportoidaan välittömästi sen jälkeen,

kun käsky on loppuunviety ja ne tulevat arkkitehtuurisesti näkyviksi. Kumoamiset ilmoittavat sellaisesta selvittämättömästä virheestä, joka ei salli uudelleenkäynnistystä kumouksen laukaisseelle tehtävälle.

5 Spectre-tyyppiset hyökkäykset

Spectre-tyyppiset hyökkäykset ovat transientisuoittamiseen kohdistuvia hyökkäyksiä, jotka hyödyntävät väärinennakointia [7]. Spectre-tyyppisessä hyökkäyksessä on hyökkääjä ja uhri, jotka jakavat resurssuja [8]. Spectre-tyyppiset hyökkäykset saavat uhrin spekulatiivisesti suorittamaan operaatioita, joita ei esiinny ohjelman normaalina suorittamisen aikana ja jotka vuotavat piilokanavien kautta uhrin luottamuksellista tietoa hyökkääjälle [6]. Spectre-tyyppiset hyökkäykset olettavat, että spekulatiivisesti suoritettavat käskyt voivat lukea vain sellaisia muistialueita, joihin uhriprossessilla on käyttäjätilassa pääsy. Vaikka prosessori estäisi spekulatiivisen suorittamisen etuoikeutetussa tilassa, niin Spectre-tyyppinen hyökkäys toimii.

Spectre-tyyppisten hyökkäysten rajoite on, että ne vuotavat vain sellaista dataa, joka on ladattu virtuaaliosoiteavaruuteen (virtual address space) [19]. Lisäksi hyökkääjän tulee houkutella uhri tekemään spekulatiivista suoritusta. Spectre-tyyppisiä hyökkäyksiä vastaan on rakennettu ohjelmistopohjaisia ratkaisuja. Esimerkiksi spekulatiivinen suoritus voidaan pysyttää, kun haetaan ei-luotettuja osoitinmuuttujia tai indeksejä, tai estää kokonaan, kun on kyse haavoittuvaista haaroista.

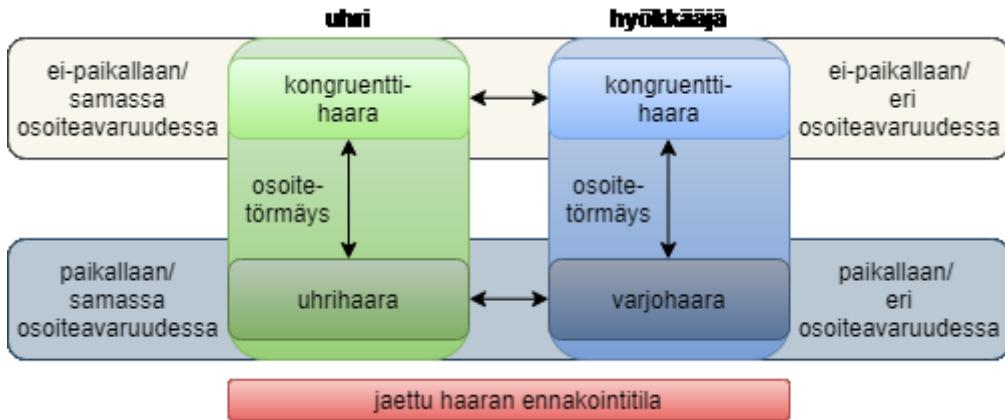


Kuva 5.1 Spectre-tyyppisen hyökkäyksen vaiheet

Spectre-tyyppinen hyökkäys koostuu kolmesta vaiheesta, jotka näkyvät kuvassa 5.1 ja jotka esitellään seuraavaksi.

5.1 Pohjustusvaihe

Yleensä Spectre-tyyppinen hyökkäys alkaa pohjustusvaiheella, jolloin hyökkääjä suorittaa operaatioita, joilla se väärinkouluttaa uhrin tekemään virheellistä spekulointia [6]. Uhrin väärinkouluttaminen voidaan tehdä joko samassa osoiteavaruudessa (same-address-space) tai hyökkääjän hallinnoimassa eri osoiteavaruudessa (cross-address-space) [7]. Lisäksi jos vain virtuaaliositteen (virtual address) osajoukko käytetään ennakoinnin tekemiseen, niin väärinkouluttaminen onnistuu käyttämällä haarautumiskäskyn kongruenttia eli yhteneväistä virtuaaliosoitetta.



Kuva 5.2 Haaran ennakoinnin väärinkouluttamisen strategiat [7]

Kuvassa 5.2 näkyy Spectre-tyyppisten hyökkäysten neljä erilaista hyökkäysstrategiaa, joilla voidaan väärinkouluttaa haaran ennakointi [7]. Ne ovat:

- suorittaa uhrihaaraa uhriprosessissa eli paikallaan (in-place) ja samassa osoiteavaruudessa,
- suorittaa kongruenttihaaraa uhriprosessissa eli ei-paikallaan (out-of-place) ja samassa osoiteavaruudessa,
- suorittaa varjohaaraa (shadow branch) eri prosessissa eli paikallaan ja eri osoiteavaruudessa,
- suorittaa kongruenttihaaraa eri prosessissa eli ei-paikallaan ja eri osoiteavaruudessa.

Pohjustusvaiheessa vihollinen voi myös valmistella piilokanavan, jota käytetään uhrin tietojen keräämiseen [6].

5.2 Spekulatiivinen suorittaminen ja salaisuuden paljastuminen

Toisessa vaiheessa väärinennakoiva prosessori spekulatiivisesti suorittaa käskyjä, jotka siirtävät tietoa uhrin kontekstista mikroarkkitehtuuriseen sivukanavaan [6]. Spectre-tyyppisiä hyökkäyksiä voi luokitella sen mukaan, mitä mikroarkkitehtuurisia ominaisuuksia ne hyväksikäyttäävät [7].

- Spectre-PHT hyväksikäyttää käskysarjamallin historiataulua (pattern history table) eli PHT:a [7]. Käskysarjamallin historiataulu ylläpitää tietoa siitä miin suuntaan haara todennäköisesti lähtee, kun kohdataan jokin tietty käskysarjamalli [20]. PHT ennakoii ehdollisten haarojen lopputuleman [7]. Ensimmäinen esitely Spectre-hyökkäys oli Spectre-PHT.
- Spectre-BTB hyväksikäyttää haaran kohdepuskuria (branch target buffer) eli BTB:ia. Haaran kohdepuskuriissa säilötään niitä laskettuja kohdeosoitteita, joihin mennään haarautumiskäskyissä [16]. Kun vastaavaa haarautumiskäskyä ollaan suorittamassa, niin haaran kohdepuskurista noudetaan kohdeosoitteita. BTB ennakoii haarojen kohdeosoitteita [7]. Toinen esitely Spectre-hyökkäys oli Spectre-BTB.
- Spectre-RSB hyväksikäyttää paluupinopuskuria (return stack buffer) eli RSB:ia [7]. RSB säilöö käskyjen paluuosoitteita (return address) pinoon ja palauttaa päälimmäisen, kun ennakoii paluuosoitteen.
- Spectre-STL hyväksikäyttää muistiosoituksen yksinkertaistajaa (memory disambiguator). Talletus ennen latausta (store to load) eli STL-riippuvuudet vaativat, että muistiin lataamista ei lopputunviedä ennen kuin kaikki samaan osoitteeseen kirjoittavat edeltävät talletukset ovat valmistuneet.

Muistiosoituksen yksinkertaistaja ennakoii, mitkä lataukset voi suorittaa spekulatiivisesti.

Taulukosta 5.1 näkyy miten eri prosessoreilla (Intel, ARM, AMD) on mahdollista suorittaa eri tyyppisiä Spectre-hyökkäyksiä ja millä väärinkoulutusstrategialla hyökkäys onnistuu [7]. Taulukossa ei ole huomioitu Spectre-tyyppisiä hyökkäyksiä vastaan tehtyjä tietoturvapaikkauskia.

valmistaja	prosessi	haara	Spectre-PHT	Spectre-BTB	Spectre-RSB	Spectre-STL
Intel	uhri	uhri	kyllä	kyllä	kyllä	kyllä
	uhri	kongruentti	kyllä	kyllä	kyllä	ei
	hyökkääjä	varjo	kyllä	kyllä	kyllä	ei
	hyökkääjä	kongruentti	kyllä	kyllä	kyllä	ei
ARM	uhri	uhri	kyllä	kyllä	kyllä	kyllä
	uhri	kongruentti	kyllä	ei tietoa	kyllä	ei
	hyökkääjä	varjo	kyllä	kyllä	ei tietoa	ei
	hyökkääjä	kongruentti	kyllä	ei tietoa	ei tietoa	ei
AMD	uhri	uhri	kyllä	kyllä	kyllä	kyllä
	uhri	kongruentti	kyllä	ei tietoa	kyllä	ei
	hyökkääjä	varjo	kyllä	kyllä	kyllä	ei
	hyökkääjä	kongruentti	kyllä	ei tietoa	kyllä	ei

Taulukko 5.1 Spectre-tyyppiset hyökkäykset [7]

Lopuksi hyökkääjä onkii välimuistista salaisen tiedon itselleen käyttämällä piilokanavaa, esimerkiksi Flush+Reload -hyökkäystä [6].

5.3 Spectre-hyökkäyksiä

Spectre Variant 1 ja Spectre Variant 2 ovat ensimmäiset esitellyt Spectre-tyyppiset hyökkäykset [6]. Spectre Variant 1 on Spectre-PHT, joka myrkyttää PHT:n niin, että prosessori väärinennakoi mihiin suuntaan ehdollisessa haarassa lähdetään transienttisuorittamaan [7]. Haaran myrkyttämiseksi eli väärinkouluttamiseksi Spectre Variant 1 suorittaa uhrihaaraa uhriprosessissa eli paikallaan ja samassa osoiteavaruudessa. Ensimmäisessä Spectre-julkaisussa kuvailaan Spectre Variant 1:n tekninen toteutus C:llä ja JavaScriptillä [6].

Spectre Variant 2 on Spectre-BTB-hyökkäys ja se tekee uhrin väärinkouluttamisen paikallaan ja eri osoiteavaruudessa [7]. Toisin kuin Spectre-PHT, jossa transienttisuorittamista voidaan tehdä vain ohjelman väärinennakoidulla polulla, niin Spectre-BTB sallii transienttisuorittamisen mennä myös muihin kohdeosoitteisiin. Jos kohdeosoitteen määrittäminen viivästyy välimuistihudin (cache miss) takia ja haaran ennakkointiyksikkö on väärinkoulutettu ohjaamaan pahansuopiin kohdeosoitteisiin, niin transienttisuorittaminen voi tapahtua hyökkääjän valitsemassa paikassa [6]. Tällöin transienttisuorittaminen voidaan ohjata sellaisiin paikkoihin, joihin ei mennä ohjelman sallitun suorittamisen aikana. Uhrin muisti altistuu paljastumiselle, vaikka ehdollinen haara ei tekisi ollenkaan väärinennakointia. Ensimmäisessä Spectre-julkaisussa kuvailaan Spectre Variant 2:n tekninen toteutus 32-bittisellä Windowsilla [6].

6 Meltdown-tyyppiset

hyökkäykset

Siinä missä Spectre-tyyppiset hyökkäykset hyödyntävät haaran väärinennakointia, niin Meltdown-tyyppiset hyökkäykset hyödyntävät poikkeusten käsittelyä [7]. Transienttisuorittamisen aikana ei käsitellä poikkeuksia. Poikkeukset tulevat arkkitehtuurisesti näkyviksi vasta, kun poikkeuskäsky loppuunviedään. Joissakin mikroarkkitehtuureissa tämä ominaisuus sallii suoritusliukuhihnalla edessäpäin olevien transienttikäskyjen tehdä laskentaa poikkeuksen laukaisevan käskyn eiväautorisoiduille tuloksiille. Prosessorin käskyjen loppuunvienti järjestyksessä – mekanismi (in-order instruction retirement mechanism) hylkää tällaisen laskennan tulokset, mutta salaisuudet voivat vuotaa mikroarkkitehtuuristen piilokanavien kautta.

Meltdown-tyyppinen hyökkäys koostuu kahdesta osasta [2]. Ensinnä prosessori saadaan suorittamaan sellaisia käskyjä, jotka eivät tapahtuisi oikealla suorituspolulla. Toiseksi siirretään transienttikäskysarjan mikroarkkitehtuurinen sivuvaikutus arkkitehtuuriseen tilaan, jotta vuodettua salaisuutta saadaan käsiteltyä. Tähän voi käyttää esimerkiksi Flush+Reload –hyökkäystä [12]. Tunnettuja Meltdown-tyyppisiä hyökkäyksiä ovat muun muassa Meltdown ja Foreshadow [7].

hyökkäys	muita nimiä	#GP	#NM	#BR	#PF	U/S	P	R/W	RSVD	XD	PK
MD-GP	(Variant 3a)	x				-	-	-	-	-	-
MD-NM	(Lazy FP)		x			-	-	-	-	-	-
MD-BR				x		-	-	-	-	-	-
MD-US	(Meltdown)				x	x					
MD-P	(Foreshadow)				x		x		x		
MD-RW	(Variant 1.2)				x			x			
MD-PK					x						x

Taulukko 6.1 Meltdown-tyyppisiä hyökkäyksiä [7]

Taulukosta 6.1 näkyy miten Meltdown-tyyppisiä hyökkäyksiä voi luokitella sen mukaan, millä poikkeustyyppillä ne käynnistävät transienttisuorittamisen [7]. Sivuvika- (page fault) eli PF-poikkeusta hyödyntäviä hyökkäyksiä voi lisäksi vielä jaotella sen mukaan, mitä pääsyoikeusbittiä (permission bit) ne hyödyntävät. Poikkeuksista vian jälkeen tehdään transienttisuorittamista, mutta ansan ja kumoamisen jälkeen sitä ei ole havaittu. Tästä voi olettaa, että Meltdown-tyyppinen hyökkäys voi hyödyntää poikkeuksista vain vikoja. Tätä ei kuitenkaan voi tietää aivan varmasti. Vikoja voi tapahtua koska tahansa käskyjä suorittaessa.

Meltdown-tyyppiset hyökkäykset voivat vuotaa muistin, välimuistin ja rekistereiden salaisuuksia ja osa voi ylittää nykyisen käyttöoikeustason (current privilege level) [7].

	<i>MD-GP</i>	<i>MD-NM</i>	<i>MD-BR</i>	<i>MD-US</i>	<i>MD-P</i>	<i>MD-RW</i>	<i>MD-PK</i>
<i>Intel</i>	1	1	1	1	1	1	1
<i>ARM</i>	1	-	-	1	0	1	-
<i>AMD</i>	0	0	1	0	0	0	-

Taulukko 6.2 Meltdown-tyyppisille hyökkäyksille haavoittuvaiset prosessorit [7]

Taulukosta 6.2 näkyy, minkä valmistajien prosessorit ovat haavoittuvaisia millekin Meltdown-tyyppisille hyökkäyksille [7]. Numero yksi tarkoittaa, että valmistajan jonkin prosessorin tiedetään olevan haavoittuvainen; numero nolla tarkoittaa, että valmistajan prosessorien ei tiedetä olevan haavoittuvaisia ja viiva sitä, että hyökkäystyyppi ei ole sovellettavissa kyseisen valmistajan prosessoriin. Taulukossa ei ole huomioitu Meltdown-tyyppisiä hyökkäyksiä vastaan tehtyjä tietoturvapaikauksia.

6.1 Meltdown

Meltdown lukee käyttöjärjestelmämuistia käyttäjätilasta [2]. Meltdown tekee sen hyödyntämällä sivuvikapoikkeusta ja U/S-pääsyoikeusbittiä [7]. Meltdown pääsee käsiksi muistiin ja välimuistiin ja voi ylittää nykyisen käyttöoikeustason. Sen pitää kuitenkin käyttää valideja muistiosoitteita [19].

Meltdown koostuu kolmesta vaiheesta:

1. luetaan salaisuus
2. lähetetään salaisuus
3. vastaanotetaan salaisuus [2].

Salaisuuden lukemiseksi pitää data tuoda rekisteriin [2]. Jotta data saadaan ladattua keskusmuistista rekisteriin, viitataan keskusmuistiin virtuaaliositteella. Kääntäessään virtuaaliosoitetta fyysiseksi osoitteeksi prosessori tarkastaa pääsyoikeusbitistä onko virtuaaliosoite saavutettavissa käyttäjätilasta vai etuoikeutetusta tilasta. Muistin eristämistä pääsyoikeusbitillä on pidetty turvallisena ratkaisuna ja käyttöjärjestelmät ovat sijoittaneet koko käyttöjärjestelmäytimen (kernel) jokaisen käyttäjäprosessin virtuaalisosoiteavaruuteen. Kaikki käyttöjärjestelmäytimessä olevat osoitteet kääntyvät valideiksi fyysisiksi osoitteiksi ja prosessori pääsee käsiksi niiden sisältöön.

Kun tehdään haku käyttöjärjestelmäytimen osoitteeseen, niin prosessori luo poikkeuksen pääsyoikeustason (permission level) takia [2]. Meltdown kuitenkin hyväksikäyttää epäjärjestykssä suorittamista, joka suorittaa käskysarjan (instruction sequence) laittoman muistihauun (memory access) ja poikkeuksen luomisen välissä.

Salaisuuden lähettämiseksi epäjärjestykssä suoritettava käskysarja tulee valita niin, että siitä tulee transienttikäskysarja [2]. Jos transienttikäskysarja suorittaa laskentaa salaisuuden perusteella ennen kuin poikkeuskäsky loppuunviedään, niin salaisuus saadaan sijoitettua välimuistiin.

Hyökkääjä vastaanottaa salaisuuden käyttämällä mikroarkkitehtuurista piilokanavaa, joka kuljettaa salaisuuden välimuistista arkkitehtuuriseen tilaan [2]. Toistamalla näitä vaiheita hyökkääjä voi urkkia koko muistin iteroimalla läpi kaikki osoitteet.

Meltdownilta voi suojautua niin, että käyttöjärjestelmä eristää käyttöjärjestelmäytimen omaan osoiteavaruuteen eikä luota pääsyoikeusbitin huolehtivan eristämisestä [19].

6.2 Foreshadow

Foreshadow tai toiselta nimitykseltä L1TF eksittiin, kun tutkittiin Intelin Software Guard eXtensions:n eli SGX:n turvallisuutta [21]. SGX on joillakin Intelin prosessoreilla oleva kokoelma käskykantalaajennuksia, jotka lisäävät tietoturvaa. SGX on laitteistopohjainen luotettu suoritusympäristö (trusted execution environment) eli TEE [16]. SGX lupaa turvallisen suorittamisen enklaaville (enclave) eli muistin yksityisille alueille, vastapuolen hallinnoimilla koneilla. Esimerkiksi pilvipalveluiden yhteydessä SGX takaa, että kun prosessori suorittaa asiakkaan modifioimatonta koodia, niin pilvipalveluiden tarjoaja ei pysty tarkkailemaan suorituksen aikana käsiteltyä dataa.

Meltdownia ei voi käyttää SGX-enklaavia vastaan, koska ei-autorisoidut pääsyyritykset enklaavin muistiin eivät luo poikkeusta [7]. Ilman poikkeusta ei Meltdown-hyökkäys toimi. Foreshadow:ssa hyökkääjä poistaa pääsyoikeusluvat enklaavin muistista. Seuraava pääsy-yritys sivulle luo poikkeuksen. Tämän jälkeen hyökkääjä etenee transienttikäskysarjalla, joka vuotaa salaisuuden piilokanavan kautta.

Foreshadow pääsee käsiksi välimuistiin, muistiin ja rekistereihin [7]. Se voi ylittää nykyisen käyttöoikeustason ja tekee tämän hyödyntämällä sivuvikapoikkeusta ja ”läsnä” -pääsyoikeusbittiä (”present” bit).

Foreshadow:ssa hyökkääjä pääsee käsiksi fyysisiin osoitteisiin, jotka ovat epäkelpoja, toisin kuin Meltdownissa [19]. Foreshadow tekee mahdolliseksi hyökkäävän satunnaisesti eri osoiteavaruuksiin. Tämä on kuitenkin mahdollista vain, jos hyökkääjä voi kontrolloida epäkelpoja fyysisiä osoitteita. Foreshadow:lta voi suojahtua sillä, että käyttöjärjestelmä naamioi epäkelvot fyysiset osoitteet.

7 Pohdintaa

Tietokoneiden prosessorien suunnittelussa on vuosien ajan pyritty mahdollisimman suureen tehokkuuteen ja nopeuteen. Samalla on luotu laitteistotasolla tietokoneisiin sellaisia ominaisuuksia, jotka sisältävät haavoittuvuuksia transienttisuorittamisen takia. Transienttisuorittamisesta johtuvia haavoittuvuuksia on hoidettu ohjelmistotason ratkaisuilla, jotka paikkaavat tietoturva-aukkoja yksi kerrallaan. Prosessorien mikroarkkitehtuurin toteutus on liikesalaisuus eikä avoimesti saatavilla. Tämän takia on perusteltua olettaa, että useita haavoittuvuuksia on vielä löytämättä. Täten voi sanoa, että kaikkia transienttisuorittamiseen kohdistuvia hyökkäyksiä vastaan ei ole mahdollista suojauduttaa. Transienttisuorittamiseen kohdistuvat hyökkäykset ovat ikäviä myös siksi, että mikroarkkitehtuurissa tapahtuvaa epäilyttäävää toimintaa ei voi havaita suoraan ohjelmatasolla. Haavoittuvuuksien juurisyiden korjaaminen laitteistotasolla vaatii aivan uudenlaisia ratkaisuja, joiden takia pitää luopua vuosien aikana tehdystä kehitystyöstä.

Onneksi vaikean korjaamisen lisäksi transienttisuorittamiseen kohdistuvia hyökkäyksiä on vaikea toteuttaa. Niiden tekeminen vaatii monimutkaista teknistä osaamista, joka on kokonaan jätetty tämän tutkielman ulkopuolelle. Vaikka transienttisuorittamiseen kohdistuvat hyökkäykset ovat periaatteessa houkutteleva hyökkäys, niin käytännössä on olemassa paljon helpompia hyökkäyksiä. Tällä hetkellä ei ole tietoa siitä, että transienttisuorittamiseen kohdistuvia hyökkäyksiä olisi toteutettu muuten kuin tutkijoiden toimesta. Toisaalta ne ovat niin uudenlainen hyökkäystyyppi, että tutkijat ovat vasta aloittaneet aiheen parissa eikä niiden merkittävyyttä tulevaisuudessa voi vielä pätevästi arvioida.

Lähteet

- [1] C. Cobb, S. Sudar, N. Reiter, R. Anderson, F. Roesner ja T. Kohno, "Computer security for data collection technologies," *Development Engineering*, osa/vuosik. 3, pp. 1-11, 2018.
- [2] M. Lipp, M. Schwarz, D. Gruss, T. Prescher, W. Haas, A. Fogh, J. Horn, S. Mangard, P. Kocher, D. Genkin, Y. Yarom ja M. Hamburg, "Meltdown: Reading kernel memory from user space," tekijä: *27th USENIX Security Symposium*, 2018.
- [3] Y. Lyu ja P. Mishra, "A Survey of Side-Channel Attacks on Caches and Countermeasures," *Journal of Hardware and Systems Security*, osa/vuosik. 2, nro 1, pp. 33-50, 2018.
- [4] N. Lawson, "Side-Channel Attacks on Cryptographic Software," *IEEE Security & Privacy*, osa/vuosik. 7, nro 6, pp. 65-68, 2009.
- [5] P. C. Kocher, "Timing attacks on implementations of diffie-hellman, RSA, DSS, and other systems," tekijä: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 1996.
- [6] P. Kocher, J. Horn, A. Fogh, D. Genkin, D. Gruss, W. Haas, M. Hamburg, M. Lipp, S. Mangard, T. Prescher, M. Schwarz ja Y. Yarom, "Spectre Attacks: Exploiting Speculative Execution," tekijä: *40th IEEE Symposium on Security and Privacy*, 2019.

- [7] C. Canella, J. Van Bulck, M. Schwarz, M. Lipp, B. Von Berg, P. Ortner, F. Piessens, D. Evtyushkin ja D. Gruss, "A Systematic Evaluation of Transient Execution Attacks and Defenses," tekijä: *SEC'19: Proceedings of the 28th USENIX Conference on Security Symposium*, 2019.
- [8] M. D. Hill, J. Masters, P. Ranganathan, P. Turner ja J. L. Hennessy, "On the Spectre and Meltdown Processor Security Vulnerabilities," *IEEE Micro*, osa/vuosik. 39, nro 2, pp. 9-19, 2019.
- [9] S. Langkemper, "The password guessing bug in Tenex," 1. 11. 2016. [Online]. Available: <https://www.sjoerdlangkemper.nl/2016/11/01/tenex-password-bug/>. [Haettu 20. 4. 2020].
- [10] F. C. Freiling ja S. Schinzel, "Detecting Hidden Storage Side Channel Vulnerabilities in Networked Applications," tekijä: *SEC 2011: Future Challenges in Security and Privacy for Academia and Industry*, 2011.
- [11] A. Johnson ja R. Davies, "Speculative Execution Attack Methodologies (SEAM): An overview and component modelling of Spectre, Meltdown and Foreshadow attack methods," tekijä: *2019 7th International Symposium on Digital Forensics and Security (ISDFS)*, 2019.
- [12] Y. Yarom ja K. Falkner, "FLUSH+RELOAD: a High Resolution, Low Noise, L3 Cache Side-Channel Attack," tekijä: *Proceedings of the 23rd USENIX Security Symposium*, 2014.
- [13] M. Schwarz, M. Lipp, D. Moghimi, J. Van Bulck, J. Stecklina, T. Prescher ja D. Gruss, "ZombieLoad: Cross-Privilege-Boundary Data Sampling,"

- tekijä: *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, 2019.
- [14] F. Liu, Y. Yarom, Q. Ge, G. Heiser ja R. B. Lee, "Last-level cache side-channel attacks are practical," tekijä: *Proceedings - IEEE Symposium on Security and Privacy*, 2015.
- [15] N. Vedula, A. Shriraman, S. Kumar ja W. N. Sumner, "NACHOS: Software-Driven Hardware-Assisted Memory Disambiguation for Accelerators," tekijä: *IEEE Conference Publications; IEEE Xplore; IEEE Journals & Magazines*, 2018.
- [16] S. Lee, M.-W. Shih, P. Gera, T. Kim, H. Kim ja M. Peinado, "Inferring fine-grained control flow inside SGX enclaves with branch shadowing," tekijä: *SEC'17: Proceedings of the 26th USENIX Conference on Security Symposium*, 2017.
- [17] D. Evtyushkin, R. Riley, N. Abu-Ghazaleh ja D. Ponomarev, "BranchScope: A New Side-Channel Attack on Directional Branch Predictor," tekijä: *ASPLOS '18: Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems*, 2018.
- [18] INTEL, Intel® 64 and IA-32 Architectures Software Developer Manuals, osa/vuosik. 1 Basic Architecture, 2019, pp. 6-12.
- [19] S. Van Schaik, A. Milburn, S. Osterlund, P. Frigo, G. Maisuradze, K. Razavi, H. Bos ja C. Giuffrida, "RIDL: Rogue in-flight data load," tekijä: *Proceedings - 2019 IEEE Symposium on Security and Privacy*, 2019.

- [20] P.-Y. Chang, M. Evers ja Y. Patt, "Improving branch prediction accuracy by reducing pattern history table interference," *International Journal of Parallel Programming*, osa/vuosik. 25, nro 5, pp. 339-362, 1997.
- [21] J. Van Bulck, M. Minkin, O. Weisse, D. Genkin, B. Kasikci, F. Piessens, M. Silberstein, T. F. Wenisch, Y. Yarom ja R. Strackx, "Breaking Virtual Memory Protection and the SGX Ecosystem with Foreshadow," *IEEE Micro*, osa/vuosik. 39, nro 3, pp. 66-74, 2019.