

# MELTDOWN & SPECTRE

A new era of microarchitectural attacks

# Indice

- 1) Overview
- 2) Premesse
- 3) Meltdown [CVE-2017-5754]
  - a) Generalità
  - b) Implementare Meltdown
  - c) Impatti
  - d) Proof
  - e) Fixes
- 4) Spectre
  - a) Generalità e varianti a confronto
  - b) Variante 1 [CVE-2017-5753]
  - c) Variante 2 [CVE-2017-5715]
  - d) Fixes
- 5) Impatti sulle prestazioni
- 6) Le varianti continuano
- 7) Conclusioni



# Chi e quando

03/01/2018

Il 3 Gennaio 2018, 6 giorni prima della data di uscita prevista, sono state rese pubbliche le due vulnerabilità Meltdown e Spectre a causa di leak<sup>[1]</sup> e notizie premature sull'accaduto.

## MELTDOWN: (3 team)

- » Jann Horn (**Google Project zero**).
- » Werner Haas, Thomas Prescher (**Cyberus Technology**).
- » Daniel Gruss, Moritz Lipp, Stefan Mangard, Michael Schwarz (**Graz University of Technology**).

## SPECTRE: (2 people)

- » Jann Horn (**Google Project zero**).
- » Paul Kocher. \*

\*in collaboration with, in alphabetical order, Daniel Genkin (University of Pennsylvania and University of Maryland), Mike Hamburg (Rambus), Moritz Lipp (Graz University of Technology), and Yuval Yarom (University of Adelaide and Data61)

# Chi ne può essere affetto?

Praticamente **tutti**.

La maggior parte dei dispositivi ad oggi quali **Desktop**, **Laptop**, **Cloud** e **Smartphone** sono soggetti a queste due famiglie di vulnerabilità.

Il problema di fondo è l'**architettura dei dispositivi**, che per questioni di tendenza, dagli anni '90 ad oggi, adottano tutti un modello molto simile...



## Quali sono i danni

La portata di questi due tipi di attacchi sono principalmente **dati sensibili e password**.



Inoltre gli attacchi possono essere eseguiti in maniera totalmente silenziosa, in modo tale da **non essere individuati** una volta avvenuti.





# Possibili contromisure

## » Antivirus:

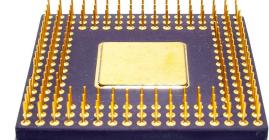
- ◊ **Pro:** bloccano l'attacco alla sua esecuzione.
- ◊ **Cons:** possibile solo se si conoscono in anticipo le firme dei programmi attaccanti.

## » Kernel Patch:

- ◊ **Pro:** risolvono i problemi per Meltdown e Spectre (variante 1).
- ◊ **Cons:** non eliminano completamente tutti gli attacchi.

## » CPU Microcode Update.

- ◊ **Pro:** risolve alla radice ogni problema.
  - ◊ **Cons:** può sacrificare dal 5% fino al 30% di prestazioni, latenza o impossibilità nel ricevere aggiornamenti.
- 



## Reingegnerizzare le CPU

# Vulnerabilità a confronto

## MELTDOWN

Questa vulnerabilità “**rompe**” l’isolamento fondamentale che c’è tra l’userspace e il Sistema Operativo, consentendo all’attaccante di accedere e quindi leggere aree di memoria protette, consultando indisturbatamente informazioni di altri processi e del Sistema Operativo.

## SPECTRE

Questa vulnerabilità “**rompe**” il normale isolamento che c’è tra un’applicazione e un’altra (sandboxing), accedendo ad aree di memoria private dell’applicazione vittima, dando la possibilità all’attaccante di leggere informazioni sensibili e password.



## Andiamo sul tecnico

Per parlare nel dettaglio di queste due vulnerabilità  
dobbiamo chiarire alcuni concetti sulle architetture  
moderne.

In particolare ci soffermeremo su termini chiave quali,  
**esecuzione speculativa, out-of-order execution,**  
**branch prediction, cache e spazi di indirizzamento.**

Infine discuteremo dei **side-channel attack**.



# Andiamo sul tecnico pt. 1

## UN BREVE RIPASSO

### Esecuzione Speculativa

L'esecuzione speculativa e il pipelining dei dati sono delle tecniche utilizzate dai processori moderni per **parallelizzare le unità di elaborazione** in modo tale da aumentare le performance del calcolatore.

### Out-of-order execution

E' una delle tecniche adottate dal processore per attuare l'esecuzione speculativa. Praticamente le operazioni non vengono schedulate in base all'ordine di arrivo, ma in base a come conviene al processore per incrementare le prestazioni, senza alterarne però il risultato finale.

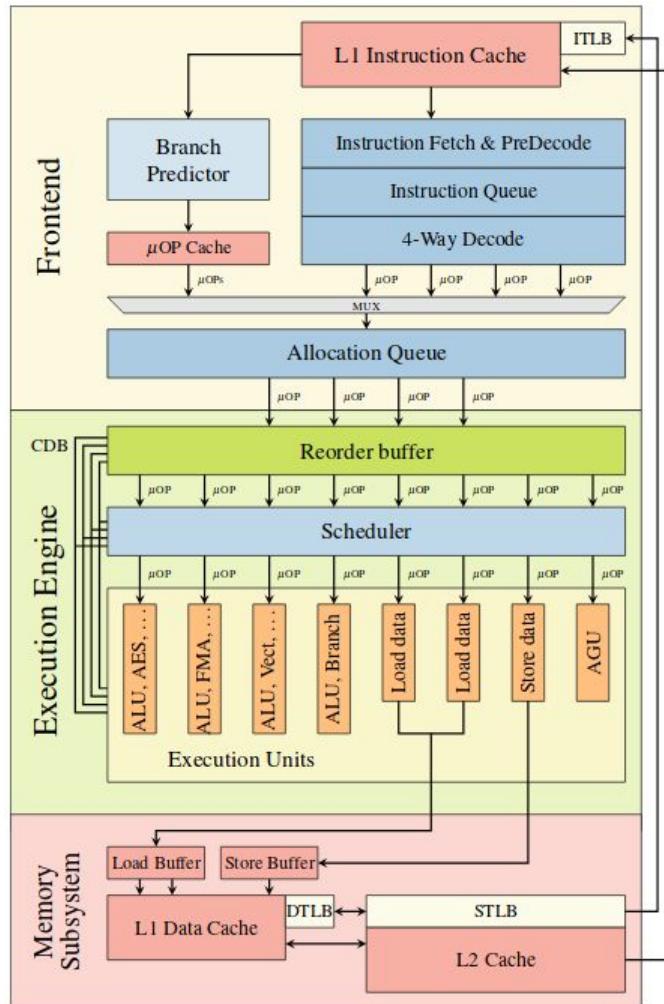
### Branch prediction

E' un'altra delle funzionalità adottate del processore per incrementare le prestazioni.

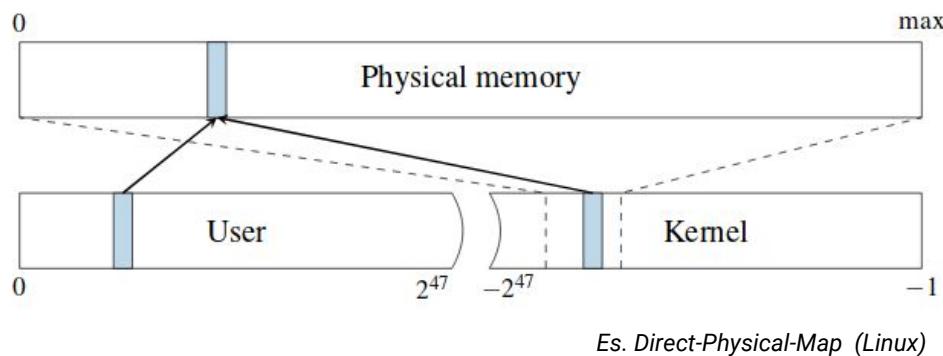
In pratica, attraverso l'uso di un BTB (Branch Target Buffer) cerca di predire quali rami verranno presi nelle istruzioni condizionali.

### Cache

Sono memorie di piccole dimensioni che garantiscono una elevata velocità di accesso. Sono usate all'interno del processore per salvare dati che sono acceduti frequentemente



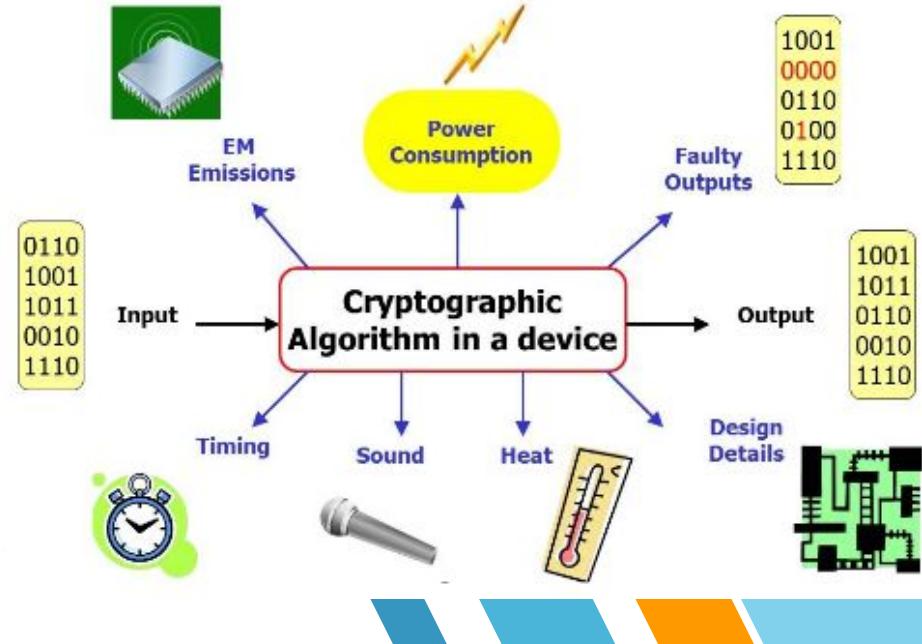
# Andiamo sul tecnico pt. 1.1 (Esecuzione speculativa e Spazio di indirizzamento)

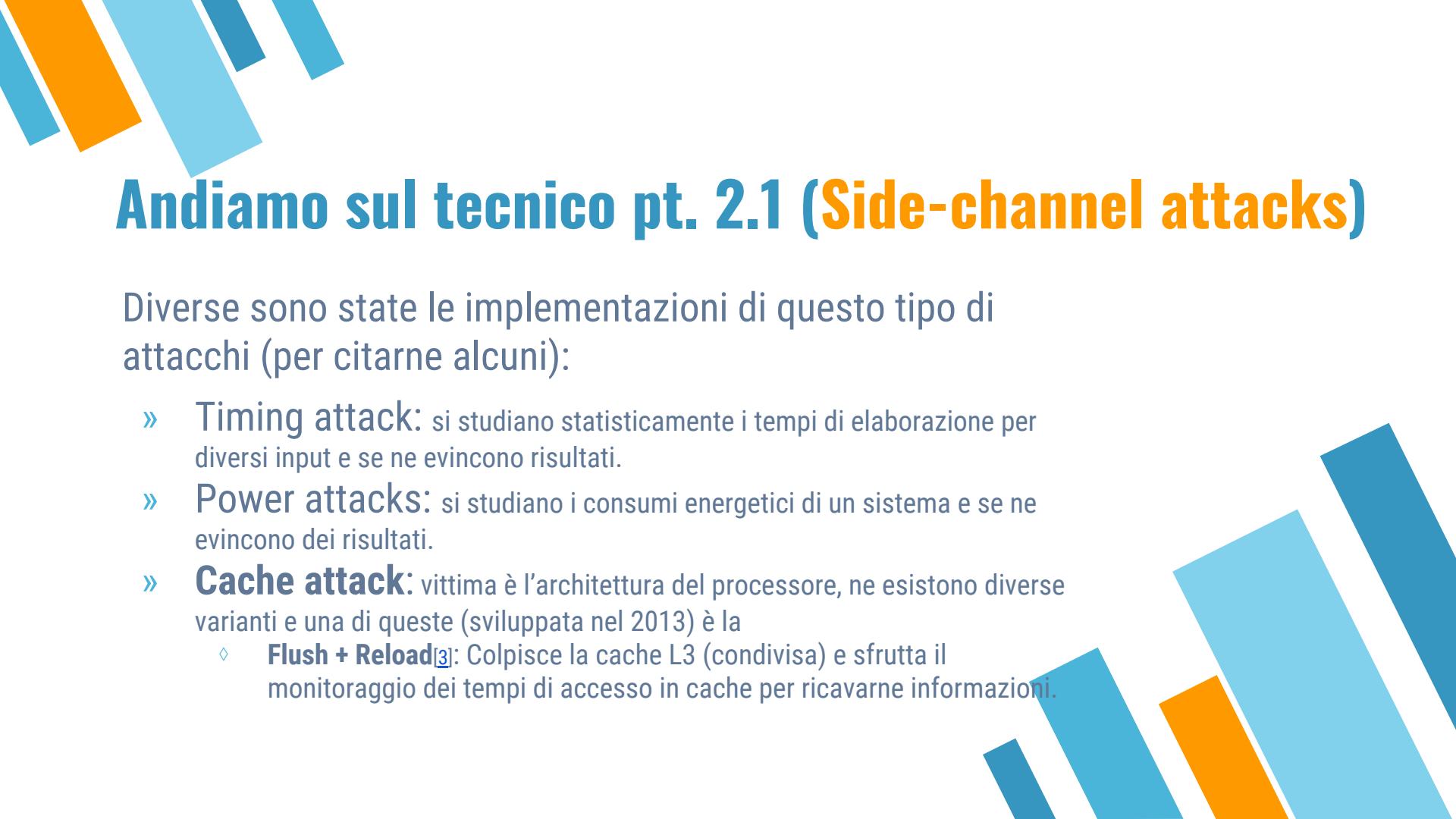


# Andiamo sul tecnico pt. 2 (Side-channel attacks)

Un attacco di tipo side-channel sfrutta un **fuga non intenzionale di informazioni** (es. *consumo di energia, il tempo di esecuzione, le radiazioni elettromagnetiche...*).

Storicamente nati per attaccare sistemi crittografici, il primo attacco risale al 1996 formalizzato da Paul Kocher [2].





# Andiamo sul tecnico pt. 2.1 (Side-channel attacks)

Diverse sono state le implementazioni di questo tipo di attacchi (per citarne alcuni):

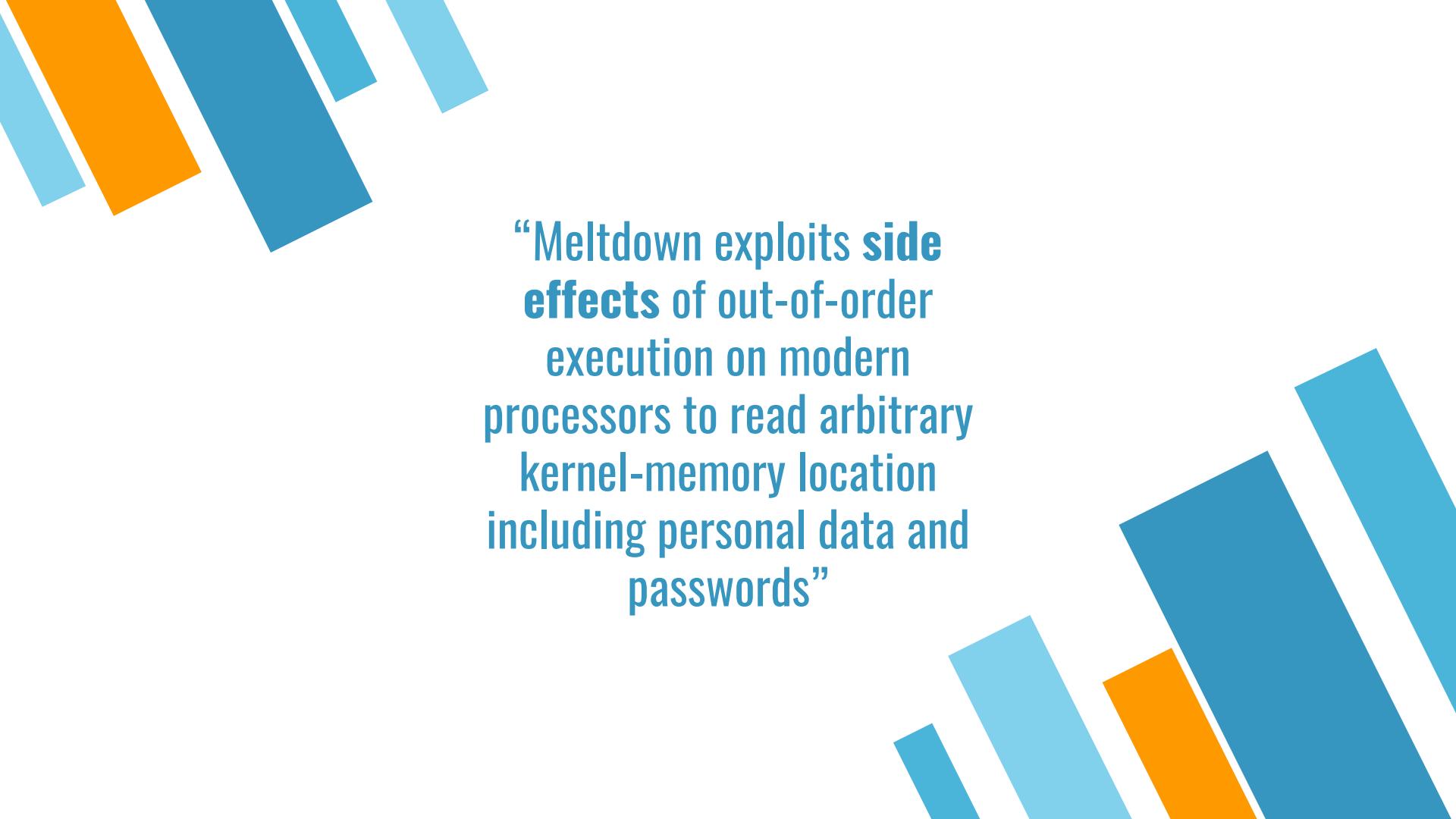
- » **Timing attack:** si studiano statisticamente i tempi di elaborazione per diversi input e se ne evincono risultati.
- » **Power attacks:** si studiano i consumi energetici di un sistema e se ne evincono dei risultati.
- » **Cache attack:** vittima è l'architettura del processore, ne esistono diverse varianti e una di queste (sviluppata nel 2013) è la
  - ◊ **Flush + Reload**<sup>[3]</sup>: Colpisce la cache L3 (condivisa) e sfrutta il monitoraggio dei tempi di accesso in cache per ricavarne informazioni.

1.

# MELTDOWN

A first look [CVE-2017-5754]

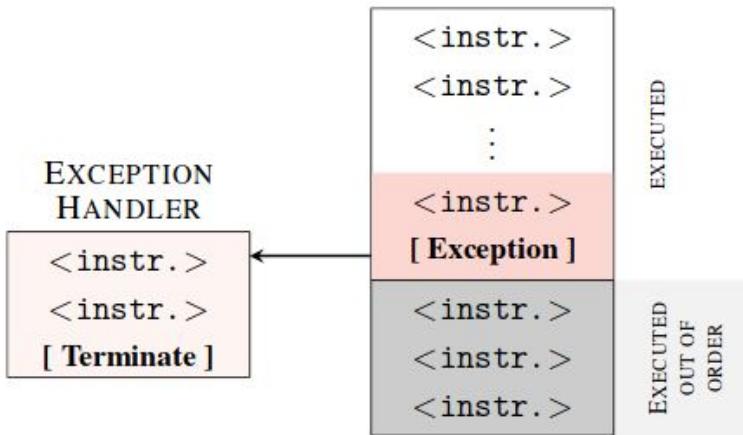




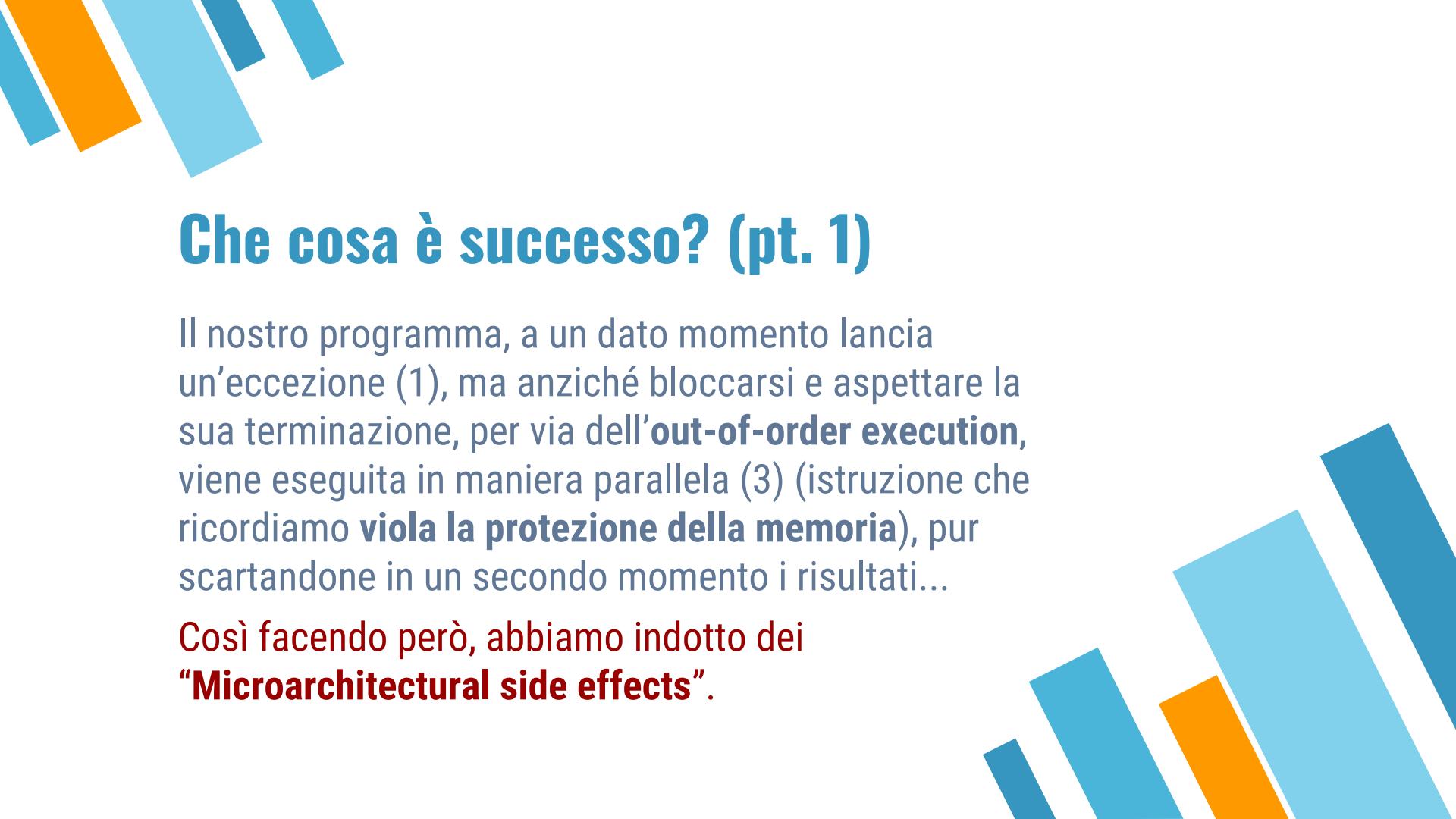
**“Meltdown exploits side effects of out-of-order execution on modern processors to read arbitrary kernel-memory location including personal data and passwords”**

# A “toy example”

```
1. raise_exception();
2. //the line below is never reached
3. access(probe_array[data * 4096]);
```



//è eseguita out of order  
Prende il nome di:  
**Transient Instruction**



# Che cosa è successo? (pt. 1)

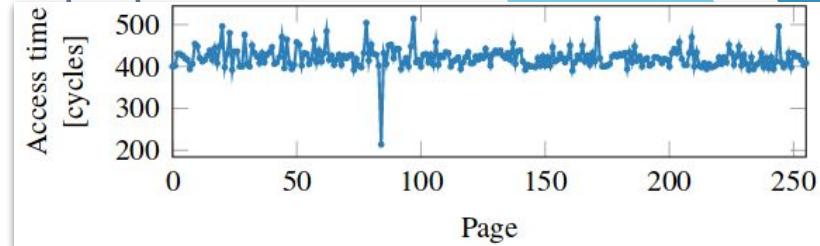
Il nostro programma, a un dato momento lancia un'eccezione (1), ma anziché bloccarsi e aspettare la sua terminazione, per via dell'**out-of-order execution**, viene eseguita in maniera parallela (3) (istruzione che ricordiamo **viola la protezione della memoria**), pur scartandone in un secondo momento i risultati...

Così facendo però, abbiamo indotto dei  
**"Microarchitectural side effects"**.

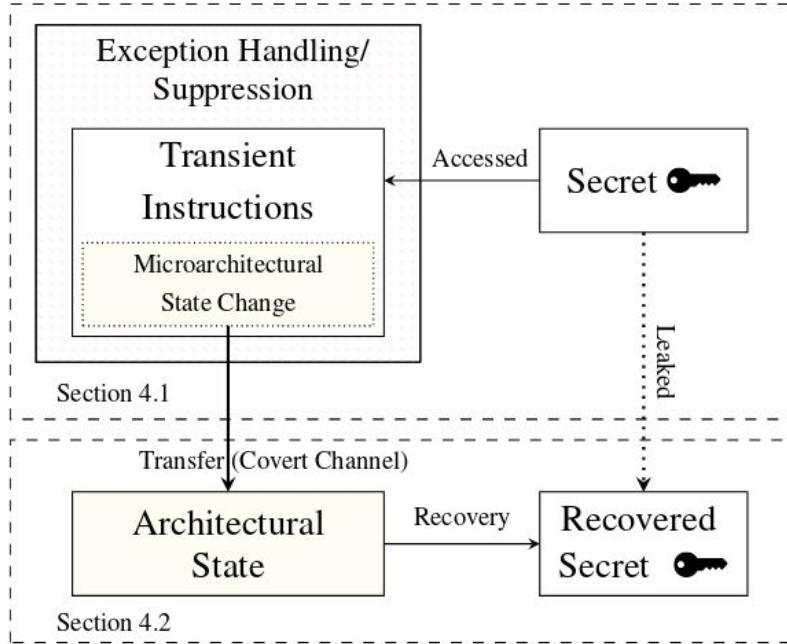
## Che cosa è successo? (pt. 2)

L'esecuzione fuori ordine ha quindi “cambiato lo stato microarchitetturale” del sistema.

Da qui in poi un eventuale **“listener”** può controllare le modifiche avvenute nella cache ed effettuando dei veri e propri **cache attack** (es. Flus + Reload) può, in un secondo momento, far transitare i dati nel proprio spazio.



# How to: Meltdown (una vista ad alto livello)



Per implementare un attacco di tipo Meltdown bisogna principalmente implementare **due Building Blocks**.

1. **Transient Instruction Block**  
(Section 4.1)
2. **Covert Channel Block** (Section 4.2)



## Meltdown: (1. Transient Instruction Block)

Bisogna creare un set di istruzioni che funzioni in maniera simile al “toy” che abbiamo visto prima, cercando di accedere a zone di memoria inaccessibili (es. Indirizzo base del kernel).

La logica è sempre la stessa, sfruttare l'**out-of-order execution** per provocare effetti **side-channel**.

I papers suggeriscono due strade diverse:

- » Exception Handling
  - » Exception Suppression
- 



# Meltdown: (1. Transient Instruction Block)

## Exception handling

Un approccio facile sarebbe quello di **forkare l'applicazione attaccante**, e far eseguire la *Transient Instruction* al figlio (che crasherà), per poi controllare i **side effect** ottenuti in cache al padre.

//Oppure installare un “signal handler” (+ efficente)

## Exception suppression

Un approccio ortogonale sarebbe quello di evitare il nascere dell'eccezione **sopprimendola**.

Questa tecnica risulta più efficiente ma più complicata poichè richiede di effettuare una sorta di training sul **branch predictor**.



## Meltdown: (2. Covert Channel Block)

La seconda parte, è quella che si occupa di riversare i dati dalla cache allo spazio di indirizzamento dell'attaccante (*from microarchitectural state to architectural state*).

Una delle tecniche, efficiente e “noise resistant” è la tecnica del **Flush + Reload** attack (cache attack).

(es. Può essere un processo diverso (anzi lo è) da quello che lancia le Transient Instructions.)



## Meltdown: (RECAP)

1. Il contenuto (inaccessibile) da accedere dall'attaccante deve essere caricato in un **registro**.
  2. Una **transient instruction** accede alla linea di cache relativa al contenuto segreto del registro.
  3. L'attaccante usa la tecnica del **Flush+Reload** per determinare che linea di cache è stata acceduta e ne scruta il contenuto.
- 

# Che impatti può avere?



Un attacco di tipo Meltdown può arrivare a eseguire il  
**dump di tutta la memoria.**

Ma come?

Linux/macOS: direct-physical-map

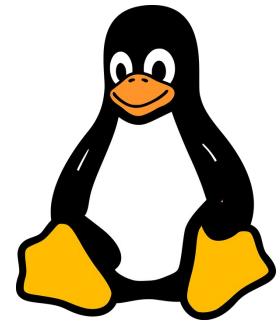
Windows: Paged pool, non paged pool, cached memory

## Ma come?

Nella maggior parte dei sistemi operativi moderni la **memoria virtuale** di un processo è divisa in **memoria utente** e **memoria kernel**, quest'ultima per questioni di efficienza mappa tutta o quasi tutta la memoria fisica (proteggendola comunque da accessi indesiderati).

**Ma...** l'inghippo è proprio qua, **Meltdown**, una volta scoperto l'**indirizzo base del kernel**, può dumpare l'intera memoria. (*Privilege Escalation*)

//Es. con Linux (la metodologia cambia di poco se siamo su Windows, o macOS)



## Come fa a scoprire l'indirizzo base?

- » **nonKASLR**: Linux fino alla versione Kernel 4.12 mappava la direct-physical map all'indirizzo: 0xfffff 8800 0000 0000
- » **KASLR**: Nelle nuove versioni di linux (4.13+) KASLR è attivo di default, ma **ciò non impedisce comunque di ricavare l'indirizzo**, dato che comunque la memoria kernel è allocata contiguamente.

**How to break KASLR:**  
Processore a 64bit ->  $2^{64}$  possibili indirizzi, ma di fatto utilizzati al più 40bit ->  $2^{40}$  possibili indirizzi.  
Supponendo una memoria da 8GB =  $2^{33}$  bit per cercare l'indirizzo base della memoria basta sondarlo in max  $2^{40} - 2^{33} = 2^7 = 128$  accessi.

\***KASLR**: Kernel Access Space Layout Randomization



## Bonus: il caso dei Containers

L'attacco è stato provato essere valido anche su containers come Docker, LXC e OpenVZ.

Ciò è di **importanza critica**, poiché l'attacco non si ferma al solo container da cui parte, ma è capace di leggere anche il contenuto di altri container. Questo poichè il Kernel in sistemi di tipo Docker è **shared**.

Ciò rappresenta un serio problema di sicurezza per i provider dei servizi citati.

# UN PO' DI NUMERI (Meltdown)

CHI NE È AFFETTO?



Qualsiasi processore Intel prodotto dal 1995 sino ad oggi. (Esclusa la serie Itanium e Atom)

PERFORMANCE

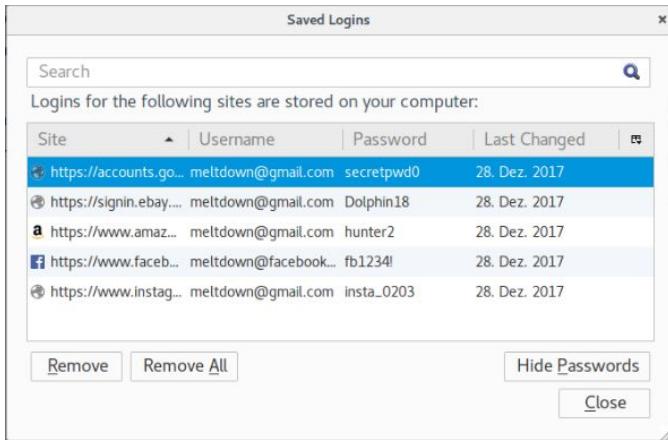


Average reading rates: 503 KB/s

Error rate: 0,02%

(dati con tecnica dell' exception suppression)

# Proof



```
f94b77a0: XX | .....  
f94b77b0: XX XX XX XX XX XX XX XX | .....secr  
f94b77c0: 65 74 70 77 64 30 e5 | etpwd0.....  
f94b77d0: 30 b4 18 7d 28 7f XX XX XX XX | .....  
f94b77e0: XX XX XX XX XX XX XX XX | .....  
f94b77f0: XX XX XX XX XX XX XX XX | .....  
f94b7800: e5 | .....  
f94b7810: 68 74 74 70 73 3a 2f 2f 61 64 64 6f 6e 73 2e 63 | https://addons.c...  
f94b7820: 64 6e 2e 6d 6f 7a 69 6c 6c 61 2e 6e 65 74 2f 75 | dn.mozilla.net/u...  
f94b7830: 73 65 72 2d 6d 65 64 69 61 2f 61 64 64 6f 6e 5f | ser-media/addon_...  
f94b7840: 69 63 6f 6e 73 2f 33 35 34 2f 33 35 34 33 39 39 | icons/354/354399...  
f94b7850: 2d 36 34 2e 70 6e 67 3f 6d 6f 64 69 66 69 65 64 | -64.png?modified...  
f94b7860: 3d 31 34 35 32 32 34 34 38 31 35 XX XX XX XX XX | =1452244815....
```

# Another proof (leggere un immagine dalla memoria)



# CONTROMISURE pt 1



## KAISER - a Short-Term Solution

KAISER<sup>[4]</sup> nasce come meccanismo di protezione per KASLR, ma **si è scoperto essere efficace** contro Meltdown.

KAISER introduce una modifica sostanziale nel kernel, che consiste nel **non mappare il kernel stesso nello user space**.

\*Kernel Address Isolation to have Side-channels Efficiently Removed

# CONTROMISURE pt 1



## KAISER - a Short-Term Solution

### Perchè a Short-Term Solution?

A causa dell'architettura x86, alcune locazioni di memoria privilegiate **devono** essere comunque mappate nell'user space.

Di conseguenza KAISER fornisce solo una **basic protection**.

Overhead trascurabile a favore di **nessun degrado prestazionale e rapidità nel ricevere una cura**.

//Tantochè **verrà introdotta** nelle successive (e precedenti ) versioni di linux con il nome di **KPTI**.

# CONTROMISURE pt 2

## Update Intel microcode

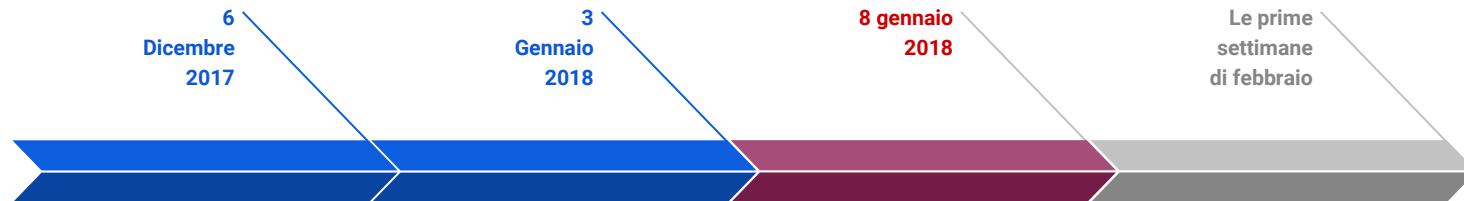


La soluzione migliore in termini di sicurezza è quella di **aggiornare l'hardware**, a scapito però delle prestazioni.

Ma che modifiche adottare?

- » Disabilitare l'out-of-order execution? => **NO!**
- » Serializzare il “controllo permessi” e il “fetch istruzioni”
- » Hard-Split della memoria (Consigliata)

# Roadmap fix



## macOS 10.3.2

L'Apple rilascia una correzione a livello kernel del suo sistema operativo, che sarà successivamente migliorata con la versione di macOS 10.13.3 (23/01/2018)

## Windows 10

Windows 10 riceve la patch KB4056892[5] che risolve i problemi per Meltdown.

Successivamente per il "Patch Tuesday" riceveranno la patch anche altre versioni di Windows (9/01/2018)

## Intel rilascia aggiornamenti del microcodice per alcune CPU [6]

DragonflyBSD, OpenBSD, FreeBSD, NetBSD

Ricevono un fix contro meltdown chiamato SVS (Separate Virtual Space)

2.

# SPECTRE

A first look

[CVE-2017-5753 & CVE-2017-5715]





“Spectre attacks **trick** the processor into speculatively executing instruction sequences that should not have executed during correct program execution”



## Una vista dall'alto

Un attacco di tipo Spectre generalmente si divide in **3 fasi**.

1. Nella prima fase l'avversario esegue operazioni di **training** verso il **Branch Predictor**.
  2. Nella seconda fase si esegue in maniera speculativa l'istruzione che serve a trasferire le informazioni della vittima via **side channel**.
  3. Nella terza fase si recuperano le informazione attraverso un **covert channel** (es. Flush + Reload attack).
- 



## Ancora sull'Esecuzione Speculativa

Se l'esecuzione di un'istruzione dipende da un valore che non sta in cache, il suo raggiungimento in memoria principale richiede qualche centinaio di cicli di clock. Perciò è qui che **l'esecuzione speculativa** entra in gioco, cercando di “**indovinare**” il percorso seguito del programma e quindi eseguendo le istruzioni seguenti in via anticipata.

Il processore fa un **check point** dei suoi registri e va avanti in esecuzione speculativa mentre il valore viene pescato in memoria.

Una volta ritornato il valore, **se il risultato non è quello previsto**, si eliminano i risultati intermedi, e si ripristinano i registri.





# Ma...

L'esecuzione speculativa porta inevitabilmente a dei **side effect**.

**La microarchitettura (in questo caso la cache) è in grado di rivelarci informazioni più o meno sensibili.**



Vediamo adesso il funzionamento delle **due varianti** di Spectre...

# Varianti a confronto

## V1 (Exploiting Conditional Branches)

**CVE-2017-5753**

La variante uno di Spectre consiste in un primo periodo di “**training**” per il **Branch Predictor**, in modo tale da poter fargli eseguire in un secondo momento in maniera speculativa codice che non dovrebbe essere eseguito.

## V2 (Exploit Indirect Branches)

**CVE-2017-5715**

Funziona in maniera simile ad un attacco **ROP**.

L’attaccante deve trovare dei “**gadget**” sul codice della vittima e deve influenzarla (sempre “trainando” il BTB) ad eseguire una **indirect branch instruction**.

# Spectre V1 - Exploiting Conditional Branches

Consideriamo questo esempio (codice di un **programma vittima**):

Il valore x è passato da un origine non fidata (es. Input utente, ritorno da una libreria, ecc...)

```
if(x) < array1_size)
    y = array2[array1[x] * 256];
```

Array2 è lungo 64KB

# Spectre V1 - Exploiting Conditional Branches

Il valore **x deve essere giustamente controllato** se non si vogliono avere accessi non consentiti in memoria (perlomeno è ciò che suggerisce una buona pratica di programmazione)

```
if(x< array1_size)
    y = array2[array1[x] * 256];
```

# Spectre V1 - Exploiting Conditional Branches

Supponiamo ora che:

1. **X** sia scelto in maniera maligna in modo tale che **array1[x]** punti ad un **byte segreto k** (valore da leakare) nello spazio di memoria della vittima.
2. **array1\_size** e **array2** non siano in **cache**, ma **k** lo sia.
3. Nelle esecuzioni precedenti X sia stato passato volutamente < di **array1\_size** (**training del Branch Predictor**)

# Spectre V1 - Exploiting Conditional Branches

Come ottenere una configurazione di cache simile?

2. **array1\_size** e **array2** non sono in **cache**, ma **k** lo sia.

Es1. Tentando di occupare una grossa porzione di cache dal programma attaccante.

Es2. Se si conosce l'architettura (es. x86) con istruzioni primitive come: **clflush**.

```
if(x < array1_size)  
    y = array2[array1[x] * 256];
```

# Spectre V1 - Exploiting Conditional Branches

Cosa succede ora?

1. Viene richiesto di leggere **array1\_size** ---> cache miss ---> CPU ordina di caricarlo.
2. Il Branch Predictor ((mis)trainato a dovere) **assume if=true** eseguendo il corpo.
3. Così viene **letto array1[x]** che è il byte segreto **k (chache hit!)**.
4. L'esecuzione speculativa continua usando k per computare la **lettura di array2[k \* 256]** ---> cache miss ---> se ne ordina la lettura.
5. Nel frattempo però è arrivato il valore di **array1\_size** e ops...  
L'esecuzione speculativa si sbagliava ---> **si ripristinano i vecchi registri**.

```
if(x < array1_size)  
    y = array2[array1[x] * 256];
```

# Spectre V1 - Exploiting Conditional Branches

Qualcosa però è andato storto.

6. La lettura speculativa di array2 ha lasciato dei **side effects** in cache.
7. Per completare l'attacco, l'avversario ha bisogno di controllare i cambiamenti in cache per recuperare il **byte segreto k**.
8. Per accedere al **byte k**, se l'array2 è leggibile dall'attaccante, si possono misurare i timing di accesso in **array2[n \* 256]** dove:
  - a. se **n=k** il tempo di accesso sarà brevissimo
  - b. se **n!=k** i tempi di accessi saranno lunghidandoci informazioni sulla linea di cache corrispondente.



# Spectre V2 - Exploit Indirect Branches

## Come funziona

Similmente a un attacco ROP, l'exploit deve trovare e successivamente usare dei **gadget** (ad es. Guardando i binari della vittima, shared DLL, ecc...).

In aggiunta, l'attaccante allena il **Branch Target Buffer (BTB)** a sbagliare una **indirect branch instruction** e a saltare al gadget.

I soliti **side effect** verranno riversati in cache, e usati dai gadget per leggere arbitrariamente la memoria della vittima.





# Spectre V2 - Exploit Indirect Branches

## How to misstrain the BTB

Dopo aver trovato gli **indirizzi virtuali dei gadget** della vittima,  
**l'attaccante** performa degli indirect branches a questi indirizzi,  
direttamente **dal suo spazio di indirizzamento!**

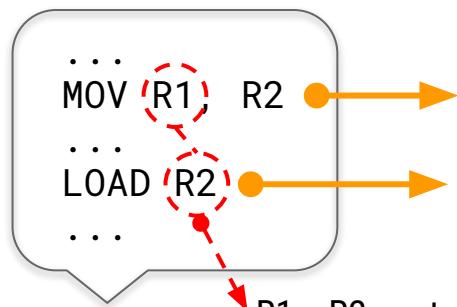
Non importa cosa vi sia mappato nello spazio dell'attaccante, con  
un giusto allenamento del BTB e con i giusti indirizzi virtuali,  
l'attacco andrà a segno.



# Spectre V2 - Exploit Indirect Branches

Per questo attacco di notevole importanza è la scelta dei **gadget nel codice della vittima**.

- Un esempio di gadget può essere questo:



- 1) La prima istruzione muove una locazione di memoria R1 in un registro R2
- 2) La seconda istruzione accede ad R2

R1 e R2 controllati dall'attaccante

# Spectre V2 - Exploit Indirect Branches

```
...  
MOV R1, R2  
...  
LOAD R2  
...
```

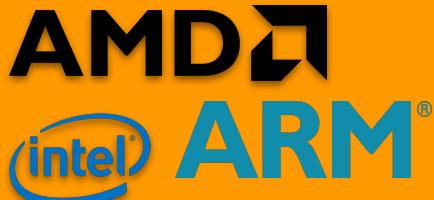
Ciò consente all'attaccante di ottenere il controllo via R1 su **quale** indirizzo di memoria leggere, e via R2 su **come** viene leakata in memoria.

Ma proviamo ad eseguire direttamente un esempio:

<https://www.exploit-db.com/exploits/43427/>

# UN PO' DI NUMERI (Spectre)

CHI NE È AFFETTO?



PERFORMANCE



Average reading rates: 10 KB/s

(Meno efficiente che Meltdown, ma sufficiente in quanto il target è lo spazio di memoria di un'applicazione vittima che generalmente è limitato)

CONTROMISURE



# SPECTRE: HARD TO LEVERAGE BUT HARD TO MITIGATE

# CONTROMISURE



- Software patch by Google: Reptoline (**Return Trampoline**) (per la variante 1) [\[7\]](#)
- La più efficace resta sempre un **aggiornamento del microcodice**. (Soprattutto per la variante 2)

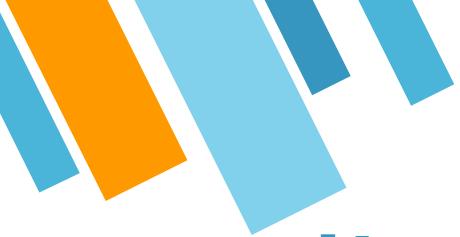
Tutto ciò **però** comporta:

1. Latenza nel ricevere l'aggiornamento
2. Degrado delle prestazioni
3. Non tutti processori potrebbero ricevere gli aggiornamenti [\[8\]](#)

## Intel first and second microcode update...

Come detto, non è stato facile fronteggiare il problema in maniera rapida dall'accaduto.

Intel ha dovuto infatti ricorrere a **due update** del microcodice, in quanto il primo ha portato casi di instabilità in alcune macchine e ha costretto la stessa a dover invitare aziende, OEM e consumatori finali a non diffondere il primo update. [9]



# Linus Torvald vs Intel vs Spectre and Meltdown

**“All of this is pure garbage. Is Intel really planning on making this shit architectural? Has anybody talked to them and told them they are f\*cking insane? Please, any Intel engineers here - talk to your managers.”**[\[10\]](#)



# Impatto sulle prestazioni.

Benchmark	Workload Description	8th Generation Desktop Intel® Core™ i7 8700K Processor	8th Generation Mobile Intel® Core™ i7-8650U Processor	7th Generation Mobile Intel® Core™ i7 7920HQ Processor	6th Generation Desktop Intel® Core™ i7 6700K Processor		
CPU Code Name		Coffee Lake	Kaby Lake	Kaby Lake	Skylake	Skylake	Skylake
OS		Windows 10	Windows 10	Windows 10	Windows 10	Windows 7	Windows 7
Storage		SSD	SSD	SSD	SSD	SSD	HDD
Introduction Date		Q4'17	Q3'17	Q1'17	Q3'15	Q3'15	Q3'15
Relative Performance (Fully Mitigated System / Non Mitigated System at 100%)							
SYSMark 2014 SE Overall	Windows Application-based Office Productivity, Data/Financial Analysis and Media Creation.	94%	95%	93%	92%	94%	100%
SYSMark 2014 SE Office Productivity		95%	95%	95%	90%	93%	96%
SYSMark 2014 SE Media Creation		96%	97%	96%	96%	97%	97%
SYSMark 2014 SE Data/Finance Analysis		97%	98%	98%	103%	99%	106%
SYSMark 2014 SE Responsiveness		88%	86%	86%	79%	89%	101%
PCMark 10 - Overall	Windows application based benchmark covering essentials, content creation and productivity.	96%	96%	97%	96%	96%	96%
PCMark 10 - Essentials		96%	96%	97%	96%	93%	95%
PCMark 10 - Productivity		96%	94%	95%	94%	97%	97%
PCMark 10 - Digital Content Creation		98%	98%	98%	99%	97%	97%
3DMark Sky Diver - Overall		100%	99%	100%	101%	100%	100%
3DMark Sky Diver - Graphics	DX11 Gaming performance	100%	99%	100%	101%	100%	100%
3DMark Sky Diver - Physics		99%	98%	100%	99%	97%	99%
3DMark Sky Diver - Combined		100%	99%	100%	101%	100%	101%
WebXPRT 2015 Notw. Windows 10 on Edge Browser Windows 7 on IE Browser	Web applications using six usage scenarios: Photo Enhancement, Organize Album, Stock Option Pricing, Local Notes, Sales Graphs, Explore DNA Sequencing.	92%	90%	93%	90%	95%	92%

Source: Intel

Note: The data above is based on multiple runs and expected system benchmark variation is assumed to be +/- 3%

# Varianti come se piovesse

Data	Nome vulnerabilità	Chi ne è afflitto?	Rimedi
11/Febbraio/2018	MeltdownPrime & SpectrePrime <a href="#">[11]</a>	Intel/AMD	Software
03/Maggio/2018	Spectre-NG <a href="#">[12]</a> (8 vulnerabilità)	Intel/ARM/AMD(?)	Software/Firmware
21/Maggio/2018	Spectre v3a/Spectre v4 <a href="#">[13]</a>	Intel/ARM/AMD	Firmware
13/Giugno/2018	Lazy FP <a href="#">[14]</a>	Intel Core/Xenon	Software

# Conclusion

THE MELTDOWN AND SPECTRE EXPLOITS USE "SPECULATIVE EXECUTION?" WHAT'S THAT?

YOU KNOW THE TROLLEY PROBLEM? WELL, FOR A WHILE NOW, CPUs HAVE BASICALLY BEEN SENDING TROLLEYS DOWN BOTH PATHS, QUANTUM-STYLE, WHILE AWAITING YOUR CHOICE. THEN THE UNNEEDED "PHANTOM" TROLLEY DISAPPEARS.



THE PHANTOM TROLLEY ISN'T SUPPOSED TO TOUCH ANYONE. BUT IT TURNS OUT YOU CAN STILL USE IT TO DO STUFF. AND IT CAN DRIVE THROUGH WALLS.



THAT SOUNDS BAD.

HONESTLY, I'VE BEEN ASSUMING WE WERE DOOMED EVER SINCE I LEARNED ABOUT ROWHAMMER.

WHAT'S THAT?

IF YOU TOGGLE A ROW OF MEMORY CELLS ON AND OFF REALLY FAST, YOU CAN USE ELECTRICAL INTERFERENCE TO FLIP NEARBY BITS AND—  
DO WE JUST SUCK AT...COMPUTERS?

YUP. ESPECIALLY SHARED ONES.



SO YOU'RE SAYING THE CLOUD IS FULL OF PHANTOM TROLLEYS ARMED WITH HAMMERS.

...YES. THAT IS EXACTLY RIGHT.  
OKAY. I'LL, UH...  
INSTALL UPDATES?

GOOD IDEA.

# Grazie.

Stefano Spadola

» [stefano.spadola@outlook.it](mailto:stefano.spadola@outlook.it)