Vulnerabilità hardware

Luca Mannini   
Corso di Calcolatori Elettronici e Reti di Calcolatori – Università Politecnica delle Marche

# **Premessa**

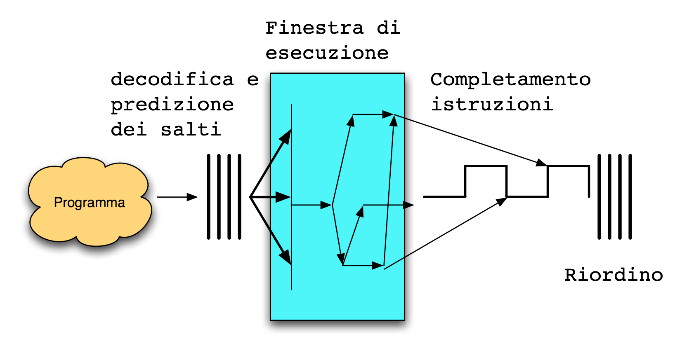
Nel 2018 dei ricercatori di Project Zero di Google sono riusciti, con la collaborazione della Cyberus Technology e dell’[Università tecnica di Graz](https://it.wikipedia.org/wiki/Universit%C3%A0_tecnica_di_Graz), a scoprire una delle più pericolose vulnerabilità hardware nei sistemi con processori Intel.  
È di uso comune dare dei nomi alle vulnerabilità di maggior impatto, tra quelle che andremo a studiare la prima è stata chiamata *Meltdown*, alla quali seguirono contromisure e patch immediate per limitare i danni.  
Meltdown ha dato il via alla scoperta di altre, numerose vulnerabilità hardware, la seconda che tratteremo è stata denominata *Spectre*. Scoperta successivamente, si è dimostrata molto più difficile da contrastare.  
La scoperta di queste due falle nei processori Intel ha spianato la strada verso la scoperta di altre gravi falle, tra cui 3 denominate *Fallout, RIDL* e *Zombieload* che non tratteremo in questo documento.

# **Background**

Gli attacchi funzionano basandosi su due principi di funzionamento della CPU: la cosiddetta “esecuzione fuori ordine” combinata con la “esecuzione speculativa”. Il primo è un metodo utilizzato dalle CPU per processare le singole istruzioni senza rispettare l’ordine imposte dal programma e la seconda invece è l’esecuzione contemporanea di più istruzioni.

# **Esecuzione fuori ordine**

Supponiamo che un processo maligno voglia accedere a locazioni di memoria non autorizzate, il tentativo da parte del processo provoca la generazione di un’eccezione con il conseguente fallimento nella lettura dei dati.



L’istruzione non viene completata e il processo viene interrotto e nonostante i dati della memoria riservata siano stati letti effettivamente dal processore, nessun dato è fornito al processo malevolo.  
  
Accade però che il dato della memoria riservata viene memorizzato nella cache della CPU e qualora il processo tentasse di ri-accedere all’informazione, verrebbe nuovamente generata un’eccezione successivamente al controllo privilegi.  
Meltdown interviene in questo punto:  
misurando il tempo di accesso alla locazione di memoria, riesce quindi a capire se il dato rimane memorizzato nella memoria temporaneamente.  
Piuttosto che eseguire il programma un’istruzione alla volta, il processore prende diverse istruzioni contemporaneamente, nel modo più rapido possibile e solitamente senza ordine (motivo per cui si chiama Esecuzione fuori ordine) e memorizza l’output di ogni istruzione nella cache.  
  
Utilizziamo la seguente istruzione come esempio:  
  
; rcx = kernel address  
 mov al, byte [rcx]

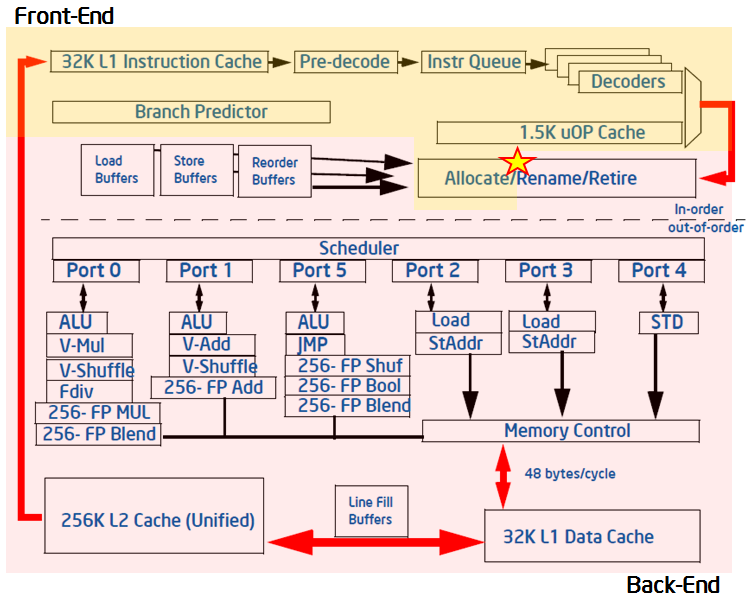
Questo codice copia un byte di dati dalla memoria kernel e li inserisce in una porzione temporanea di memoria della CPU. Con l’ esecuzione in ordine normalmente la CPU non permetterebbe l’esecuzione dell’istruzione, ma con la l’esecuzione fuori ordine quest’ultima esegue l’istruzione e la memorizza nella cache temporanea, a questo punto si genera l’eccezione e termina il programma.

**Il byte però rimane memorizzato nella cache.**

La CPU dovrebbe svuotare la cache il prima possibile ma ciò non avviene e quindi la rende vulnerabile ad un attacco chiamato “cache attack”.  
Gli attacchi alla cache, che fanno parte di una categoria chiamata “side-channel attacks”, esistono da anni e possono essere usati congiuntamente con Meltdown per ottenere i dati nella cache.

# **Esecuzione Speculativa**

La figura sotto mostra un generico diagramma che descrive l’esecuzione di istruzioni all'interno di una CPU moderna.



Le moderne CPU sono incredibilmente complicate e non eseguono semplicemente le istruzioni della macchina in ordine. Ogni thread della CPU (che sarebbe un insieme di registri) ha un complicato sistema di pipeline che è in grado di eseguire le istruzioni fuori servizio. Il motivo di ciò ha a che fare con il caching. Ogni CPU utilizza più livelli di memorizzazione nella cache. Ogni “miss” di cache aggiunge un notevole ritardo all’esecuzione del programma. Al fine di mitigare questo problema, i processori sono in grado di eseguire istruzioni*“ahead and out of order”,* ovvero *“in anticipo e fuori ordine”*, in attesa che si carichi la memoria. Questo è noto come esecuzione speculativa. Il seguente frammento di codice lo dimostra:

if (x < array1\_size) {  
y = array2[array1[x] \* 256];  
}

Nello snippet precedente, immagina che array1\_size non sia disponibile nella cache, ma che lo sia l'indirizzo di array1. La CPU potrebbe indovinare (speculare) che x è minore di array1\_size e andare avanti eseguendo i calcoli all'interno dell'istruzione if. Una volta letto array1\_size dalla memoria, la CPU può determinare se ha indovinato. Se lo ha fatto, può continuare con il vantaggio di aver risparmiato un sacco di tempo. Se così non fosse, può buttare via i calcoli speculativi e ricominciare da capo; il caso peggiore quindi sarebbe equivalente al caso in cui le istruzioni siano processate senza esecuzione speculativa.

# **Meltdown**

La prima vulnerabilità, nota come Meltdown, è sorprendentemente semplice da spiegare e quasi banale da sfruttare.

Il codice di exploit è approssimativamente simile al seguente:

1. uint8\_t \* probe\_array = new uint8\_t [256 \* 4096];   
  
2. // ... Assicurarsi che probe\_array non sia memorizzato nella cache   
  
3. uint8\_t kernel\_memory =   
\* (uint8\_t \*) (indirizzo\_kernel);   
  
4. uint64\_t final\_kernel\_memory = kernel\_memory \* 4096;   
  
5. uint8\_t dummy = probe\_array [final\_kernel\_memory];   
  
6. // ... catch page fault

7. // ... determina quale dei 256 slot in probe\_array è memorizzato nella cache

Prendiamo ogni passaggio sopra, descriviamo cosa fa e come porta a essere in grado di leggere la memoria dell'intero computer da un programma utente .

1. Nella prima riga viene allocato un "probe array” ovvero la memoria nel nostro processo che viene usata come “canale laterale” per recuperare i dati dal kernel.

2. Seguendo l'allocazione, l'attaccante si assicura che nessuna parte della memoria nell'array sia memorizzato nella cache. Ci sono vari modi per farlo, il più semplice dei quali include istruzioni specifiche della CPU per cancellare una posizione di memoria dalla cache.

3. L'utente malintenzionato procede quindi a leggere un byte dallo spazio di indirizzi del kernel. Tutti i kernel moderni ormai mappano l'intero spazio degli indirizzi virtuali del kernel nel processo utente. I sistemi operativi si basano sul fatto che ogni voce della tabella delle pagine ha dei registri di autorizzazione e che i programmi in modalità utente non possono accedere alla memoria del kernel. Qualsiasi accesso di questo tipo provocherà un errore di pagina. Questo, invece, è ciò che succederà una volta arrivati allo step 3.

4. I processori moderni, tuttavia, utilizzano anche l'esecuzione speculativa e eseguiranno prima delle istruzioni di errore. Pertanto, i passaggi 3-5 possono essere eseguiti nella pipeline della CPU prima che venga generato l’errore. In questo passaggio, il byte della memoria del kernel (che va da 0-255) viene moltiplicato per la dimensione della pagina del sistema, che in genere è 4096.

5. A questo punto, il byte moltiplicato della memoria del kernel viene quindi utilizzato per leggere dal probe array in un valore fittizio. La moltiplicazione del byte per 4096 è per evitare che una funzione della CPU chiamata "prefetcher" legga più dati di quanti ne vogliamo nella cache.

6. A questo punto, la CPU ha realizzato l'errore ritorna al passaggio 3. Tuttavia, i risultati delle istruzioni speculate sono ancora visibili nella cache . L'utente malintenzionato utilizza la funzionalità del sistema operativo per intercettare le istruzioni di errore e continuare l'esecuzione (ad esempio, la gestione di SIGFAULT).

7. Nel passaggio 7, l'utente malintenzionato esegue un'iterazione e vede quanto tempo impiega a leggere ciascuno dei 256 byte possibili nell'array probe che potrebbe essere stato indicizzato dalla memoria del kernel. La CPU avrà caricato una delle posizioni nella cache e questa posizione verrà caricata sostanzialmente più velocemente di tutte le altre posizioni (che devono essere lette dalla memoria principale). Questa posizione è il valore del byte nella memoria del kernel .

Grazie alla tecnica qui sopra, e il fatto che per i sistemi operativi moderni è prassi comune mappare tutta la memoria fisica nello spazio degli indirizzi virtuali del kernel, un utente malintenzionato può leggere l'intera memoria fisica del computer .

Tutto ciò avviene a causa di un bug nei processori Intel . Tutto l'accesso alla memoria virtuale deve avvenire tramite il TLB. È possibile verificare, durante l'esecuzione speculativa, che una mappatura memorizzata nella cache abbia le autorizzazioni compatibili con il livello di privilegio di esecuzione attuale. L'hardware Intel semplicemente non lo fa. Altri fornitori di processori eseguono un controllo di autorizzazione e bloccano l'esecuzione speculativa. Quindi, per quanto ne sappiamo, Meltdown è una vulnerabilità quasi esclusivamente *Intel* (fatta eccezione per qualche processore ARM).

# **Spectre**

Spectre condivide alcune proprietà di Meltdown ed è composto da due varianti. Diversamente da Meltdown, Spectre è sostanzialmente più difficile da sfruttare, ma riguarda quasi tutti i processori moderni prodotti negli ultimi vent'anni. Essenzialmente, Spectre è un attacco contro la moderna progettazione di CPU e sistemi piuttosto che una specifica vulnerabilità di sicurezza.

# **Bounds check bypass (Prima variante di Spectre)**

La prima variante di Spectre è conosciuta come "bounds check bypass". Prendiamo lo stesso frammento di codice di prima come esempio:

if (x < array1\_size) {   
 y = array2[array1[x] \* 256];   
}

Per spiegare la variante presupponiamo la seguente sequenza di eventi:

* L'attaccante controlla x.
* array1\_size non è memorizzato nella cache
* array1 è memorizzato nella cache.
* La CPU suppone che x sia inferiore a array1\_size. (Gli algoritmi sulla speculazione sono diversi a seconda dei fornitori e dei vari modelli di processore, motivo per cui anche il modo di “attaccare” di Spectre cambia)
* La CPU esegue il body dell'istruzione if mentre è in attesa del caricamento di array1\_size, influenzando la cache in modo simile a Meltdown.
* L'utente malintenzionato può quindi determinare il valore effettivo array1[x] tramite uno dei vari metodi.

Spectre è molto più difficile da sfruttare di Meltdown perché questa vulnerabilità *non* dipende dall'escalation dei privilegi. L'utente malintenzionato deve convincere il kernel a eseguire codice e speculare in modo errato. Detto questo, i ricercatori hanno mostrato la possibilità di scrivere diversi exploit e proof-of-concept.

# **Branch target injection (Seconda variante di Spectre)**

La ramificazione indiretta è molto comune nei programmi moderni. La variante 2 di Spectre utilizza la previsione del ramo indiretto per indurre la CPU all'esecuzione speculativa in una posizione di memoria che altrimenti non avrebbe mai eseguito. Se l'esecuzione di tali istruzioni può lasciare lo stato nascosto nella cache che può essere rilevato utilizzando gli “attacchi di inferenza della cache”, l'utente malintenzionato può quindi scaricare tutta la memoria del kernel. Come la variante Spectre 1, la variante Spectre 2 è molto più difficile da sfruttare di Meltdown, tuttavia i ricercatori hanno mostrato diversi exploit e proof-of-concept della variante 2.

# **Test Vulnerabilità**

Abbiamo provato a eseguire dei test di queste vulnerabilità. Ci sono innumerevoli repository su Github che contengono PoC (Proof of Concept) di Meltdown e Spectre. La parte difficile dei nostri test è riuscire a trovare una macchina vulnerabile sulla quale poter sperimentare le potenzialità di Meltdown e Spectre.

A causa del fatto che la maggior parte dei computer utilizzati oggi sono stati aggiornati e “patchati” tramite tecniche di mitigazione apposite, abbiamo pensato che l’ideale sarebbe stato trovare un vecchio pc inutilizzato che non abbia subito aggiornamenti.

Abbiamo condotto i nostri test su un Acer Intel Pentium Dual CPU 2.16GHz utilizzando Ubuntu 16.04 in live Boot su USB come sistema operativo e per prima cosa abbiamo scaricato un tool, sempre da Github, per esaminare le versioni e le varianti a cui il sistema fosse vulnerabile.

E’ possibile scaricare il tool dal seguente link: <https://github.com/speed47/spectre-meltdown-checker>.

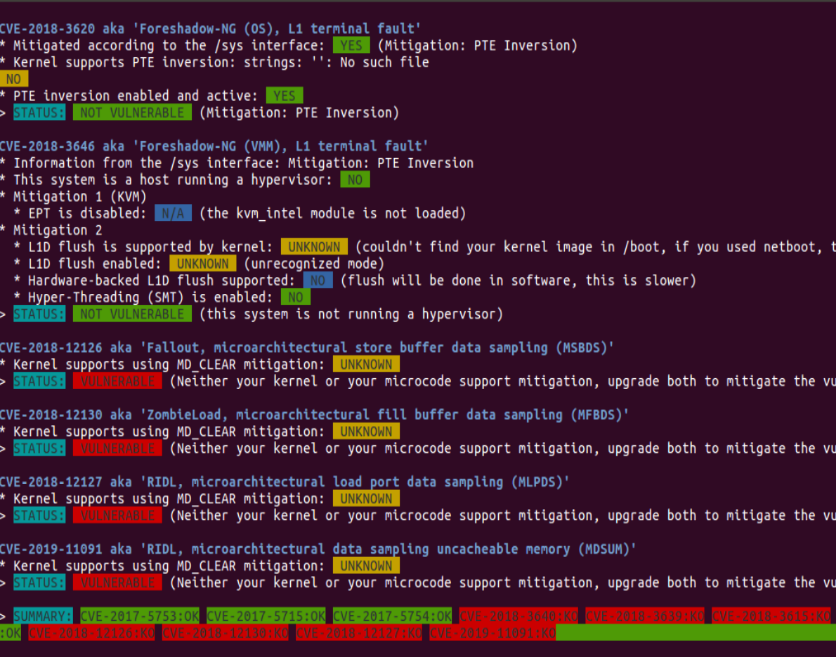
Le vulnerabilità che controlla sono le seguenti:

* CVE-2017-5753 aka 'Spectre Variant 1'
* CVE-2017-5715 aka 'Spectre Variant 2'
* CVE-2017-5754 aka 'Meltdown' aka 'Variant 3'
* CVE-2018-3640 aka 'Variant 3a'
* CVE-2018-3639 aka 'Variant 4'
* CVE-2018-3615 aka 'Foreshadow (SGX)'
* CVE-2018-3620 aka 'Foreshadow-NG (OS)'
* CVE-2018-3646 aka 'Foreshadow-NG (VMM)'
* CVE-2018-12126 aka 'Fallout'
* CVE-2018-12130 aka 'ZombieLoad'
* CVE-2018-12127 aka 'RIDL'
* CVE-2019-11091 aka 'RIDL'

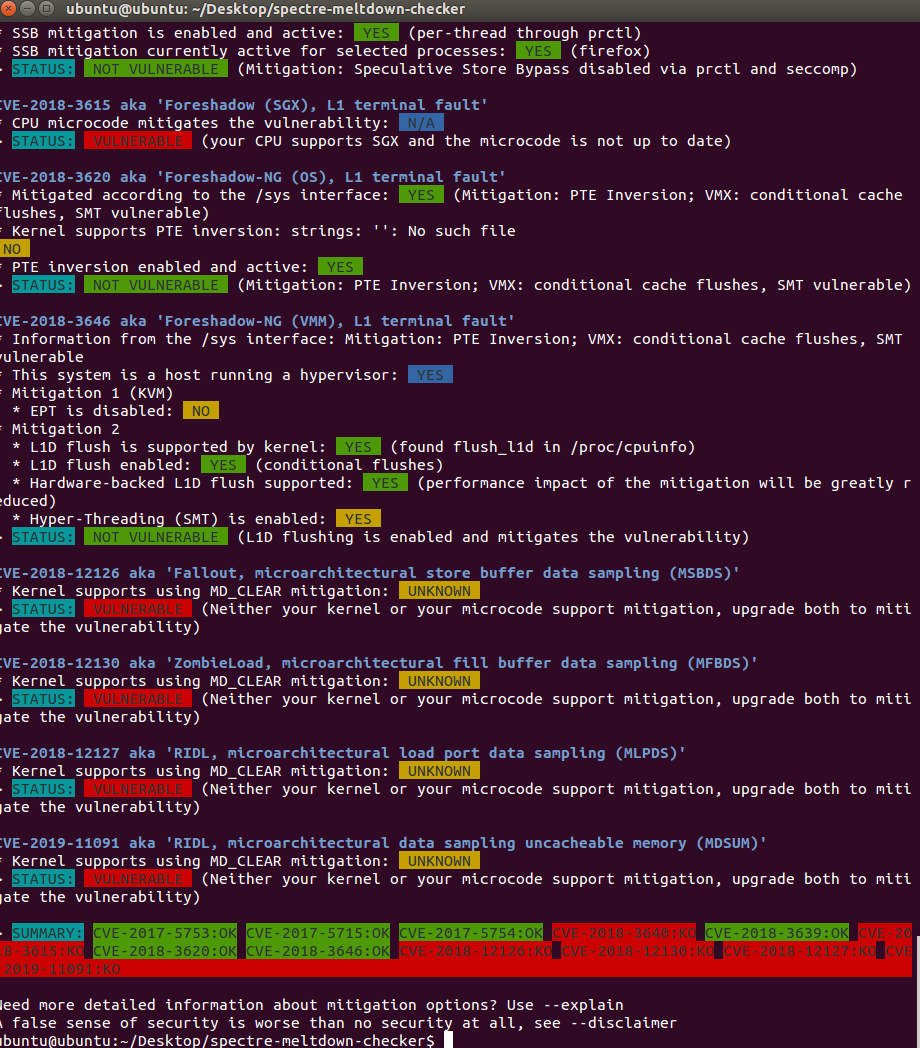
Grazie ad esso possiamo verificare se il sistema è effettivamente vulnerabile o meno agli attacchi Meltdown e Spectre.

Per eseguire il tool bisogna scaricare la cartella o clonarla da terminale, dopodiché è necessario dare i permessi al programma tramite il comando chmod +x /nome\_del\_file.sh.

A questo punto si può eseguire il programma tramite il comando ./nome\_del\_file.sh che controllerà in automatico le possibili falle nel sistema su cui è in esecuzione:

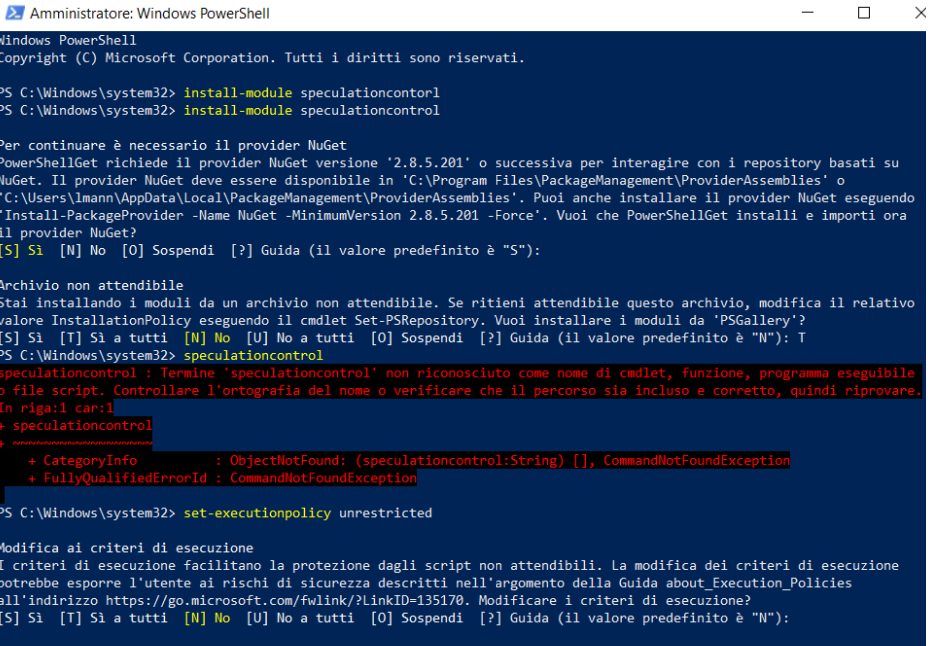


Per cercare di ottenere un risultato migliore abbiamo testato anche una versione meno aggiornata, sempre di Ubuntu, la 14.04, questo è stato il risultato:



La prevalenza del rosso nella schermata fa presagire che il sistema sia effettivamente vulnerabile, ma in effetti se si osservano le singole vulnerabilità, si nota come esse facciano riferimento a falle più nuove, tra cui le sovracitate Fallout, RIDL e Zombiload.  
Purtroppo non è facile trovare in rete Proof of Concept o Exploit per sfruttare queste falle, indi per cui non abbiamo modo di analizzarne gli effetti.

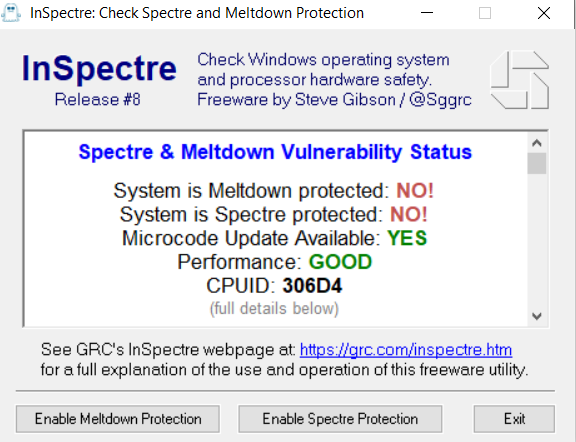
A questo punto ci siamo resi conto che forse sarebbe stato possibile disabilitare le patch su computer già aggiornati e dopo alcune ricerche, per lo più su siti o forum di Cybersecurity inglesi e americani, siamo riusciti a trovare il modo di disabilitare le difese su Windows 10:



Il sito ufficiale della Microsoft offre una guida di supporto per poter abilitare e disabilitare le patch utilizzate per mitigare Spectre e Meltdown:  
<https://support.microsoft.com/en-us/help/4073119/protect-against-speculative-execution-side-channel-vulnerabilities-in>

Per poterlo fare bisogna creare un file bat contenente le istruzioni presenti nel sito, lanciare il file e riavviare.

Su Windows però è necessario utilizzare un altro tool apposito per poter verificare lo stato della sicurezza del sistema, googlando si trova facilmente InSpectre, che ci fornisce il seguente risultato:

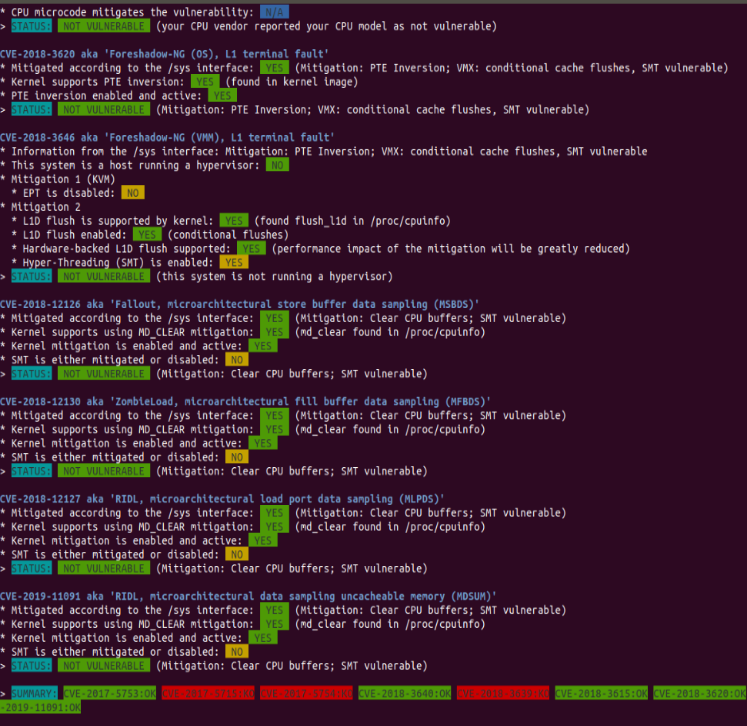


Per quanto lo screenshot faccia ben sperare, anche stavolta c’è il problema che per Windows non sono presenti Exploit a riguardo.

Il prossimo tentativo allora si baserà sulla disabilitazione delle patch per Linux, la versione che avevamo preinstallata in dual boot con Windows.

La ricerca per Linux è stata più difficoltosa, siamo capitati per lo più su forum sui quali degli utenti chiedevano di poter disabilitare le patch per poter incrementare le prestazioni della CPU, questo perché, ovviamente, le patch, modificando il modo in cui la CPU esegue le istruzioni, vanno ad infierire sulla velocità del processore, rendendolo fino al 30% più lento nei casi peggiori.

Una volta riusciti a disabilitare anche su Linux le patch il risultato è stato il seguente:

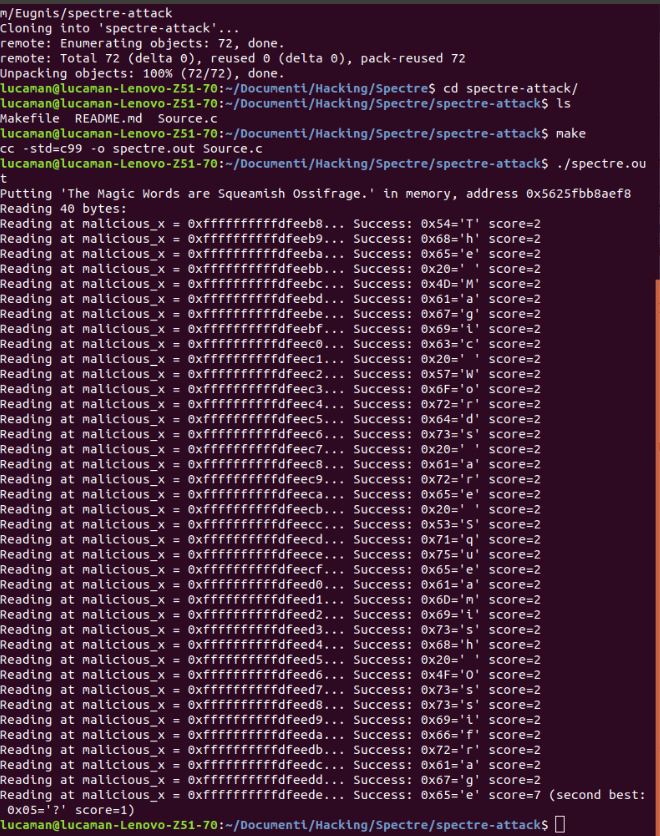


Finalmente il risultato che aspettavamo.  
Il sistema pare essere vulnerabile alle CVE-2017-5715 aka “Spectre variant 2, branch target injection”, alla CVE-2017-5754 aka “Variant 3, Meltdown, rogue data cache load” e alla CVE-2018-3639, aka “Variant 4, speculative store bypass”.

# **Test Exploit**

Per trovare gli exploit sulle falle di nostro interesse serve una ricerca veloce sempre su Github che ci porterà a decine di risultati.  
Abbiamo provato gli exploit più scaricati e il primo testato è un exploit per Spectre scaricabile al seguente link:   
<https://github.com/Eugnis/spectre-attack>

L’esempio in questione è un programma che mette in memoria il testo "*The Magic Words are Squeamish Ossifrage*" e, se il sistema è vulnerabile, il programma lanciato sfruttando Spectre legge dalla memoria stessa la frase riportandola su terminale. Ecco qua il risultato del test condotto sul nostro Linux:



Spectre funziona e la frase viene letta.  
Questo è il codice della vittima:

unsigned int array1\_size = 16;  
uint8\_t unused1[64];  
uint8\_t array1[160] = {1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16};  
uint8\_t unused2[64];  
uint8\_t array2[256 \* 512];

char\* secret = "The Magic Words are Squeamish Ossifrage.";

uint8\_t temp = 0; /\* Used so compiler won't optimize out victim\_function() \*/

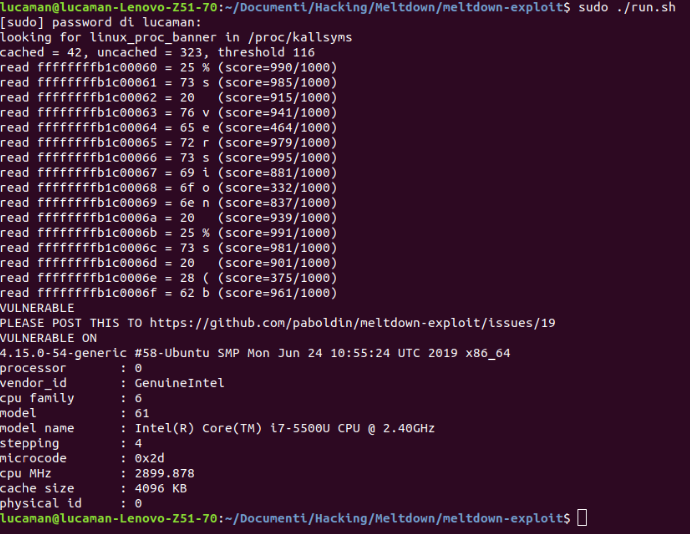
void victim\_function(size\_t x)  
{ if (x < array1\_size)  
 { temp &= array2[array1[x] \* 512]; }  
}

Nel codice dell’exploit, se le istruzioni compilate in victim\_function() venissero eseguite in ordine la funzione leggerebbe solamente da array[0…15] dato che la dimensione di array1 è 16. Con l’esecuzione speculativa, sono possibili letture fuori dai limiti imposti dai permessi.]

Il secondo test è stato effettuato tramite un exploit trovato al seguente indirizzo:

<https://github.com/paboldin/meltdown-exploit>

e il funzionamento del programma è molto simile al precedente. Ecco il risultato:



Anche in questo caso è evidente come il programma riesce a leggere dalla memoria e ottenere i caratteri stampati su terminale.

# **Considerazioni finali**

Dai test condotti si è dimostrato come, sfruttando falle di sistema quali Spectre e Meltdown, si possa ottenere un accesso illimitato ai dati in memoria, rendendo di fatto il sistema potenzialmente indifeso ad attacchi basati sull’esecuzione speculativa e fuori ordine. È opportuno da evidenziare come per alcune versioni di Spectre non siano state trovate mitigazioni adeguate, seppur per la maggior parte dei dispositivi siano stati rilasciati aggiornamenti che nella maggior parte dei casi funzionano efficacemente.

In ogni caso “un falso senso di sicurezza è peggio che non avere alcuna sicurezza”.

References:

* Microsoft Support 24 Maggio 2019, Windows client guidance for IT Pros to protect against speculative execution side-channel vulnerabilities:  
  <https://support.microsoft.com/en-us/help/4073119/protect-against-speculative-execution-side-channel-vulnerabilities-in>
* Wikipedia, 10 Giugno 2019‎, Spectre (vulnerabilità di sicurezza):  
  <https://it.wikipedia.org/wiki/Spectre_(vulnerabilit%C3%A0_di_sicurezza)>
* Wikipedia, 18 Maggio 2019‎, Meltdown (vulnerabilità di sicurezza):  
  <https://it.wikipedia.org/wiki/Meltdown_(vulnerabilit%C3%A0_di_sicurezza)>
* “Spectre and Meltdown explained: What they are, how they work, what's at risk”, 15 Gennaio 2018: <https://www.csoonline.com/article/3247868/spectre-and-meltdown-explained-what-they-are-how-they-work-whats-at-risk.html>
* “Spectre and Meltdown explained”:  
  <https://linuxaria.com/article/spectre-and-meltdown-explained>
* “Anyone disable their Meltdown and Spectre patches for max CPU performance?”, 2019:  
  <https://www.reddit.com/r/PUBATTLEGROUNDS/comments/8y3oyf/anyone_disable_their_meltdown_and_spectre_patches/>
* “Exploiting modern microarchitectures: Meltdown, Spectre, and other attacks”:  
  <https://people.redhat.com/jcm/talks/FOSDEM_2018.pdf>
* “Meltdown: Reading Kernel Memory from User Space”:  
  <https://meltdownattack.com/meltdown.pdf>
* “Spectre Attacks: Exploiting Speculative Execution”:  
  <https://spectreattack.com/spectre.pdf>