

Spectre y Meltdown

Seguridad de Sistemas Operativos



Juan Carlos Pulido Poveda
jcpulido97@correo.ugr.es
Jaime Amate Ramírez
jaimeamate@correo.ugr.es

Índice

Índice	1
Introducción	2
Desarrollo	4
Contexto	4
Ejecución fuera de orden (Out-of-order Execution)	4
Ejecución especulativa (Speculative Execution)	4
Predicción de salto (Branch Prediction)	5
Jerarquía de memoria	5
Espacio de direcciones	6
Spectre	6
Explicación	6
Variante 1 - 'Bounds check bypass (CVE-2017-5753)'	7
Variante 2 - 'Branch target injection (CVE-5715)'	7
Mitigaciones	8
Prevenir el ejecutamiento especulativo	8
Prevenir el acceso de los datos secretos	9
Prevenir que los datos entren por canales encubiertos	9
Prevenir el envenenamiento del predictor de saltos	9
Meltdown	10
Explicación	10
Mitigación	12
Aislamiento de tabla de páginas del Kernel (KPTI)	12
Demostración	13
Spectre	13
Meltdown	15
Conclusiones	16
Bibliografía	17

Introducción

Este trabajo trata de las vulnerabilidades encontradas a principios de 2018 que conciernen a la gran mayoría de CPU modernas en el mercado. Estas vulnerabilidades son Spectre y Meltdown y se nombran juntas debido a que ambas se aprovechan de la ejecución especulativa de los procesadores modernos.

Las vulnerabilidades fueron anunciadas de forma oficial por los siguientes códigos:

- Spectre: [CVE-2017-5753](#) y [CVE-2017-5715](#)
- Meltdown: [CVE-2017-5754](#)

Current Description

Systems with microprocessors utilizing speculative execution and branch prediction may allow unauthorized disclosure of information to an attacker with local user access via a side-channel analysis.

Source: MITRE

Description Last Modified: 01/04/2018

[+View Analysis Description](#)

Impact

CVSS v3.0 Severity and

Metrics:

Base Score: 5.6 MEDIUM

Vector: AV:L/AC:H/PR:L/UI:N
/S:C/C:H/I:N/A:N (V3 legend)

Impact Score: 4.0

Exploitability Score: 1.1

Attack Vector (AV): Local

Attack Complexity (AC): High

Privileges Required (PR): Low

User Interaction (UI): None

Scope (S): Changed

Confidentiality (C): High

Integrity (I): None

Availability (A): None

CVSS v2.0 Severity and

Metrics:

Base Score: 4.7 MEDIUM

Vector: (AV:L/AC:M/Au:N/C:C
/I:N/A:N) (V2 legend)

Impact Subscore: 6.9

Exploitability Subscore: 3.4

Access Vector (AV): Local

Access Complexity (AC):
Medium

Authentication (AU): None

Confidentiality (C): Complete

Integrity (I): None

Availability (A): None

Additional Information:

Allows unauthorized disclosure
of information

Descripción CVE-2017-5753



Spectre, más que una vulnerabilidad es un tipo de ataque que puede crear variedad de potenciales amenazas. Este limita su radio de acción dentro un mismo proceso por lo que podríamos suponer que no es tan peligroso, pero debemos de recordar que en la actualidad usamos un mismo proceso para acciones cotidianas como puede ser el navegador Web. Es por eso que una web podría ejecutar código Javascript para acceder a memoria de otra página arbitraria como podría ser la del banco, etc. De esta forma se violaría la seguridad de la memoria del navegador pudiendo dejar nuestras claves de acceso en memoria al alcance de personas sin permisos.



Meltdown sin embargo si se considera una gran amenaza para los sistemas modernos ya que permite acceder a memoria kernel y por tanto escalar privilegios de lectura de memoria para poder acceder a cualquier dirección de memoria del sistema. Esto conlleva un gran problema de violación de privilegios de memoria y por tanto una gran amenaza ya que toda información sensible del sistema podría ser leída por procesos sin suficientes permisos con objetivos maliciosos.

Desarrollo

Contexto

En esta sección, describiremos algunos conceptos de la microarquitectura de las CPUs modernas que nos harán falta para entender las dos vulnerabilidades.

A. Ejecución fuera de orden (Out-of-order Execution)

La ejecución fuera de orden incrementa la utilización de los componentes del procesador permitiendo la ejecución en paralelo de varias instrucciones. En lugar de ejecutar las instrucciones secuencialmente, la CPU las ejecuta siempre que los recursos requeridos estén disponibles. Por lo tanto, las instrucciones pueden ejecutarse en paralelo siempre y cuando sus resultados sigan la definición especificada.

B. Ejecución especulativa (Speculative Execution)

A menudo, el procesador desconoce el flujo de instrucciones (por ejemplo, depende valores de entrada o ciertas condiciones). Esto ocurre cuando la ejecución fuera de orden alcanza un salto condicional cuya dirección depende de instrucciones precedentes que no han sido ejecutadas aún. En ese caso, el procesador puede almacenar el estado de registro actual y hacer una predicción del salto que el programa seguirá y “especulativamente” ejecutar instrucciones. Si la predicción resulta ser incorrecta (error de predicción) se revierten los cambios perpetrados por la especulación y vuelve al estado almacenado.

El límite de instrucciones que se pueden ejecutar especulativamente depende del tamaño del buffer de reordenamiento.

C. Predicción de salto (Branch Prediction)

Los predictores de salto de los procesadores modernos, por ejemplo, Haswell Xeon, tienen múltiples mecanismos para saltos directos e indirectos. Los saltos indirectos son instrucciones que saltan a direcciones de memoria arbitrarias computadas en tiempo de ejecución.

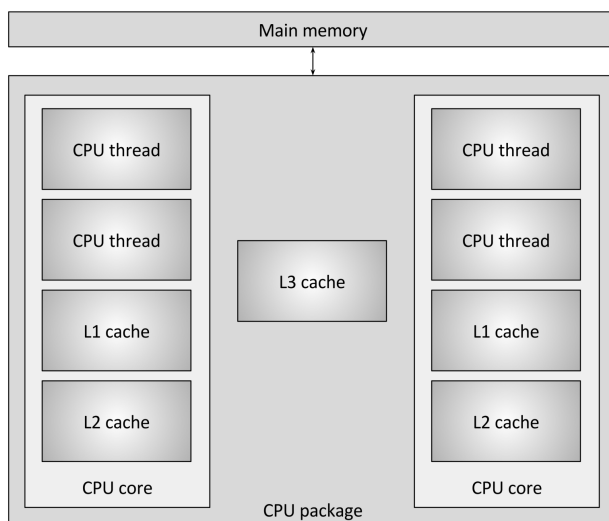
Ejemplo: Salto indirecto compilado para x86

```
jmp *%rax;
```

Para mejorar las predicciones, los procesadores mantienen un registro de resultados de salto, tanto para saltos directos como indirectos.

D. Jerarquía de memoria

Los procesadores modernos incluyen un tipo de memoria más rápida y pequeña entre los registros del procesador y la memoria principal con el fin de contrarrestar el cuello de botella que supone los accesos a memoria principal. A su vez la caché está jerarquizada en distintos niveles (típicamente L1, L2 y L3).



Cuando el procesador necesita un dato de memoria accede al nivel 1 de memoria caché (L1) para comprobar si tiene una copia del dato.

Si no la contiene se produce un fallo de caché y baja al siguiente nivel de la jerarquía.

E. Espacio de direcciones

Para aislar los procesos los unos de los otros, las CPUs proporcionan un espacio de memoria virtual cuyas direcciones se traducen en direcciones físicas de memoria. Un espacio virtual de direcciones se divide en un conjunto de páginas que pueden ser individualmente mapeadas a memoria física mediante una tabla de traducción multinivel. La traducción define la traducción de dirección virtual a física además y además las propiedades de protección de la página.

Cada espacio de direcciones virtual está dividido en un espacio de usuario y otro espacio de kernel (memoria física del kernel mapeada en cada espacio de direcciones). El espacio de direcciones del kernel solo se puede ejecutar si la CPU se está ejecutando en modo privilegiado (llamadas al sistema).

Spectre

Explicación

En la mayoría de los procesadores, la ejecución especulativa que surge de un fallo de la predicción puede dejar efectos observables colaterales que pueden revelar información privada a un atacante. Por ejemplo, si el patrón de accesos a la memoria realizados por la mencionada ejecución especulativa depende de datos privados, el estado resultante de la caché de datos constituye un canal lateral mediante el cual un atacante puede ser capaz de obtener información acerca de los datos privados empleando un ataque sincronizado.

Spectre se basa en explotar los efectos secundarios de la ejecución especulativa. Esta técnica se utiliza para reducir la latencia de memoria. En particular, Spectre se centra en la predicción de saltos, un caso especial de la ejecución especulativa.

Variante 1 - 'Bounds check bypass (CVE-2017-5753)'

La primera variante de Spectre es conocida como "bounds check bypass". Esta se demuestra en el siguiente trozo de código.

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

En este ejemplo, asumimos la siguiente secuencia de eventos:

1. El atacante controla `x`.
2. `array1_size` no está en caché.
3. `array1` está en caché
4. La CPU adivina que `x` es menor que `array1_size`. (Las CPUs utilizan varios algoritmos propietarios y heurísticas para determinar la especulación, esta es la razón por la cuál los detalles del ataque pueden variar entre distintos modelos de procesador).
5. La CPU ejecuta el cuerpo de la instrucción `if` mientras espera que `array1_size` se cargue en caché.
6. El atacante puede entonces determinar el valor de `array1[x]`.

Variante 2 - 'Branch target injection (CVE-5715)'

La segunda variante utilizada la predicción de salto indirecto para envenenar la CPU con el fin de ejecutar especulativamente en una zona de memoria que no se ejecutaría de lo contrario.

Considere el siguiente código de herencia en C++ como ejemplo:

```
class Shape {  
    public:  
        virtual void Draw() = 0;  
};  
  
class Circle : public Shape {
```



```
public:
    void Draw() override {...}
};
```

Donde `Shape` es la clase base y `Circle` es la clase derivada. Ahora considere el siguiente fragmento de código:

```
Shape *obj = new Circle;
obj->Draw();
```

La dirección de destino `Draw()` no se puede determinar en tiempo de compilación lo que resulta en un salto indirecto que se tiene que resolver en tiempo de ejecución. Durante el tiempo de ejecución, se utiliza una tabla de consulta para encontrar la función correspondiente. Mientras esto pasa, el procesador adivina la dirección de destino e inmediatamente empieza a ejecutar las instrucciones especulativamente del código.

El atacante debe encontrar un código similar al ejemplo mostrado que cuando manipulado a través del predictor de salto indirecto, pueda llevar al procesador a ejecutar especulativamente código que resulta en la variante 1 de Spectre. El atacante usará entonces la primera variante de ataque para obtener información sensible del espacio de memoria de la víctima.

Spectre es considerablemente más difícil de explotar que Meltdown puesto que esta vulnerabilidad no depende del escalado de privilegios. El atacante debe convencer al kernel de ejecutar el código y especular de forma incorrecta. El atacante típicamente tiene que “envenenar” el predictor de saltos de la CPU y engañarlo para especular de forma incorrecta.

Mitigaciones

A. Prevenir el ejecutamiento especulativo

El ejecutamiento especulativo es requerido para los ataques Spectre. Deshabilitar esta característica del procesador prevendría dichos ataques, sin embargo, supondría una degradación considerable del rendimiento del sistema. Como alternativa se

podría plantear deshabilitar la ejecución especulativa dinámicamente vía microcódigo, aunque esta solución no proporciona un arreglo inmediato del problema.

Una alternativa viable consiste en insertar instrucciones de serialización para ayudar a mitigar el envenenamiento del salto indirecto. Insertar una instrucción lfence (la cual AMD recomienda su uso), asegura que se haya completado la operación de carga de memoria (bloquea la ejecución especulativa).

B. Prevenir el acceso de los datos secretos

Como solución, Google Chrome ejecuta cada web como un proceso independiente. De esta forma, un ataque perpetrado por un código JavaScript no sería capaz de atacar datos alojados en otra página web.

Otra enfoque propuesto por WebKit (motor de búsqueda open-source) utiliza index masking (enmascaramiento de índices) en lugar de los chequeos de límites del array.

C. Prevenir que los datos entren por canales encubiertos

Los futuros procesadores podrían implementar algún mecanismo para rastrear si los datos fueron accedidos como resultado de una operación especulativa y en ese caso prevenir que estos datos sean usados en próximas operaciones que puedan filtrarlos.

D. Prevenir el envenenamiento del predictor de saltos

Google propuso como solución el uso de “Retpoline”. Retpoline es una construcción software que previene el branch-target-injection, denominación que utiliza la división de Google responsable del proyecto para referirse a ataques Spectre. Las secuencias Retpoline permiten aislar los saltos incondicionales de la ejecución especulativa. Esto se puede aplicar para proteger binarios sensibles de ataques de salto indirecto.

Meltdown

Explicación

Meltdown afecta a una amplia gama de sistemas. En el momento de hacerse pública su existencia se incluían todos los dispositivos que no utilizasen una versión convenientemente parcheada de iOS, GNU/Linux, macOS, Windows y Windows 10 Mobile y contasen con una CPU Intel moderna o algún modelo específico de ARM.

Por lo tanto, muchos servidores y servicios en la nube se han visto impactados, así como potencialmente la mayoría de dispositivos inteligentes y sistemas embebidos que utilizan procesadores con arquitectura ARM (dispositivos móviles, televisores inteligentes y otros), incluyendo una amplia gama de equipo usado en redes.

Meltdown se aprovecha de explotar una condición de carrera inherente al diseño de muchas CPU actuales. Esta condición se da entre los accesos a la memoria y la comprobación de privilegios durante el procesamiento de instrucciones. Además, en combinación con un ataque de canal lateral a la caché de la CPU, esta vulnerabilidad permite que un proceso se salte las comprobaciones habituales de nivel de privilegio que normalmente aislarían al proceso maligno e impedirían que accediese a datos que pertenecen al sistema operativo y otros procesos concurrentes.

Ahora comentaremos un ejemplo de código de meltdown:

```
1. uint8_t* probe_array = new uint8_t[256 * 4096];
2. clflush(probe_array) // ... Asegurarnos que probe_array no está en caché
3. uint8_t kernel_memory = *(uint8_t*)(kernel_address);
4. uint64_t final_kernel_memory = kernel_memory * 4096;
5. uint8_t dummy = probe_array[final_kernel_memory];
6. // ... catch del fallo de página
7. // ... determinar cuál de los 256 elementos de probe_array está en caché
```

1. En la primera línea se declara `probe_array`. Esta memoria de nuestro proceso que se usa como side channel para traer información del kernel.

2. Después de la declaración, el atacante se asegura de que no se almacene en la memoria caché `probe_array`. Hay varias formas de lograr esto, la más simple de las cuales incluye instrucciones específicas de la CPU para borrar una ubicación de memoria de la memoria caché, la función `clflush` se encarga de limpiar la memoria caché del procesador
3. Luego, el atacante lee un byte del espacio de direcciones del kernel. Recordamos que todos los kernels modernos generalmente asignan todo el espacio de direcciones virtuales del kernel al proceso del usuario. Los sistemas operativos se basan en el hecho de que cada entrada de la tabla de páginas tiene configuraciones de permisos y que los programas de modo de usuario no tienen permiso para acceder a la memoria del kernel. Cualquier acceso de este tipo dará como resultado un error de página.
4. Sin embargo, los procesadores modernos también realizan ejecuciones especulativas y se ejecutarán antes de la instrucción de fallo. Por lo tanto, los pasos 3 a 5 pueden ejecutarse en el pipeline de la CPU antes de que se genere el fallo. En este paso, el byte de la memoria del kernel (que oscila entre 0 y 255) se multiplica por el tamaño de página del sistema, que suele ser 4096.
5. En este paso, el byte multiplicado de la memoria del kernel se usa para leer desde la variable `probe_array` en un valor ficticio. La multiplicación del byte por 4096 es para evitar que una característica de la CPU llamada `prefetcher` lea más datos de los que queremos en caché.
6. En este paso, la CPU se da cuenta de su error y regresa al paso 3. Sin embargo, los resultados de las instrucciones especuladas aún son visibles en el caché. El atacante usa la funcionalidad del sistema operativo para interceptar la instrucción de fallos y continuar la ejecución (por ejemplo, el manejo de `SEGFAULT`).
7. El atacante recorre y ve cuánto tiempo lleva leer cada uno de los 256 bytes posibles en la variable `probe_array` que podrían haber sido indexados por la memoria del núcleo. La CPU habrá cargado una de las ubicaciones en la memoria caché y esta ubicación se cargará sustancialmente más rápido que todas las demás posiciones (que deben leerse desde la memoria principal). Esta ubicación es el valor del byte en la memoria del núcleo.

La razón por la que podemos acceder memoria de kernel es debido a que los criterios de seguridad de acceso a memoria no se tienen en cuenta cuando se realiza la ejecución

especulativa. Este problema se limita a procesadores de Intel y algún procesador ARM, por lo que parece el problema en este caso es causado por fallo de Intel.

Esta vulnerabilidad no está presente en los procesadores de la familia AMD puesto que en estos durante la ejecución especulativa también se comprueba que las páginas en caché también tengan los permisos compatibles con el actual nivel de privilegio.

Mitigación

Meltdown es fácil de entender y también fácil de explotar. Afortunadamente también es fácil de mitigar ya que podemos corregir la implementación del kernel para que tenga en cuenta estos aspectos de seguridad para evitar que el usuario pueda acceder al espacio de kernel indiscriminadamente.

Aislamiento de tabla de páginas del Kernel (KPTI)

KPTI corrige estas fugas de información separando completamente las tablas de páginas del espacio de usuario de las del espacio del núcleo. En los procesadores que soportan los identificadores de contexto de proceso (PCID) puede evitarse la limpieza del TLB, pero incluso entonces esto supone una pérdida significativa de rendimiento, particularmente con tareas donde intervienen muchas llamadas al sistema o interrupciones.

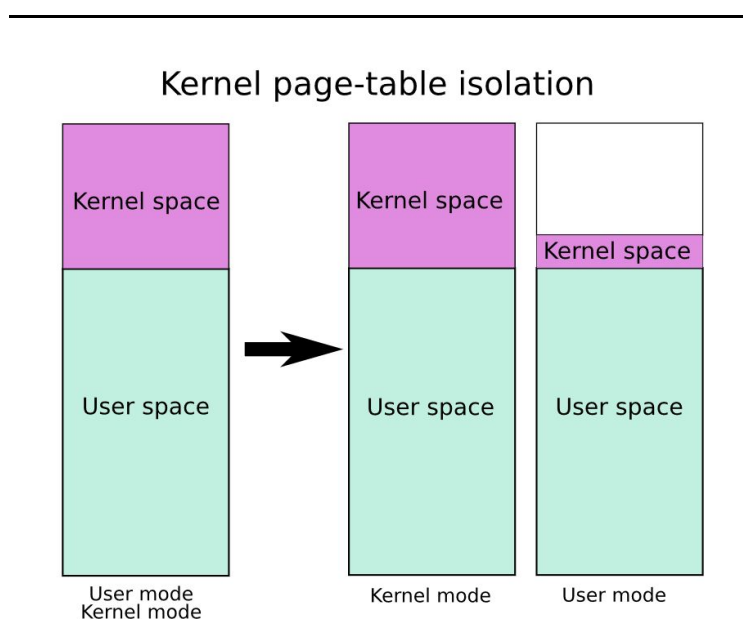


Diagrama aislamiento de tablas de páginas del núcleo

La sobrecarga de trabajo se estableció en un 0.28 % según los autores originales de KAISER un desarrollador de Linux la fijó en aproximadamente un 5 % para la mayoría de tareas típicas y en un máximo de un 30 % en determinados casos aún contando con la optimización de PCID

Demostración

El entorno de trabajo utilizado para las demostraciones comprende sigue estas especificaciones Hardware en materia de procesador:

Como se puede observar, el procesador empleado es el Intel Core i5-5200U. Dicha CPU es vulnerable a Spectre como a Meltdown [].

El sistema operativo empleado ha sido Ubuntu 14.04 del año 2014. Con lo que nos aseguramos de que no tenga instalados los parche de seguridad para mitigar tanto Spectre como Meltdown.

Spectre

Código obtenido del siguiente repositorio [github](#)

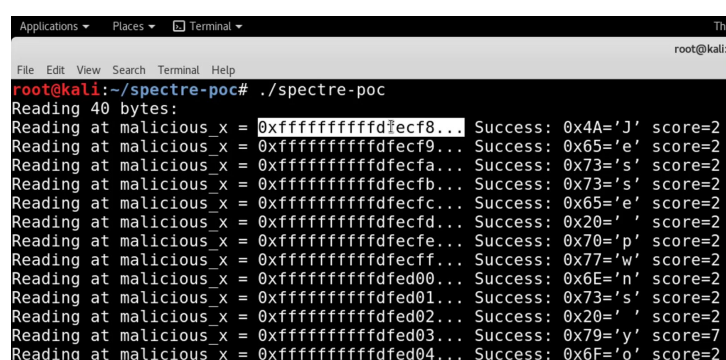
```
ubuntu@ubuntu:~/Desktop$ ./a.out
Reading 40 bytes:
Reading at malicious_x = 0xffffffffdffb08... Success: 0x54='T' score=2
Reading at malicious_x = 0xffffffffdffb09... Success: 0x68='h' score=2
Reading at malicious_x = 0xffffffffdffb0a... Success: 0x65='e' score=2
Reading at malicious_x = 0xffffffffdffb0b... Success: 0x20=' ' score=2
Reading at malicious_x = 0xffffffffdffb0c... Success: 0x4D='M' score=2
Reading at malicious_x = 0xffffffffdffb0d... Success: 0x61='a' score=2
Reading at malicious_x = 0xffffffffdffb0e... Success: 0x67='g' score=2
Reading at malicious_x = 0xffffffffdffb0f... Success: 0x69='i' score=2
Reading at malicious_x = 0xffffffffdffb10... Success: 0x63='c' score=2
Reading at malicious_x = 0xffffffffdffb11... Success: 0x20=' ' score=2
Reading at malicious_x = 0xffffffffdffb12... Success: 0x57='W' score=2
Reading at malicious_x = 0xffffffffdffb13... Success: 0x6F='o' score=2
Reading at malicious_x = 0xffffffffdffb14... Success: 0x72='r' score=2
Reading at malicious_x = 0xffffffffdffb15... Success: 0x64='d' score=2
Reading at malicious_x = 0xffffffffdffb16... Success: 0x73='s' score=2
Reading at malicious_x = 0xffffffffdffb17... Success: 0x20=' ' score=2
```

```

Reading at malicious_x = 0xffffffffdffb18... Success: 0x61='a' score=2
Reading at malicious_x = 0xffffffffdffb19... Success: 0x72='r' score=2
Reading at malicious_x = 0xffffffffdffb1a... Success: 0x65='e' score=2
Reading at malicious_x = 0xffffffffdffb1b... Success: 0x20=' ' score=2
Reading at malicious_x = 0xffffffffdffb1c... Success: 0x53='S' score=2
Reading at malicious_x = 0xffffffffdffb1d... Success: 0x71='q' score=2
Reading at malicious_x = 0xffffffffdffb1e... Success: 0x75='u' score=2
Reading at malicious_x = 0xffffffffdffb1f... Success: 0x65='e' score=2
Reading at malicious_x = 0xffffffffdffb20... Success: 0x61='a' score=2
Reading at malicious_x = 0xffffffffdffb21... Success: 0x6D='m' score=2
Reading at malicious_x = 0xffffffffdffb22... Success: 0x69='i' score=2
Reading at malicious_x = 0xffffffffdffb23... Success: 0x73='s' score=2
Reading at malicious_x = 0xffffffffdffb24... Success: 0x68='h' score=2
Reading at malicious_x = 0xffffffffdffb25... Success: 0x20=' ' score=2
Reading at malicious_x = 0xffffffffdffb26... Success: 0x4F='O' score=2
Reading at malicious_x = 0xffffffffdffb27... Success: 0x73='s' score=2
Reading at malicious_x = 0xffffffffdffb28... Success: 0x73='s' score=2
Reading at malicious_x = 0xffffffffdffb29... Success: 0x69='i' score=2
Reading at malicious_x = 0xffffffffdffb2a... Success: 0x66='f' score=2
Reading at malicious_x = 0xffffffffdffb2b... Success: 0x72='r' score=2
Reading at malicious_x = 0xffffffffdffb2c... Success: 0x61='a' score=2
Reading at malicious_x = 0xffffffffdffb2d... Success: 0x67='g' score=2
Reading at malicious_x = 0xffffffffdffb2e... Success: 0x65='e' score=2
Reading at malicious_x = 0xffffffffdffb2f... Success: 0x2E='.' score=2

```

Como ya se ha explicado con anterioridad este programa lee memoria que se aloja en su mismo proceso, lo cual supone una gran amenaza cuando el código ejecutado es proporcionado por una fuente externa como puede ser una página web. El motor de javascript ejecutaría el código de forma normal pero podría acceder a memoria reservada para otra web, pudiendo ser posible acceder a contraseñas, Tokens de autenticación, cookies, etc.



```

Applications ▾ Places ▾ Terminal ▾ Thu
root@kali:
File Edit View Search Terminal Help
root@kali:~/spectre-poc# ./spectre-poc
Reading 40 bytes:
Reading at malicious_x = 0xffffffffdfeb8... Success: 0x4A='J' score=2
Reading at malicious_x = 0xffffffffdfeb9... Success: 0x65='e' score=2
Reading at malicious_x = 0xffffffffdfebfa... Success: 0x73='s' score=2
Reading at malicious_x = 0xffffffffdfebfb... Success: 0x73='s' score=2
Reading at malicious_x = 0xffffffffdfebfc... Success: 0x65='e' score=2
Reading at malicious_x = 0xffffffffdfebfd... Success: 0x20=' ' score=2
Reading at malicious_x = 0xffffffffdfebfe... Success: 0x70='p' score=2
Reading at malicious_x = 0xffffffffdfebff... Success: 0x77='w' score=2
Reading at malicious_x = 0xffffffffdfeb00... Success: 0x6E='n' score=2
Reading at malicious_x = 0xffffffffdfeb01... Success: 0x73='s' score=2
Reading at malicious_x = 0xffffffffdfeb02... Success: 0x20=' ' score=2
Reading at malicious_x = 0xffffffffdfeb03... Success: 0x79='y' score=7
Reading at malicious_x = 0xffffffffdfeb04... Success: 0x6F='o' score=2

```

[Vídeo demostración](#) de Spectre en acción

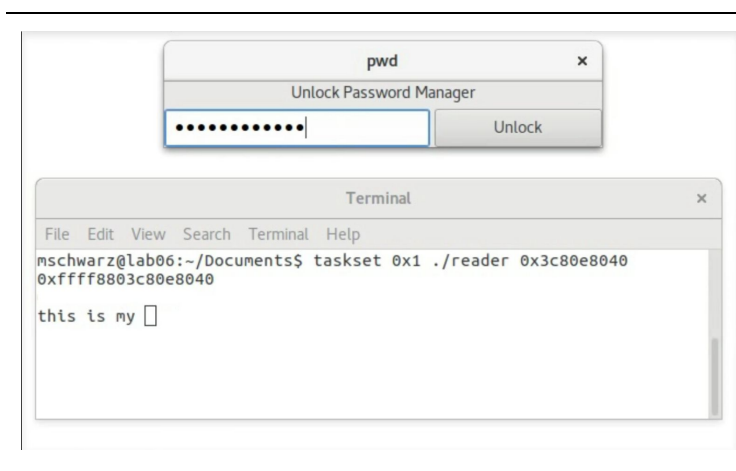
Meltdown

Código obtenido del siguiente repositorio [github](#)

```
$> ./run.sh
looking for linux_proc_banner in /proc/kallsyms
protected. requires root
+ find_linux_proc_banner /proc/kallsyms sudo
+ sudo sed -n -re s/^[0-9a-f]*[1-9a-f][0-9a-f]*).* linux_proc_banner$/\1/p
/proc/kallsyms
+ linux_proc_banner=ffffffff81800060
+ set +x
ffffffff81800060
cached = 27, uncached = 245, threshold 81
read ffffffff81800060 = 25 % (score=983/1000)
read ffffffff81800061 = 73 s (score=992/1000)
read ffffffff81800062 = 20 (score=545/1000)
read ffffffff81800063 = 76 v (score=904/1000)
read ffffffff81800064 = 65 e (score=972/1000)
read ffffffff81800065 = 72 r (score=995/1000)
read ffffffff81800066 = 73 s (score=997/1000)
read ffffffff81800067 = 69 i (score=988/1000)
read ffffffff81800068 = 6f o (score=977/1000)
read ffffffff81800069 = 6e n (score=988/1000)
read ffffffff8180006a = 20 (score=988/1000)
read ffffffff8180006b = 25 % (score=994/1000)
read ffffffff8180006c = 73 s (score=989/1000)
read ffffffff8180006d = 20 (score=982/1000)
read ffffffff8180006e = 28 ( (score=678/1000)
read ffffffff8180006f = 62 b (score=979/1000)
VULNERABLE
PLEASE POST THIS TO https://github.com/paboldin/meltdown-exploit/issues/19
VULNERABLE ON
4.4.0-31-generic #50~14.04.1-Ubuntu SMP Wed Jul 13 01:07:32 UTC 2016 x86_64
processor      : 0
vendor_id     : GenuineIntel
cpu family    : 6
model         : 61
model name    : Intel(R) Core(TM) i5-5200U CPU @ 2.20GHz
stepping      : 4
microcode     : 0x1f
cpu MHz       : 2399.976
cache size    : 3072 KB
physical id   : 0
```


Como ya se ha comentado en la parte de desarrollo este código se aprovecha del fallo de las CPUs de Intel a la hora de la ejecución especulativa ya que no se comprueban los bits de permisos de las páginas mientras se está ejecutando especulativamente, es por ello que el contenido se queda almacenado en caché y queda accesible al espacio de usuario

En el siguiente enlace podemos ver un video de la demostración de meltdown para leer memoria del proceso encargado de dar permisos de superusuario al introducir la contraseña de algún usuario que esté en el archivo 'sudoers'.



[Vídeo demostración](#) de meltdown en acción

Conclusiones

Es muy raro que un resultado de investigación cambie fundamentalmente la forma en que las computadoras se construyen y ejecutan. Meltdown y Spectre han hecho precisamente eso. Estos hallazgos alterarán sustancialmente el diseño de hardware y software durante los próximos 7 a 10 años (el próximo ciclo de hardware de la CPU) a medida que los diseñadores toman en cuenta la nueva realidad de las posibilidades de fuga de datos a través de los canales laterales de caché.

En esencia Meltdown y Spectre solo han servido para mejorar nuestra visión sobre los procesadores y enseñarnos un foco donde centrar nuestros esfuerzos para mejorar la seguridad sin perder las mejoras de rendimiento obtenido gracias a la ejecución especulativa.

Bibliografía

Medium. (2018). *Meltdown and Spectre, explained – Matt Klein – Medium*. [en línea] Disponible en: <https://medium.com/@mattklein123/meltdown-spectre-explained-6bc8634cc0c2> [Accedido 6 Dec. 2018].

Nvd.nist.gov. (2018). *NVD - CVE-2017-5715*. [en línea] Disponible en: <https://nvd.nist.gov/vuln/detail/CVE-2017-5715> [Accedido 6 Dec. 2018].

Nvd.nist.gov. (2018). *NVD - CVE-2017-5753*. [en línea] Disponible en: <https://nvd.nist.gov/vuln/detail/CVE-2017-5753> [Accedido 6 Dec. 2018].

Nvd.nist.gov. (2018). *NVD - CVE-2017-5754*. [en línea] Disponible en: <https://nvd.nist.gov/vuln/detail/CVE-2017-5754> [Accedido 6 Dec. 2018].

GitHub. (2018). *paboldin/meltdown-exploit*. [en línea] Disponible en: <https://github.com/paboldin/meltdown-exploit> [Accedido 6 Dec. 2018].

Gist. (2018). *Spectre example code*. [en línea] Disponible en: <https://gist.github.com/ErikAugust/724d4a969fb2c6ae1bbd7b2a9e3d4bb6> [Accedido 6 Dec. 2018].

Kurru, J. (2018). *Video demostración Spectre*. [video] Disponible en: <https://youtu.be/0kHFvUcQsWQ> [Accedido 6 Dec. 2018].

Vídeo demostración de meltdown en acción. (2018). [video] Disponible en: <https://cdn.rawgit.com/IAIK/meltdown/master/videos/spy.mp4> [Accedido 6 Dec. 2018].

Es.wikipedia.org. (2018). *Aislamiento de tablas de páginas del núcleo*. [en línea] Disponible en: https://es.wikipedia.org/wiki/Aislamiento_de_tablas_de_p%C3%A1ginas_del_n%C3%BAcleo [Accedido 6 Dec. 2018].

Wong, D. (2018). *Complete List Of CPUs Vulnerable To Meltdown / Spectre Rev. 8.0*. [en línea] Tech ARP. Disponible en: <https://www.techarp.com/guides/complete-meltdown-spectre-cpu-list/> [Accedido 6 Dec. 2018].

En.wikichip.org. (2018). *CVE-2017-5715 (Spectre, Variant 2) - CVE - WikiChip*. [en línea] Disponible en: <https://en.wikichip.org/wiki/cve/cve-2017-5715> [Accedido 6 Dec. 2018].