Spectre & Meltdown

Estamos acostumbrados a pensar en procesadores de computadoras como máquinas ordenadas que proceden de una simple instrucción a la siguiente con completa regularidad. Pero la verdad es que desde hace décadas, han estado haciendo sus tareas fuera de orden y adivinando lo que debería venir después. Los procesadores son muy buenos en eso, de hecho tan buenos, que esta capacidad, llamada ejecución especulativa, que ha apuntalado gran parte de la mejora en la potencia informática durante los últimos 25 años más o menos. Pero el 3 de enero de 2018, el mundo aprendió que este truco, que había hecho tanto por la informática moderna, era ahora una de sus mayores vulnerabilidades.

En materia de seguridad informática entendemos como vulnerabilidades a los puntos débiles de los sistemas que son comúnmente aprovechados por personas que buscan la manera de acceder y realizar alguna acción maliciosa para su propio beneficio.

Meltdown y Spectre aprovechan vulnerabilidades críticas en los procesadores modernos. Estas vulnerabilidades de hardware permiten a los programas robar datos que se procesan actualmente en el equipo. Mientras que a los programas normalmente no se les permite leer datos de otros programas, un programa malicioso puede explotar Meltdown y Spectre para obtener secretos almacenados en la memoria de otros programas en ejecución. Esto puede incluir sus contraseñas almacenadas en un administrador de contraseñas o navegador, sus fotos personales, correos electrónicos, mensajes instantáneos e incluso documentos críticos para algún negocio.

Meltdown y Spectre funcionan en ordenadores personales, dispositivos móviles y en la nube. Dependiendo de la infraestructura del proveedor de nube, es posible robar datos de otros clientes.

★ Meltdown

La seguridad de los sistemas informáticos se basa fundamentalmente en el aislamiento de la memoria, por ejemplo, los intervalos de direcciones del kernel se marcan como no accesibles y están protegidos contra el acceso de los usuarios.

Meltdown explota los efectos secundarios de la ejecución fuera de orden en procesadores modernos para leer ubicaciones arbitrarias de memorias del kernel, incluidos datos personales y contraseñas.

La ejecución fuera de orden es una característica de rendimiento indispensable y está presente en una amplia gama de procesadores modernos. El ataque funciona en diferentes

microarquitecturas Intel desde al menos 2010 y potencialmente otros procesadores se ven afectados. La causa principal de Meltdown es el hardware. El ataque es independiente del sistema operativo, y no se basa en ninguna vulnerabilidad de software.

Meltdown rompe todos los supuestos de seguridad dados por el aislamiento del espacio de direcciones, así como los entornos paravirtualizados y, por lo tanto, todos los mecanismos de seguridad basados en esta base. En los sistemas afectados, Meltdown permite a un adversario leer la memoria de otros procesos o máquinas virtuales en la nube sin ningún permiso o privilegio, afectando a millones de clientes y prácticamente a todos los usuarios de un ordenador personal.

Meltdown explota la información de canal lateral disponible en la mayoría de los procesadores modernos.

Mientras que los ataques de canal lateral normalmente requieren un conocimiento muy específico sobre la aplicación de destino y solo filtran información sobre los secretos de la aplicación de destino. Meltdown es un ataque novedoso que permite superar completamente el aislamiento de memoria al permitir que un adversario que puede ejecutar código en el procesador vulnerable, se le proporciona una manera sencilla para que cualquier proceso de usuario lea toda la memoria del kernel de la máquina en la que se ejecuta, incluida toda la memoria física asignada en la región del kernel.

La causa principal de la simplicidad y la fuerza de Meltdown son efectos secundarios causados por la ejecución fuera de orden. La ejecución fuera de orden es una característica de rendimiento importante de los procesadores actuales para superar las latencias de las unidades de ejecución ocupadas, por ejemplo, una unidad de recuperación de memoria (fetch) debe esperar a que los datos lleguen desde la memoria. En lugar de detener la ejecución, los procesadores modernos ejecutan operaciones fuera de orden, es decir, miran hacia adelante y programan las operaciones posteriores en unidades de ejecución inactivas del procesador. Sin embargo, tales operaciones a menudo tienen efectos secundarios no deseados, pueden filtrar información de la ejecución secuencial y fuera de orden.

Las CPU vulnerables permiten que un proceso sin privilegios cargue datos desde una dirección privilegiada (kernel o física) en un registro temporal de CPU. Además, la CPU incluso realiza más cálculos basados en este valor de registro, por ejemplo, el acceso a un arreglo basada en el valor de registro. El procesador garantiza la correcta ejecución del programa, simplemente descartando los resultados de las búsquedas de memoria (por ejemplo, los estados de registro modificados), si resulta que no se debería haber ejecutado una instrucción. Por lo tanto, en el nivel arquitectónico (por ejemplo, la definición abstracta de cómo el procesador debe realizar los cálculos), no surge ningún problema de seguridad.

Sin embargo, observamos que las búsquedas de memoria fuera de orden influyen en la memoria caché, que a su vez se puede detectar a través del canal lateral de la memoria caché. Como resultado, un atacante puede volcar toda la memoria del kernel leyendo memoria con privilegios en una secuencia de ejecución fuera de orden y transmitir los datos desde este estado esquiva a través de un canal cobertizo microarquitectónico (por ejemplo, Flush+Reload) al mundo exterior. En el extremo receptor del canal encubierto, se reconstruye el valor del registro. Por lo tanto, en el nivel microarquitectónico (por ejemplo, la implementación de hardware real), hay un problema de seguridad explotable.

Si bien el rendimiento depende en gran medida de la máquina específica, por ejemplo, la velocidad del procesador, los tamaños de caché y TLB, y la velocidad de la DRAM, podemos volcar el kernel y la memoria física con hasta 503KB/s.

- ➔ **Bloques en los que se divide**

El ataque completo de Meltdown consta de 2 bloques de construcción, de manera general consiste en lo siguiente:

El primer bloque consiste en hacer que el CPU ejecute una o más instrucciones que nunca ocurrirán en una ruta determinada. Llamaremos instrucciones transitorias, a las instrucciones que se ejecutan fuera de orden y dejan efectos secundarios medibles.

Para aprovechar las instrucciones transitorias para un ataque, la secuencia de instrucciones transitorias debe utilizar un valor secreto que el atacante quiere filtrar.

El segundo bloque de construcción consiste en transferir los efectos secundarios de la secuencia de instrucciones transitorias de un estado arquitectónico para seguir procesando el secreto filtrado.

- ◆ **Ejecutando instrucciones transitorias**

Las instrucciones transitorias ocurren todo el tiempo, ya que la CPU se ejecuta continuamente por delante de la instrucción actual para minimizar la latencia experimentada y, por lo tanto, para maximizar el rendimiento.

Las instrucciones transitorias introducen un canal explotable si su funcionamiento depende de un valor secreto.

Acceder a páginas inaccesibles para el usuario, como las páginas del kernel, desencadena una excepción que generalmente termina la aplicación. Si el atacante apunta a un valor secreto en una dirección inaccesible el atacante tiene que hacer frente a esta excepción. Se proponen 2 enfoques: con el manejo de excepciones capturamos la excepción que ocurre

efectivamente después de ejecutar la secuencia de instrucción transitoria, y con la supresión de excepciones, evitamos la excepción y, en su lugar,se redirige el control de flujo después de ejecutar la secuencia de instrucciones transitorias.

- **Exception handling**

Un enfoque trivial es bifurcar la aplicación de ataque antes de acceder a una ubicación de memoria inválida que finaliza el proceso y de esta manera solo acceder a la ubicación de memoria no válida en el proceso secundario.

De esta manera la CPU ejecuta la secuencia de instrucciones transitorias en el proceso hijo antes de estrellarse. El proceso padre puede luego recuperar el secreto observando el estado de la microarquitectura, por ejemplo, a través de un canal lateral.

También es posible instalar una señal controladora que se ejecuta cuando ocurre una cierta excepción, por ejemplo, un fallo de segmentación. Esto permite al atacante emitir la secuencia de instrucciones y evitar que la aplicación se bloquee, reduciendo la sobrecarga ya que no es necesario crear un nuevo proceso.

- **Exception suppression**

Un enfoque diferente para tratar con excepciones es evitar que sean lanzadas en primer lugar. La memoria transaccional permite agrupar los accesos a memoria en una operación aparentemente atómica, dando la opción de retroceder a un estado anterior si se produce un error. Si ocurre una excepción dentro de la transacción, el estado arquitectónico se restablece y la ejecución del programa continúa sin interrupciones.

Además la ejecución especulativa emite instrucciones que podrían no ser ejecutadas debido a una rama de predicción errónea. Dichas instrucciones que dependen de una rama condicional precedente puede ejecutarse especulativamente.

Por lo tanto, el acceso no válido a la memoria se coloca dentro de una secuencia de instrucción especulativa que sólo se ejecuta si una condición de ramificación anterior se evalúa como verdadera. Al asegurarnos de que la condición nunca se evalúe como verdadera en la ruta del código ejecutado, podemos suprimir la excepción que se produce ya que el acceso a la memoria solo se ejecuta de manera especulativa.

- **Construyendo el canal encubierto**

El extremo receptor del canal secreto recibe el cambio de estado de microarquitectura y deduce el secreto de este estado.

Aprovechamos las técnicas de los ataques de caché, ya que el estado de caché es un estado de microarquitectura que se puede transferir de manera confiable a un estado arquitectónico utilizando diversas técnicas. Por ejemplo se puede utilizar Flush + Reload ya que permite construir un canal secreto rápido y silencioso.

Después de que la secuencia de instrucciones transitorias accedió a una dirección accesible, la dirección se almacena en caché para accesos posteriores.

El receptor puede monitorear si la dirección ha sido cargada en el caché midiendo el tiempo de acceso a la dirección. Por lo tanto, el remitente puede transmitir un bit '1' para acceder a una dirección que se carga en el caché y un bit '0' al no acceder a dicha dirección.

- ➔ **Descripción del ataque**

Meltdown consta de 3 pasos:

Paso 1 (Leyendo el secreto): Se carga el contenido de una ubicación de memoria elegida por el atacante, que es inaccesible para el atacante en un registro.

Para cargar datos desde la memoria principal a un registro, los datos en la memoria principal se referencian utilizando una dirección virtual. En paralelo a la traducción de una dirección virtual a una dirección física, el CPU también verifica los bits de permiso de la dirección virtual, es decir, si esta dirección virtual es accesible para el usuario o solo accesible por el núcleo. Este aislamiento está basado en hardware a través de un bit de misión, que se considera seguro y recomendado por vendedores de hardware. Por lo tanto, los sistemas operativos modernos siempre asignan todo el núcleo al espacio de direcciones virtuales de cada proceso de usuario.

Como consecuencia, todas las direcciones del núcleo conducen a una dirección física al traducirlos, y la CPU puede acceder al contenido de tales direcciones. La única diferencia con acceder a una dirección de espacio de usuario es que la CPU genera una excepción para el nivel de permiso actual y no permite el acceso a dicha dirección. Por lo tanto, el espacio de usuario no puede simplemente leer el contenido de dicha dirección.
Sin embargo, Meltdown explota la ejecución fuera de orden de las CPU modernas, que ejecuta instrucciones en el pequeño intervalo de tiempo entre el acceso ilegal a la memoria y el lanzamiento de la excepción.

Paso 2 (Transmitiendo el secreto): Una instrucción transitoria accede a una línea de caché basado en el contenido secreto del registro.

Paso 3 (Recibiendo el secreto): El atacante usa Flush + Reload para determinar el línea de caché accedida y, por lo tanto, el secreto almacenado en el ubicación de memoria elegida.

Al repetir estos pasos para diferentes ubicaciones de memoria, el atacante puede volcar la memoria del núcleo, incluida la memoria física completa.

➔ Contramedidas

En esta parte tendremos 2 enfoques, como ya se discutió el problema radica en el hardware por lo que se discuten posibles actualizaciones de microcódigo y cambios generales en el diseño del hardware. En segundo lugar, se presenta KAISER una contramedida basada en software.

◆ Hardware

Meltdown evita el aislamiento forzado por hardware de dominios de seguridad No hay vulnerabilidad de software involucrada en Meltdown.

Cualquier parche de software (por ejemplo KAISER) dejará pequeñas cantidades de memoria expuestas. No hay documentación para saber si la solución requiere el desarrollo de completamente nuevo hardware, o puede repararse mediante una actualización de microcódigo.

Mientras Meltdown explota la ejecución fuera de orden, una trivial contramedida sería deshabilitar la ejecución fuera de orden completamente. Sin embargo, los impactos en el rendimiento serían devastadores, ya que el paralelismo de las CPU modernas ya no podría aprovecharse. Por lo tanto, esta no es una solución viable.

Meltdown presenta una especie de condición de carrera entre el buscar una dirección de memoria y la verificación de permisos correspondiente para esta dirección. Serializar la verificación de permisos y la recuperación de registros puede evitar Meltdown, ya que la dirección de memoria nunca se recupera si falla la verificación de permisos. Sin embargo, esto implica una sobrecarga significativa para cada recuperación de memoria, ya que la recuperación de memoria tendría que detenerse hasta que se complete la verificación de permisos.

Una solución más realista sería introducir un división dura del espacio de usuario y espacio de kernel. Esto podría habilitarse opcionalmente por los núcleos modernos utilizando un nuevo bit de división dura en un registro de control de la CPU.

Si se establece el nuevo bit de división, el núcleo debe residir en la mitad superior del espacio de direcciones, y el espacio de usuario tiene que residir en la mitad inferior del espacio de direcciones. Con esta división dura, una búsqueda de memoria puede identificar

inmediatamente si tal la búsqueda del destino violaría un límite de seguridad, ya que el nivel de privilegio puede derivarse directamente de la dirección virtual sin más búsquedas.

- **KAISER**

Como el hardware existente no es tan fácil de parchar, hay un necesidad de soluciones de software hasta que el nuevo hardware pueda ser desplegado.

Gruss propuso KAISER, una modificación del núcleo para no tener el núcleo asignado en el espacio de usuario. Esta modificación tenía la intención de pre ventilar los ataques de canal lateral que rompen KASLR. Sin embargo, también evita Meltdown, ya que garantiza que no hay mapeo válido al espacio de memoria del kernel o físico disponible en el espacio del usuario.

★ Spectre

Explotación de la Ejecución Especulativa. Los procesadores modernos utilizan la predicción de rama (*Branch Prediction*) y la ejecución especulativa (*Speculative Execution*) para maximizar el rendimiento. Los ataques Spectre implican inducir a una víctima a realizar operaciones especulativas que no ocurrirían durante la ejecución correcta del programa y que filtran la información confidencial de la víctima a través de un canal lateral al adversario.

Los cálculos realizados por dispositivos físicos a menudo dejan efectos secundarios observables más allá de los resultados nominales del cálculo. Los ataques de canal lateral se centran en explotar estos efectos secundarios para extraer información secreta que de otro modo no estaría disponible,se pueden utilizar para extraer información secreta de dispositivos complejos como PC y teléfonos móviles, estos ataques explotan las vulnerabilidades del software (como desbordamientos del búfer o errores dobles libres), otros ataques de software aprovechan las vulnerabilidades del hardware para filtrar información confidencial. Los ataques de este último tipo incluyen ataques de microarquitectura que explotan el tiempo de caché, historial de predicción de sucursales, búferes de destino de sucursal o filas de DRAM abiertas.

- **Ejecución Especulativa (Speculative Execution)**

A menudo, el procesador no conoce el flujo de instrucción futuro de un programa. Por ejemplo, esto ocurre cuando la ejecución fuera de orden alcanza una instrucción de rama condicional cuya dirección depende de instrucciones anteriores cuya ejecución aún no se ha completado. En tales casos, el procesador puede conservar su estado de registro actual,

hacer una predicción de la ruta que seguirá el programa y ejecutar instrucciones especulativas a lo largo de la ruta. Si la predicción resulta ser correcta, los resultados de la ejecución especulativa se comprometen (es decir, se guardan), lo que proporciona una ventaja de rendimiento sobre el retraso durante la espera. De lo contrario, cuando el procesador determina que siguió el camino equivocado, abandona el trabajo que realizó especulativamente al revertir su estado de registro y reanudar a lo largo de la ruta correcta.

Nos referimos a las instrucciones que se realizan erróneamente (es decir, como resultado de una predicción errónea), pero que pueden dejar rastros de microarquitectura, *instrucciones transitorias.* Aunque la ejecución especulativa mantiene el estado arquitectónico del programa como si la ejecución siguiera el camino correcto, los elementos microarquitecturales pueden estar en un estado diferente (pero válido) que antes de la ejecución transitoria.

La ejecución especulativa en las CPU modernas puede ejecutar cientos de instrucciones por delante. El límite generalmente se rige por el tamaño del búfer de reordenamiento en la CPU. Por ejemplo, en la microarquitectura Haswell, el buffer de reordenamiento tiene espacio suficiente para 192 microoperaciones. Como no existe una relación uno a uno entre el número de microoperaciones e instrucciones, el límite depende de las instrucciones que se utilicen.

➔ **Predicción de Rama (Branch Prediction)**

Durante la ejecución especulativa, el procesador hace conjeturas sobre el resultado probable de las instrucciones de ramificación. Las mejores predicciones mejoran el rendimiento al aumentar el número de operaciones especulativas ejecutadas que se pueden realizar con éxito.

Los predictores de rama de los procesadores Intel modernos, tienen múltiples mecanismos de predicción para ramas directas e indirectas. Las instrucciones indirectas de rama pueden saltar a direcciones de destino arbitrarias calculadas en tiempo de ejecución. Por ejemplo, las instrucciones x86 pueden saltar a una dirección en un registro, en una dirección de memoria o en la pila. Las ramas indirectas también son compatibles con ARM y otros procesadores. Para compensar la flexibilidad adicional en comparación con las ramas directas, los saltos indirectos y las llamadas se optimizan utilizando al menos dos mecanismos de predicción diferentes. Intel describe que el procesador predice :

- “Llamadas y saltos directos” de manera estática o monotónica
- “Llamadas y saltos indirectos”, ya sea de manera monótona o de manera variable, lo que depende del comportamiento reciente del programa, y para
- “Ramas condicionales”, el objetivo de la rama y si se tomará la rama.

    ◆ **Branch Predictor**

En arquitectura de computadoras, un predictor de rama es un circuito digital que intenta adivinar en qué dirección irá una rama antes de que esto se conozca definitivamente. El propósito del predictor de rama es mejorar el flujo en el pipeline de instrucciones. Los predictores de rama desempeñan un papel fundamental para lograr un alto rendimiento efectivo en muchas arquitecturas modernas de microprocesadores canalizados como x86.

La primera vez que se encuentra una instrucción de salto condicional, no hay mucha información sobre la cual basar una predicción. Pero el predictor de rama mantiene registros de si las ramas se toman o no. Cuando se encuentra con un salto condicional que se ha visto varias veces antes, puede basar la predicción en la historia. El predictor de rama puede, por ejemplo, reconocer que el salto condicional se realiza con mayor frecuencia o que se realiza cada dos veces.

- **Branch Target Predictor**

Un predictor de objetivo de rama es la parte de un procesador que predice el objetivo de una rama condicional tomada o una instrucción de rama incondicional antes de que la unidad de ejecución del procesador calcule el objetivo de la instrucción de rama.

La predicción de rama no es lo mismo que la predicción de objetivo de rama. La predicción de rama intenta adivinar si se realizará un salto condicional o no. La predicción de objetivo de rama intenta adivinar el objetivo de un salto condicional o incondicional realizado antes de que se calcule decodificando y ejecutando la instrucción en sí.

➔ Jerarquía de la Memoria

Cuando el procesador necesita datos de la memoria, primero verifica si el caché L1, en la parte superior de la jerarquía, contiene una copia. En el caso de un golpe de caché, es decir, los datos se encuentran en el caché, los datos se recuperan de este y se usan. De lo contrario, en el caso de una falta de memoria caché, el procedimiento se repite para intentar recuperar los datos de los siguientes niveles de caché y, finalmente, la memoria externa. Una vez que se completa una lectura, los datos generalmente se almacenan en la memoria caché (y se desaloja un valor previamente almacenado en caché para hacer espacio) en caso de que sea necesario recuperarlo en el futuro cercano.

Los procesadores Intel modernos suelen tener tres niveles de caché, cada núcleo tiene cachés dedicados L1 y L2 y todos los núcleos comparten un caché L3 común, también conocido como caché de último nivel (LLC). Un procesador debe asegurarse de que el caché L1 y L2 por núcleo sea un caché coherente, a menudo basado en el protocolo MESI. En particular, el uso del protocolo MESI o algunas de sus variantes implica que una operación de escritura de memoria en un núcleo hará que las copias de los mismos datos en los cachés L1 y L2 de otros núcleos se marquen como inválidas, lo que significa que los futuros accesos a estos datos en otros núcleos no podrá cargar rápidamente los datos de la caché L1 o L2.

Cuando esto sucede repetidamente en una ubicación de memoria específica, esto se llama informalmente salto de línea de caché. Debido a que la memoria se almacena en caché con

una granularidad de línea, esto puede suceder incluso si dos núcleos acceden a diferentes ubicaciones de memoria cercanas que se asignan a la misma línea de caché. Este comportamiento se denomina intercambio falso y es conocido como una fuente de problemas de rendimiento.

- ➔ **Ataques microarquitectónicos de canal lateral**

Cuando varios programas se ejecutan en el mismo hardware, ya sea simultáneamente o por tiempo compartido, los cambios en el estado de microarquitectura causados por el comportamiento de un programa puede afectar a otros programas. Esto, a su vez, puede dar lugar a fugas de información no intencionadas de un programa a otro.

Los ataques iniciales de canales laterales microarquitectónicos explotaron la variabilidad de tiempo y la fuga a través del caché de datos L1 para extraer claves de primitivas criptográficas. A lo largo de los años, se han demostrado canales a través de múltiples componentes microarquitectónicos, incluidos el caché de instrucciones, los cachés de nivel inferior, el BTB y el historial de ramificaciones. Los objetivos de los ataques tácticos se han ampliado para abarcar la detección de ubicación conjunta, el control de pulsaciones de teclas, las huellas digitales en un sitio web Los resultados recientes incluyen ataques de núcleo cruzado y CPU cruzada, ataques basados en la nube, ataques en y desde entornos de ejecución de confianza, ataques de código móvil , y nuevas técnicas de ataque.

Usando estas técnicas, el atacante comienza desalojando una línea de caché del caché que se comparte con la víctima. Después de que la víctima se ejecuta por un tiempo, el atacante mide el tiempo que le toma realizar una lectura de memoria en la dirección correspondiente a la línea de caché expulsada. Si la víctima accedió a la línea de caché monitoreada, los datos estarán en el caché y el acceso será rápido. De lo contrario, si la víctima no ha accedido a la línea, la lectura será lenta. Por lo tanto, al medir el tiempo de acceso, el atacante aprende si la víctima accedió a la línea de caché monitoreada entre los pasos de desalojo y sondeo.

- ➔ **Programación Orientada al Retorno**

Es una técnica de explotación de seguridad informática que permite a un atacante ejecutar código ante la presencia de defensas de seguridad como protección de espacio ejecutable y código firmado.

En esta técnica, un atacante gana control de la pila de llamadas para secuestrar el flujo de control del programa y entonces ejecuta secuencias de instrucciones de máquina elegidas cuidadosamente entre las que ya están presentes en la memoria del computador, las cuales se denominan "gadgets". Cada gadget típicamente termina con una instrucción de retorno y está localizado en una subrutina dentro del programa existente y/o en el código de alguna biblioteca compartida. Estos gadgets encadenados le permiten a un atacante realizar operaciones arbitrarias en un computador que emplee defensas contra ataques más simples.

- ➔ **Vista General del Ataque**

Los ataques Spectre inducen a una víctima a realizar operaciones especulativas que no ocurrirían durante el procesamiento en orden de las instrucciones del programa, y que filtran la información confidencial de la víctima a través de un canal secreto al adversario.

En la mayoría de los casos, el ataque comienza con una fase de configuración, donde el adversario realiza operaciones que maltratan al procesador para que luego realice una predicción especulativa explícitamente errónea. Además, la fase de configuración generalmente incluye pasos que ayudan a inducir la ejecución especulativa, como la manipulación del estado de la memoria caché para eliminar los datos que el procesador necesitará para determinar el flujo de control real. Durante la fase de configuración, el adversario también puede preparar el canal encubierto que se utilizará para extraer la información de la víctima.

Durante la segunda fase, el procesador ejecuta especulativamente instrucciones que transfieren información confidencial del contexto de la víctima a un canal secreto microarquitectural. Esto puede activarse haciendo que el atacante solicite a la víctima que realice una acción, por ejemplo, mediante una llamada al sistema, un socket o un archivo.

En otros casos, el atacante puede aprovechar la ejecución especulativa (errónea) de su propio código para obtener información confidencial del mismo proceso. Por ejemplo, el código de ataque que está encerrado por un intérprete, un compilador justo a tiempo o un "lenguaje seguro" puede querer leer la memoria que se supone que no debe acceder. Si bien la ejecución especulativa puede exponer datos sensibles a través de una amplia gama de canales encubiertos, los ejemplos dados provocan que la ejecución especulativa lea primero un valor de memoria en una dirección elegida por el atacante y luego realice una operación de memoria que modifica el estado de la memoria caché de una manera que expone el valor.

Para la fase final, se recuperan los datos confidenciales. Para los ataques Spectre, el proceso de recuperación consiste en cronometrar el acceso a las direcciones de memoria en las líneas de caché que se monitorean. Los ataques de Spectre solo suponen que las instrucciones ejecutadas especulativamente pueden leer de la memoria que se podría acceder normalmente, sin activar una falla de página o una excepción.

Por lo tanto, Spectre es ortogonal a Meltdown que explota escenarios en los que algunas CPU permiten la ejecución fuera de orden de las instrucciones del usuario para leer la memoria del kernel. En consecuencia, incluso si un procesador impide la ejecución especulativa de las instrucciones en los procesos del usuario para acceder a la memoria del kernel, los ataques Spectre aún funcionan

➔ Opciones de Mitigación

Se han propuesto varias contramedidas para los ataques de Spectre. Cada uno aborda una o más de las características en las que se basa el ataque.

◆ Prevención de la ejecución especulativa

Asegurarse de que las instrucciones se ejecutan sólo cuando se determina el flujo de control que las conduce evitaría la ejecución especulativa y, con ello, los ataques de Spectre. Si bien es efectivo como una contramedida, evitar la ejecución especulativa causaría una degradación significativa en el rendimiento del procesador. Aunque los procesadores actuales no parecen tener métodos que permitan que el software desactive la ejecución especulativa, tales modos podrían agregarse en procesadores futuros, o en algunos casos podrían introducirse potencialmente a través de cambios de microcódigo.

Alternativamente, algunos productos de hardware (como los sistemas integrados) podrían cambiar a modelos de procesadores alternativos que no implementan la ejecución especulativa. Aún así, es poco probable que esta solución proporciona una solución inmediata al problema.

Alternativamente, el software podría modificarse para utilizar instrucciones de serialización o bloqueo de especulación que aseguren que las instrucciones que siguen no se ejecuten especulativamente. El enfoque más seguro (pero más lento) para proteger las ramas condicionales sería agregar tal instrucción sobre los dos resultados de cada rama condicional. Sin embargo, esto equivale a deshabilitar la predicción de rama, esto reduciría drásticamente el rendimiento.

- **Prevenir el acceso a datos secretos**

Otras contramedidas pueden evitar que el código ejecutado especulativamente acceda a datos secretos. Una de esas medidas, utilizada por el navegador web Google Chrome, es ejecutar cada sitio web en un proceso separado. Debido a que los ataques de Spectre solo aprovechan los permisos de la víctima.

- **Evitar que los datos ingresen a canales encubiertos**

Los procesadores futuros podrían rastrear si los datos se recuperaron como resultado de una operación especulativa y, de ser así, evitar que esos datos se utilicen en operaciones posteriores que podrían filtrarlos. Sin embargo, los procesadores actuales generalmente no tienen esta capacidad.

- **Limitar la extracción de datos de canales encubiertos**

Los principales proveedores de navegadores han degradado aún más la resolución del temporizador de JavaScript, potencialmente agregando jitter. Estos parches también deshabilitan SharedArrayBuffers, que pueden usarse para crear una fuente de sincronización.

Si bien esta contramedida requeriría un promedio adicional para ataques, el nivel de protección que proporciona no está claro ya que las fuentes de error simplemente reducen la tasa en el que los atacantes pueden filtrar datos. Además, los procesadores actuales carecen de los mecanismos necesarios para la eliminación completa del canal encubierto. Por lo tanto, si bien este enfoque puede disminuir el rendimiento del ataque, no garantiza que los ataques no sean posibles.

- **Prevención del envenenamiento de rama**

Para evitar el envenenamiento indirecto de ramas, Intel y AMD extendieron el ISA con un mecanismo para controlar las ramas indirectas. El mecanismo consta de tres controles:

El primero, la especulación restringida indirecta de rama (IBRS), evita que las ramas indirectas en código privilegiado se vean afectadas por ramas en código menos privilegiado. El procesador entra en un modo IBRS especial, que no está influenciado por ningún cálculo fuera de los modos IBRS.

El segundo, predicción de rama indirecta de subproceso único (STIBP), restringe el intercambio de predicción de rama entre el software que se ejecuta en los subprocesos del mismo puntaje.

Finalmente, la barrera de predicción indirecta de rama (IBPB) evita que el software se ejecute antes de establecer que la barrera afecte la predicción de ramificación mediante el software que se ejecuta después de la barrera, es decir, eliminando el estado BTB. Estos controles se habilitan después de un parche de microcódigo y requieren el sistema operativo o el soporte del BIOS para su uso.

Bibliografía

- Lipp, M. ,Schwarz M.,Gruss D. , Prescher T.,Haas W., Fogh A.,Horn J. ,Mangard S.,Kocher P. , Genkin D., Yarom Y., Hamburg M., Graz University of Technology,Cyberus Technology GmbH, G-Data Advanced Analytics, Google Project Zero, Independent (www.paulkocher.com), University of Michigan, University of Adelaide & Data61, Rambus, Cryptography Research Division. (2018). *Spectre Attacks: Exploiting Speculative Execution.* 29 de Octubre del 2019, de Graz University of Technology Sitio web: https://spectreattack.com/spectre.pdf

- Lipp, M. ,Schwarz M.,Gruss D. , Prescher T.,Haas W., Fogh A.,Horn J. ,Mangard S.,Kocher P. , Genkin D., Yarom Y., Hamburg M., Graz University of Technology,Cyberus Technology GmbH, G-Data Advanced Analytics, Google Project Zero, Independent (www.paulkocher.com), University of Michigan, University of Adelaide & Data61, Rambus, Cryptography Research Division. (2018). *Meltdown: Reading Kernel Memory from User Space*. Recuperado el 28 de octubre de 2019 de: https://meltdownattack.com/meltdown.pdf