Universidad Nacional Autónoma de México

Facultad de Ingeniería

Spectre y Meltdown

Sistemas Operativos (0840)

Profesor: Gunnar Eyal Wolf Iszaevich

Grupo 6 Semestre 2022-1

Integrantes:

Manzanares Peña Jorge Luis

Salazar Domínguez Jesús Eduardo

23 de noviembre de 2021

Índice general

Introducción

Los procesadores modernos tienen un funcionamiento muy diferente al que la mayoría de las personas cree. Ya no son aquellos dispositivos sencillos encargados de seguir instrucciones en un orden específico. En realidad, se han vuelto inteligentes; capaces, incluso, de predecir el futuro (con ciertas limitaciones). Esta habilidad de vaticinar las acciones venideras ha significado un cambio radical en la forma de operar de las computadoras. El rendimiento y la velocidad de ejecución han mejorado enormemente, pues el procesador lleva a cabo sus tareas incluso antes de que se le solicite. La ejecución especulativa, nombre que recibe esta forma de accionar, ha revolucionado al mundo.

Sin embargo, cuando la ejecución especulativa se encontraba en su punto más álgido, todo se vino abajo. El 3 de enero de 2018, las vulnerabilidades Spectre y Meltdown salieron a la luz pública. Como espectros que se habían mantenido invisibles hasta ese momento, ambos problemas aparecieron para sumergir en el terror a toda la industria computacional. Además, como aprenderíamos por las malas, llegaron para quedarse.

Spectre y Meltdown son dos vulnerabilidades que afectan a la mayoría de los procesadores. Estas se aprovechan de la ejecución especulativa, así como de otros conceptos que revisaremos más adelante, para obtener información confidencial del sistema operativo o del usuario víctima. Haciendo alusión a su nombre, derriten las barreras que deberían prevenir las filtraciones. Así, la seguridad de todo tipo de

datos se vio comprometida, desde nombres de usuario o contraseñas hasta información personal.

El propósito de este trabajo es explicar el funcionamiento general de Spectre y Meltdown, de forma que sea posible identificar las similitudes y diferencias de sus accionares. Además, se dará un breve repaso por las características de los procesadores que propiciaron la aparición de estas vulnerabilidades. Para cumplir con este objetivo, dividiremos el contenido en varias secciones y subsecciones. En cada una de ellas se explicará un concepto relacionado con el tema, a fin de que su lectura y comprensión sean lo más sencillas posible.

Segmentación de instrucciones (*Pipelining*)

El software que utilizamos hoy en día es escrito en lenguajes entendibles para los seres humanos, lenguajes de alto nivel, los cuales son compilados (de ser el caso) para generar instrucciones en lenguaje ensamblador. Estas instrucciones pueden ser comunicadas directamente a la Unidad Central de Procesamiento (CPU) para su ejecución. Sería normal pensar que las instrucciones máquina de un programa son ejecutadas de forma secuencial: empezando la siguiente instrucción solo después de haber completado la actual, sin embargo, ese no es el caso en los procesadores modernos.

La segmentación de instrucciones (*pipelining*) logra paralelizar la ejecución de instrucciones aprovechando el hecho de que el proceso de completar una instrucción se suele componer de varias etapas. Esto mejora considerablemente el desempeño de los sistemas de cómputo. Por ejemplo, dos etapas comúnmente presentes en la ejecución de una instrucción son su extracción de la memoria y su decodificación. Es posible que, después de haber leído la primera instrucción de un programa, se comience a decodificarla y, al mismo tiempo que está siendo decodificada, como la etapa anterior está desocupada, se inicie la lectura de la segunda instrucción. Cada que se libera una etapa en el *pipeline* de ejecución, la siguiente instrucción puede utilizarla, logrando incrementar la tasa de ejecución de instrucciones del CPU.

Ejecución fuera de orden

Aunque el *pipelining* implementa paralelismo a nivel del de instrucciones, estas todavía entran al *pipeline* en orden. Si una instrucción permanece en una etapa determinada por un tiempo prolongando el *pipeline* (cuya traducción al español es 'tubería') puede llegar a un estado en el cual varias instrucciones antecedentes estén listas para avanzar, pero la instrucción prolongada atore la tubería. La ejecución fuera de orden es una optimización al *pipelining* que permite aliviar este problema dando paso a instrucciones antecedentes del *pipeline*, en lugar de esperar a que la instrucción prolongada termine su trabajo.

Otra situación que complica la segmentación de instrucciones se da cuando un se debe lidiar con saltos condicionales (*branching*). Un salto condicional ocurre cuando el flujo del programa cambia después de haberse cumplido (o no) cierta condición. En lenguajes de alto nivel, estructuras de itera-

ción y decisión (sentencias *while* e *if*) así como llamadas a funciones e interrupciones causadas por excepciones requieren de saltos condicionales. Naturalmente, no es posible saber con certeza absoluta si un programa dará un salto condicional o no y, consecuentemente, se desconoce qué instrucción debería entrar a continuación al *pipeline*. Entonces, es necesario esperar a qué la instrucción condicional termine de ejecutarse para saber qué instrucción subsecuente se debe extraer de la memoria, contrarrestando por completo el propósito del *pipelining*. El problema es todavía más grave si se considera que los saltos condicionales llegan a ser alrededor del 20 % de las instrucciones totales de un programa. [1]

Ejecución especulativa

Para combatir las inconveniencias provocadas por los saltos condicionales, los procesadores modernos cuentan con una unidad encargada de predecir estos saltos. El predictor de saltos especula acerca de los saltos condicionales del programa. En particular, intenta adivinar si se dará o no un salto condicional que altere el flujo de ejecución del programa y, de así serlo, hacia dónde se dirigirá el programa, es decir, qué instrucción se ejecutará a continuación. Las especulaciones proporcionadas por el predictor de saltos son tomadas en cuenta para meter nuevas instrucciones al *pipeline* y evitar el estancamiento de la ejecución.

Evidentemente, si la especulación es correcta, el desempeño de la ejecución se mejora significativamente y el problema presentado por el salto condicional es superado. De otra manera, los resultados obtenidos en la ejecución paralela de instrucciones especuladas incorrectamente deben ser descartados y revertidos sin que el programa del cual provinieron dichas instrucciones se entere. Los grandes beneficios que pueden obtenerse de la ejecución especulativa han sido motivo de una extensa investigación acerca de la implementación de predictores de saltos y, actualmente, los predictores basados en la historia de ejecución del programa llegan a alcanzar un 95 % de especulación correcta en ciertos programas. [1] No obstante, siempre existe la posibilidad de obtener especulaciones incorrectas.

Ataques a la memoria caché

El caché es una memoria pequeña y rápida ubicada dentro del chip del procesador, cuyo objetivo es almacenar datos solicitados frecuentemente por el procesador durante la ejecución de un programa; fue concebida como resultado de las pérdidas de desempeño causadas por la relativa lentitud de respuesta de la memoria principal ante solicitudes del CPU. Existen diferentes algoritmos, políticas y mecanismos para determinar qué almacenar dentro de la memoria caché y cuándo, sin embargo, el funcionamiento termina siendo el mismo. Cuando un programa requiere extraer un dato de la memoria el procesador revisa primero el caché, en caso de que el dato se encuentre ahí (*caché hit*) el dato se recupera rápidamente y se evita la lectura a memoria principal. En el caso contrario (*caché miss*), de acuerdo a la implementación de caché, se hace espacio para almacenar el dato en el caché.

Una característica importante de la memoria caché presente en los sistema de cómputo modernos es que se divide en niveles, los cuales se asocian a direcciones de memoria física. [1]

Por sus características, la memoria caché a menudo se vuelve un canal lateral de acceso. Un canal lateral es una ruta de acceso accidental (no planeada) a un sistema, usualmente toma ventaja de algún recurso compartido. Aunque no es posible acceder directamente a sus contenidos, la organización detallada de la memoria caché hace posible medir y distinguir diferentes tiempos de acceso a datos. Con ello, un atacante puede obtener información acerca de las direcciones de memoria de la víctima.

Una método de ataque a la memoria caché que ha sido utilizado para para comprometer algoritmo criptográficos, llamadas a funciones en servidores web, entradas de usuaria e información acerca de direcciones del kernel es Flush+Reload. [5] Este consiste en remover algún dato del caché (usando una instrucción como *clflush* en x86). La siguiente vez que el usuario necesite usar este dato, será necesario volver a recuperarlo de la memoria principal. El atacante puede medir cuánto tiempo tomó para acceder al dato y, con base en ello, determinar de qué conjunto de cachés fueron accedidos por la víctima.

Meltdown

Los ataques de tipo Meltdown se aprovechan de la especulación presente en la mayoría de los procesadores, especialmente en aquellos con arquitectura x86. La naturaleza de este mecanismo provoca que la mayoría de las operaciones, incluida la lectura de memoria, se lleven a cabo antes de verificar si el programa solicitante cuenta con los permisos para acceder a dicha localidad [1]. Antes del descubrimiento de estas vulnerabilidades, se creía que este tipo de especulación era inofensiva, puesto que los datos recuperados no se entregaban a la aplicación hasta que se comprobaban los permisos. Sin embargo, como veremos a continuación, este planteamiento es incorrecto.

Meltdown utiliza un canal lateral, basado en el tiempo que toma realizar la lectura de memoria, para recuperar la información que ya ha sido leída por el procesador. Siguiendo este principio, un ataque Meltdown es capaz de acceder a cualquier localidad mapeada en el espacio de memoria del proceso, independientemente de los permisos que este tenga. Así, datos referentes al usuario o al sistema operativo pueden verse comprometidos, incluso si el procesador lanza una excepción a causa de los permisos posteriormente.

Meltdown puede dividirse en tres pasos principales [5]. El primero de ellos consiste en cargar datos desde la memoria principal de la víctima, originalmente inaccesible para el atacante, hasta la caché. Para conseguir esto, el programa hace referencia a la localidad que le interesa mediante una dirección de memoria virtual. Mientras el procesador traduce esta referencia a una localidad física, también debería revisar los permisos de la aplicación que realizó la solicitud. Sin embargo, como hemos establecido previamente, Meltdown se aprovecha de la especulación y ejecuta acciones en el breve espacio existente entre la lectura de la información y el lanzamiento de la excepción.

El segundo paso consiste en transmitir la información "secreta". Para conseguir esto, se debe ejecu-

tar una instrucción transitoria, o sea un instrucción que no debería realizarse, debido al lanzamiento de una excepción, pero que se lleva a cabo de forma especulativa. Cuando esta se ejecuta, sus resultados no serán entregados formalmente al programa, pero sí serán almacenados en la caché. Como consecuencia, la instrucción transitoria dejará rastros medibles que permitirán identificar en cuál página se guardó la información. En otras palabras, la instrucción transitoria altera la microarquitectura de la caché, provocando que el canal lateral pueda recuperar los datos leídos, incluso cuando estos nunca se comunicaron directamente.

Finalmente, lo único que resta es recibir el secreto. Para ello, el canal lateral se aprovecha de la modificación microestructural que llevó a cabo la instrucción transitoria del paso anterior. El atacante itera sobre cada una de las páginas candidatas a contener la información deseada, midiendo los tiempos de acceso para la primera linea de caché en la página. Un tiempo mucho menor al promedio revelará que esa página fue accedida durante la instrucción transitoria, por lo que contiene el secreto.

Spectre

Similar a Meltdown, Spectre es un ataque que se aprovecha de la especulación de los procesadores, específicamente de los sistemas predictores de saltos. Esta clase de sistema se compone de tres partes principales: el predictor de la dirección de saltos, encargado de vaticinar si un salto condicional será tomada o no; el predictor del objetivo de saltos, cuyo objetivo es determinar cuál será la localidad de memoria objetivo de las saltos condicionales indirectos, y el buffer de retorno, diseñado con el fin de predecir el punto donde debería continuar el programa tras encontrar una instrucción return [1].

El objetivo de los ataques Spectre es provocar que la víctima ejecute operaciones que no se llevarían a cabo si se siguiera un orden de realización secuencial. Estas acciones, llevadas a cabo de forma especulativa, llevarán a la computadora vulnerable a filtrar información [4]. A continuación, el atacante empleará un canal lateral para recuperar los datos expuestos, tal como se explicó para Meltdown.

Existen múltiples variantes de Spectre. Cada una de ellas cuenta con particularidades únicas y se aprovecha de alguna de las partes del sistema predictor de saltos de forma distinta. No obstante, la mayoría de estos ataques siguen tres fases bien definidas [4]. La primera, conocida como etapa de preparación o configuración, se caracteriza porque el atacante entrena al sistema víctima a su conveniencia. Este falso entrenamiento provocará predicciones especulativas erróneas. En adición a esto, el agresor puede manipular al procesador para que elimine ciertos datos de la caché e ir preparando el canal lateral para recuperar los datos que obtenga.

En la segunda fase ocurre la ejecución especulativa errónea. Spectre se aprovecha de otro proceso y del entrenamiento previo, de forma que se recuperan los datos de cierta localidad de memoria, a la cual el proceso víctima tiene acceso, y se transfieren a la caché. Si bien esto se realizará únicamente de forma especulativa, el atacante podrá recuperar los datos mediante el canal lateral, lo cual constituye la tercera y última fase.

Mediante este proceso, Spectre es capaz de engañar a otras aplicaciones para que accedan a una

localidad de memoria arbitraria. Esto representa la mayor diferencia entre Meltdown y Spectre, pues mientras el primero es capaz de traspasar permisos y acceder a datos que normalmente estarían restringidos al propio sistema operativo, el segundo se limita a aprovechar la especulación para leer localidades de memoria a las que la aplicación víctima tiene acceso por default. No obstante, esto no hace que Spectre sea menos peligroso, puesto que es más difícil de prevenir. Incluso si el procesador evita la lectura especulativa de localidades fuera del alcance del programa, Spectre continuará existiendo.

Consecuencias y Soluciones

Meltdown y Spectre representaron un duro golpe para la industria de la computación. Por sus naturalezas, estos ataques hicieron que prácticamente todos los sistemas fueran susceptibles a ellos, desde los celulares inteligentes, pasando por las computadoras personales y llegando hasta la nube. El desafío generado para la seguridad informática fue especialmente duro porque las vulnerabilidades se originan en el hardware. Si bien el problema fue especialmente grave para los procesadores con arquitectura x86 de Intel, también se detectaron variantes de Spectre en ciertos procesadores AMD y ARM.

Las primeras soluciones para el problema fueron a nivel de software. Ciertas actualizaciones, lanzadas por los propios sistemas operativos, tuvieron como objetivo limitar el alcance de los ataques. Sin embargo, estas eran respuestas incompletas, puesto que no eliminaban por completo la posibilidad de que el dispositivo fuera afectado por Meltdown o Spectre. Además, estos parches significaron una reducción considerable de rendimiento. Recordemos que la especulación se implementó para ahorrar tiempo, por lo que, al limitarla, se redujo la velocidad de los ordenadores.

En general, las opciones de mitigación de estas vulnerabilidades intentan prevenir uno o varios de los siguientes factores de riesgo [4]:

- La ejecución especulativa.
- El acceso a información confidencial.
- La recuperación de datos por canales laterales.
- La manipulación de ramas y predicciones.

El aislamiento de tablas de páginas del núcleo (KPTI, por su siglas en inglés) fue una de las alternativas utilizadas para prevenir Meltdown. Fundamentada en KAISER, KPTI fue una característica añadida al núcleo de Linux tras el descubrimiento de estos ataques. Esta defensa se basa en el concepto de memoria virtual. Es importante recordar que cada aplicación observa la memoria de la computadora como una serie de localidades contiguas. No obstante, esto no corresponde con la memoria física, puesto que esta última se comparte entre los diferentes procesos y se distribuye de forma distinta. Para

poder traducir las direcciones de memoria virtual a memoria física, el sistema operativo, a través de la unidad de manejo de memoria (MMU), emplea una tabla de páginas. Lo que KPTI hace es eliminar de la tabla aquellas páginas que corresponden a información del sistema operativo, u otros datos confidenciales. Este descarte se mantiene activo mientras se ejecuta un programa del usuario. Una vez que este termina, las páginas secretas vuelven a incluirse en la tabla. Como se puede esperar, esta solución es muy costosa para el sistema operativo, ya que requiere de constantes modificaciones de la tabla de páginas.

El problema también fue abordado con premura por los fabricantes de procesadores. Tanto Intel como AMD se vieron forzados a modificar su microcódigo. Este término hace referencia a una capa de instrucciones ubicada entre el lenguaje ensamblador y los circuitos digitales propios del procesador. La alteración del microcódigo tuvo como objetivo limitar el uso de la especulación, lo cual implicó cambiar el funcionamiento de ciertas operaciones del ensamblador [1].

Conclusiones

Spectre y Meltdown se distinguen de la gran mayoría de las fallas de seguridad en el mundo del cómputo por ser vulnerabilidades presentes en el hardware de la computadora, particularmente en el CPU. Las condiciones que hicieron posible su existencia surgieron del producto de años y décadas de innovaciones en tecnología de procesadores. Desde tecnologías tan reconocidas y ubicuas como las memorias caché y la segmentación de instrucciones hasta otras más especializadas como la ejecución especulativa, las cuales se investigaron e implementaron con el objetivo de mejorar el desempeño de nuestras computadoras, crearon el ambiente exacto para dar lugar a estas vulnerabilidades. Una serie de interacciones previamente no consideradas entre todos estos mecanismos, así como las particularidades del manejo de memoria de los sistemas operativos modernos, comprometieron prácticamente todos los sistemas de cómputo del mundo durante al menos dos décadas.

Aunque no se sabe a ciencia cierta que tanto hayan sido utilizadas (si acaso) estas vulnerabilidades para llevar a cabo ataques maliciosos, su impacto ha sido sentido en el desempeño de todas las computadoras del mundo. Además, han demostrado que la seguridad informática no es solo una preocupación de los desarrolladores de software. De ahora en adelante, los diseñadores de CPU's se enfrentan al desafío de recuperar la pérdidas de desempeño sacrificadas en la mitigación de estas vulnerabilidades, cuidando, al mismo tiempo, que algo como esto no vuelva a ocurrir.

Bibliografía

[1] Nael Abu-Ghazaleh, Dmitry Ponomarev y Dmitry Evtyushkin. "How the spectre and meltdown hacks really worked". En: *IEEE Spectrum* 56.3 (2019), págs. 42-49. DOI: 10.1109/MSPEC.2019.8651934.

[2] Benedict Herzog y col. "The Price of Meltdown and Spectre: Energy Overhead of Mitigations at Operating System Level". En: *Proceedings of the 14th European Workshop on Systems Security*. EuroSec '21. Online, United Kingdom: Association for Computing Machinery, 2021, págs. 8-14. ISBN: 9781450383370. DOI: 10.1145/3447852.3458721. URL: https://doi.org/10.1145/3447852.3458721.

[3] Mark D. Hill y col. "On the Spectre and Meltdown Processor Security Vulnerabilities". En: *IEEE Micro* 39.2 (2019), págs. 9-19. DOI: 10.1109/MM.2019.2897677.

[4] Paul Kocher y col. "Spectre Attacks: Exploiting Speculative Execution". En: *40th IEEE Symposium on Security and Privacy (S&P'19)*. 2019.

[5] Moritz Lipp y col. "Meltdown: Reading Kernel Memory from User Space". En: *27th USENIX Security Symposium (USENIX Security 18)*. 2018.

[6] Andrew Prout y col. "Measuring the Impact of Spectre and Meltdown". En: *2018 IEEE High Performance extreme Computing Conference (HPEC)*. 2018, págs. 1-5. DOI: 10.1109/HPEC.2018.8547554.