Vulnerabilidades de las arquitecturas: exploits más peligrosos y protección de procesadores

Helen Elizabeth Chen, Diego Maximiliano Medina, Gian Luca Spagnolo,
Estudiantes de la Facultad de Ingeniería, Universidad de Buenos Aires

Abstract—La arquitectura de los procesadores modernos evoluciona a lo largo de las generaciones, donde su protección es sumamente importante para la seguridad informática. En este informe, se discuten las vulnerabilidades más peligrosas presentes en las arquitecturas, las cuales permiten la realización de exploits que ponen en peligro la privacidad de datos del usuario. Asimismo, se busca analizar dos vulnerabilidades en específico: Spectre y Meltdown, las cuales dependen de errores en la ejecución producto de diferentes técnicas de optimización utilizadas por los CPUs, que afectan a casi todos los dispositivos modernos. Además, se desarrollan sus variantes y otros tipos de vulnerabilidades presentes, y se destacan áreas de investigación en curso para brindar información con respecto a la formas de protección y mitigación de los procesadores ante estos tipos de ataques.

Index Terms—Ejecución fuera de orden, Ejecución especulativa, Ejecución transitoria, Predictor de saltos, Canales laterales, Vulnerabilidades, Spectre, Meltdown, Variantes, Protección.

1 INTRODUCCIÓN

DURANTE las últimas décadas, las principales empresas productoras de procesadores han aplicado metodologías de optimización para desarrollar dispositivos cada vez más rápidos. Una de estas metodologías fue la adaptación de la arquitectura de los CPUs al pipeline, lo que permite ejecutar instrucciones en paralelo, dividiendo procesos en etapas independientes para garantizar una ejecución de múltiples objetos en todo momento [6].

En cada ciclo de instrucción, al momento de realizar la ejecución de esta, se aplican distintas técnicas de optimización: la ejecución fuera de orden y la ejecución especulativa. Estos tipos de ejecuciones, sin embargo, son capaces de producir errores que pueden vulnerar información privada del usuario, donde numerosas vulnerabilidades se han detectado explotando estos agujeros de seguridad.

El objetivo de este documento es analizar el origen, funcionamiento y alcance de dos de las vulnerabilidades más importantes descubiertas en estos últimos años: Spectre y Meltdown; detallar cómo han evolucionado a lo largo de este último tiempo destacando sus variantes más peligrosas; informar sobre el funcionamiento de los exploits más amenazantes de dichos agujeros de seguridad; y explicar cómo los investigadores intentan contrarrestar y mitigar los exploits de estas vulnerabilidades en los procesadores, proveyendo información sobre las áreas de investigación actuales para la detección y protección de estos agujeros de seguridad.

En la sección 2 del presente informe, se introducen conceptos fundamentales para la comprensión y el desarrollo del funcionamiento de las vulnerabilidades detalladas en la sección 3. A su vez, se otorga especial énfasis en Spectre, Meltdown y sus variantes, para la posterior explicación en la sección 4 de las diversas medidas y técnicas de protección de los procesadores frente a estos agujeros de seguridad.

Chen, Helen Elizabeth, Ingeniería Informática, Universidad de Buenos Aires (mail: hchen@fi.uba.ar).
Medina, Diego Maximiliano, Ingeniería Informática, Universidad de Buenos Aires (mail: dmedina@fi.uba.ar).
Spagnolo, Gian Luca, Ingeniería Informática, Universidad de Buenos Aires (mail: gspagnolo@fi.uba.ar).

2 BACKGROUND

2.1 Técnicas de optimización

La **ejecución fuera de orden** [1], [2], [4], [5] (o ejecución dinámica) es una técnica de optimización que permite la maximización del uso de los componentes del procesador, procesando las instrucciones en un orden definido por la disponibilidad de los recursos que necesita. El orden de ejecución es determinado por un algoritmo de planificación dinámica [16] (desarrollado por Robert Tomasulo en 1967) que garantiza una correcta y eficiente ejecución de múltiples unidades de ejecución.

La **ejecución especulativa** [1], [4], [5] es otra técnica de optimización que ocurre cuando la ejecución fuera de orden alcanza una instrucción condicional, donde la dirección depende de otra instrucción cuya ejecución no se ha completado aún. En este caso, el procesador realiza una predicción en base al estado actual de los registros y corre especulativamente las instrucciones en aquel camino, continuando así la ejecución en base a un comportamiento futuro esperado.

Aquellas instrucciones ejecutadas especulativamente forman parte de la **ejecución transitoria** [10], las cuales son capaces de alterar el estado de elementos de la microarquitectura, quedando registradas en forma de trazas.

La predicción de la ejecución especulativa es determinada por el **predictor de saltos** (o branch predictor) [1] el cual cuenta con múltiples mecanismos de predicción para saltos directos o indirectos, lo que permite predecir un salto condicional en base al comportamiento de la unidad de ejecución en saltos anteriores, con el objetivo de optimizar el rendimiento del dispositivo.

Por otro lado, la **verificación de límites** (o bounds checks) [1] es un método de la ejecución especulativa para comprobar si efectivamente una variable (como una

dirección) se encuentra dentro de los límites permitidos (referencia en fig. 2).

2.2 Canales laterales

Los **canales laterales** [1], [9] representan vías secundarias a través de las cuales es posible la obtención de información filtrada, explotando variaciones temporales; o a través del nivel L1 de la caché, monitoreando accesos a direcciones de memoria con información privada de la víctima.

Los **ataques de canales laterales** de la caché explotan las diferencias temporales introducidas por las caches. Una de estas técnicas de ataque es `Flush + Reload` [1], [2] donde un atacante libera frecuentemente una parte de la memoria en específico utilizando la instrucción `clflush`, para luego medir el tiempo que le toma recargar aquel dato y así poder identificar si este fue cargado por otro proceso.

Un caso especial de un ataque de canal lateral es un **canal encubierto** (o covert channels) [2], donde el atacante ejerce control sobre tanto la inducción del efecto secundario como la medición de este, con el objetivo de filtrar información sobrepasando los límites existentes en el nivel de la microarquitectura o superiores.

3 PRINCIPALES VULNERABILIDADES: SPECTRE Y MELTDOWN

Numerosas vulnerabilidades presentes en la microarquitectura de los procesadores han surgido por errores producto de la ejecución transitoria de instrucciones (aprovechando los errores de la ejecución especulativa o la ejecución fuera de orden) donde buscan ejecutar código que permita leer sin permiso parte de la memoria y dejar vulnerada dicha información para poder ser accedida vía un ataque de canal lateral [8].

Las más importantes son Spectre y Meltdown. Ambas fueron descubiertas por numerosos grupos de investigadores aproximadamente el primero de junio de 2017, pero fueron reveladas al público en enero de 2018. Y, al tratarse de agujeros de seguridad en la arquitectura de los CPUs, se vuelven extremadamente difíciles de contrarrestar definitivamente.

3.1 Spectre

Este agujero de seguridad aprovecha los errores producidos durante la ejecución especulativa [1], permitiendo realizar operaciones que no deberían ocurrir durante un correcto funcionamiento del programa y garantizando acceso a la memoria y los registros de cualquier programa de la víctima, con el objetivo de filtrar información confidencial vía un canal lateral del atacante.

El objetivo de este ataque es manipular el estado del predictor de saltos de la víctima, induciendo a información aparentemente inocente para luego ejecutar una instrucción garantizando un error en la predicción de la ejecución especulativa [8].

El ataque se segmenta en diferentes fases [1] (como se puede visualizar en la fig. 1): primero, la manipulación del predictor de saltos (como se ha mencionado anteriormente). Segundo, la ejecución de aquellas instrucciones maliciosas que, bajo el error de la ejecución especulativa, puede filtrar información confidencial de la víctima de ciertas direcciones de memoria enviándola a la caché. Por último, aquellos datos vulnerados pueden ser identificados y accedidos vía un ataque de canal lateral, como `Flush + Reload`, que permite identificar el dato presente en la memoria caché a través de accesos a sus líneas en un tiempo monitoreado.

Esta vulnerabilidad afecta a aquellos procesadores cuyas arquitecturas soporten la ejecución especulativa, como Intel, ARM y AMD; y ha generado la aparición de múltiples variantes [1], [3] de funcionamiento similar que explotan diferentes errores hasta hoy en día. A continuación se va a presentar y describir el funcionamiento de dos de las variantes de Spectre más peligrosas.

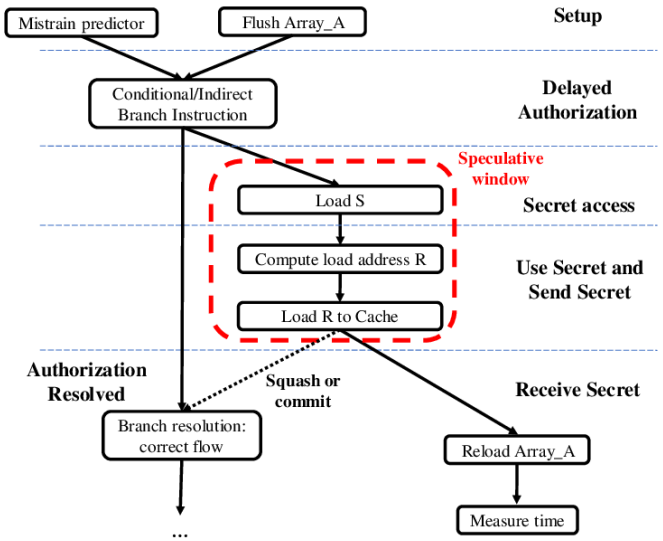

Fig. 1. Secuencia de ejecución especulativa explotada por Spectre

3.1.1 *Spectre-PHT: Bounds Check Bypass (CVE-2017-5753)*

Este tipo de variante de Spectre (variante 1, omisión de la verificación de límites) [3] es capaz de leer memoria arbitraria de otro proceso, explotando errores del predictor de saltos.

Esto es posible debido a que, en caso de disponer de una dirección de memoria inválida (y luego de verificar que esta no se encuentra en su línea correspondiente de la caché) se intenta acceder a ella luego de ejecutar una instrucción condicional. En ese escenario, la ejecución especulativa mediante el predictor de saltos (como se muestra en la fig. 2) intenta predecir la validez de esta dirección. En caso de acertar, se reducen drásticamente los tiempos de ejecución; pero, en caso de fallar en la predicción y asumir que aquella dirección es válida (cuando no lo es) permite la ejecución de aquella instrucción y, consecuentemente, envia aquel dato hacia la memoria caché, para luego poder identificarlo y extraerlo mediante un ataque de canal lateral [1].

Como consecuencia, un atacante puede explotar los errores de la ejecución especulativa mediante diferentes escenarios utilizando esta variante [1]. Por ejemplo, en vez de realizar una verificación de límites, el error en la predicción de saltos puede producirse debido a una verificación incorrecta de un valor computado anteriormente; o, de manera

similar, código ejecutado especulativamente puede filtrar el resultado de una comparación en una dirección de memoria determinada.

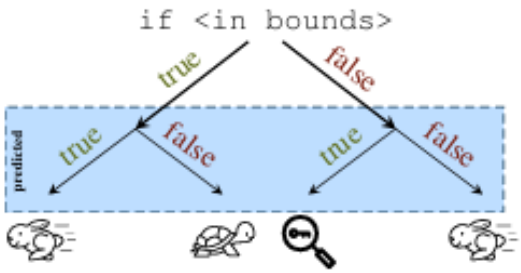

Fig. 2. Ejemplificación de las posibilidades de la verificación de límites, utilizado por el predictor de saltos para confirmar si hubo un error en su predicción [1]

3.1.2 Spectre-BTB: Branch Target Injection (CVE-2017-5715)

Esta variante diferente de Spectre (variante 2, inyección de destino de salto) [3] consiste en un caso particular que, bajo específicas condiciones en las fases del ataque, permite alterar el estado del predictor de saltos en base a información maliciosa, determinada por el código ejecutado previamente, con el objetivo de localizar un fragmento de código colocado por el atacante para transferir la información privada de la víctima vía un canal encubierto [1], [8].

En este caso (y tal como está dibujado en la fig. 3), el atacante es capaz de condicionar los saltos condicionales con el objetivo de ejecutar especulativamente instrucciones manipulando el predictor de saltos a su favor, orientando la predicción hacia una dirección objetivo del atacante para poder vulnerar el dato [1].

Debido a su funcionamiento, numerosos ataques son posibles en base al estado de la arquitectura del procesador que controla el atacante, aprovechando esta vulnerabilidad. Por ejemplo, una función criptográfica que devuelve un valor secreto en un registro puede ser explotada si el atacante manipula la ejecución especulativa con el objetivo de llevar el contenido de aquel registro a una dirección de memoria [1].

Este exploit es altamente peligroso ya que pone en peligro los programas del usuario frente a ataques de diferentes programas. Dicha amenaza también afecta a los proveedores de los servicios de nubes ya que no pueden garantizar que los datos de sus clientes estén protegidos correctamente [8].

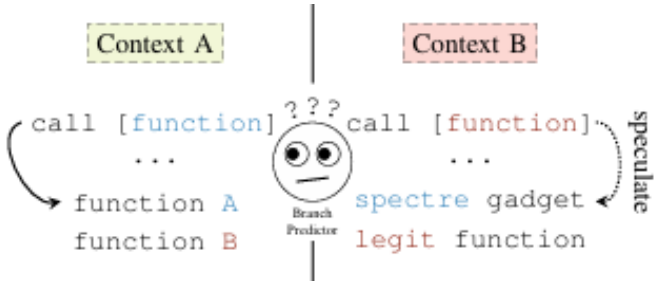

Fig. 3. Ejemplificación de cómo el predictor de saltos puede ser manipulado para ejecutar instrucciones en un contexto incorrecto [1]

3.2 Meltdown

Esta vulnerabilidad presente en los procesadores, a diferencia de Spectre, explota los errores de la ejecución fuera de orden con el objetivo de filtrar memoria del kernel. Las dos bases de esta vulnerabilidad son: la ejecución de instrucciones transitorias y la capacidad de conformar un canal encubierto.

Meltdown no requiere el predictor de saltos, sino que (tal cual como está diagramado en la fig. 4) se basa en la ejecución fuera de orden de instrucciones luego de que una de estas haya elevado una excepción. También, Meltdown explota la ejecución transitoria de ciertas instrucciones con el objetivo de sobrepasar la protección de la memoria. Combinando estos problemas [1], [2], este agujero de seguridad puede acceder a la memoria del kernel desde el espacio del usuario, y puede filtrar el contenido de la memoria accedida a través de un canal encubierto de la caché.

Este ataque se segmenta en tres partes diferentes: primero, el contenido de una dirección de memoria elegida por el atacante (la cual normalmente se encuentra protegida) es cargada a un registro; luego, la ejecución transitoria accede a una línea de la caché basándose en el contenido de aquel registro, cargando temporalmente la información a la caché; y por último, se realiza un ataque de canal lateral de la caché (`Flush + Reload` por ejemplo) con el objetivo de determinar la línea de la caché accedida y, con eso, la información de aquella dirección de memoria seleccionada en el principio [2], [7], [8].

Mientras que los tiempos de extracción de información son significativamente mejores si los datos se encuentran en la caché de antemano, Meltdown ha demostrado ser muy efectivo para poder extraer información desde la memoria [10].

Este agujero de seguridad afecta a los microprocesadores de Intel x86, IBM Power y algunos basados en ARM (usados por Apple). Y, al igual que Spectre, hay numerosas variantes de esta misma vulnerabilidad que continúan afectando hasta el día de hoy. A continuación, se detallarán tres variantes importantes de Meltdown.

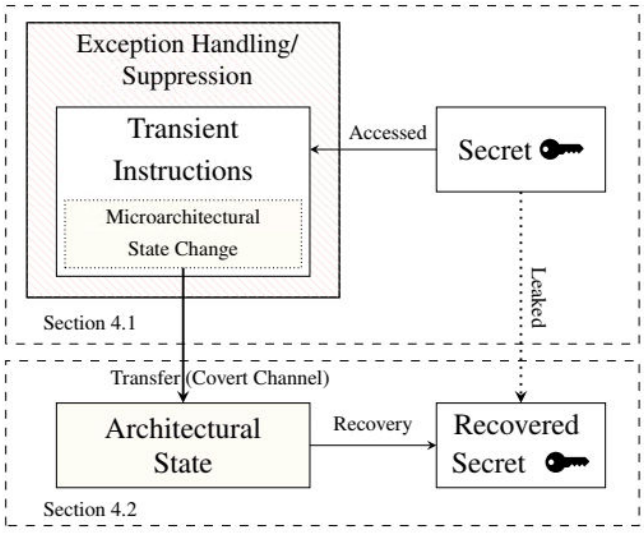

Fig. 4. Visualización del manejo o supresión de las excepciones por parte de Meltdown, con el objetivo de correr instrucciones transitorias

3.2.1 *Meltdown-US: Supervisor-only Bypass (CVE-2017-5754)*

Los procesadores modernos poseen en sus tablas de paginación atributos de "usuario/supervisor" para definir si una página de memoria pertenece al kernel del sistema operativo [10].

Teniendo en cuenta esto, esta variante de Meltdown (Omisión solo para supervisores, que fue la primera variante hecha pública, y también es conocida como variante 3 / Rogue Data Cache Load) [3], [10] busca leer la memoria del kernel desde el espacio del usuario en el CPU que no fuerce transitoriamente este atributo de usuario/supervisor, con el objetivo de desreferenciar una dirección del kernel sin autorización, generando un page fault.

Sin embargo, el atacante puede ejecutar una secuencia de instrucciones transitorias antes de que aquel page fault se vuelva visible, para poder acceder a una línea de la caché basándose en el privilegio de la información leída al ejecutar dicha instrucción. Al final, todo esto permite reconstruir la información filtrada por un canal encubierto (como `Flush + Reload`) una vez se eleva la excepción.

De esta forma, un atacante es capaz de tener acceso al kernel en su totalidad si se itera byte por byte sobre su espacio y se suprimen o manejan las excepciones que son arrojadas. Y, en caso de que el sistema operativo disponga de un mapa directo físico en el kernel, un atacante también puede tener completo acceso a la memoria física [10].

3.2.2 *Meltdown-GP / Meltdown-CPL-REG: System Register Bypass (CVE-2018-3640)*

Esta variante (Omisión de registros del sistema, también denominada como variante 3-a / Rogue System Register Read) le permite al atacante leer registros privilegiados del sistema explotando la ejecución transitoria luego de ejecutar instrucciones que elevan un "general protection fault", debido a no tener acceso a dichos registros. Sin embargo, instrucciones transitorias pueden computar los datos sin autorización igualmente, filtrando el contenido de los registros del sistema a través de un canal encubierto de la microarquitectura (`Flush + Reload`) [10].

En la figura 5 se puede observar de manera simplificada como un atacante, ejecutando instrucciones sin privilegio, es capaz de filtrar y decodificar contenido de registros privilegiados (como contadores de rendimiento) mediante esta vulnerabilidad.

Fig. 5. Visualización de la filtración de registros de Meltdown-GP (Meltdown 3a) [7]

En un análisis más sistemático [7], esta vulnerabilidad no afecta a los mismos registros privilegiados de la misma forma que otras variantes, por lo que no necesariamente todos los registros del sistema pueden verse afectados. Por otro lado, esto también quiere decir que aquellos procesadores completamente protegidos frente a los ataques originales de Meltdown pueden seguir siendo afectados por Meltdown-CPL-REG.

3.2.3 *Meltdown-P / L1FT / Foreshadow: Virtual Translation Bypass (CVE-2018-3615)*

Foreshadow (también considerada como Omisión de traducción virtual) es una vulnerabilidad de la ejecución transitoria de instrucciones que ataca los "enclaves SGX" propios de algunos procesadores de Intel, los cuales son sets de instrucciones implementadas para encriptar una parte de la memoria [3]. Si bien pueden prevenir numerosos tipos de ataques, al ser operadas en un contexto desconfiado, se vuelven vulnerables a los ataques de canales laterales.

Este tipo de ataque [11] busca extraer información del nivel L1 de la memoria caché, explotando la ejecución especulativa al intentar acceder a una parte de la memoria encriptada por estos enclaves, utilizando información sensible de dicho nivel de la caché antes de que el procesador note la falta de permisos y eleve un terminal fault. Por esto, Intel se ha referido a esta vulnerabilidad como ataques de "L1 Terminal Fault".

De esta forma, debido a la pobre conexión entre la caché L1 y la traducción de direcciones en las microarquitecturas modernas, los bits de una entrada de la tabla de paginación de la memoria que sufre un terminal fault pueden ser enviados a la caché L1. De modo que, cualquier acceso a una dirección física que cae en la caché L1 puede ser ejecutado transitoriamente, sin tener en cuenta los permisos de accesos [10].

También existen diferentes variantes de este agujero de seguridad [10], como Foreshadow-NG o Foreshadow-VMM, que amenazan de formas diferentes con el mismo objetivo de provocar un terminal fault en la caché L1.

3.3 Diferentes vulnerabilidades, exploits y áreas de investigación

Como las variantes que han sido mencionadas anteriormente, existe una gran cantidad de vulnerabilidades de ejecución transitoria que amenazan de manera similar a Spectre o Meltdown, las cuales pueden presentar una gran cantidad de variantes diferentes [3], [10] y tomar un enfoque único dentro de los errores presentes en la microarquitectura de los procesadores. Una de estas vulnerabilidades se denomina Microarchitectural Data Sampling (o muestreo de datos de la microarquitectura) [12], [13] que produjo la aparición de ataques como ZombieLoad, Rogue In-Flight Data Load, Fallout, CacheOut, los cuales también se comportan como ataques en la ejecución transitoria de instrucciones.

Un exploit diferente que aprovecha en cierta medida la vulnerabilidad Meltdown es Load Value Injection (CVE-2020-0551) [14] que permite inyectar datos, y es resistente a la mayoría de sus mitigaciones. Este agujero de seguridad es

capaz de secuestrar el resultado de una operación de carga a memoria confiable, y forzar a la víctima a realizar una ejecución transitoria para terminar cargando datos indeseados.

El funcionamiento de este ataque es el siguiente: primero se prepara el estado adecuado de la microarquitectura con un dato indeseado, para que este sea cargado erróneamente en caso de un page fault, mediante una ejecución transitoria de estas instrucciones de cargado. En ese entonces, la información de la víctima puede verse vulnerada debido a cambios del estado de la microarquitectura, siendo subsecuentemente accedida por análisis de canales laterales [14].

En la actualidad, numerosos grupos de investigadores y empresas continúan derivando recursos hacia la investigación para la detección de nuevas vulnerabilidades presentes en la arquitectura de los procesadores, pero también para encontrar variantes y exploits nuevos de aquellos agujeros de seguridad que continúan amenazando hasta hoy en día [15].

4 PROTECCIÓN DE LOS PROCESADORES

La seguridad de software fundamentalmente depende de un entendimiento claro entre los desarrolladores de hardware y software, para poder definir qué información de la CPU puede exponerse en las computaciones [1].

Debido al constante avance en la optimización de la ejecución de los procesadores, distintas técnicas de optimización que manipulan la ejecución de instrucciones (que bajo ciertas condiciones pueden resultar innecesarias) son capaces de poner en peligro la información del usuario.

Sin embargo, al tratarse de problemas en la arquitectura de los procesadores, no existe una solución definitiva que pueda erradicar estas vulnerabilidades, ya que la mayoría de estas proviene en formas de parches de microcódigo y actualizaciones de software (como se puede visualizar en la fig. 6), cosas que no son capaces de solucionar el problema en la microarquitectura de raíz, por lo que existe la garantía de que estos problemas persistirán. Mientras tanto, se desarrollan nuevas arquitecturas de los procesadores que introducen estructuras al CPU dedicadas a la especulación y su separación de la memoria caché del procesador y otros componentes del hardware, con el objetivo de aislar la ejecución especulativa para resguardarse en caso de haber un error en la predicción [8].

A continuación se van a detallar las más útiles medidas para contrarrestar tanto Spectre como Meltdown, como también para incrementar el nivel de seguridad de los procesadores y así evitar filtración de información.

Speculative side channel variant	Coffee Lake-S Refresh: 9th Gen Intel® Core™ Desktop Processors	Basin Falls Refresh: Intel® Core™ X-Series Desktop Processors	Skylake-X: Intel® Xeon® W-3175X Processor
Spectre V2 (Branch Target Injection)	Microcode + Software	Microcode + Software	Microcode + Software
Meltdown V3 (Rogue Data Cache Load)	Hardware	Microcode	Microcode
Meltdown V3a (Rogue System Register Read)	Microcode	Microcode	Microcode
V4 (Speculative Store Bypass)	Microcode + Software	Microcode + Software	Microcode + Software
L1 Terminal Fault	Hardware	Microcode + Software	Microcode + Software

Fig. 6. Primeras actualizaciones de seguridad de Intel en sus nuevos procesadores para enfrentar determinadas variantes de vulnerabilidades, en 2018

4.1 Defensas para mitigar Spectre

Spectre ha permanecido amenazando durante cinco años los procesadores modernos. Y, ante la ausencia de una solución definitiva, se toman en consideración diferentes técnicas para intentar mitigar este agujero de seguridad [1], [10], [17].

Una posibilidad es deshabilitar la ejecución especulativa, modificando el software utilizando instrucciones de bloqueo de especulación (o de serialización) que garantizan una ejecución completa de las instrucciones de manera secuencial (como por ejemplo la instrucción `lfence` recomendada por Intel y AMD) aplicándolas al inicio y al final de la condición, eliminando la especulación y también mitigando la manipulación de saltos.

Otra posibilidad también busca prevenir que los datos secretos sean accedidos por la ejecución especulativa. Debido a que la efectividad de Spectre depende de los niveles de acceso concedidos, se puede buscar correr cada proceso en un entorno de ejecución independiente denominado sandbox [18], limitando el acceso a recursos del sistema operativo, y obstaculizando el acceso de un proceso malicioso a datos de otro proceso.

En el caso de la variante 1 de Spectre se maneja la posibilidad de impedir que el hardware ejecute instrucciones de carga especulativamente en base a instrucciones anteriores, evitando una posible instrucción transitoria errónea de cargado de un dato malicioso. En el caso de la variante 2, se busca entrenar el predictor de saltos para evitar que sea vulnerable a una manipulación, y deshabilitando la predicción de saltos cuando un proceso solicite servicios del sistema operativo, o de información a la cual no tiene acceso.

Otros métodos de protección buscan evitar el ingreso de información a los canales encubiertos, mitigando la precisión de estos y también limitando la extracción de estos datos. Sin embargo, estas propuestas sacrifican un gran porcentaje del rendimiento de los dispositivos.

4.2 Defensas para mitigar Meltdown

Meltdown es una vulnerabilidad presente en el hardware, como ha sido explicado anteriormente, donde los parches de software todavía dejan pequeñas cantidades de memoria expuestas. Por este motivo, se toman en consideración otras propuestas [2], [10], [17] con el objetivo de mitigar este agujero de seguridad.

Se puede considerar, en primera medida, una garantía de que aquellos datos inaccesibles desde la arquitectura permanezcan inaccesibles desde el nivel de la microarquitectura, debido a que el procesador permite que instrucciones transitorias computen (y filtren) valores sin permisos de acceso. Para este caso se propuso KAISER en 2018 [2], una modificación del kernel con el objetivo de no tener al kernel mapeado en el espacio del usuario. Esta metodología provee una protección básica frente a esta vulnerabilidad, pero (en la arquitectura x86) hay partes privilegiadas de la memoria que todavía requieren ser mapeadas en el espacio del usuario, dejando un espacio residual para que Meltdown pueda seguir atacando.

Otra medida aplicada para mitigar este agujero de seguridad es un cambio en la forma que se gestiona la memoria entre el software de la aplicación y el sistema operativo, incorporando KPTI (Kernel Page Table Isolation),

una tecnología que separa la memoria de forma que aquellos datos privilegiados no se puedan cargar a la memoria caché mientras se ejecuta código de usuario. Sin embargo, el rendimiento varía en función de la frecuencia con la que una aplicación necesita utilizar tales servicios del sistema operativo.

Estas medidas propuestas para mitigar Meltdown no son capaces de evitar variantes más modernas de esta vulnerabilidad, ni tampoco para mitigar ataques de Spectre, por lo que este es considerado un tema de investigación muy importante [19] para combatir estos agujeros de seguridad en la espera de una solución definitiva.

5 CONCLUSIÓN

En el marco del funcionamiento de las vulnerabilidades previamente descritas y la seguridad de los CPUs, sostenemos que es fundamental abordar estas cuestiones ante la necesidad de preservar la integridad y confidencialidad de la información almacenada en los sistemas informáticos. Estos ataques representan amenazas en el nivel más bajo de los dispositivos, afectando a una parte considerable de la población mundial. Es de conocimiento general que los errores de diseño en la arquitectura de los procesadores perdurarán durante un extenso período dado a la inexistencia de una solución definitiva. En ausencia de una resolución inmediata, la esperanza se centra en las futuras generaciones de CPUs que, al reemplazar a las actuales, puedan erradicar estos agujeros de seguridad.

Si bien la causa de estos ataques se atribuye a las técnicas de optimización en el diseño de la arquitectura de los procesadores, se reconoce que algunas de las soluciones propuestas podrían llegar a implicar una significativa ralentización en el rendimiento de los procesadores. En este sentido, ha surgido un amplio debate en la comunidad, incluso dentro de este mismo grupo de investigación, donde diversas opiniones en relación a la noción de sacrificar velocidad de ejecución a cambio de garantizar seguridad han entrado en conflicto. A pesar de ello, sostenemos que estos ataques, que ponen en peligro la información confidencial de los usuarios, deben constituir un foco de investigación crucial en el contexto de la planificación y estructuración de nuevas generaciones de procesadores.

AGRADECIMIENTOS

Queremos expresar nuestro agradecimiento a los investigadores del LINTI (Laboratorio de Investigación en Nuevas Tecnologías Informáticas de La Plata) por su invaluable orientación en la búsqueda de información con respecto a las vulnerabilidades más críticas, sus variantes y el comportamiento de cada una.

Asimismo, queremos agradecer a los integrantes del grupo 10, nuestros compañeros en la cursada de la materia "Organización del Computador 1 (9557)", por asistirnos con respecto al funcionamiento y la historia de la ejecución fuera de orden, y los algoritmos de planificación dinámica.

Por último, les otorgamos nuestro sincero agradecimiento a los docentes Ing. Marchi, Edgardo e Ing. Cervetto, Marcos, y a la colaboradora Jáuregui, Melina, que nos han enseñado y asistido a lo largo de la cursada este cuatrimestre.

REFERENCES

[1] P. Kocher, J. Horn, A. Fogh, D. Genkin, D. Gruss, W. Haas, M. Hamburg, M. Lipp, S. Mangard, T. Prescher, M. Schwarz, Y. Yarom, *"Spectre Attacks: Exploiting Speculative Execution"*, mayo de 2019. Disponible: https://spectreattack.com/spectre.pdf

[2] M. Lipp, M. Schwarz, D. Gruss, T. Prescher, W. Haas, A. Fogh, J. Horn, S. Mangard, P. Kocher, D. Genkin, Y. Yarom and M. Hamburg, *"Meltdown: Reading Kernel Memory from User Space"*, 2018. Disponible: https://meltdownattack.com/meltdown.pdf

[3] J. Sanders, *"Spectre and Meltdown explained: A comprehensive guide for professionals"*, mayo de 2019.

[4] J. Masters, *"Seminario de la universidad de Stanford. Computer systems colloquium. Exploiting modern microarchitectures: Meltdown, Spectre, and other hardware attacks"*, 31 de enero, 2018.

[5] cs.washington.edu, *"CSE 471 - Introduction to Out-of Order Execution"*, 2006.

[6] V. Zyuban, D. Brooks, V. Srinivasan, M. Gschwind, P. Bose, P. N. Strenski, P. G. Emma, *"Integrated Analysis of Power and Performance for Pipelined Microprocessors"*, agosto de 2004.

[7] D. Weber, F. Thomas, L. Gerlach, R. Zhang, and M. Schwarz, *"Reviving Meltdown 3a"*, 2023.

[8] N. Abu-Ghazaleh, D. Ponomarev, D. Evtyushkin, *"How the Spectre and Meltdown hacks really worked"*, 28 de febrero de 2019.

[9] L. Lerman, G. Bontempi, O. Markowitch, *"Side channel attack: an approach based on machine learning"*, febrero de 2011.

[10] C. Canella, J. V. Bulck, M. Schwarz, M. Lipp, B. von Berg, P. Ortner, F. Piessens, D. Evtyushkin, D. Gruss, *"A Systematic Evaluation of Transient Execution Attacks and Defenses"*, noviembre de 2018.

[11] O. Weisse, J. V. Bulck, M. Minkin, D. Genkin, B. Kasikci, F. Piessens, M. Silberstein, R. Strackx, T. F. Wenisch, a. Y. Yarom and W. Ofir, *"Foreshadow: Extracting the Keys to the Intel SGX Kingdom with Transient Out-of-Order Execution"*, agosto de 2018.

[12] M. Minkin, D. Moghimi, M. Lipp, M. Schwarz, J. V. Bulck, D. Genkin, D. Gruss, F. Piessens, B. Sunar, and Y. Yarom, *"Fallout: Reading Kernel Writes From User Space"*, 2019.

[13] S. V. Schaik, D. Genkin, M. Minkinç, A. Kwong, Y. Yarom, *"Cache-Out: Leaking Data on Intel CPUs via Cache Evictions"*, 2020.

[14] J. Van Bulck, D. Moghimi, M. Schwarz, M. Lipp, M. Minkin, D. Genkin, Y. Yarom, B. Sunar, D. Gruss, F. Piessens, *"LVI: Hijacking Transient Execution through Microarchitectural Load Value Injection"*, 2020.

[15] Z. Tong, Z. Zhu, Y. Zhang, Y. Liu, D. Liu, *"Attack detection based on machine learning algorithms for different variants of Spectre attacks and different Meltdown attack implementations"*, 30 de agosto de 2022.

[16] R. M. Tomasulo, *"An Efficient Algorithm for Exploiting Multiple Arithmetic Units"*, 1967.

[17] J. Masters, *"Meltdown and Spectre - Kernel Side-Channel Attacks - CVE-2017-5754 CVE-2017-5753 CVE-2017-5715"*, 3 de enero de 2018.

[18] *"The Chromium Projects: Site Isolation"*, 2021. Disponible: https://www.chromium.org/Home/chromium-security/site-isolation/

[19] M. D. Hill, J. Masters, P. Ranganathan, P. Turner, J. L. Hennessy, *"On the Spectre and Meltdown Processor Security Vulnerabilities"*, 2019.